Kaggle Competition - Online News Popularity

Team DUN - Dom, Uwe, Neiti

March 2016

Abstract

In this report we present a detailed description of the methods used for the in-class Kaggle Online News Popularity competition. Our final model is based on a popular multi-layer approach, inspired by the winners Gilberto Titericz & Stanislav Semenov of the Otto Group Product Classification challenge on Kaggle. The solution is based on a two-fold approach: Feature engineering and machine learning techniques. The latter consists of a 3-layer learning architecture as shown in the picture below. The result on the private Kaggle scoreboard was 52.83% accuracy.

Introduction

The Online News Popularity dataset provided by Kelwin Fernandes et al. consists of news articles published by Mashable (www.mashable.com). The raw data is split into 30,000 * 62 training set and 9,644 * 61 test set. For illustration purpose, but also to avoid compiling issues we will use merely a small subsample in this report. The goal is to predict individual popularity, i.e softmax classification. The class value to be predicted was originally based on the number of times the article was shared elsewhere (e.g. on social networks).This continuously valued variable was removed from the dataset and replaced by a label with 5 levels: unpopular (1), moderately popular (2), popular (3), super popular (4), viral (5).

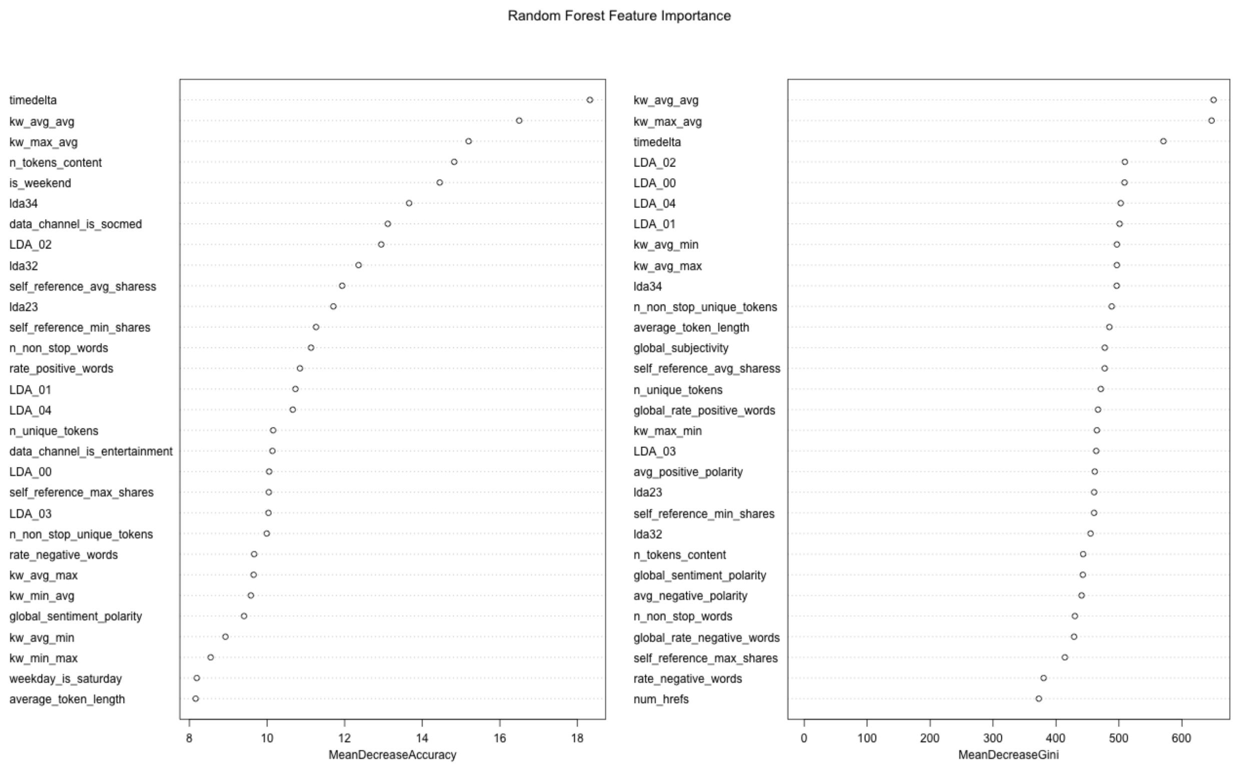
Data and considered models

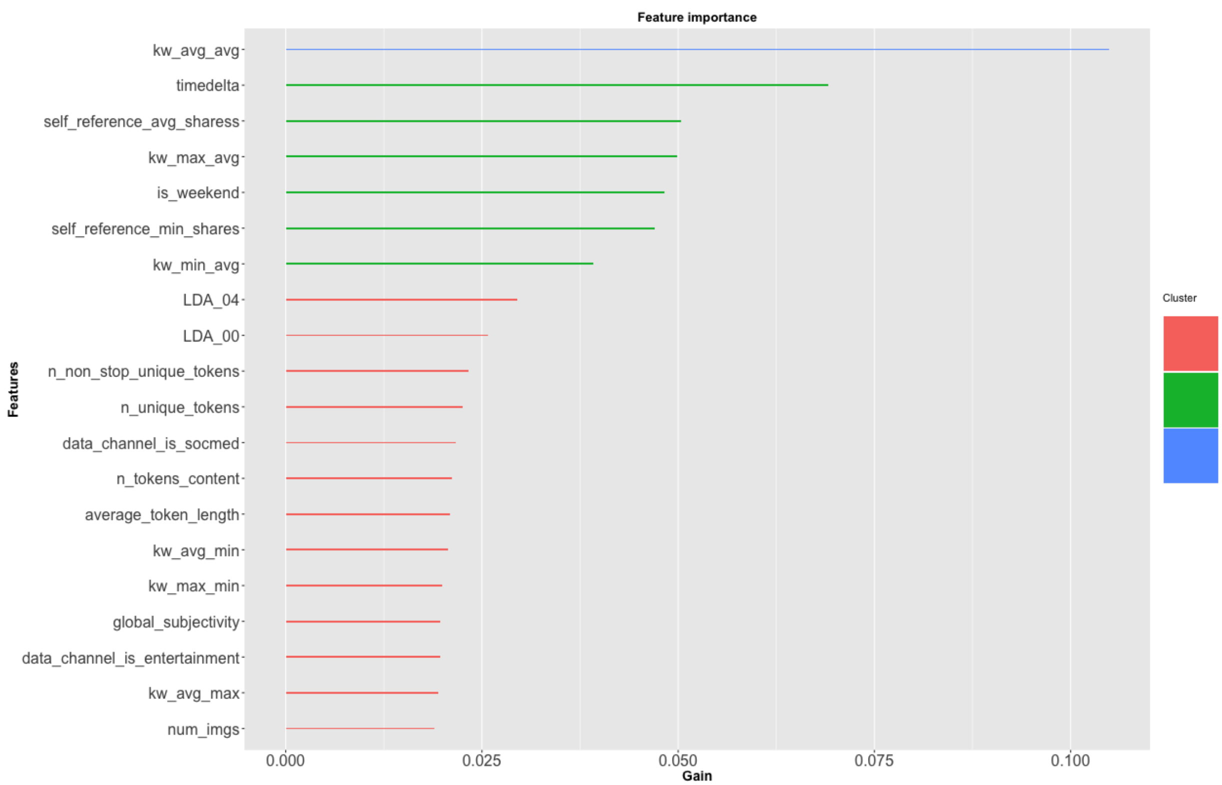
Created features

After testing various interactive variables we found these to be important:

- LDA_02 / LDA_03
- LDA_03 / LDA_02
- LDA_03 / LDA_04

Plotting these features against article popularity shows some significant differences between different classes. However, the improvement in the first layer was too small and we dropped them. These features were ranked important by Random Forests but not by Xgboost.





