Machine Learning Code

Kee-Young Shin

2019-04-22

```
library(tidyverse) # map_df();
library(caret) # createDataPartition();
library(AppliedPredictiveModeling) # transparentTheme();
library(pROC) # roc();
library(rpart.plot) # rpart.plot();
library(ranger) # for variable importance of rf
library(pdp) # PDP
library(lime) # prediction explaination

library(factoextra) # clustering visualization
library(corrplot)
```

```
# import the raw data
DATA = read.csv(file = './OnlineNewsPopularity/OnlineNewsPopularity.csv', head = TRUE, sep = ',')
```

```
# check for missing data
sum( map_df(DATA, function(x) sum(is.na(x))) )
# NOTES: The original data contains no missing data.
```

1 Preprocessing

```
# raw data cleaning part1
Data_prep_1 =
  DATA[,-1] %>%
  mutate(., data_channel_is_other = ifelse(data_channel_is_lifestyle+data_channel_is_entertainment+data_
  gather(., key = channel_type, value = data_channel_is,
            data_channel_is_lifestyle:data_channel_is_world, data_channel_is_other) %>%
  filter(., data_channel_is == '1') %>%
  select(., -data_channel_is) %>%
  mutate(., channel_type = str_replace(channel_type, 'data_channel_is_', ''),
            channel_type = forcats::fct_relevel(channel_type, 'other'))
# In this part, we deal with the channel types. First, we created a new variable 'other' that refers to
```

```
# raw data cleaning part2
Data_prep_2 =
  gather(Data_prep_1, key = publish_day, value = day_is, weekday_is_monday:weekday_is_sunday) %>%
  filter(., day_is == '1') %>%
  select(., -day_is) %>%
  mutate(., publish_day = str_replace(publish_day, 'weekday_is_', ''),
            publish_day = forcats::fct_relevel(publish_day, 'monday'))
# In this part, similarly, we gather 7 columns into one single variable 'publish_day', and make 'monday
```

```
# delete uninformative variables
Data_prep_3 =
  select(Data_prep_2, -n_unique_tokens, -n_non_stop_words, -n_non_stop_unique_tokens, -timedelta)
# NOTES: the first three variables has nearly 0 variance, in addition to one or two outliers.
# NOTES: timedelta cannot be used to predict 'popularity' before the article get published.
```

```r
# Using 1500 as threshold to separate 'shares' into 'popular' and 'unpopular'.
Data_prep_4 =
  mutate(Data_prep_3, popularity = case_when(shares >= 1500 ~ 'popular',
                                             shares < 1500 ~ 'unpopular')) %>%
  mutate(., popularity = forcats::fct_relevel(popularity, 'unpopular')) %>%
  select(., -shares)
#Data_prep_4 %>% count(., popularity)
# NOTES: make 'unpopular' to be the reference category.
# NOTES: the raw dataset ended up with 19562 'popular' and 20082 'unpopular' articles.
```

```r
# Randomly take a subset of the whole data as the original dataset.
set.seed(4)
subset = createDataPartition(Data_prep_4$popularity, p = 0.05, list = FALSE) # 5% of the original datas
Data_prep_5 = Data_prep_4[subset,]
# NOTES: randomly choose 5% of the whole dataset.
```

```r
# because of collinearity issue, we decided to take out these two variables.
Data_prep_6 = Data_prep_5 %>% select(., -publish_day, -LDA_04)
```

```r
# the whole dataset after preprocessing
News_Data = Data_prep_6
```

```r
# training and test dataset split
set.seed(4)
trRows = createDataPartition(News_Data$popularity, p = 0.7, list = FALSE)

###training data###
News_Data_train = News_Data[trRows,]
###test data###
News_Data_test = News_Data[-trRows,]

# training + test data
x_total = model.matrix(popularity~., News_Data)[,-1]
y_total = News_Data$popularity

# training data with 1390 observations
x_train = model.matrix(popularity~., News_Data_train)[,-1]
y_train = News_Data_train$popularity

# test data with 594 observations
x_test = model.matrix(popularity~., News_Data_test)[,-1]
y_test = News_Data_test$popularity

# NOTES: 70% training + 30% test
```

Explorataty Analysis

```r
# plots of response(shares) vs. predictors
#################################
### takes about 20 min to run ###
#################################
transparentTheme(trans = 0.4)
```

```r
featurePlot(x = x_total,
            y = y_total,
            scales = list(x = list(relation = 'free'), y = list(relation = 'free')),
            plot = 'density', pch = '|',
            layout = c(3, 3),
            auto.key = list(columns = 2))
```

```r
# correlation coefficient matrix
corrplot::corrplot(cor(x_total), tl.cex = 0.45)
```

