Cybernetic Ducks

Statistics 101C Lecture 4 Final Project Report

Oscar Monroy, Zoe Wang,
Christine Marie Castle

2020-12-14

1 Introduction

YouTube is the largest online video-sharing platform in the world. A plethora of factors ranging from the duration of the video to the length of the video title to the average pixel value in the thumbnail image could be associated increased engagement. Our goal, simply put, was to find the best model and the optimal set of predictors to successfully predict the success, measured in growth of views between two and six hours after publishing, of any given YouTube video.

2 Methodology

2.1 Preprocessing

We used the base `R` and `caret` packages to clean the data. We began by looking for any missing values, nonsensical values, or duplicated rows but none were found. The data set did contain several constant variables (i.e. variables with only one unique value); these were eliminated as they provided no information. `id` was removed from the training data set because it contains arbitrary labels that only serve to distinguish the observations. `PublishedDate` was removed because dates are not stored as numeric in `R`, which resulted in errors (e.g. when calculating the correlation matrix).

2.2 Variable Selection

We removed highly correlated variables to curb multicollinearity and overfitting. To remove these variables, we used the `findCorrelation` function from the `caret` package, which takes in a correlation matrix and a specified pairwise correlation cutoff (Kuhn, n.d.). It outputs the indices of the variables that have correlations above the cutoff. We removed these variables from the data set.

To determine the correlation cutoff, we created five separate data sets with variables removed based on correlation cutoffs of 0.5, 0.6, 0.7, 0.8, and 0.9, and fit a random forest

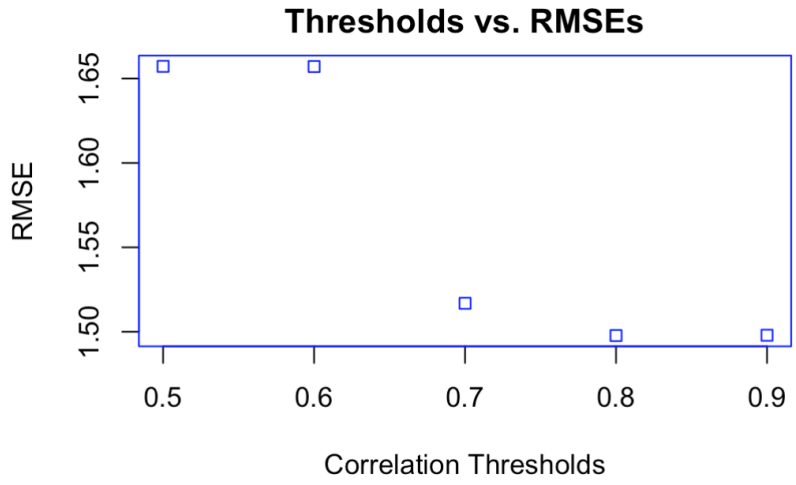

Figure 1: Correlation thresholds and corresponding out-of-bag RMSE's

model to each of these sets. We used RMSE and percentage of variance in the response explained, which are both based on predictions on the out-of-bag samples. We considered thresholds no less than 0.5 because we did not want to remove so many variables that our model suffered. The plot in Figure 1 compares the RMSE's with different thresholds. Appendix Table 2 shows the percentage of variance explained and optimal `mtry` value (i.e. number of predictors sampled at each split) for each model. The threshold of 0.8 resulted in the lowest RMSE and the highest percentage of variance explained. Using the 0.8 cutoff, we removed 74 variables.

Further, we repeatedly trained a random forest model then used `varImpPlot` in the `randomForest` package to determine and remove variables which were not important to the model. All of the `hog_*` variables and nine other thumbnail features were removed in this process. The final training data set had 73 predictors.

2.3 Statistical Model

Our final model was a random forest of 500 trees created with 48 randomly selected predictors considered at each split. To find the optimal `mtry`, we used the `tuneRF` function, which by default started at an `mtry` of $\lfloor \#\text{predictors}/3 \rfloor = 24$. The initial `mtry` value is halved multiple times then doubled multiple times, continuing in each direction until model improvement becomes insignificant. `mtry` values and the resulting out-of-bag estimated MSE's are shown in Appendix Figure 2.