We also explored several binary features (1 if greater than zero):

- min keyword
- number of self references
- number of images
- number of videos

We also considered logs of kw_avg_avg and kw_max_avg without any improvement.

Titles and Text Mining

We also extracted titles from the url column. We discovered that n_tokens_title feature (number of words in title) is not always correct. Creating our own title length feature did not contribute to accuracy. We looked at most common words in titles, and after excluding connectors and articles we explored words like "google", "facebook", "video" etc. We were looking particularly at word frequency among class labels and tried to find common words that had significantly different class distribution. However, most common words appear in less than 1000 titles (<0.03) so they did not end up contributing to accuracy.

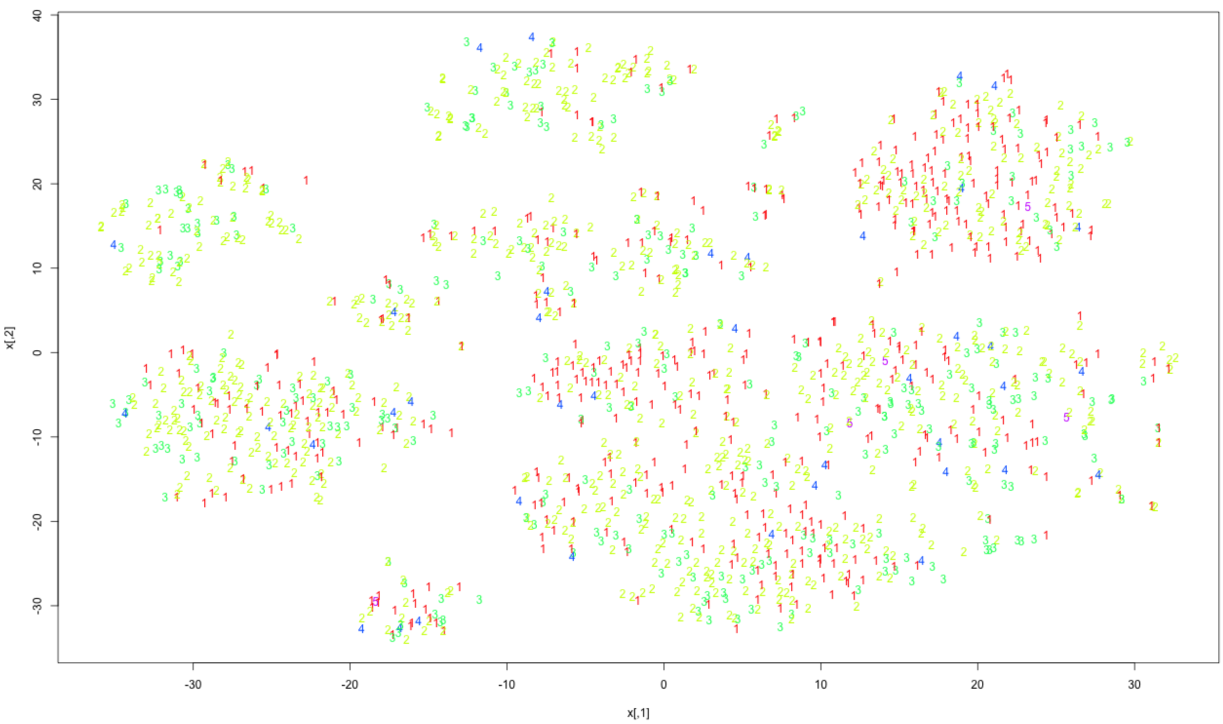
Feature selection

We tried different models on subset of features without improvement.

- top n features given by Random Forest
- top n features given by XGBoost
- top n features given by Gibbs Sampler (mombf package)

Dimensionality reduction

- PCA: top n eigenvalues
- t-Distributed Stochastic Neighbor Embedding (tsne package). The plot below shows that the data will not be easily classified by a k-nn algorithm.

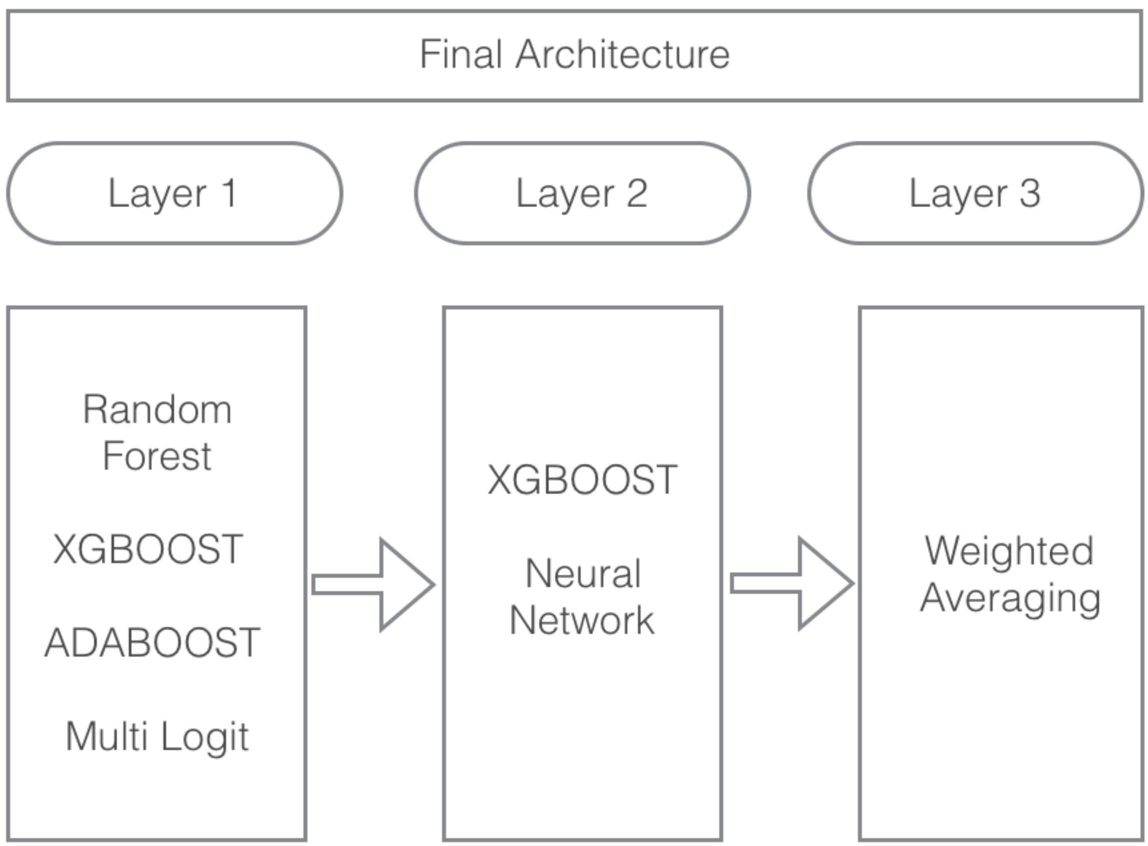


Considered models

K-Nearest Neighbors metafeatures ($K \in 1, 2, 4, 8, 16, 32, 64, 128, 256$). Cross validated accuracy is below 49%. Poor performance in correlation with poor performance of tSNE.

