Machine Learning - Lab 1

Yiran Wang, Tore Andersson and Shashi Nagarajan Iyer

11/11/2020

Question 1

Question 2

Bayesian ridge regression model for the data (without the intercept parameter) is:

$$\mathbf{y}|\mathbb{X}, w_0, \mathbf{w} \sim N(w_0 + \mathbb{X}\mathbf{w}, \sigma^2\mathbb{I}),$$

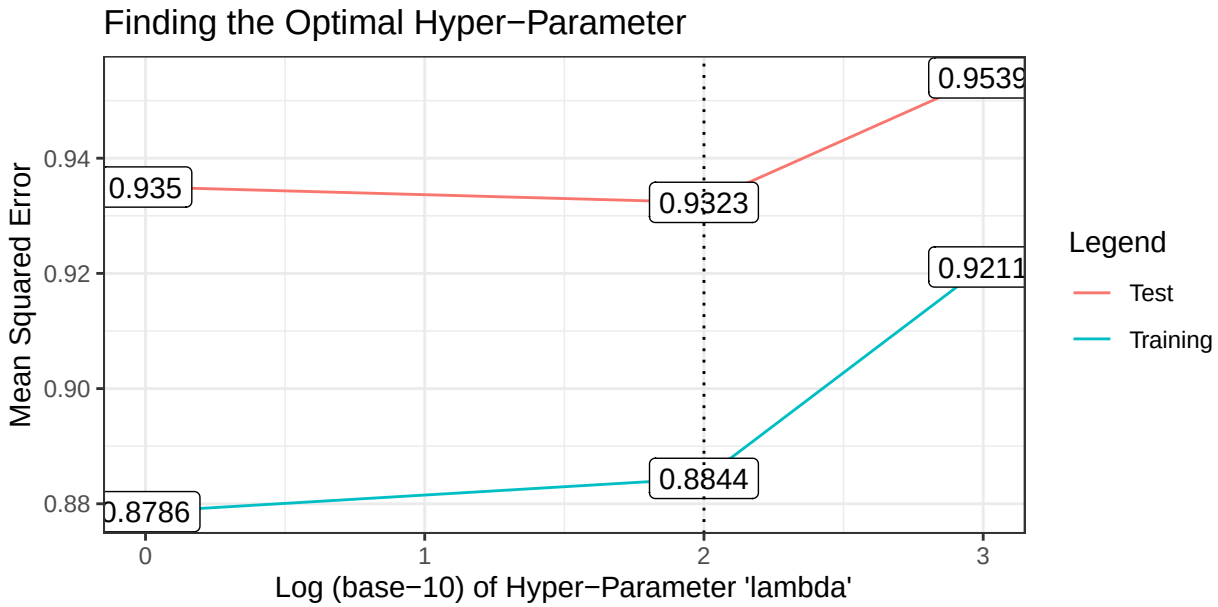
where:

- $\mathbf{y}$ is the vector of Parkinson's disease symptom scores, "motor UPDRS", the dependent variable
- $\mathbb{X}$ is the design matrix containing the observed values of all the independent variables corresponding to $\mathbf{y}$ (voice characteristics in this case)
- $\boldsymbol{w}$ is the vector of bayesian ridge regression parameters
- σ is a scalar such that $\sigma^2 \cdot \mathbb{I}$ is the covariance matrix of the linear regression error terms

The Bayesian prior for $\boldsymbol{w}$ is:

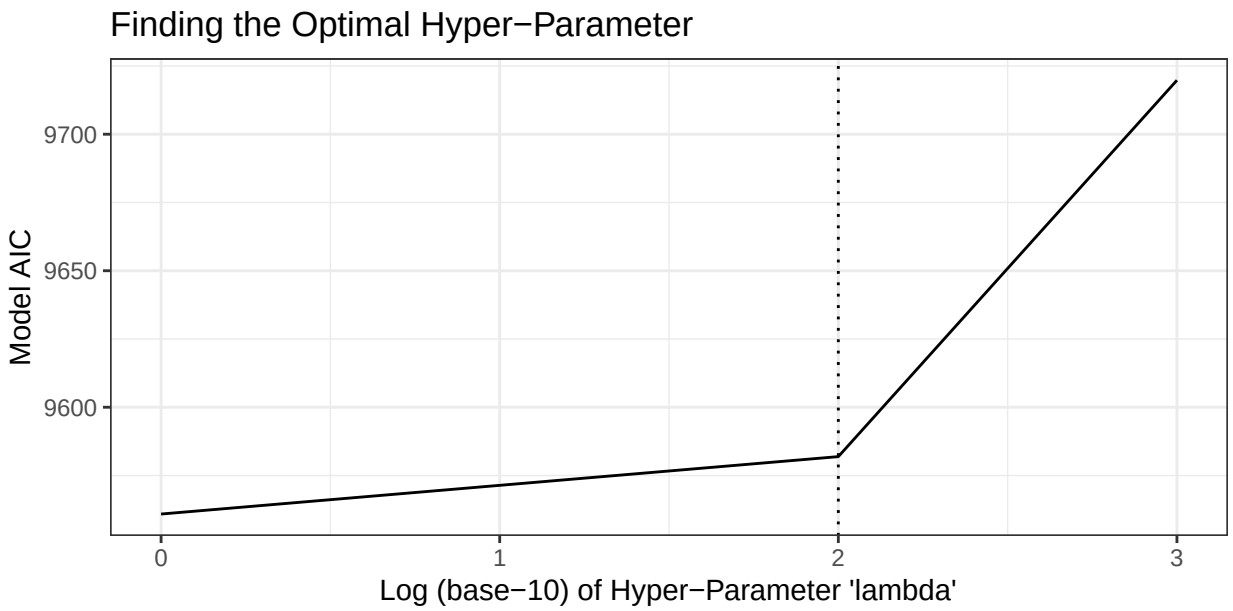
$$\mathbf{w} \sim N(0, \frac{\sigma^2}{\lambda}\mathbb{I})$$

Fitting the above mentioned ridge regression model on the training dataset, with $\lambda = 1, 100 and 1,000$, we see the training and test MSE values as shown in the plot below.



We can see that the optimal value of λ within the set $\{1, 100, 1000\}$ is 100. At $\lambda = 1$, there is overfitting, evidenced by low training error and large test error; at $\lambda = 1000$, on the other hand, there is underfitting, evidenced by high training and test error.

In terms of AIC as well, the optimal value of λ within the set $\{1, 100, 1000\}$ is 100. One theoretical advantage of using AIC is that unlike the holdout method, it can be used for model selection in unsupervised learning models, where losses, unlike MSE, will not be a function of 'labels' and where one cannot ascertain under/overfitting based on test MSEs.



Question 3

The linear regression model for the data is:

$$\mathbf{y} \sim \mathcal{N}(\mathbb{X}\boldsymbol{\beta}, \sigma^2\mathbb{I}),$$

where:

- $\mathbf{y}$ is the vector "Fat" (the dependent variable)
- $\mathbb{X}$ is the design matrix containing the observed values of all the independent variables corresponding to $\mathbf{y}$ (absorbance characteristics, in thtis case), prefixed with a column of 1s for the intercept
- $\boldsymbol{\beta}$ is the vector true linear regression parameters
- σ is a scalar such that $\sigma^2 \cdot \mathbb{I}$ is the covariance matrix of the linear regression error terms

Fitting the above mentioned model on the given dataset we see low training error and high test error; errors were measured using the mean-squared error loss function.

```
## Training MSE is: 0.007108636
```

```
## Test MSE is:  635.6713
```

These statistics indicate overfitting. On average training predictions are very close to actual training labels, but test predictions are relatively very different from test labels. As a result we cannot use satisfactorily use this model to predict fat content in any new meat sample.