We decided to pursue a random forest model because the initial RMSE estimate was noticeably less than that of the other models we tried, as shown in Table 1. The RMSE estimates were obtained via 10-fold cross-validation (except for the random forest model, where we had an out-of-bag estimate). For the regularization methods, the shrinkage parameter λ used was chosen from a range of 100 values to minimize the cross-validated

Model	Lowest initial CV RMSE estimate	Test RMSE (Kaggle)
Random Forest**	1.48982	1.49149
Linear Regression	1.65061	1.62673
Elastic Net ($\alpha = 0.5$)	1.64885	1.62927
Ridge regression	1.67215	1.63813
LASSO regression	1.67114	1.63971
PCR*	2.07693	—
PLS Regression*	2.15084	—

Table 1: Lowest cross-validated (out-of-bag for random forest) RMSE estimates of each model we attempted initially. *Predictions were not submitted to Kaggle. **This estimate is not a cross-validated estimate but an out-of-bag estimate

RMSE estimate. All of the models were trained on the training data set after removing variables above the 0.8 correlation threshold.

3 Results

Our best evaluation metric value with this model as shown in the Kaggle public leaderboard was an RMSE of 1.39935.

4 Conclusions

As shown in the Kaggle private leaderboard, the RMSE with this model was 1.41372. One of our models that was trained on more predictors and had a higher public RMSE had a lower private RMSE than this model.

The five most important predictors in our final model (as measured by total decrease in node impurities from splitting on the variable, averaged over all trees) were `Num_Views_Base_mid_high`, `views_2_hours`, `cnn_25`, `cnn_89`, and `Num_Subscribers_Base_mid_high`, as shown in Appendix Figure 3. Three of these predictors relate to how popular the channel or video already is. The other two are thumbnail features.

Our model works well because random forest creates an average of trees that is less variable and thus better at prediction. This is because at each split, random forest considers a random subset of predictors rather than all the predictors unlike bagging. As a result, the trees created are more varied and thus less correlated. Additionally, this reduces test error and the out-of-bag error estimate. Random forest is also computationally inexpensive and easily accessible as the function `tuneRF` takes the work out of optimizing the `mtry` value.

We believe our model could potentially be improved by continuing to analyze the predictors and potentially creating new predictors by combining or transforming predictors.

References

Kuhn, M. (n.d.). FindCorrelation. Retrieved December 12, 2020, from https://www.rdocumentation.org/packages/caret/versions/6.0-86/topics/findCorrelation

Liaw, A (n.d.). Importance. Retrieved December 12, 2020, from https://www.rdocumentation.org/packages/randomForest/versions/4.6-14/topics/importance

Statement of Contributions

All members contributed to writing the report and creating the presentation slides.

Zoe Wang contributed by creating the final model, as well as submitting 10 other predictions on Kaggle using the random forest method.

Oscar Monroy contributed by cleaning up the data and submitting a few submissions to Kaggle based on models formed by both ridge regression and lasso regression.

Christine Marie Castle contributed by submitting three sets of predictions to Kaggle based on linear regression and elastic net models; PCR and PLS regression were also performed but not submitted to Kaggle. They also typeset the report in LaTeX.