GBM (gbm package) - already did adaBoost and xgboost. Accuracy bellow 50.5%.

Method



In the first layer we used five folds to create metafeatures using 4 different classifiers. The metafeatures for the test set were created by using all available training data. After a thorough investigation of different types of classifiers for the first level we decided to stick with the following list:

- Random Forest (1000 trees, probability output)
- Xgboost (250 rounds, softprob, optimised using grid search in caret package)
- AdaBoost (250 rounds, maboost package)
- Multinomial logistic regression (glmnet package)

In the second layer we again optimize parameters of Xgboost using only metafeatures created in the first layer. We do the same for h2o Neural Net. Both models are then trained using the optimal parameters and with the softprob output.

In the third layer we use arithmetic/geometric averaging to combine Xgboost and Neural Networks to produce the final classification. Our final model is based on a quite popular multi-layered approach. In the first layer we used fixed five folds to create Metafeatures using 4 different classifiers. Metafeatures for the test set were created by training on the full train set. We considered several different classifiers for the first level and in the end kept the following: Random Forest (1000 trees, probability output) Xgboost (250 rounds, softprob, optimised using grid search in caret package) AdaBoost (250 rounds, maboost package) Multinomial logistic (glmnet package)

Accuracy tables

Raw train set is the original set excluding id and url columns. Cross validation is always performed on the same 5 folds.

Model	Data used	CV accuracy	Public	Private
Xgboost, 250 rounds, optimized	Raw	52.39	54.037	53.044
Geometric average of layer 1 metafeatures. 0.3 RF 0.7 Xgb	Meta RF and Xgb	/	54.814	53.033
Xgboost, 250, optimized on raw	Raw + meta	52.27	54.503	52.889
Full 3-layer, geom average (0.76 Xgb – 0.24 h2o neural net	Full meta (RF, Xgb, Ada, Logistic)	52.73	55.901	52.833
Random Forest, 500 trees	Raw	51.9	54.037	52.333
Random Forest, 100 trees	Raw	/	54.969	51.033

Instructions for reproducing the method

1 Load data

```
trainfull <- read.csv("news_popularity_training.csv", stringsAsFactors=FALSE)
test  <- read.csv("news_popularity_test.csv",  stringsAsFactors=FALSE)
```

Store the test set id column as it is needed later for creating the submission file.

```
idcol <- test[,1]
```

Remove id and url columns.

```
trainfull <- trainfull[,-c(1,2)]
test <- test[,-c(1,2)]
```

Transform target variable into a factor.

```
trainfull$popularity <- as.factor(trainfull$popularity)
```

Label frequency of full train set:

```
round(table(trainfull$popularity)/nrow(trainfull),3)
```

```
##
##     1     2     3     4     5
## 0.316 0.459 0.190 0.033 0.002
```

As mentioned above we will subsample just 1347 rows from the trainset. Due to this small sample size, we increase the proportion of label 4 and 5 to ensure proper functionality of the models used later. As a matter of fact, having too few labels of one kind could create problems in the cross validation part.

```
library(dplyr)
set.seed(2949)
a1 <- sample_n(filter(trainfull,popularity==1), size=400)
a2 <- sample_n(filter(trainfull,popularity==2), size=600)
a3 <- sample_n(filter(trainfull,popularity==3), size=200)
a4 <- sample_n(filter(trainfull,popularity==4), size=100)
a5 <- sample_n(filter(trainfull,popularity==5), size=47)
train <- rbind(a1,a2,a3,a4,a5)
```

```
round(table(train$popularity)/nrow(train),3)
```

```
##
##     1     2     3     4     5
## 0.297 0.445 0.148 0.074 0.035
```

2 Creating metafeatures for layer 1

Random Forest layer 1

Next, we run random forest and create first set of metafeatures. Our function splits data into 5 folds, trains on 4 and predicts on 1. We used 1000 trees per fold on the full trainset.

```
rfmetatrain <- DUN::fmeta.rf(train, trees = 30, verbose = FALSE)
```

```
## [1] "No test set, cross validating train set."
## [1] "fold"
## [1] 1
## [1] "fold"
## [1] 2
## [1] "fold"
## [1] 3
## [1] "fold"
## [1] 4
## [1] "fold"
## [1] 5
```

```
rfmetatest <- DUN::fmeta.rf(train,test = test,trees = 30, verbose = FALSE)
```

```
## [1] "test set loaded, learning on train and predicting on test"
```

Head:

```
head(rfmetatrain)
```

```
##        rfp1      rfp2       rfp3       rfp4       rfp5 rflabel
## 1 0.5000000 0.3666667 0.06666667 0.03333333 0.03333333       1
```

```
## 2 0.1666667 0.4333333 0.23333333 0.03333333 0.13333333       2
## 3 0.1333333 0.5000000 0.23333333 0.10000000 0.03333333       2
## 4 0.4000000 0.5000000 0.06666667 0.00000000 0.03333333       2
## 5 0.7666667 0.2000000 0.00000000 0.03333333 0.00000000       1
## 6 0.3000000 0.6000000 0.10000000 0.00000000 0.00000000       2
```

```r
head(rfmetatest)
```

```
##        rfp1      rfp2       rfp3       rfp4       rfp5 rflabel
## 1 0.2333333 0.5333333 0.10000000 0.13333333 0.00000000       2
## 2 0.4000000 0.3333333 0.16666667 0.06666667 0.03333333       1
## 3 0.2333333 0.3000000 0.20000000 0.20000000 0.06666667       2
## 4 0.6666667 0.3000000 0.03333333 0.00000000 0.00000000       1
## 5 0.3666667 0.5000000 0.10000000 0.03333333 0.00000000       2
## 6 0.1333333 0.4000000 0.30000000 0.03333333 0.13333333       2
```

Notice that the sixth column is the predicted label. This will not be used as a metafeature later on.

```r
rfmetatrain <- rfmetatrain[,1:5]
rfmetatest <- rfmetatest[,1:5]
```

Xgboost layer 1

Now create meta features with Xgboost. These are created on the same five folds. Xgboost parameters were optimised using caret package grid search. Please refer to the vignette appendix.

```r
xgmetatrain <- DUN::fmeta.xgb(train,nrounds = 30, verbose = 0)
```

```
## [1] "No test set, cross validating train set."
## [1] "fold"
## [1] 1
## [1] "fold"
## [1] 2
## [1] "fold"
## [1] 3
## [1] "fold"
## [1] 4
## [1] "fold"
## [1] 5
## [1] "fold looping complete"
```

```r
xgmetatest <- DUN::fmeta.xgb(train, test = test, nrounds = 30, verbose = 0)
```

```
## [1] "test set loaded, learning on train and predicting on test"
```