K means clustering

```r
# using train + test dataset
# scale
x_total_scaled = scale(x_total)

fviz_nbclust(x_total_scaled,
             FUNcluster = kmeans,
             method = 'silhouette')
```

Hierarchical clustering

```r
hc.complete = hclust(dist(x_total_scaled), method = 'complete')
fviz_dend(hc.complete, k = 4,
          cex = 0.3,
          palette = 'jco',
          color_labels_by_k = TRUE,
          rect = TRUE, rect_fill = TRUE, rect_border = 'jco',
          labels_track_height = 2.5)
# NOTES: results in a compact plot.
```

PCA

```r
pca = prcomp(x_total_scaled)
pca$rotation
```

```r
corrplot(pca$rotation %*% diag(pca$sdev), tl.cex = 0.45)
```

```r
# scree plot
fviz_eig(pca, addlabels = TRUE)
```

```r
fviz_contrib(pca, choice = 'var', axes = 1)
```

2 Model Selection all using 'caret' package

Logistic Regression without regularization

```r
# logistic regression (glm) without regularization
# train model with training data using 'caret' package
set.seed(4)
glm.fit = train(x = x_train, y = y_train,
```

```
                method = 'glm',
                metric = 'Accuracy',
                trControl = trainControl(method = 'cv', number = 10, classProbs = TRUE)) # 10 fold CV

# NOTES: use the default for 'summaryFunction', which is using Accuracy and Kappa.
```

```
summary(glm.fit)
#glm.fit$resample

##glm.pred = predict(glm.fit, newdata = x_train, type = 'prob')[, 2]
##roc(y_train, glm.pred)
##plot( roc(y_train, glm.pred), legacy.axes = TRUE)
```

Logistic Regression with regularization

```
# logistic regression (glmn) with regularization (elastic)
# train model with training data using 'caret' package
glmnGrid = expand.grid(.alpha = seq(0, 1, length = 6),
                       .lambda = exp(seq(-10, -1, length = 20)))
set.seed(4)
glmn.fit = train(x = x_train, y = y_train,
                 method = 'glmnet',
                 tuneGrid = glmnGrid,
                 metric = 'Accuracy',
                 trControl = trainControl(method = 'cv', number = 10, classProbs = TRUE))
```

```
# tuning result for alpha, lambda
glmn_tune = glmn.fit$bestTune
glmn_tune
```

```
# plot of the tuning result
#plot(glmn.fit, xTrans = function(x) log(x), highlight = TRUE)
ggplot(glmn.fit, highlight = TRUE) + scale_x_log10()
```

Linear Discriminant Analysis LDA

```
# LDA (lda)
# train model with training data using 'caret' package
set.seed(4)
lda.fit = train(x = x_train, y = y_train,
                method = 'lda',
                metric = 'Accuracy',
                trControl = trainControl(method = 'cv', number = 10, classProbs = TRUE))
```

```
#summary(lda.fit)
#lda.fit$resample
```

Quadratic Discriminant Analysis QDA

```
# QDA (qda)
# train model with training data using 'caret' package
set.seed(4)
qda.fit = train(x = x_train, y = y_train,
                method = 'qda',
                metric = 'Accuracy',
                trControl = trainControl(method = 'cv', number = 10, classProbs = TRUE))
```

```
# x_train[, -(43:54)][, -(22:26)]
#x_train[, -(51:54)][, -(26:26)]
# collinearity problem
```

```
#summary(qda.fit)
#qda.fit$resample
```

Naive Bayes

```
# Naive Bayes (nb)
# train model with training data using 'caret' package
# library("klaR")
############################################
### takes too long to run, very lagging ###
############################################
set.seed(4)
nbGrid = expand.grid(usekernel = TRUE,
                     fL = 1,
                     adjust = 10)
nb.fit = train(x = x_train[, 1:2], y = y_train,
               preProcess = c('center', 'scale'),
               method = 'nb',
               tuneGrid = nbGrid,
               metric = 'Accuracy',
               trControl = trainControl(method = 'cv', number = 10, classProbs = TRUE))
dim(x_train)
# NOTES: because of the '0' values in the dataset, trainig for this model didn't work.
```

```
#plot(nb.fit)
```

Classification Tree

```
# Classification Tree (CART)
# train model with training data using 'caret' package
set.seed(4)
ct.fit = train(popularity~., News_Data_train,
               method = 'rpart',
               tuneGrid = data.frame(cp = exp(seq(-10, -1, len = 30))),
               metric = 'Accuracy',
               trControl = trainControl(method = 'cv', number = 10, classProbs = TRUE))
```

```
# tuning result for cp
ct_tune = ct.fit$bestTune
ct_tune
```

```r
# plot of the tuning result
ggplot(ct.fit, highlight = TRUE)
```

```r
# plot the tree
rpart.plot(ct.fit$finalModel)
```