Appendix

Figures and tables

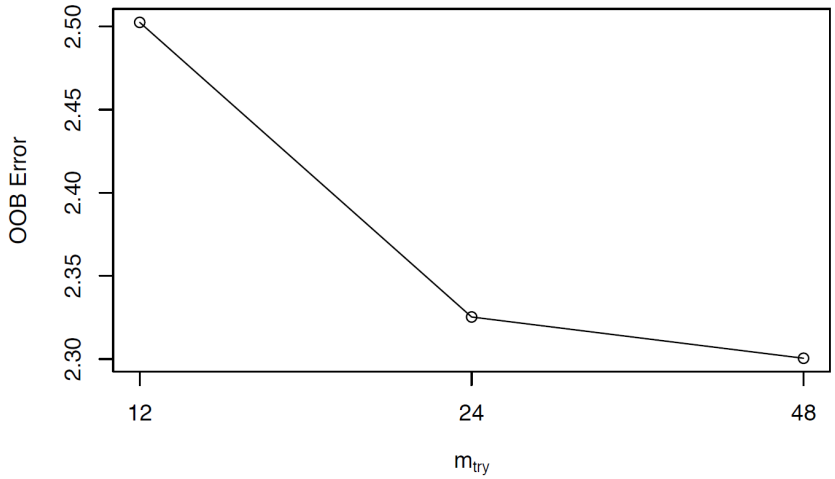

Figure 2: Out-of-bag MSE estimates for each value of m tried in final data set

Correlation Threshold	Optimal `mtry`	% Variance Explained
0.5	100	60.31%
0.6	108	60.32%
0.7	110	66.75%
0.8	114	67.58%
0.9	116	67.58%

Table 2: Comparison of correlation thresholds, optimal numbers of predictors considered at each split m, and percentage of variance in the response explained

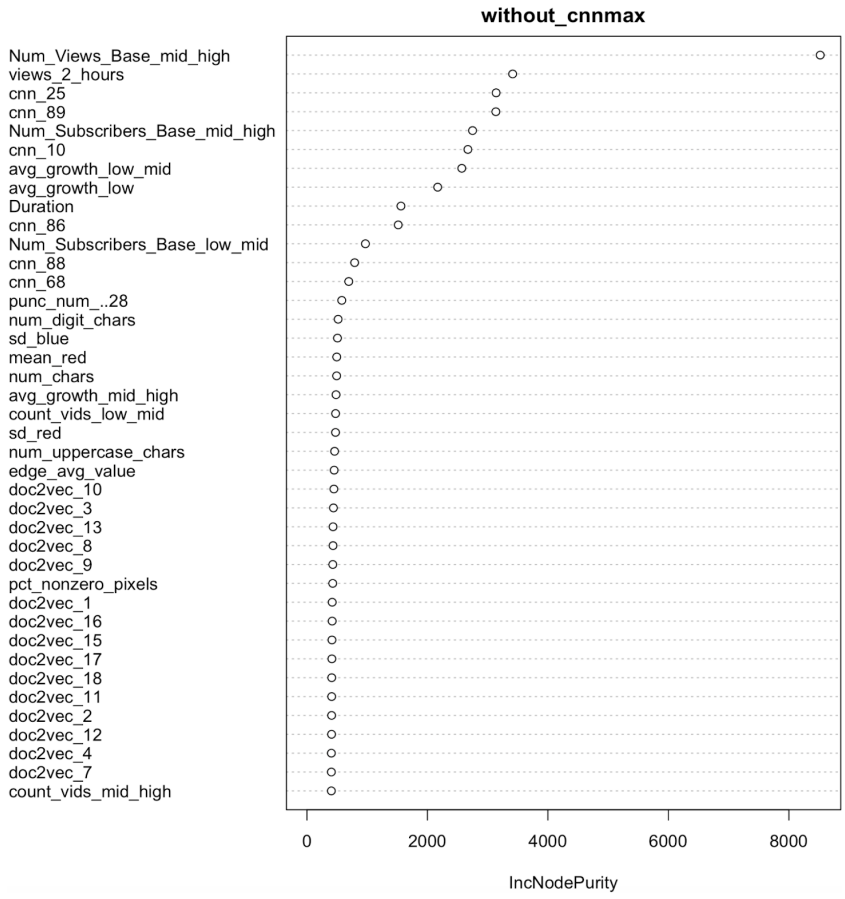

Figure 3: Variable importance plot of 40 most important variables in the final model. Variable importance measured by total decrease in node impurities from splitting on the variable, averaged over all trees.