Print head:

```r
head(xgmetatrain)
```

```
##       xgbp1     xgbp2     xgbp3     xgbp4     xgbp5
## 1 0.3067535 0.1128550 0.1408965 0.1122514 0.4101729
## 2 0.2023819 0.3287976 0.2788310 0.2161369 0.3141597
## 3 0.1962564 0.1252225 0.1204360 0.1646375 0.3873491
## 4 0.3747770 0.1736327 0.1160845 0.1437512 0.1519284
## 5 0.2967321 0.4243000 0.1248465 0.1097733 0.1703701
## 6 0.1736208 0.2461592 0.1941922 0.1410360 0.1359991
```

```r
head(xgmetatest)
```

```
##       xgbp1     xgbp2     xgbp3     xgbp4     xgbp5
## 1 0.2615191 0.3318696 0.1522494 0.1325101 0.1218518
## 2 0.2603344 0.3052537 0.1718682 0.1429314 0.1196121
## 3 0.2708087 0.1722364 0.1905442 0.2429086 0.1235021
## 4 0.4219496 0.2118131 0.1265353 0.1237301 0.1159721
## 5 0.2912273 0.3036822 0.1527278 0.1291661 0.1231965
## 6 0.2095685 0.2663572 0.1956177 0.1852869 0.1431697
```

maboost(AdaBoost) layer 1

Next create metafeatures on same folds with AdaBoost. We use maboost package which enables multiclass AdaBoost.

```r
mabmetatrain <- DUN::fmeta.mab(train,rounds = 30)
```

```
## [1] "No test set, cross validating train set."
## [1] "fold"
## [1] 1
## [1] "Multiclass boosting is selected"
## [1] "fold"
## [1] 2
## [1] "Multiclass boosting is selected"
## [1] "fold"
## [1] 3
## [1] "Multiclass boosting is selected"
## [1] "fold"
## [1] 4
## [1] "Multiclass boosting is selected"
## [1] "fold"
## [1] 5
## [1] "Multiclass boosting is selected"
```

```r
mabmetatest <- DUN::fmeta.mab(train,test = test, rounds = 30)
```

```
## [1] "test set loaded, learning on train and predicting on test"
## [1] "Multiclass boosting is selected"
```

Print head:

```r
head(mabmetatrain)
```

```
##       mabp1     mabp2      mabp3      mabp4      mabp5        mabF1
## 1 0.4572706 0.3733595 0.11009286 0.05927710 0.00000000 0.0005181254
## 2 0.2713871 0.3875300 0.18977748 0.09426166 0.05704379 0.0002834809
## 3 0.2562182 0.3443722 0.36286118 0.03654840 0.00000000 0.0002848129
## 4 0.2835928 0.4755093 0.18364732 0.04109925 0.01615134 0.0003152426
## 5 0.5118863 0.3302087 0.07616845 0.08173654 0.00000000 0.0005800095
## 6 0.4571852 0.3718660 0.06554734 0.07943201 0.02596946 0.0005082084
##          mabF2        mabF3        mabF4        mabF5
## 1 0.0004230472 1.247443e-04 6.716586e-05 0.000000e+00
## 2 0.0004047995 1.982346e-04 9.846225e-05 5.958584e-05
## 3 0.0003828052 4.033576e-04 4.062731e-05 0.000000e+00
## 4 0.0005285775 2.041429e-04 4.568605e-05 1.795388e-05
## 5 0.0003741538 8.630516e-05 9.261426e-05 0.000000e+00
## 6 0.0004133674 7.286262e-05 8.829686e-05 2.886772e-05
```

```r
head(mabmetatest)
```

```
##        mabp1     mabp2      mabp3      mabp4 mabp5        mabF1
## 1 0.47600402 0.2927019 0.18587365 0.04542045     0 3.594168e-04
## 2 0.41157420 0.4208645 0.10616159 0.06139968     0 3.107677e-04
## 3 0.33245581 0.2103433 0.21378307 0.24341781     0 2.510277e-04
## 4 0.62062616 0.3579223 0.02145151 0.00000000     0 4.686167e-04
## 5 0.34835601 0.5385614 0.09163106 0.02145151     0 2.630335e-04
## 6 0.03478679 0.3939962 0.33691926 0.23429775     0 2.626649e-05
##          mabF2        mabF3        mabF4 mabF5
## 1 0.0002210107 1.403478e-04 3.429566e-05     0
## 2 0.0003177826 8.015952e-05 4.636111e-05     0
## 3 0.0001588241 1.614214e-04 1.837977e-04     0
## 4 0.0002702567 1.619741e-05 0.000000e+00     0
## 5 0.0004066520 6.918795e-05 1.619741e-05     0
## 6 0.0002974951 2.543979e-04 1.769114e-04     0
```

The first 5 columns are probability class estimates. Columns 6:10 are ensamble averages produced by selecting type= "F" in predict.maboost. These have 0.99 correlation with class probabilites and we use only columns 1:5 further on.

```r
mabmetatrain <- mabmetatrain[,1:5]
mabmetatest <- mabmetatest[,1:5]
```

Multinomial Logistic layer 1

Next we create metafeatures with multinomial logistic regression. We use glmnet package.

```r
glmmetatrain <- DUN::fmeta.glm(train)
```

```
## [1] "No test set, cross validating train set."
## [1] "fold"
## [1] 1
```

```
## [1] "fold"
## [1] 2
## [1] "fold"
## [1] 3
## [1] "fold"
## [1] 4
## [1] "fold"
## [1] 5
```

```
glmmetatest <- DUN::fmeta.glm(train,test = test)
```

```
## [1] "test set loaded, learning on train and predicting on test"
```

Print head:

```
head(glmmetatrain)
```

```
##       glmp1     glmp2     glmp3      glmp4      glmp5
## 1 0.3821564 0.3901132 0.1310679 0.06553396 0.03112863
## 2 0.3140781 0.4021883 0.1638895 0.08194469 0.03789941
## 3 0.3169875 0.4576818 0.1316698 0.06317013 0.03049089
## 4 0.4652534 0.3584626 0.1042513 0.04887125 0.02316145
## 5 0.3899101 0.3780845 0.1335283 0.06676414 0.03171297
## 6 0.3213549 0.4529156 0.1334084 0.06285855 0.02946264
```

```
head(glmmetatest)
```

```
##       glmp1     glmp2     glmp3      glmp4      glmp5
## 1 0.1700220 0.5101357 0.1843472 0.09217354 0.04332155
## 2 0.3138844 0.4085017 0.1600080 0.08000397 0.03760186
## 3 0.3562509 0.3501097 0.1692447 0.08462227 0.03977246
## 4 0.4046689 0.3767434 0.1259872 0.06299354 0.02960696
## 5 0.3121061 0.4352565 0.1456124 0.07280615 0.03421888
## 6 0.1653496 0.4920719 0.1974517 0.09872577 0.04640110
```