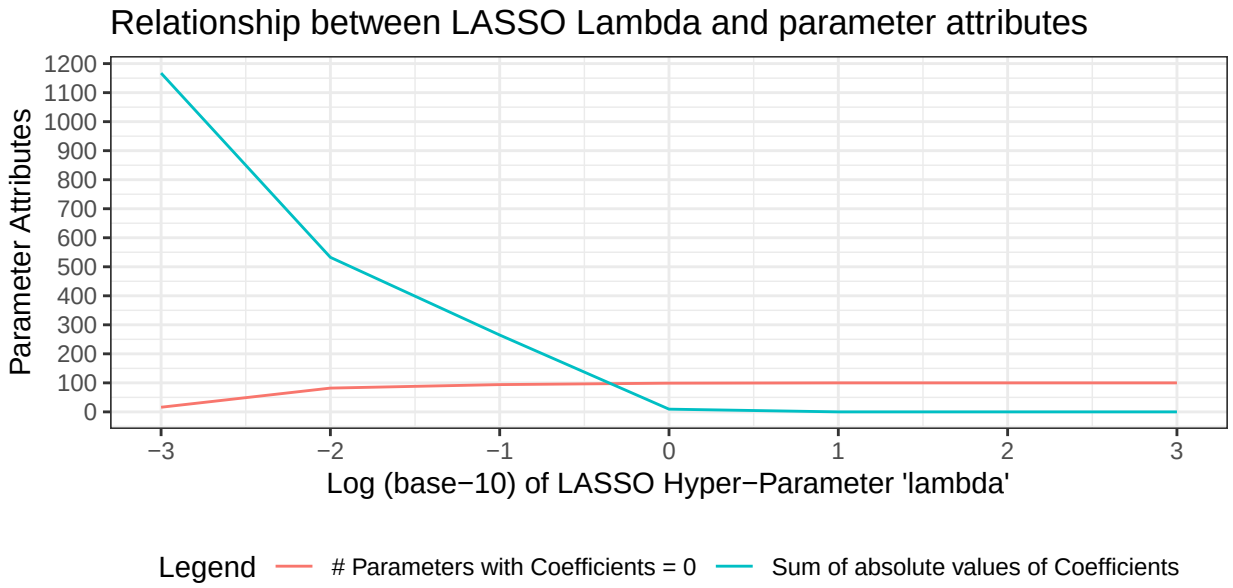
If we turn to the LASSO regression method, however, we would be selected $\boldsymbol{\beta}$ such that the following loss function is minimised:

$$\sum_{i=1}^{N} (y_i - \mathbb{X}_i \boldsymbol{\beta})^2 + \lambda \sum_{j=1}^{p} |\beta_i|$$

where, additionally:

- N is the total number of observations in the dataset
- i is an index the observations in the dataset, and runs from 1, 2, 3, ... N
- λ is the LASSO hyper-parameter

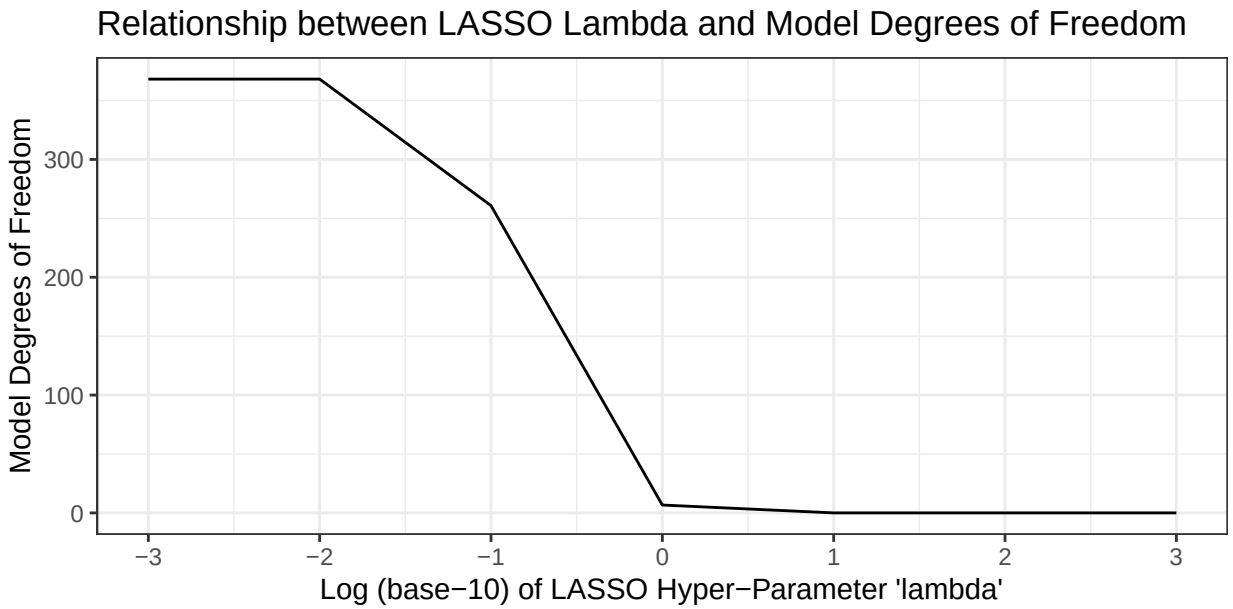
With $\lambda \in \{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$, LASSO regression estimates parameters with attributes as shown in the plot below:



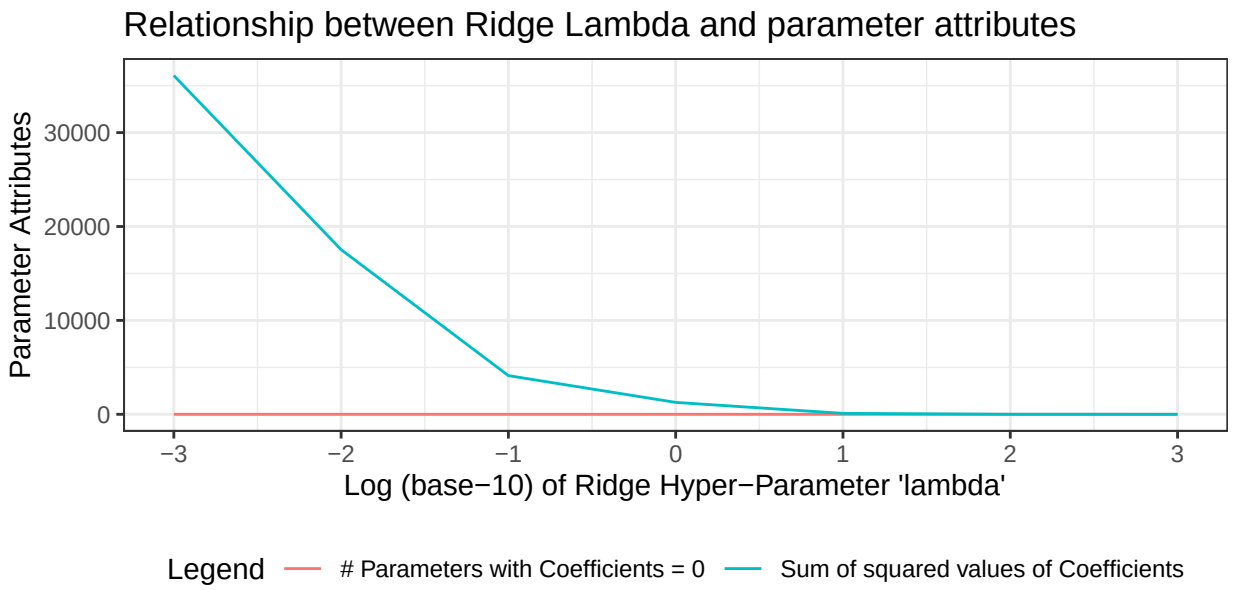
What we see above, as expected in theory, is that increases in λ result in increased penalty on the absolute values of parameter estimates. When $\lambda >= 1$, LASSO reduces to estimates of all 100 parameters to zero and brings down the value of the sum of absolute values of all parameters estimates to zero.

The most appropriate value for lambda in a case where there are only 3 independent variables can, as is common, be estimated through cross-validation or holdout sample methods, but one should not expect the estimates to be very large.

Plotting model degrees of freedom versus λ on log (base 10) scale, we can see the former decrease as λ increases; with very large values of lambda (such as $\lambda >= 100$ in this case), degrees of freedom will drop to zero, the lowest possible value for this measure.