Script

```
# Packages used ------------------------------------------------------------
library(boot)
library(caret)
library(glmnet)
library(pls)
library(randomForest)
library(tidyverse)

### FINAL MODEL ###############################################################

# Step 0: Importing data ---------------------------------------------------

yt_train_uncleaned <- read.csv("training.csv")
test <- read.csv("test.csv")

# Step 1: Cleaning the data ------------------------------------------------

# Any NA values? No
any(apply(yt_train_uncleaned, 2, function(x) sum(is.na(x))) > 0)
# Any infinite values? No
any(apply(yt_train_uncleaned, 2, function(x) sum(is.infinite(x))) > 0)
# Any NaN values? No
any(apply(yt_train_uncleaned, 2, function(x) sum(is.nan(x))) > 0)
# Any duplicated rows (excluding id)? No
sum(duplicated(yt_train_uncleaned[, -1]))
# Are there any duplicated id's? No
sum(duplicated(yt_train_uncleaned$id))

# Are there unusual duration values? No
summary(yt_train_uncleaned$Duration)
# No 0 second videos
# A video of 42895 seconds (or ~12 hours) isn't unusual on YouTube

# Are there unusual views_2_hours values? No
summary(yt_train_uncleaned$views_2_hours)
# No negative values
# Nothing too unusual here either unless you count the outliers

# Indices of constant variables (i.e. all same value)
useless_variables <- which(
  apply(yt_train_uncleaned, 2, function(x) sum(length(unique(x))) == 1)
)
```

```r
# Remove constant variables
training <- yt_train_uncleaned[, -c(useless_variables)]
test <- test[,-c(useless_variables)]

# write.csv(training, "training_clean.csv", row.names = FALSE)
# write.csv(test, "test_clean.csv", row.names = FALSE)
#
# #Loading in the cleaned data
# training <- read.csv("training_clean.csv")
# test <- read.csv("test_clean.csv")

# Step 2: Optimizing correlation cutoff ----------------------------------------

set.seed(1)

#Removing PublishedDate
training <- training[,-2]
test <- test[,-2]

#Saving the test id's for later
testid <- cbind(test$id)

#Removing the id's
training <- training[,-1]
test <- test[,-1]

#Creating a correlation matrix for use in findCorrelation
corrmx <- cor(training)

#Running through the different thresholds

#Testing 0.5
correlated_vars <- findCorrelation(corrmx, cutoff = 0.5)
temp_training <- training[,-correlated_vars]
temp_test <- test[,-correlated_vars]

ncol(temp_training)
ncol(temp_test)

forest_uncorr05 <- tuneRF(
  x = temp_training[,-153], y = temp_training$growth_2_6,
  plot = TRUE, doBest = TRUE
)
print(forest_uncorr05)
```

```r
#Testing 0.6
correlated_vars <- findCorrelation(corrmx, cutoff = 0.6)
temp_training <- training[,-correlated_vars]
temp_test <- test[,-correlated_vars]

ncol(temp_training)
ncol(temp_test)

forest_uncorr06 <- tuneRF(
  x = temp_training[,-165], y = temp_training$growth_2_6,
  plot = TRUE, doBest = TRUE
)
print(forest_uncorr06)

#Testing 0.7
correlated_vars <- findCorrelation(corrmx, cutoff = 0.7)
temp_training <- training[,-correlated_vars]
temp_test <- test[,-correlated_vars]

ncol(temp_training)
ncol(temp_test)

forest_uncorr07 <- tuneRF(
  x = temp_training[,-167], y = temp_training$growth_2_6,
  plot = TRUE, doBest = TRUE
)
print(forest_uncorr07)

#Testing 0.8
correlated_vars <- findCorrelation(corrmx, cutoff = 0.8)