3 Training layer 2 models

First combine metafeatures from the first layer which are used for training models in the second layer.

```
metatrain2 <- data.frame(train$popularity,rfmetatrain,xgmetatrain,mabmetatrain)
names(metatrain2)[1]<-"popularity"
metatest2 <- data.frame(rfmetatest,xgmetatest,mabmetatest)
```

Xgboost layer 2

```
library(xgboost)

param <- list("objective"="multi:softprob",
```

```
               "eval_metric"="merror",
               "num_class"=5,
               "booster"="gbtree",
               "eta"=0.01,
               "max_depth"=6,
               "subsample"=0.8,
               "colsample_bytree"=0.6)

y<-as.integer(metatrain2$popularity)-1

bst <- xgboost(params = param, data = as.matrix(metatrain2[,-1]),label = y,
               nrounds = 30, verbose = 0)

preds <- predict(bst,as.matrix(metatest2))
xgbprob <- matrix(preds,ncol=5,byrow=TRUE)
```

h2o Neural networks layer 2

```
library(h2o)
localH2O = h2o.init(nthreads=-1)
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     /var/folders/7x/9xd00gzx2h973_8dzyl2w22w0000gn/T//RtmpZJjv0g/h2o_uhoenig_started_from_r.out
##     /var/folders/7x/9xd00gzx2h973_8dzyl2w22w0000gn/T//RtmpZJjv0g/h2o_uhoenig_started_from_r.err
##
##
## Starting H2O JVM and connecting: .. Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         2 seconds 205 milliseconds
##     H2O cluster version:        3.8.1.3
##     H2O cluster name:           H2O_started_from_R_uhoenig_gmk639
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   0.12 GB
##     H2O cluster total cores:    4
##     H2O cluster allowed cores:  4
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     R Version:                  R version 3.2.3 (2015-12-10)
```

```
data_train_h <- as.h2o(metatrain2,destination_frame = "h2o_data_train")
```

```
##
  |
  |                                                                   |   0%
```

```
  |
  |=================================================================| 100%
```

```r
data_test_h <- as.h2o(metatest2,destination_frame = "h2o_data_test")
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```r
y <- "popularity"
x <- setdiff(names(data_train_h), y)

data_train_h[,y] <- as.factor(data_train_h[,y])

model <- h2o.deeplearning(x = x,
                          y = y,
                          training_frame = data_train_h,
                          #validation_frame = data_test_h,
                          distribution = "multinomial",
                          activation = "RectifierWithDropout",
                          hidden = c(20,20),
                          input_dropout_ratio = 0.2,
                          l1 = 1e-7,
                          epochs = 10)
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```r
#
pred <- h2o.predict(model, newdata = data_test_h)
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```r
nnprob <- as.matrix(pred[,2:6])

h2o.shutdown()
```

```
## Are you sure you want to shutdown the H2O instance running at http://localhost:54321/ (Y/N)?
```

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

4 Third and final layer

In this layer we perform arithmetic and geometric averaging of second layer model predictions.

Head of 2nd layer probability class estimates.

```
head(xgbprob)
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 0.2392463 0.2378115 0.1831146 0.1710245 0.1688032
## [2,] 0.2214223 0.2279395 0.2030976 0.1748551 0.1726855
## [3,] 0.2029420 0.2325867 0.1967959 0.1941885 0.1734868
## [4,] 0.2382281 0.2292276 0.1892001 0.1689848 0.1743594
## [5,] 0.2598159 0.2264598 0.1823115 0.1666395 0.1647732
## [6,] 0.1877049 0.2404355 0.2173201 0.1768157 0.1777239
```

```
head(nnprob)
```

```
##             p1        p2        p3         p4          p5
## [1,] 0.2636335 0.4793931 0.1883500 0.05312270 0.015500684
## [2,] 0.3090568 0.4748807 0.1672684 0.03776690 0.011027214
## [3,] 0.2612516 0.4730412 0.1658626 0.07771487 0.022129740
## [4,] 0.3270305 0.4722246 0.1708941 0.02326315 0.006587685
## [5,] 0.2823760 0.5265641 0.1518469 0.03045962 0.008753415
## [6,] 0.2576321 0.4443548 0.1871802 0.08670306 0.024129946
```

Arithmetic average

Optimal weights were computed using the avg.arit function found in the appendix.

```
arit <- function(vec){
  pred <- which.max(vec[1:5]*(0.76) + vec[6:10]*(0.24))
  return(pred)
}

combined <- cbind(xgbprob,nnprob)
finallabelsarit <- apply(combined,1,arit)
head(finallabelsarit)
```

```
## [1] 2 2 2 2 2 2
```

Geometric average

Optimal weights were computed using the avg.geom function found in the appendix.

```
geom <- function(vec){
  pred <- which.max(vec[1:5]^(0.76) * vec[6:10]^(0.24))
  return(pred)
}

combined <- cbind(xgbprob,nnprob)
finallabelsgeom <- apply(combined,1,geom)
head(finallabelsgeom)
```

```
## [1] 2 2 2 2 2 2
```

Creating submission files

```
submissionarit <- data.frame(id = idcol)
submissionarit$popularity <- finallabelsarit
write.csv(submissionarit, file = "arithmetic_r_gsenews.csv", row.names=FALSE)

submissiongeom <- data.frame(id = idcol)
submissiongeom$popularity <- finallabelsgeom
write.csv(submissiongeom, file = "geometric_r_gsenews.csv", row.names=FALSE)
```

Appendix

Optimising Xgboost

Below code is not executed for this document as it takes some time to run. On the full train set it can easily run for more than 3 hours.

```
library(caret)
library(xgboost)
library(readr)
library(dplyr)
library(tidyr)

df_train <- train

# set up the cross-validated hyper-parameter search
xgb_grid_1 = expand.grid(
  nrounds = c(150,200,250,300),
  eta = c(0.03, 0.01, 0.001),
  max_depth = c(2, 4, 6),
  gamma = c(0,1),
  colsample_bytree = c(0.6, 0.8, 1),    #default=1
  min_child_weight = 1      #default=1
)


# pack the training control parameters
xgb_trcontrol_1 = trainControl(
  method = "cv",
  number = 5,
  verboseIter = TRUE,
  returnData = FALSE,
  returnResamp = "all",                       # save losses across all models
  classProbs = TRUE,                     # set to TRUE for AUC to be computed
  #summaryFunction = twoClassSummary,
  summaryFunction = defaultSummary,
  allowParallel = TRUE
)
```

```r
z <- unlist(lapply("Label", paste0, df_train$popularity ))

xgb_train_1 = train(
  x = as.matrix(df_train %>%
                  select(-popularity)),
  y = as.factor(z),
  trControl = xgb_trcontrol_1,
  tuneGrid = xgb_grid_1,
  method = "xgbTree"
)
```