Random Forest

```r
# Random Forest (rf)
# train model with training data using 'caret' package
##################################
### takes about 70 min to run ###
##################################
rf.grid = expand.grid(mtry = 1:47,
                      splitrule = 'gini',
                      min.node.size = 1:10)
set.seed(4)
rf.fit = train(popularity~., News_Data_train,
               method = 'ranger',
               tuneGrid = rf.grid,
               metric = 'Accuracy',
               trControl = trainControl(method = 'cv', number = 10, classProbs = TRUE))
```

```r
# save the object
#saveRDS(rf.fit, file = './rf.fit.rda')
# read in the object
rf.fit = readRDS('/Users/junyuanzheng/Desktop/Data Science II/Final_Project/rf.fit.rda')
```

```r
# tuning result for mtry, min.node.size
rf_tune = rf.fit$bestTune
rf_tune
```

```r
# plot of the tuning result
ggplot(rf.fit, highlight = TRUE)
```

```r
#rf.fit$resample
```

Boosting

```r
# Gradient Boosting Machines (gbm)
# train model with training data using 'caret' package
##################################
### takes about 30 min to run ###
##################################
gbm.grid = expand.grid(n.trees = c(2000, 3000, 4000),
                       interaction.depth = 1:5,
                       shrinkage = c(0.0006, 0.0009, 0.003, 0.006),
                       n.minobsinnode = 1)
set.seed(4)
gbm.fit = train(popularity~., News_Data_train,
                method = 'gbm',
```

```r
                 tuneGrid = gbm.grid,
                 trControl = trainControl(method = 'cv', number = 10, classProbs = TRUE),
                 distribution = 'adaboost',
                 metric = 'Accuracy',
                 verbose = FALSE)
```

```r
# save the object
#saveRDS(gbm.fit, file = './gbm.fit.rda')
# read in the object
gbm.fit = readRDS('/Users/junyuanzheng/Desktop/Data Science II/Final_Project/gbm.fit.rda')
```

```r
gbm.fit$bestTune
ggplot(gbm.fit, highlight = TRUE)
```

```r
#gbm.fit$resample
```

SVM with linear kernel

```r
# Support Vector Machine with linear kernel (svml)
# train model with training data using 'caret' package
##################################
### takes about 60 min to run ###
##################################
set.seed(4)
svml.fit = train(popularity~., data = News_Data_train,
                 method = 'svmLinear2',
                 preProcess = c('center', 'scale'),
                 tuneGrid = data.frame(cost = exp(seq(-8, 8, len=30))),
                 trControl = trainControl(method = 'cv', number = 10))
```

```r
# save the object
#saveRDS(svml.fit, file = './svml.fit.rda')
# read in the object
svml.fit = readRDS('/Users/junyuanzheng/Desktop/Data Science II/Final_Project/svml.fit.rda')
```

```r
#summary(svml.fit)
ggplot(svml.fit, highlight = TRUE)
#best.svml.fit = svml.fit$bestTune # cost =
```

SVM with radial kernel

```r
# Support Vector Machine with radial kernel (svml)
# train model with training data using 'caret' package
svmr.grid = expand.grid(C = exp(seq(-8, 8, len=20)),
                        sigma = exp(seq(-8, 2, len=10)))

set.seed(4)
svmr.fit = train(popularity~., data = News_Data_train,
                 method = 'svmRadial',
                 preProcess = c('center', 'scale'),
                 tuneGrid = svmr.grid,
                 trControl = trainControl(method = 'cv', number = 10))
```

```r
# save the object
#saveRDS(svmr.fit, file = './svmr.fit.rda')
# read in the object
svmr.fit = readRDS('/Users/junyuanzheng/Desktop/Data Science II/Final_Project/svmr.fit.rda')
```

```r
#summary(svmr.fit)
ggplot(svmr.fit, highlight = TRUE)
#best.svmr.fit = svmr.fit$bestTune # sigma = 0.00309, C = 8.209
```

Summary

```r
# comparing the models based on resampling results
set.seed(4)
resamp = resamples(list(GLM = glm.fit,
                        GLMN = glmn.fit,
                        LDA = lda.fit,
                        QDA = qda.fit,
                        CART = ct.fit,
                        RF = rf.fit,
                        GBM = gbm.fit,
                        SVML = svml.fit,
                        SVMR = svmr.fit))

summary(resamp)
```