temp_training <- training[,-correlated_vars]
temp_test <- test[,-correlated_vars]

ncol(temp_training)
ncol(temp_test)

forest_uncorr08 <- tuneRF(
  x = temp_training[,-172], y = temp_training$growth_2_6,
  plot = TRUE, doBest = TRUE
)
print(forest_uncorr08)

#Testing 0.9
```

```r
correlated_vars <- findCorrelation(corrmx, cutoff = 0.9)
temp_training <- training[,-correlated_vars]
temp_test <- test[,-correlated_vars]

ncol(temp_training)
ncol(temp_test)

forest_uncorr09 <- tuneRF(
  x = temp_training[,-176], y = temp_training$growth_2_6,
  plot = TRUE, doBest = TRUE
)
print(forest_uncorr09)

# Step 3: Variable selection ------------------------------------------------

set.seed(1)

#Finding and removing variables with correlations > 0.8
correlated_vars <- findCorrelation(corrmx, cutoff = 0.8)
training_uncorr <- training[,-correlated_vars]
test_uncorr <- test[,-correlated_vars]

#Finding the optimal mtry Random Forest model
#using the new data without the correlated variables
forest_uncorr <- tuneRF(
  x = training_uncorr[,-172], y = training_uncorr$growth_2_6,
  plot = TRUE, doBest = TRUE
)
# OOB RMSE
sqrt(min(forest_uncorr$mse))

#Plotting to see what variables are unimportant
varImpPlot(forest_uncorr, sort = FALSE, n.var = 40)

#Removing all hog variables up until hog_485
training_uncorr <- training_uncorr[,-(3:40)]
test_uncorr <- test_uncorr[,-(3:40)]

#Fitting a new RF to this new dataset
without_hog485 <- tuneRF(
  x = training_uncorr[,-134], y = training_uncorr$growth_2_6,
  plot = TRUE, doBest = TRUE
)
# OOB RMSE
```

```r
sqrt(min(without_hog485$mse))

#Plotting to see what variables are unimportant
varImpPlot(without_hog485, sort = FALSE, n.var = 40)

#Remove the rest of the hog variables
training_uncorr <- training_uncorr[,-(3:53)]
test_uncorr <- test_uncorr[,-(3:53)]

ncol(training_uncorr)
ncol(test_uncorr)

#Fitting a new RF to this new dataset
without_allhog <- tuneRF(
  x = training_uncorr[,-83], y = training_uncorr$growth_2_6,
  plot = TRUE, doBest = TRUE
)

# OOB RMSE
sqrt(min(without_allhog$mse))

#Remove more variables deemed unimportant from varImpPlot
varImpPlot(without_allhog, sort = FALSE, n.var = 40)

#The cnn and max variables are found to be unimportant
#Removing max_blue,green,red plus cnn_0,9,20,36,39,65
unnecessary <- c(3, 4, 6, 8, 9, 10, 16, 19, 20)

training_uncorr <- training_uncorr[,-unnecessary]
test_uncorr <- test_uncorr[,-unnecessary]

ncol(training_uncorr)
ncol(test_uncorr)

# Step 4: Final tuning ------------------------------------------------------

#Fitting a new RF to this new dataset
without_cnnmax <- tuneRF(
  x = training_uncorr[,-74], y = training_uncorr$growth_2_6,
  plot = TRUE, doBest = TRUE
)

# OOB RMSE
sqrt(min(without_allhog$mse))
```

```r
final <- cbind(testid, predict(without_cnnmax, newdata = test_uncorr))
colnames(final) <- c("id", "growth_2_6")
# write.csv(final, "rmcnn_max_final.csv", row.names = FALSE)

### SHRINKAGE METHODS ############################################################

# training <- read.csv("training_clean.csv")
# test <- read.csv("test_clean.csv")

set.seed(1)
# Values to try for lambda
grid <- 10^seq(10, -3, length = 100)

# Separating predictors and response
uncorr_x <- model.matrix(growth_2_6 ~ ., data = training_uncorr)[, -1]
