HW4 – STATS305A

Isaac Kleisle-Murphy

12/8/2021

Code

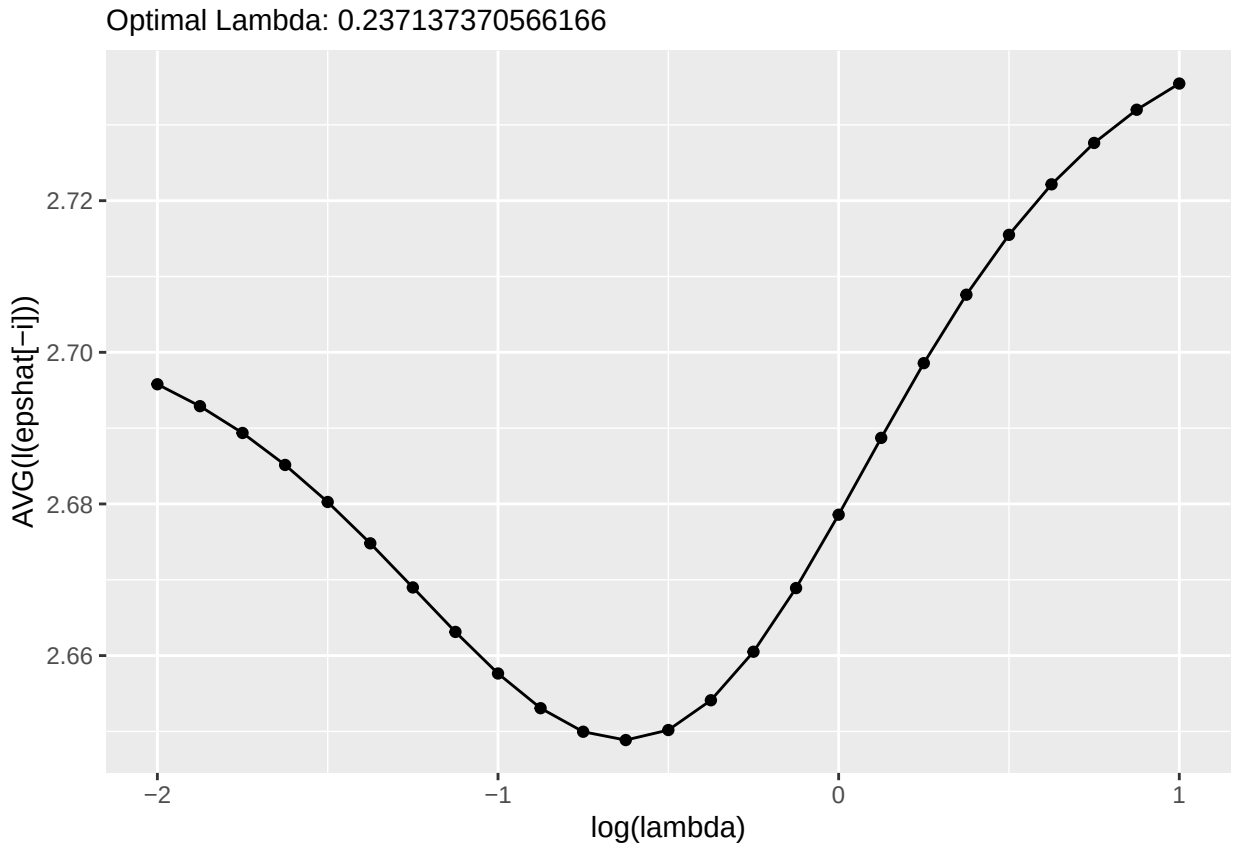
```
suppressMessages(library(dplyr))
suppressMessages(library(ggplot2))
source("~/Stanford/STATS305A/generate_outlier_data.R")


# Ingest + Setup ------------------------------------------------------------
# # make data
# data = generate.data()
# # split it up
# x_train = data$X.train; x_test = data$X.test
# y_train = data$y.train; y_test = data$y.test
# # print dims for sanity
# cat(
#   paste0(rep("#", 75), collapse=""), "\n",
#   "Dimensions:\n",
#   "\t X-train: \t", dim(x_train), "\n",
#   "\t Y-train: \t", dim(y_train), "\n",
#   "\t X-test: \t", dim(x_test), "\n",
#   "\t X-test: \t", dim(y_test), "\n",
#   paste0(rep("#", 75), collapse="")
# )


# Utils ---------------------------------------------------------------------

# establish sigmoid function
sigmoid <- function(x, op='-'){
  #' Basic sigmoid, you pick the sign
  if (op == '-'){
    x_sign = -x
  }else{
    x_sign = x
  }
  exp(x_sign) / (1 + exp(x_sign))
}

loss_fn <- function(x){log(1 + exp(x)) + log(1 + exp(-x))}
```

```r
loss_prime <- function(x){
  # sigmoid(x, "+") - sigmoid(x, "-")
  tanh(x/2)
}


loss_double_prime <- function(x){
  # second derivative, no chain rule, of loss function
  sigmoid(x, '+')/(1 + exp(x)) +
    sigmoid(x, '-')/(1 + exp(-x))
}



score_loo <- function(X, y, lambda=1e-3){
  suppressMessages(require(dplyr))
  # number of observations
  N = nrow(X)