Accuracy comparison:

```r
# parallel plot of accuracy
parallelplot(resamp, metric = 'Accuracy', lwd = 3, lty = 1, alpha = 0.7)
```

```r
# box plot of accuracy
bwplot(resamp, metric = 'Accuracy')
```

Kappa comparison:

```r
# parallel plot of accuracy
parallelplot(resamp, metric = 'Kappa', lwd = 3, lty = 1, alpha = 0.7)
```

```r
# box plot of accuracy
bwplot(resamp, metric = 'Kappa')
```

Model comparison are based mainly on Accuracy and Kappa.
As a result, SVM with radial kernel, Random Forest, and Boosting appear to have a better performance on the training dataset using Cross Validation.
Therefore, we are going to perform these three models on the test dataset.

3 Estimation on the test dataset

```r
# SVM with radial kernel on test dataset
svmr.pred_test = predict(svmr.fit, newdata = News_Data_test[, -43])
svmr_Accuracy_test = mean(svmr.pred_test == News_Data_test[, 43])
svmr_Accuracy_test

confusionMatrix(data = svmr.pred_test,
                reference = News_Data_test[, 43],
                positive = 'popular')
```

```r
# Random Forest on test dataset
rf.pred_test = predict(rf.fit, newdata = News_Data_test[, -43])
rf_Accuracy_test = mean(rf.pred_test == News_Data_test[, 43])
rf_Accuracy_test

confusionMatrix(data = rf.pred_test,
                reference = News_Data_test[, 43],
                positive = 'popular')
```

```r
# Boosting on test dataset
gbm.pred_test = predict(gbm.fit, newdata = News_Data_test[, -43])
gbm_Accuracy_test = mean(gbm.pred_test == News_Data_test[, 43])
gbm_Accuracy_test

confusionMatrix(data = gbm.pred_test,
                reference = News_Data_test[, 43],
                positive = 'popular')
```

4 Check important variables

Variable importance from random forest

```r
# variable importance using Permutation
# refit the model using 'ranger' with the tuned hyperparameter
set.seed(4)
rf_VI.per = ranger(popularity~., News_Data_train,
                      mtry = 2, splitrule = 'gini',
                      min.node.size = 2,
                      importance = 'permutation',
                      scale.permutation.importance = TRUE)
# barplot
par(mai = c(0.5, 1.2, 0.5, 0.3))
barplot(sort(ranger::importance(rf_VI.per), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.4,
        col = colorRampPalette(colors = c('red', 'cyan', 'blue'))(50), main = 'Variable Importance by Pe
        space = 0.3)
```

```r
# variable importance using Impurity
# refit the model using 'ranger' with the tuned hyperparameter
set.seed(4)
rf_VI.imp = ranger(popularity~., News_Data_train,
                   mtry = 2, splitrule = 'gini',
                   min.node.size = 2,
                   importance = 'impurity')
```

```r
# barplot
par(mai = c(0.5, 1.2, 0.5, 0.3))
barplot(sort(ranger::importance(rf_VI.imp), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.4,
        col = colorRampPalette(colors = c('red', 'cyan', 'blue'))(50), main = 'Variable Importance by In
        space = 0.3)
```

Variable importance from Boosting

```r
# variable importance from Boosting
par(mai = c(0.5, 1.2, 0.5, 0.3))
summary(gbm.fit$finalModel, las = 2, cBars = 42, cex.names = 0.4, space = 0.3)
```

NOTES:By looking at the variable importance plot from fitted RF and Boosting model, they all agree that 'kw__avg__avg' to be an important vaiable relative to the others. Therefore, we are going to focus on this variable in the following Partial Dependence Plots (PDP).

Partial Dependence Plots (PDP) of 'kw__avg__avg'

```r
pdp.rf = rf.fit %>%
  partial(pred.var = 'kw_avg_avg',
          grid.resolution = 100,
          prob = TRUE) %>%
  autoplot(rug = TRUE, train = News_Data_train) +
  ggtitle('Random Forest') +
  theme(axis.text = element_text(size = 6))

pdp.gbm = gbm.fit %>%
  partial(pred.var = 'kw_avg_avg',
          grid.resolution = 100,
          prob = TRUE) %>%
  autoplot(rug = TRUE, train = News_Data_train) +
  ggtitle('Boosting') +
  theme(axis.text = element_text(size = 6))

grid.arrange(pdp.rf, pdp.gbm, nrow = 1)
```