uncorr_y <- training_uncorr$growth_2_6
uncorr_test_x <- data.matrix(test_uncorr)

# LASSO regression
uncorr_lasso <- glmnet(uncorr_x, uncorr_y, alpha = 1, family = "gaussian",
                       lambda = grid, standardize = TRUE)
uncorr_cv <- cv.glmnet(uncorr_x, uncorr_y, alpha = 1, family = "gaussian",
                       lambda = grid, standardize = TRUE,
                       nfolds = 10, type.measure = "mse")
# Lowest RMSE (square root of mean cross-validated error)
sqrt(min(uncorr_cv$cvm))
lambda <- uncorr_cv$lambda.min
pred_lasso <- predict(uncorr_lasso, newx = uncorr_test_x, s = lambda)

# Ridge regression
uncorr_ridge <- glmnet(uncorr_x, uncorr_y, alpha = 0, family = "gaussian",
                       lambda = grid, standardize = TRUE)
uncorr_cv2 <- cv.glmnet(uncorr_x, uncorr_y, alpha = 0, family = "gaussian",
                        lambda = grid, standardize = TRUE,
                        nfolds = 10, type.measure = "mse")
# Lowest RMSE (square root of mean cross-validated error)
sqrt(min(uncorr_cv2$cvm))
lambda <- uncorr_cv$lambda.min
pred_ridge <- predict(uncorr_ridge, newx = uncorr_test_x, s = lambda)

lasso_data <- cbind(id, pred_lasso)
colnames(lasso_data) <- c("id", "growth_2_6")
# write.csv(lasso_data, "lasso_uncorr.csv", row.names = FALSE)
```

```r
ridge_data <- cbind(id, pred_ridge)
colnames(ridge_data) <- c("id", "growth_2_6")
# write.csv(ridge_data, "ridge_uncorr.csv", row.names = FALSE)

### OTHER COMPARED METHODS ###########################################################

# Including PublishedDate ------------------------------------------------------

yt_test_original <- read_csv("test.csv")
yt_train_original <- read_csv("training.csv")

# Convert PublishedDate to number of seconds since 1970-01-01 00:00:00 UTC
yt_test <- mutate(
  yt_test_original,
  PublishedDate = as.numeric(
    as.POSIXct(PublishedDate, "%m/%d/%Y %H:%M", tz = "UTC")
  )
)
yt_train <- mutate(
  yt_train_original,
  PublishedDate = as.numeric(
    as.POSIXct(PublishedDate, "%m/%d/%Y %H:%M", tz = "UTC")
  )
)

# Remove variables w/same value for all observations
yt_train <- yt_train[, -useless_variables]
yt_test <- yt_test[, -useless_variables]

# Scaling predictors -----------------------------------------------------------

# Separate id variable
yt_train_id <- yt_train$id
yt_train_scaled <- select(yt_train, -id)
yt_test_id <- yt_test$id
yt_test_scaled <- select(yt_test, -id)
# Separate binary variables and response
yt_train_end <- select(yt_train, Num_Subscribers_Base_low:last_col())
yt_train_scaled <- select(yt_train_scaled, !Num_Subscribers_Base_low:last_col())
yt_test_end <- select(yt_test, Num_Subscribers_Base_low:last_col())
yt_test_scaled <- select(yt_test_scaled, !Num_Subscribers_Base_low:last_col())
# Scale remaining numeric variables
yt_train_scaled <- scale(yt_train_scaled)
```

```r
yt_test_scaled <- scale(
  yt_test_scaled,
  # Use the same centering and scaling factors as training
  center = attr(yt_train_scaled, "scaled:center"),
  scale = attr(yt_train_scaled, "scaled:scale")
)
# Rejoin separated variables
yt_train_scaled <- bind_cols(id = yt_train_id, yt_train_scaled, yt_train_end)
yt_test_scaled <- bind_cols(id = yt_test_id, yt_test_scaled, yt_test_end)