Arithmetic and Geometric averaging

Arithmetic averaging

```r
arithmeticaverage <- DUN::avg.arit(xgmetatrain,rfmetatrain, label = train$popularity, iter = 11)
```

```
## [1] 0
## [1] 0.1
## [1] 0.2
## [1] 0.3
## [1] 0.4
## [1] 0.5
## [1] 0.6
## [1] 0.7
## [1] 0.8
## [1] 0.9
## [1] 1
```

```r
arithmeticaverage
```

```
##       [,1] [,2]      [,3]
##  [1,]  0.2  0.8 0.4357832
##  [2,]  0.5  0.5 0.4357832
##  [3,]  0.4  0.6 0.4350408
##  [4,]  0.3  0.7 0.4342984
##  [5,]  0.0  1.0 0.4320713
##  [6,]  0.1  0.9 0.4320713
##  [7,]  0.6  0.4 0.4305865
##  [8,]  0.7  0.3 0.3845583
##  [9,]  0.8  0.2 0.3221975
## [10,]  0.9  0.1 0.2613215
## [11,]  1.0  0.0 0.2078693
```

Geometric averaging

```r
geometricaverage <- DUN::avg.geom(xgmetatrain,rfmetatrain,label = train$popularity, iter = 11)
```

```
## [1] 0
## [1] 0.1
```

```
## [1] 0.2
## [1] 0.3
## [1] 0.4
## [1] 0.5
## [1] 0.6
## [1] 0.7
## [1] 0.8
## [1] 0.9
## [1] 1
```

```
geometricaverage
```

```
##       [,1] [,2]      [,3]
##  [1,]  0.4  0.6 0.4380104
##  [2,]  0.3  0.7 0.4365256
##  [3,]  0.2  0.8 0.4357832
##  [4,]  0.5  0.5 0.4335561
##  [5,]  0.0  1.0 0.4320713
##  [6,]  0.1  0.9 0.4320713
##  [7,]  0.6  0.4 0.4105419
##  [8,]  0.7  0.3 0.3704529
##  [9,]  0.8  0.2 0.3125464
## [10,]  0.9  0.1 0.2724573
## [11,]  1.0  0.0 0.2078693
```