  # does a single round of LOO
  beta_hat = minimize.robust.ridge(X, y, lambda)
  # residuals
  eps_hat = as.numeric(y - (X %*% beta_hat))
  # diagonal matrix of losses, double primed
  L = eps_hat %>%
    loss_double_prime() %>%
    diag()
  # solve H: X^T(L'')X
  H = lambda * diag(dim(X)[2]) + (t(X) %*% L %*% X) / N
  # invert it
  H_inv = solve(H)
  # compute H_ks. Forgetting broadcasting rules off top of my head for R,
  # so just going to stuff into list
  loo_errors = sapply(1:N, function(k){
    # apply problem 1: Sherman-Morrison to get Hk
    #   A = H
    #   u = l''(\hat\epsilon_k)x_k
    #   v^T = x_k^T
    # setup SMW decomp; easier to track
    A_inv = H_inv
    u = (-L[k, k] / N) * X[k, ]
    v = X[k, ]
    # solve for H_k^{-1}
    Hk_inv = A_inv -
      (A_inv %*% u %*% t(v) %*% A_inv) /
        as.numeric(
          1 + t(v) %*% A_inv %*% u
        )


    # direct solve, for comparison
    # Hk_inv_ = solve(H - (1/N) * L[k, k] * X[k, ] %*% t(X[k, ]) )

    # l'(t) = exp(t) / (1 + exp(t)) - exp(-t) / (1 + exp(-t)) = tanh(t/2)
```

```
    # by above
    # sigmoid(eps_hat[k], "+") - sigmoid(eps_hat[k], "-") == tanh(eps_hat[k] / 2)
    g_k = tanh(eps_hat[k] / 2) * X[k, ] / N

    # solve out
    yhat_not_k = as.numeric(t(X[k, ]) %*% beta_hat) - #\hat y
      as.numeric(t(X[k, ]) %*% Hk_inv %*% g_k)

    # LOO error
    eps_hat_not_k = y[k,] - yhat_not_k
    # out
    eps_hat_not_k
  })
  mean(loss_fn(loo_errors))
}
```