# Removing linear combinations ------------------------------------------------

# Initialize objects
yt_train_trimmed <- yt_train
yt_train_scaled_trimmed <- yt_train_scaled
# Find first linear combinations
linear_combos = yt_train_trimmed %>%
  select(where(is.numeric), -id, -growth_2_6) %>%
  findLinearCombos()
# Remove variables until no linear combinations remain
while (!is.null(linear_combos$remove)) {
  # Remove linear combinations
  yt_train_trimmed <- yt_train_trimmed[, -linear_combos$remove]
  yt_train_scaled_trimmed <- yt_train_scaled_trimmed[, -linear_combos$remove]
  # Check for additional linear combinations
  linear_combos <- yt_train_trimmed %>%
    select(where(is.numeric), -id, -growth_2_6) %>%
    findLinearCombos()
}

# Removing highly correlated variables ----------------------------------------

# punc_num_( and punc_num_) are obviously correlated so remove one
yt_train_trimmed <- select(yt_train_trimmed, -`punc_num_)`)
yt_train_scaled_trimmed <- select(yt_train_scaled_trimmed, -`punc_num_)`)

# Inspect correlated variables
hi_cor <- yt_train_trimmed %>%
  select(-id, -PublishedDate, -growth_2_6) %>%
  as.matrix() %>%
  cor() %>%
  findCorrelation(cutoff = 0.5, names = TRUE, exact = TRUE)
# It appears that many of the hog_* and cnn_* variables are highly correlated
# Let us create sets that do not contain the hog_* variables
```

```r
hi_cor_hog <- hi_cor[grepl(hi_cor, pattern = "hog")]
yt_train_hog <- select(yt_train_trimmed, !all_of(hi_cor_hog))
yt_train_scaled_hog <- select(yt_train_scaled_trimmed, !all_of(hi_cor_hog))
# Let us create sets that do not contain the cnn_* variables
hi_cor_cnn <- hi_cor[grepl(hi_cor, pattern = "cnn")]
yt_train_hog_cnn <- select(yt_train_hog, !all_of(hi_cor_cnn))
yt_train_scaled_hog_cnn <- select(yt_train_scaled_hog, !all_of(hi_cor_cnn))

# Multiple linear regression ------------------------------------------------

set.seed(101)

# Control parameters for train() to perform 10-fold cross-validation
tr_ctrl <- trainControl(method = "cv", number = 10)

# yt_train_trimmed
lm_trimmed <- train(
  growth_2_6 ~ ., data = yt_train_trimmed[, -1],
  method = "lm",
  trControl = tr_ctrl
)
metric <- c(lm_trimmed = lm_trimmed$results$RMSE)
# yt_train_scaled_trimmed
lm_scaled_trimmed <- train(
  growth_2_6 ~ ., data = yt_train_scaled_trimmed[, -1],
  method = "lm",
  trControl = tr_ctrl
)
metric <- c(metric, lm_sc_trimmed = lm_scaled_trimmed$results$RMSE)
# yt_train_hog
lm_hog <- train(
  growth_2_6 ~ ., data = yt_train_hog[, -1],
  method = "lm",
  trControl = tr_ctrl
)
metric <- c(metric, lm_hog = lm_hog$results$RMSE)
# yt_train_scaled_hog
lm_scaled_hog <- train(
  growth_2_6 ~ ., data = yt_train_scaled_hog[, -1],
  method = "lm",
  trControl = tr_ctrl
)
metric <- c(metric, lm_sc_hog = lm_scaled_hog$results$RMSE)
# yt_train_hog_cnn
```

```r
lm_hog_cnn <- train(
  growth_2_6 ~ ., data = yt_train_hog_cnn[, -1],
  method = "lm",
  trControl = tr_ctrl
)
metric <- c(metric, lm_hog_cnn = lm_hog_cnn$results$RMSE)
# yt_train_scaled_hog
lm_scaled_hog_cnn <- train(
  growth_2_6 ~ ., data = yt_train_scaled_hog_cnn[, -1],
  method = "lm",
  trControl = tr_ctrl
)
metric <- c(metric, lm_sc_hog_cnn = lm_scaled_hog_cnn$results$RMSE)

# RMSE
metric

# Shrinkage/Regularization ----------------------------------------------------

set.seed(3)