Cross validation function example

Below is the example of our Random forest 5-fold cross validation function.

```
cv <- DUN::cv.rf(train = train, tree = 30)
```

```
## [1] "fold"
## [1] 1
## [1] NA NA NA NA NA
## ntree      OOB      1      2      3      4      5
##     1:  67.65% 58.40% 62.01% 83.82%100.00% 88.89%
##     2:  66.30% 60.50% 59.07% 84.16% 91.67% 78.57%
##     3:  65.36% 60.83% 55.46% 83.90% 93.22% 95.65%
##     4:  65.96% 61.85% 56.68% 85.38% 92.54% 84.62%
##     5:  62.94% 62.46% 50.70% 82.84% 90.00% 90.00%
##     6:  62.70% 62.37% 50.78% 80.14% 86.84% 96.97%
##     7:  63.42% 64.17% 48.59% 84.25% 92.31%100.00%
##     8:  61.21% 62.18% 46.14% 84.00% 87.34% 94.59%
##     9:  61.14% 61.39% 45.44% 86.75% 89.87% 92.11%
##    10:  61.31% 63.84% 44.42% 85.16% 89.87% 94.74%
##    11:  61.27% 61.44% 44.96% 85.99% 91.14%100.00%
##    12:  60.39% 61.13% 42.77% 84.91% 95.00%100.00%
##    13:  59.12% 61.44% 40.59% 84.28% 92.50% 97.37%
##    14:  58.27% 60.94% 39.54% 84.38% 90.00% 94.74%
##    15:  58.68% 62.81% 38.62% 85.62% 91.25% 94.74%
##    16:  59.24% 63.12% 38.62% 88.12% 95.00% 89.47%
##    17:  58.63% 62.19% 37.50% 91.25% 92.50% 86.84%
```

```
##    18:  59.74% 64.38% 38.54% 90.00% 92.50% 92.11%
##    19:  59.55% 63.44% 37.71% 91.88% 93.75% 94.74%
##    20:  58.16% 61.56% 37.08% 90.00% 92.50% 89.47%
##    21:  59.18% 63.44% 37.50% 90.00% 92.50% 97.37%
##    22:  57.88% 61.25% 36.46% 90.62% 92.50% 89.47%
##    23:  58.07% 61.56% 35.83% 90.62% 95.00% 94.74%
##    24:  57.88% 60.31% 36.88% 90.62% 93.75% 89.47%
##    25:  58.07% 62.19% 36.46% 88.75% 93.75% 92.11%
##    26:  57.61% 61.56% 35.21% 91.25% 93.75% 89.47%
##    27:  57.51% 60.31% 35.00% 92.50% 93.75% 94.74%
##    28:  58.07% 60.00% 36.25% 92.50% 95.00% 94.74%
##    29:  58.16% 62.19% 35.00% 92.50% 95.00% 94.74%
##    30:  58.53% 62.19% 35.62% 92.50% 95.00% 97.37%
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1 29 30 11  6  0
##          2 48 81 27 10  7
##          3  2  6  2  3  2
##          4  1  2  0  1  0
##          5  0  1  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.4201
##                  95% CI : (0.3604, 0.4815)
##     No Information Rate : 0.4461
##     P-Value [Acc > NIR] : 0.8211
##
##                   Kappa : 0.064
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.3625   0.6750 0.050000 0.050000 0.000000
## Specificity            0.7513   0.3826 0.943231 0.987952 0.996154
## Pos Pred Value         0.3816   0.4682 0.133333 0.250000 0.000000
## Neg Pred Value         0.7358   0.5938 0.850394 0.928302 0.966418
## Prevalence             0.2974   0.4461 0.148699 0.074349 0.033457
## Detection Rate         0.1078   0.3011 0.007435 0.003717 0.000000
## Detection Prevalence   0.2825   0.6431 0.055762 0.014870 0.003717
## Balanced Accuracy      0.5569   0.5288 0.496616 0.518976 0.498077
## [1] "fold"
## [1] 2
## [1] 0.4200743        NA        NA        NA        NA
## ntree      OOB      1      2      3      4      5
##     1:  61.50% 57.02% 50.89% 81.67% 92.59% 90.00%
##     2:  63.31% 60.82% 52.25% 80.43% 92.45% 94.44%
##     3:  62.91% 59.83% 50.28% 84.82% 90.91% 96.30%
##     4:  64.07% 62.31% 49.88% 88.37% 92.96% 96.67%
##     5:  64.58% 62.94% 51.75% 83.80% 95.89% 96.67%
##     6:  63.71% 65.00% 48.65% 84.35% 93.51% 93.94%
```

```
##      7:  64.13% 61.18% 52.52% 83.55% 93.67% 91.18%
##      8:  61.13% 59.94% 47.30% 81.70% 92.41% 94.29%
##      9:  61.94% 61.59% 47.54% 81.82% 96.20% 91.67%
##     10:  60.09% 58.73% 45.86% 80.89% 92.50% 94.59%
##     11:  60.02% 60.82% 42.83% 83.54% 96.25% 94.59%
##     12:  61.62% 61.76% 44.65% 86.08% 98.75% 94.59%
##     13:  60.28% 60.62% 43.42% 84.28% 96.25% 94.59%
##     14:  59.94% 60.31% 42.08% 86.16% 97.50% 94.59%
##     15:  59.42% 59.38% 42.50% 84.38% 95.00% 94.59%
##     16:  60.07% 61.56% 42.29% 85.62% 93.75% 94.59%
##     17:  59.70% 62.50% 40.62% 85.62% 95.00% 94.59%
##     18:  59.61% 60.94% 40.83% 86.88% 96.25% 94.59%
##     19:  60.63% 61.88% 41.46% 90.62% 93.75% 97.30%
##     20:  59.42% 61.56% 38.96% 90.00% 95.00% 97.30%
##     21:  59.89% 64.06% 38.54% 91.25% 91.25% 97.30%
##     22:  59.89% 62.81% 39.17% 91.25% 92.50% 97.30%
##     23:  58.59% 62.19% 37.50% 88.75% 93.75% 94.59%
##     24:  59.05% 61.25% 38.54% 88.75% 96.25% 97.30%
##     25:  58.22% 59.69% 36.88% 90.62% 97.50% 97.30%
##     26:  57.75% 59.69% 35.42% 91.88% 97.50% 97.30%
##     27:  58.96% 61.88% 36.88% 93.12% 93.75% 97.30%
##     28:  59.80% 61.56% 38.54% 93.12% 96.25% 97.30%
##     29:  58.68% 60.00% 37.71% 92.50% 95.00% 94.59%
##     30:  57.94% 60.00% 35.83% 93.12% 95.00% 94.59%
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1 32 23  2  2  0
##          2 44 92 34 17  7
##          3  2  5  4  1  1
##          4  2  0  0  0  0
##          5  0  0  0  0  2
##
## Overall Statistics
##
##                Accuracy : 0.4815
##                  95% CI : (0.4205, 0.5429)
##     No Information Rate : 0.4444
##     P-Value [Acc > NIR] : 0.1224
##
##                   Kappa : 0.1471
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.4000   0.7667  0.10000 0.000000 0.200000
## Specificity            0.8579   0.3200  0.96087 0.992000 1.000000
## Pos Pred Value         0.5424   0.4742  0.30769 0.000000 1.000000
## Neg Pred Value         0.7725   0.6316  0.85992 0.925373 0.970149
## Prevalence             0.2963   0.4444  0.14815 0.074074 0.037037
## Detection Rate         0.1185   0.3407  0.01481 0.000000 0.007407
## Detection Prevalence   0.2185   0.7185  0.04815 0.007407 0.007407
```

```
## Balanced Accuracy      0.6289   0.5433  0.53043 0.496000 0.600000
## [1] "fold"
## [1] 3
## [1] 0.4200743 0.4814815        NA        NA        NA
## ntree      OOB      1      2      3      4      5
##     1:  67.40% 70.40% 56.42% 77.94% 92.59% 88.89%
##     2:  68.97% 72.59% 57.14% 82.47% 92.50% 94.12%
##     3:  66.29% 67.63% 55.10% 84.17% 85.71% 89.29%
##     4:  68.53% 69.81% 56.51% 87.60% 91.67% 90.62%
##     5:  68.27% 69.66% 55.81% 89.05% 91.04% 85.29%
##     6:  67.74% 66.56% 54.85% 90.48% 94.29% 91.67%
##     7:  65.21% 65.71% 51.42% 86.49% 90.79% 94.44%
##     8:  65.87% 65.93% 52.16% 87.92% 93.51% 91.67%
##     9:  65.28% 64.47% 50.75% 88.89% 96.20% 91.89%
##    10:  64.60% 64.89% 48.73% 91.03% 92.50% 92.11%
##    11:  63.74% 64.58% 47.79% 87.97% 93.75% 92.11%
##    12:  63.40% 64.38% 46.74% 89.24% 95.00% 89.47%
##    13:  62.85% 64.69% 45.70% 89.31% 92.50% 89.47%
##    14:  61.77% 62.19% 44.56% 89.94% 92.50% 92.11%
##    15:  61.87% 59.69% 45.62% 93.12% 95.00% 84.21%
##    16:  60.85% 57.50% 45.00% 90.00% 95.00% 94.74%
##    17:  61.97% 60.94% 44.17% 92.50% 95.00% 97.37%
##    18:  61.87% 61.88% 44.17% 90.00% 97.50% 92.11%
##    19:  60.30% 57.81% 43.54% 90.00% 96.25% 92.11%
##    20:  59.37% 58.44% 41.04% 91.25% 93.75% 92.11%
##    21:  59.18% 55.94% 41.04% 92.50% 96.25% 97.37%
##    22:  60.95% 59.06% 43.33% 91.88% 95.00% 97.37%
##    23:  59.37% 58.13% 40.21% 91.25% 97.50% 97.37%
##    24:  58.44% 58.75% 38.12% 91.25% 97.50% 92.11%
##    25:  58.26% 59.38% 38.33% 89.38% 96.25% 89.47%
##    26:  58.35% 57.19% 37.92% 94.38% 96.25% 94.74%
##    27:  58.16% 58.75% 37.08% 91.88% 97.50% 94.74%
##    28:  59.74% 60.62% 39.17% 93.12% 96.25% 94.74%
##    29:  57.61% 59.06% 35.00% 95.00% 96.25% 92.11%
##    30:  57.98% 58.44% 36.88% 93.75% 95.00% 92.11%
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1 20 21 10  2  2