Problem 1.III)

```
# # make data
data = generate.data(random_state=31)
# split it up
x_train = data$X.train; x_test = data$X.test
y_train = data$y.train; y_test = data$y.test


log_lambda_seq = seq(-2, 1, length.out=25) # provided by MC.
sapply(log_lambda_seq, function(ll){
  score_loo(X=x_train, y=y_train, lambda=10^ll)
}) -> loo_results


ggplot(data.frame(lambda = log_lambda_seq, loo_score = loo_results),
       aes(x=lambda, y=loo_score)) +
  geom_point() +
  geom_path() +
  labs(x = "log(lambda)",
       y="AVG(l(epshat[-i]))",
       subtitle = paste0("Optimal Lambda: ", as.character((10 ^ log_lambda_seq)[which.min(loo_results)])
       )
```

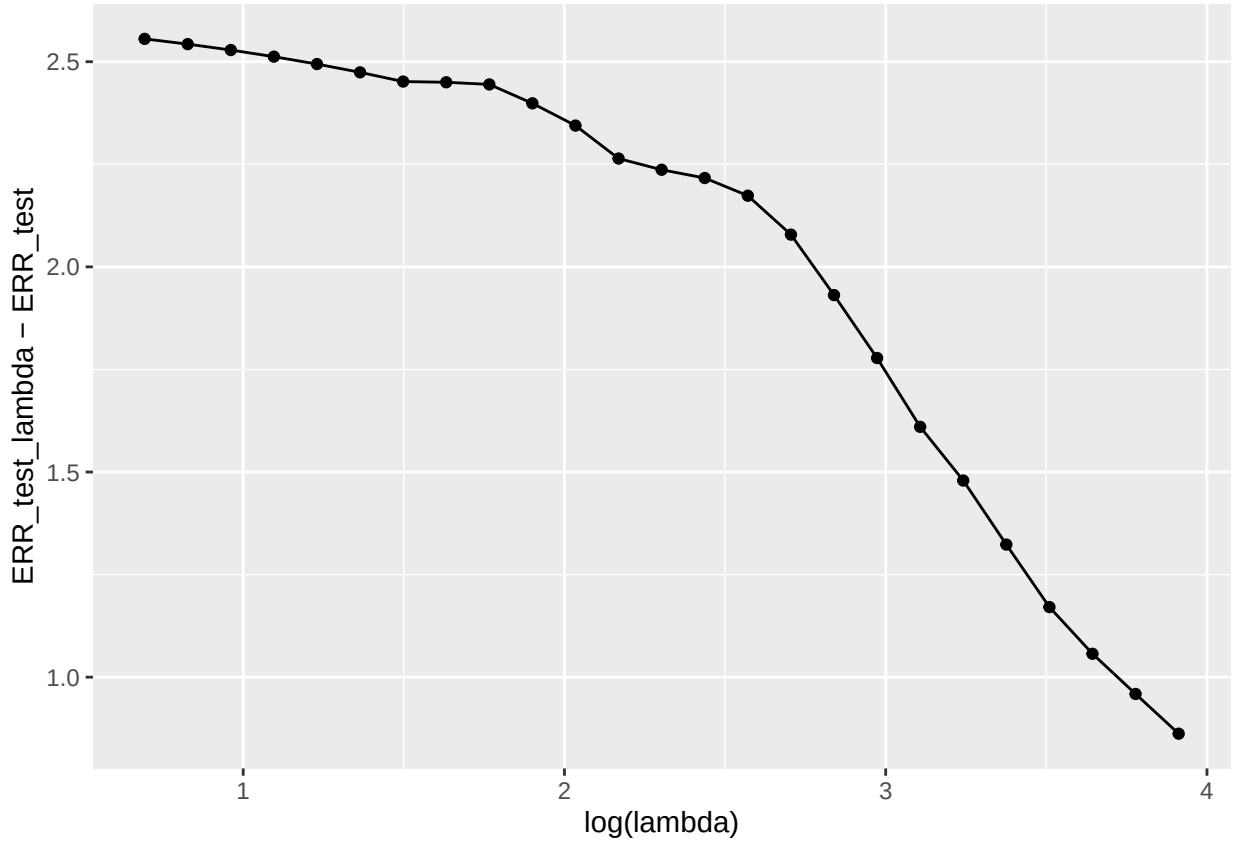


As we see, we're doing our best in the LOO sense around lambda = .24 – roughly right in the middle of our tuning grid.