# Parameters to try
alphas <- c(0, 0.25, 0.5, 0.75, 1)
lambdas <- 10^seq(10, -10, length.out = 100)
tg <- data.frame(
  alpha = rep(alphas, each = length(lambdas)),
  lambda = rep(lambdas, length(alphas))
)

# yt_train_trimmed
shr_trimmed <- train(
  growth_2_6 ~ ., data = yt_train_trimmed[, -1],
  method = "glmnet",
  tuneGrid = tg,
  trControl = tr_ctrl
)
metric <- c(metric, shr_trimmed = min(shr_trimmed$results$RMSE))
# yt_train_scaled_trimmed
shr_scaled_trimmed <- train(
  growth_2_6 ~ ., data = yt_train_scaled_trimmed[, -1],
  method = "glmnet",
  tuneGrid = tg,
  trControl = tr_ctrl
)
```

```r
metric <- c(metric, shr_sc_trimmed = min(shr_scaled_trimmed$results$RMSE))
# yt_train_hog
shr_hog <- train(
  growth_2_6 ~ ., data = yt_train_hog[, -1],
  method = "glmnet",
  tuneGrid = tg,
  trControl = tr_ctrl
)
metric <- c(metric, shr_hog = min(shr_hog$results$RMSE))
# yt_train_scaled_hog
shr_scaled_hog <- train(
  growth_2_6 ~ ., data = yt_train_scaled_hog[, -1],
  method = "glmnet",
  tuneGrid = tg,
  trControl = tr_ctrl
)
metric <- c(metric, shr_sc_hog = min(shr_scaled_hog$results$RMSE))
# yt_train_hog_cnn
shr_hog_cnn <- train(
  growth_2_6 ~ ., data = yt_train_hog_cnn[, -1],
  method = "glmnet",
  tuneGrid = tg,
  trControl = tr_ctrl
)
metric <- c(metric, shr_hog_cnn = min(shr_hog_cnn$results$RMSE))
# yt_train_scaled_hog_cnn
shr_scaled_hog_cnn <- train(
  growth_2_6 ~ ., data = yt_train_scaled_hog_cnn[, -1],
  method = "glmnet",
  tuneGrid = tg,
  trControl = tr_ctrl
)
metric <- c(metric, shr_sc_hog_cnn = min(shr_scaled_hog_cnn$results$RMSE))

# Dimension reduction methods ------------------------------------------------

set.seed(1013)

# Principal Components Regression (PCR)

# yt_train_scaled_trimmed
pcr_sc_trimmed <- pcr(
  growth_2_6 ~ ., data = yt_train_scaled_trimmed[, -1],
  center = FALSE, scale = FALSE,
```

```
  validation = "CV", segments = 10
)
metric <- c(metric, pcr_sc_trimmed = min(RMSEP(pcr_sc_trimmed)$val["CV",,]))
# yt_train_scaled_hog
pcr_sc_hog <- pcr(
  growth_2_6 ~ ., data = yt_train_scaled_hog[, -1],
  center = FALSE, scale = FALSE,
  validation = "CV", segments = 10
)
metric <- c(metric, pcr_sc_hog = min(RMSEP(pcr_sc_hog)$val["CV",,]))
# yt_train_scaled_hog_cnn
pcr_sc_hog_cnn <- pcr(
  growth_2_6 ~ ., data = yt_train_scaled_hog_cnn[, -1],
  center = FALSE, scale = FALSE,
  validation = "CV", segments = 10
)
metric <- c(metric, pcr_sc_hog_cnn = min(RMSEP(pcr_sc_hog_cnn)$val["CV",,]))

# Partial Least Squares (PLS) regression

# yt_train_scaled_trimmed
pls_sc_trimmed <- plsr(
  growth_2_6 ~ ., data = yt_train_scaled_trimmed[, -1],
  center = FALSE, scale = FALSE,
  validation = "CV", segments = 10
)
metric <- c(metric, pls_sc_trimmed = min(RMSEP(pls_sc_trimmed)$val["CV",,]))
# yt_train_scaled_hog
pls_sc_hog <- plsr(
  growth_2_6 ~ ., data = yt_train_scaled_hog[, -1],
  center = FALSE, scale = FALSE,
  validation = "CV", segments = 10
)
metric <- c(metric, pls_sc_hog = min(RMSEP(pls_sc_hog)$val["CV",,]))
# yt_train_scaled_hog_cnn
pls_sc_hog_cnn <- plsr(
  growth_2_6 ~ ., data = yt_train_scaled_hog_cnn[, -1],
  center = FALSE, scale = FALSE,
  validation = "CV", segments = 10
)
metric <- c(metric, pls_sc_hog_cnn = min(RMSEP(pls_sc_hog_cnn)$val["CV",,]))

sort(metric)
```