BA 810 Lab 2: Stepwise regression

The problem

You have now explored the computation advertising dataset provided by Cogo Labs, and you have some intution as to the determinants of email open rates. It is now time to translate this intuition into a machine learning to predict email open rates for new customers.

Preparing for the lab

Before we get started, let's load some useful libraries.

```
library(tidyverse)
library(ggplot2)
library(ggthemes)

# I like this ggplot theme but you might prefer a different one
theme_set(theme_bw())
```

The data

Training dataset

The training dataset we will use comes from Lab 1; refer to the Lab 1 description for details. Let load the training data.

IMPORTANT: you may need to update the filename below to match the location of the dataset on your computer.

```
dd <- read_tsv("data/cogo-train.tsv")
```

Train and test datasets

As usual, start by splitting your data into a training set and a test. Here's how to do it:

```
# this is needed to create reproducible results,
# otherwise your random split will be different each time you run the code
set.seed(1234)

# This will split into train and test 70-30
dd$train <- sample(c(0, 1), nrow(dd), replace = TRUE, prob = c(.3, .7))
dd_test <- dd %>% filter(train == 0)
dd_train <- dd %>% filter(train == 1)
```

The training data contains 201917 observations. This will slow down training. Let's take a random subsample, and then when we are happy with the tuning of our algorithms, we can increase the size of the training set further.

Stepwise regression

```
# these this the complete set of predictors of will use
xnames <- colnames(dd)
```

```r
# we have to be careful to remove the dependent variable p_open and the auxiliary variable train from t
xnames <- xnames[!xnames %in% c("train", "p_open")]
```

Forward selection

Recall that forward selection starts by fitting an intercept only model and then continues by adding at each step the variable that reduces MSE the most.

Let's start by fitting an intercept only model:

```r
fit_fw <- lm(p_open ~ 1, data = dd_train)
```

Now, let's calcucate MSE train and MSE test for the intercept-only model:

```r
yhat_train <- predict(fit_fw, dd_train)
mse_train <- mean((dd_train$p_open - yhat_train) ^ 2)

yhat_test <- predict(fit_fw, dd_test)
mse_test <- mean((dd_test$p_open - yhat_test) ^ 2)
```

Let's store track our results so we can examine them later:

```r
log_fw <-
  tibble(
    xname = "intercept",
    model = deparse(fit_fw$call),
    mse_train = mse_train,
    mse_test = mse_test
  )
```

Next, we have select the variable our of xnames that if added to the intercept-only model reduces MSE *test* the most. To do this we have fit as many models as the number of variables in xnames, and keep the best model. We will do this using a loop. We will use the update command, which takes an existing model and updates this to include more predictors.

```r
best_mse_train <- NA
best_mse_test <- NA
best_fit_fw <- NA
best_xname <- NA

for (xname in xnames) {
  # take a moment to examine and understand the following line
  fit_fw_tmp <- update(fit_fw, as.formula(paste0(" ~ ", xname)))

  # compute MSE train
  yhat_train_tmp <- predict(fit_fw_tmp, dd_train)
  mse_train_tmp <- mean((dd_train$p_open - yhat_train_tmp) ^ 2)

  # compute MSE test
  yhat_test_tmp <- predict(fit_fw_tmp, dd_test)
  mse_test_tmp <- mean((dd_test$p_open - yhat_test_tmp) ^ 2)

  # if this is the first predictor to be examined,
  # or if this predictors yields a lower MSE that the current
  # best, then store this predictor as the current best predictor
  if (is.na(best_mse_test) | mse_test_tmp < best_mse_test) {
```

```
    best_xname <- xname
    best_fit_fw <- fit_fw_tmp
    best_mse_train <- mse_train_tmp
    best_mse_test <- mse_test_tmp
  }
}
```

Which variable increases out-of-sample predictive accuracy the most?

```
print(best_xname)
```

```
## [1] "state"
```

Let's update our log:

```
log_fw <-
  log_fw %>% add_row(
    xname = best_xname,
    model = paste0(deparse(best_fit_fw$call), collapse = ""),
    mse_train = best_mse_train,
    mse_test = best_mse_test
  )
```

Let's examine the contents of our log:

```
log_fw
```

```
## # A tibble: 2 x 4
##   xname     model                                          mse_train mse_test
##   <chr>     <chr>                                              <dbl>    <dbl>
## 1 intercept lm(formula = p_open ~ 1, data = dd_train)         0.0288   0.0285
## 2 state     lm(formula = p_open ~ state, data = dd_trai~      0.0269   0.0267
```

It appears that the variable yielding the highest out-of-sample increase in accuracy is "state".

Let's remove "state" from xnames:

```
xnames <- xnames[xnames[xnames!=best_xname]]
```

The procedure above shows you how update a linear regression model to add the next best predictor. Continue implementing the forward selection algorithm by iteratively adding all predictors in correct order (ie, always adding the predictor that decreases MSE the most compared to the currect best model) until no predictors are left. Keep logging your progress in log_fw.

Here is a template to get you started

```
xnames <- colnames(dd_train)
xnames <- xnames[!xnames %in% c("train", "p_open", "user_id")]

fit_fw <- lm(p_open ~ 1, data = dd_train)

yhat_train <- predict(fit_fw, dd_train)
yhat_test <- predict(fit_fw, dd_test)

mse_train <- ...
mse_test <- ...
xname <- "intercept"

log_fw <-
```

```
  tibble(
    xname = xname,
    model = paste0(deparse(fit_fw$call), collapse = ""),
    mse_train = mse_train,
    mse_test = mse_test
  )

# an outer loop considers all potential predictors until all of them
# have been added to the model
while (length(xnames) > 0) {
  # keep track of which is the next best variable to add
  best_mse_train <- NA
  best_mse_test <- NA
  best_fit_fw <- NA
  best_xname <- NA

  # select the next best predictor
  for (xname in xnames) {
    # fit a model that adds the predictor xname to the current best model
    # to do this you will want to use the update() command which can add a predictor
    # to an existing model
    fit_fw_tmp <- ...

    # compute MSE train
    yhat_train_tmp <- ...
    mse_train_tmp <- ...

    # compute MSE test
    yhat_test_tmp <- ...
    mse_test_tmp <- ...

    # if this is the first predictor to be examined,
    # or if this predictors yields a lower MSE that the current
    # best, then store this predictor as the current best predictor
    if (is.na(best_mse_test) | mse_test_tmp < best_mse_test) {
      best_xname <- ...
      best_fit_fw <- ...
      best_mse_train <- ...
      best_mse_test <- ...
    }
  }

  # update the log
  log_fw <-
    log_fw %>% add_row(
      ...
    )

  # adopt the best model for the next iteration
  fit_fw <- best_fit_fw

  # remove the current best predictor from the list of predictors
  xnames <- xnames[xnames!=best_xname]
```

```
}
```

In the end, which set of variables yield the lowest MSE test?

Create a plot showing train and test MSE as you add more variables to the model. The x-axis should show the most recently added variable. The y-axis should show MSE. Two seprate lines should show train and test MSE.

Backward selection

Follow similar steps to implement backwards regression.

Produce a similar figure plotting train and test MSE as you *remove* variables iteratively from a fully saturated model.

Do forward and backward selection agree on what is the best model in terms on test MSE?