Problem 1.IV

```
# fit and score tuned
beta_tune = minimize.robust.ridge(x_train, y_train, (10^log_lambda_seq)[which.min(loo_results)])
yhat_tune = as.numeric(x_test %*% beta_tune)
median_abs_err_tune = median(abs(y_test - yhat_tune))

# fit and score others
log_lambda_seq_iv <- seq(log(2), log(50), length.out=25) # provided by MC.
sapply(log_lambda_seq_iv, function(ll){
  # beta_iter = minimize.robust.ridge(x_train, y_train, lambda=exp(ll))
  # use normal ridge equations
  beta_iter = solve(t(x_train) %*% x_train + exp(ll) * diag(dim(x_train)[2])) %*% t(x_train) %*% y_trai
  yhat_test = as.numeric(x_test %*% beta_iter)
  median_abs_err = median(abs(y_test - yhat_test))
  median_abs_err
}) -> result_part_d

ggplot(
  data.frame(
    log_lambda=log_lambda_seq_iv,
```

```
    median_abs_error_gap=result_part_d - median_abs_err_tune
  ),
  aes(x=log_lambda, y=median_abs_error_gap)
) +
  geom_point() +
  geom_path() +
  labs(x="log(lambda)",
       y="ERR_test_lambda - ERR_test"
       )
```



As we see, the gap between the iterated lambdas' median absolute error and the tuned median absolute error is overwhelmingly positive. For the default 10 outliers, the robust regression procedure with tuned regularization achieved a smaller median error in all head-to-head matchups with the ridge regression. This is likely due to the fact that it (on account of it's specialized loss function) was not as "distracted"/"misled" by the outliers as the ridge procedure – whereas the robust procedure's loss function allowed it to cut through the outliers and fit to the heart of the data, the ridge regression, on account of its squared error loss, would have to make more compromises for the outliers, resulting in a worse fit.

Problem 1.V

Now, we stuff all of the above into a big loop:

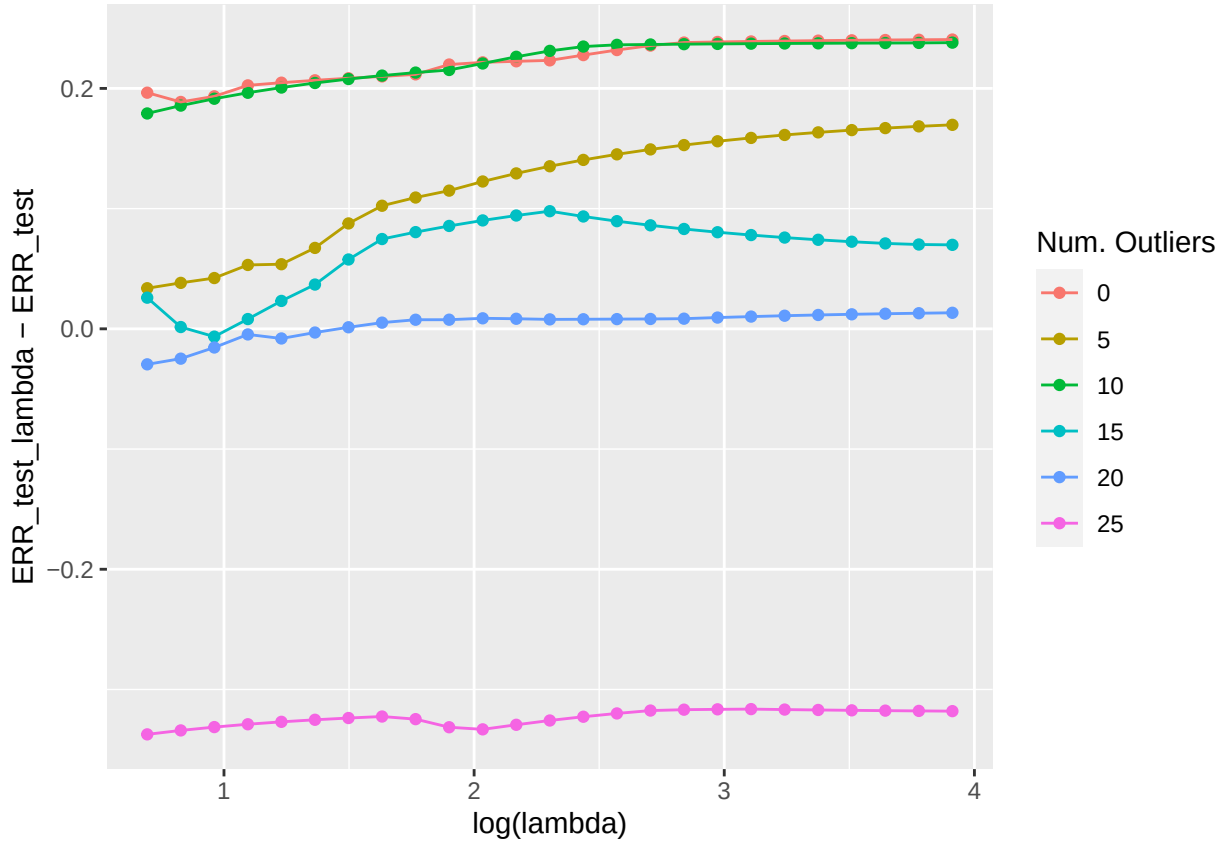
```
outlier_seq = seq(0, 25, 5)
result_outer = list()
```

```r
for (n_out in outlier_seq){
  print(n_out)
  # # make data
  data = generate.data(random_state=n_out, num.outliers=n_out)
  # split it up
  x_train = data$X.train; x_test = data$X.test
  y_train = data$y.train; y_test = data$y.test


  log_lambda_seq = seq(-2, 1, length.out=25) # provided by MC.
  sapply(log_lambda_seq, function(ll){
    score_loo(X=x_train, y=y_train, lambda=10^ll)
  }) -> loo_results

  # fit and score tuned
  beta_tune = minimize.robust.ridge(x_train, y_train, (10^log_lambda_seq)[which.min(loo_results)])
  yhat_tune = as.numeric(x_test %*% beta_tune)
  median_abs_err_tune = median(abs(y_test - yhat_tune))

  # fit and score others
  log_lambda_seq_iv <- seq(log(2), log(50), length.out=25) # provided by MC.
  sapply(log_lambda_seq_iv, function(ll){
    beta_iter = minimize.robust.ridge(x_train, y_train, lambda=exp(ll))
    yhat_test = as.numeric(x_test %*% beta_iter)
    median_abs_err = median(abs(y_test - yhat_test))
    median_abs_err
  }) -> result_part_d

  result_outer[[as.character(n_out)]] = data.frame(
      log_lambda=log_lambda_seq_iv,
      median_abs_error_gap=result_part_d - median_abs_err_tune
    ) %>%
      mutate(n_outliers = n_out)

  # ggplot(
  #   data.frame(
  #     log_lambda=log_lambda_seq_iv,
  #     median_abs_error_gap=result_part_d - median_abs_err_tune
  #   ),
  #   aes(x=log_lambda, y=median_abs_error_gap)
  # ) +
  #   geom_point() +
  #   geom_path() +
  #   labs(x="log(lambda)",
  #        y="Test data: med_ae(lambda) - med_ae(tuned)",
  #        subtitle=paste0("Outliers: ", as.character(n_out))
  #        ) -> plt
  # print(plt)

}
```

```
## [1] 0
## [1] 5
```

```
## [1] 10
## [1] 15
## [1] 20
## [1] 25
```

```
result_outer = do.call("rbind", result_outer)

ggplot(result_outer,
       aes(x=log_lambda,
           y=median_abs_error_gap,
           color=as.factor(n_outliers))) +
  geom_point() +
  geom_path() +
  labs(x="log(lambda)",
       y="ERR_test_lambda - ERR_test") +
  scale_color_discrete(name="Num. Outliers")
```



Above, we see that the robust regression retains an advantage (at least in terms of test median absolute error – ERR_test_lambda - ERR_test is positive) when there are fewer outliers in the training data, while the ridge eventually gains the upper hand as outliers exceed approximately 20. This too makes sense: when there are only a handful of outliers in the dataset, the robust procedure does a better job of seeing through them and focusing on the heart of the data. However, when the number of those outliers increase to the point that they are regularly-occuring noise (20-25 "outliers" in an N=100 dataset is a good chunk of the data), the ridge becomes preferable, as those "outliers" are now a semi-frequent occurence and demand greater attention (which squared-error loss gives them) from the fitted model.