##          2 55 96 27 17  4
##          3  3  2  2  0  1
##          4  1  1  0  0  2
##          5  1  0  1  1  0
##
## Overall Statistics
##
##                Accuracy : 0.4387
##                  95% CI : (0.3785, 0.5002)
##     No Information Rate : 0.4461
##     P-Value [Acc > NIR] : 0.6196
##
##                   Kappa : 0.0695
##  Mcnemar's Test P-Value : NA
```

```
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.25000   0.8000 0.050000  0.00000  0.00000
## Specificity           0.81481   0.3087 0.973799  0.98394  0.98846
## Pos Pred Value        0.36364   0.4824 0.250000  0.00000  0.00000
## Neg Pred Value        0.71963   0.6571 0.854406  0.92453  0.96617
## Prevalence            0.29740   0.4461 0.148699  0.07435  0.03346
## Detection Rate        0.07435   0.3569 0.007435  0.00000  0.00000
## Detection Prevalence  0.20446   0.7398 0.029740  0.01487  0.01115
## Balanced Accuracy     0.53241   0.5544 0.511900  0.49197  0.49423
## [1] "fold"
## [1] 4
## [1] 0.4200743 0.4814815 0.4386617        NA        NA
## ntree      OOB      1      2      3      4      5
##     1:  66.15% 61.16% 57.99% 88.33% 88.89% 70.00%
##     2:  68.54% 61.73% 61.35% 88.54% 93.62% 83.33%
##     3:  68.71% 65.38% 59.22% 87.20% 92.73% 88.89%
##     4:  67.10% 65.09% 56.20% 86.76% 90.62% 93.75%
##     5:  66.56% 65.52% 53.70% 87.07% 92.86% 96.97%
##     6:  65.64% 59.60% 55.06% 88.00% 94.44% 97.14%
##     7:  64.33% 60.84% 51.86% 88.00% 93.33% 94.29%
##     8:  64.59% 60.51% 51.61% 90.85% 93.59% 94.29%
##     9:  64.02% 60.57% 51.17% 88.31% 93.75% 91.67%
##    10:  64.09% 62.78% 49.15% 89.74% 92.50% 97.22%
##    11:  65.05% 63.64% 50.00% 91.82% 95.00% 91.67%
##    12:  63.62% 63.44% 47.17% 91.19% 95.00% 91.67%
##    13:  63.66% 62.19% 47.39% 94.38% 91.25% 94.59%
##    14:  62.49% 60.94% 46.67% 90.62% 92.50% 94.59%
##    15:  62.02% 62.50% 45.00% 92.50% 88.75% 89.19%
##    16:  63.32% 64.69% 45.83% 91.88% 93.75% 89.19%
##    17:  61.47% 63.44% 42.50% 91.25% 95.00% 89.19%
##    18:  61.65% 62.50% 42.50% 92.50% 96.25% 94.59%
##    19:  61.00% 63.12% 41.46% 91.88% 92.50% 94.59%
##    20:  59.89% 63.75% 39.58% 90.62% 90.00% 91.89%
##    21:  60.72% 61.88% 41.46% 91.88% 93.75% 94.59%
##    22:  59.70% 60.31% 41.25% 91.25% 90.00% 91.89%
##    23:  60.07% 62.81% 39.79% 93.12% 90.00% 91.89%
##    24:  61.28% 64.06% 41.67% 93.12% 90.00% 91.89%
##    25:  60.91% 63.75% 40.42% 95.00% 90.00% 91.89%
##    26:  59.42% 62.19% 38.75% 93.12% 91.25% 89.19%
##    27:  61.10% 61.25% 42.29% 95.00% 92.50% 89.19%
##    28:  60.35% 63.12% 40.42% 92.50% 91.25% 89.19%
##    29:  60.54% 64.06% 40.00% 91.88% 93.75% 89.19%
##    30:  61.00% 65.31% 40.00% 92.50% 92.50% 91.89%
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1 24 27  2  6  0
##          2 52 87 35  8  7
##          3  4  3  3  2  3
##          4  0  3  0  3  0
```

```
##          5  0  0  0  1  0
##
## Overall Statistics
##
##                Accuracy : 0.4333
##                  95% CI : (0.3734, 0.4948)
##     No Information Rate : 0.4444
##     P-Value [Acc > NIR] : 0.6652
##
##                   Kappa : 0.0773
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.30000   0.7250  0.07500  0.15000 0.000000
## Specificity           0.81579   0.3200  0.94783  0.98800 0.996154
## Pos Pred Value        0.40678   0.4603  0.20000  0.50000 0.000000
## Neg Pred Value        0.73460   0.5926  0.85490  0.93561 0.962825
## Prevalence            0.29630   0.4444  0.14815  0.07407 0.037037
## Detection Rate        0.08889   0.3222  0.01111  0.01111 0.000000
## Detection Prevalence  0.21852   0.7000  0.05556  0.02222 0.003704
## Balanced Accuracy     0.55789   0.5225  0.51141  0.56900 0.498077
## [1] "fold"
## [1] 5
## [1] 0.4200743 0.4814815 0.4386617 0.4333333        NA
## ntree      OOB      1      2      3      4      5
##     1:  60.54% 56.00% 49.16% 88.24% 77.78% 88.89%
##     2:  63.02% 58.25% 54.61% 82.29% 85.71% 88.24%
##     3:  62.73% 60.08% 55.59% 75.83% 83.33% 79.17%
##     4:  63.10% 60.22% 52.12% 83.21% 87.88% 93.33%
##     5:  61.86% 58.33% 52.76% 80.00% 86.96% 87.10%
##     6:  63.48% 58.78% 54.70% 82.52% 87.84% 88.57%
##     7:  63.26% 58.84% 53.90% 80.00% 90.91% 91.89%
##     8:  62.57% 61.71% 50.00% 81.46% 92.31% 89.19%
##     9:  62.04% 59.62% 48.94% 83.12% 94.94% 91.89%
##    10:  62.73% 61.01% 48.63% 86.71% 93.75% 89.19%
##    11:  61.25% 60.31% 46.01% 85.44% 93.75% 91.89%
##    12:  61.36% 62.19% 44.56% 87.34% 95.00% 86.84%
##    13:  59.44% 59.69% 44.35% 82.39% 90.00% 86.84%
##    14:  60.09% 60.94% 43.10% 86.79% 91.25% 89.47%
##    15:  59.80% 63.12% 41.46% 85.53% 88.75% 94.74%
##    16:  61.56% 64.69% 42.29% 89.94% 93.75% 92.11%
##    17:  61.19% 61.56% 43.54% 90.57% 93.75% 89.47%
##    18:  60.67% 61.25% 42.71% 90.00% 91.25% 94.74%
##    19:  59.93% 61.88% 40.00% 91.88% 91.25% 94.74%
##    20:  60.39% 63.12% 39.79% 92.50% 92.50% 94.74%
##    21:  60.39% 62.50% 41.04% 90.62% 91.25% 94.74%
##    22:  59.18% 60.62% 39.38% 91.25% 93.75% 89.47%
##    23:  59.74% 61.88% 39.79% 90.62% 93.75% 92.11%
##    24:  60.11% 62.50% 39.79% 90.62% 96.25% 92.11%
##    25:  59.55% 60.31% 40.21% 91.25% 95.00% 89.47%
##    26:  58.44% 59.69% 37.92% 91.88% 95.00% 89.47%
##    27:  59.65% 61.56% 38.96% 93.75% 92.50% 92.11%
```

```
##    28:  59.74% 60.94% 40.21% 91.25% 95.00% 89.47%
##    29:  59.09% 60.31% 39.38% 91.88% 91.25% 92.11%
##    30:  60.11% 62.50% 39.79% 92.50% 93.75% 89.47%
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1 26 34  9  5  0
##          2 49 79 29 15  6
##          3  3  5  1  0  1
##          4  2  2  1  0  0
##          5  0  0  0  0  2
##
## Overall Statistics
##
##                Accuracy : 0.4015
##                  95% CI : (0.3424, 0.4627)
##     No Information Rate : 0.4461
##     P-Value [Acc > NIR] : 0.9378
##
##                   Kappa : 0.0281
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.32500   0.6583 0.025000  0.00000 0.222222
## Specificity           0.74603   0.3356 0.960699  0.97992 1.000000
## Pos Pred Value        0.35135   0.4438 0.100000  0.00000 1.000000
## Neg Pred Value        0.72308   0.5495 0.849421  0.92424 0.973783
## Prevalence            0.29740   0.4461 0.148699  0.07435 0.033457
## Detection Rate        0.09665   0.2937 0.003717  0.00000 0.007435
## Detection Prevalence  0.27509   0.6617 0.037175  0.01859 0.007435
## Balanced Accuracy     0.53552   0.4970 0.492849  0.48996 0.611111
```

The output is accuracy per fold and average accuracy of all 5 folds.

```
cv
```

```
## $vec
## [1] 0.4200743 0.4814815 0.4386617 0.4333333 0.4014870
##
## $avg
## [1] 0.4350076
```