Individual Conditional Expectation (ICE) of 'kw__avg__avg'

```r
ice1.rf = rf.fit %>%
  partial(pred.var = 'kw_avg_avg',
          grid.resolution = 100,
          ice = TRUE,
          prob = TRUE) %>%
  autoplot(train = News_Data_train, alpha = 0.02) +
  ggtitle('Random Forest, non-centered') +
  theme(axis.text = element_text(size = 6))

ice2.rf = rf.fit %>%
  partial(pred.var = 'kw_avg_avg',
          grid.resolution = 100,
          ice = TRUE,
          prob = TRUE) %>%
```

```
  autoplot(train = News_Data_train, alpha = 0.02, center = TRUE) +
  ggtitle('Random Forest, centered') +
  theme(axis.text = element_text(size = 6))

ice1.gbm = gbm.fit %>%
  partial(pred.var = 'kw_avg_avg',
          grid.resolution = 100,
          ice = TRUE,
          prob = TRUE) %>%
  autoplot(train = News_Data_train, alpha = 0.02) +
  ggtitle('Boosting, non-centered') +
  theme(axis.text = element_text(size = 6))

ice2.gbm = gbm.fit %>%
  partial(pred.var = 'kw_avg_avg',
          grid.resolution = 100,
          ice = TRUE,
          prob = TRUE) %>%
  autoplot(train = News_Data_train, alpha = 0.02, center = TRUE) +
  ggtitle('Boosting, centered') +
  theme(axis.text = element_text(size = 6))

grid.arrange(ice1.rf, ice2.rf, ice1.gbm, ice2.gbm)
```

Partial Dependence Plots (PDP) of 'self_reference_min_shares'

```
pdp_2.rf = rf.fit %>%
  partial(pred.var = 'self_reference_min_shares',
          grid.resolution = 100,
          prob = TRUE) %>%
  autoplot(rug = TRUE, train = News_Data_train) +
  ggtitle('Random Forest') +
  theme(axis.text = element_text(size = 6))

pdp_2.gbm = gbm.fit %>%
  partial(pred.var = 'self_reference_min_shares',
          grid.resolution = 100,
          prob = TRUE) %>%
  autoplot(rug = TRUE, train = News_Data_train) +
  ggtitle('Boosting') +
  theme(axis.text = element_text(size = 6))

grid.arrange(pdp_2.rf, pdp_2.gbm, nrow = 1)
```

Individual Conditional Expectation (ICE) of 'self_reference_min_shares'

```
ice1_2.rf = rf.fit %>%
  partial(pred.var = 'self_reference_min_shares',
          grid.resolution = 100,
          ice = TRUE,
          prob = TRUE) %>%
  autoplot(train = News_Data_train, alpha = 0.02) +
  ggtitle('Random Forest, non-centered') +
```

```
  theme(axis.text = element_text(size = 6))

ice2_2.rf = rf.fit %>%
  partial(pred.var = 'self_reference_min_shares',
          grid.resolution = 100,
          ice = TRUE,
          prob = TRUE) %>%
  autoplot(train = News_Data_train, alpha = 0.02, center = TRUE) +
  ggtitle('Random Forest, centered') +
  theme(axis.text = element_text(size = 6))

ice1_2.gbm = gbm.fit %>%
  partial(pred.var = 'self_reference_min_shares',
          grid.resolution = 100,
          ice = TRUE,
          prob = TRUE) %>%
  autoplot(train = News_Data_train, alpha = 0.02) +
  ggtitle('Boosting, non-centered') +
  theme(axis.text = element_text(size = 6))

ice2_2.gbm = gbm.fit %>%
  partial(pred.var = 'self_reference_min_shares',
          grid.resolution = 100,
          ice = TRUE,
          prob = TRUE) %>%
  autoplot(train = News_Data_train, alpha = 0.02, center = TRUE) +
  ggtitle('Boosting, centered') +
  theme(axis.text = element_text(size = 6))

grid.arrange(ice1_2.rf, ice2_2.rf, ice1_2.gbm, ice2_2.gbm)
```

```
pdp2d.rf = partial(rf.fit, pred.var = c('kw_avg_avg', 'self_reference_min_shares'), plot = TRUE,
                   chull = TRUE, train = News_Data_train, prob = TRUE, plot.engine = 'ggplot2',
                   palette = 'viridis')
pdp2d.rf
```

LIME

```
new_obs = News_Data_test[, -43][1,]
explainer.gbm = lime(News_Data_test[, -43], gbm.fit)
explaination.gbm = explain(new_obs, explainer.gbm, n_features = 10,
                           labels = 'pos')
plot_features(explaination.gbm)
# NOTES: not included in the report
# NOTES: try scale the dataset.
```