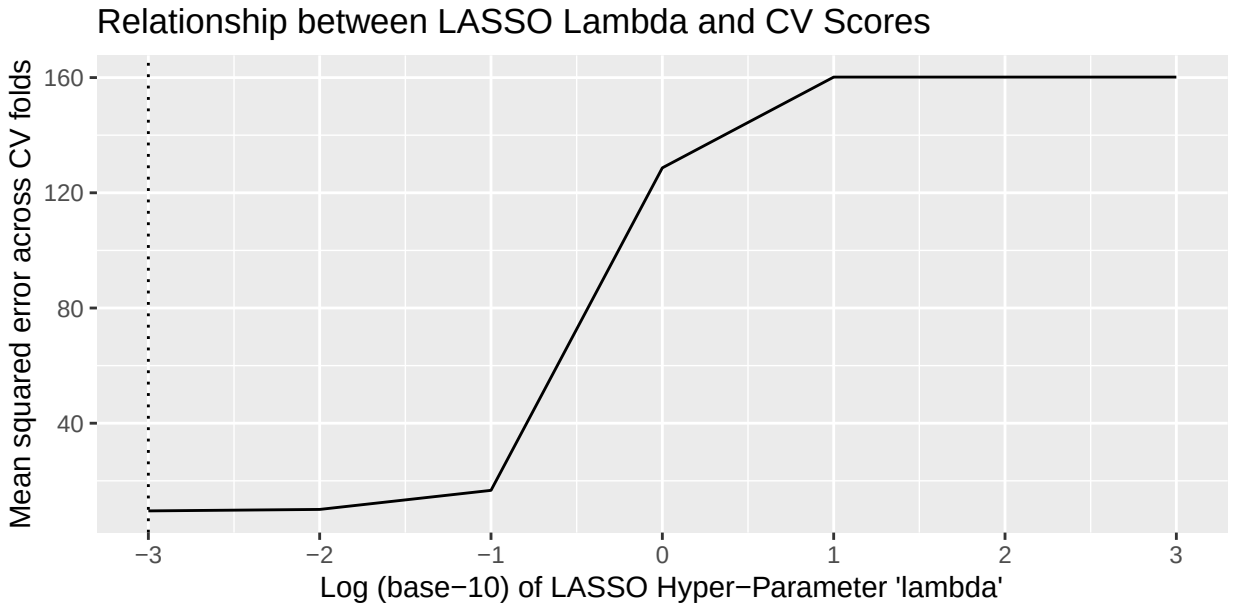


With $\lambda \in \{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$, ridge regression, on the other hand, estimates parameters with attributes as shown in the plot below:

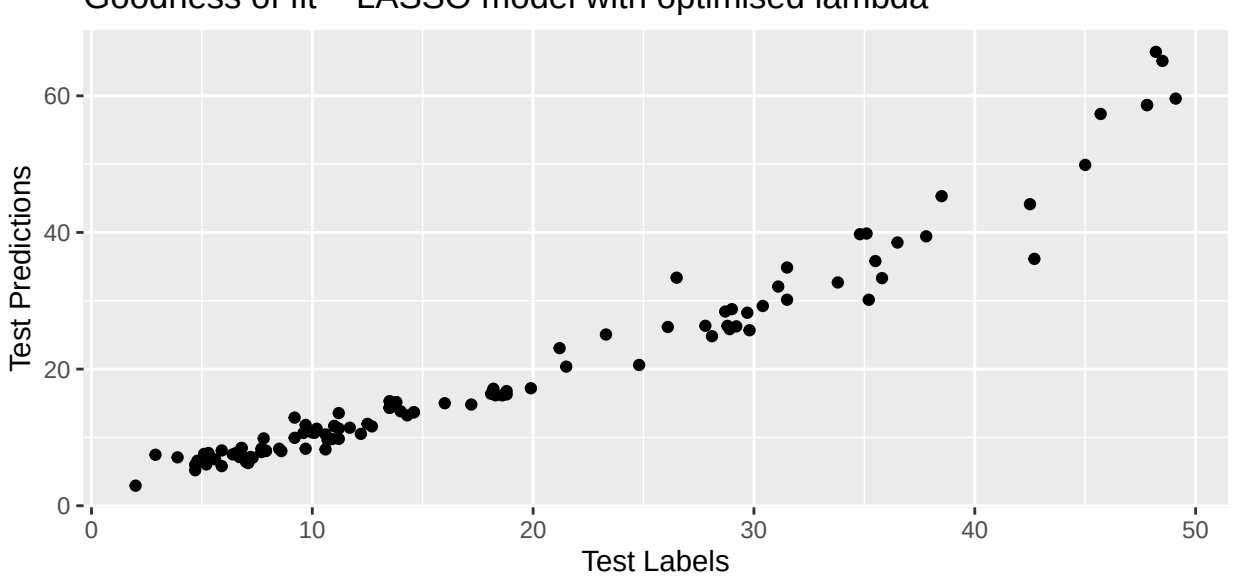


Upon comparison, it becomes clear that ridge regression, unlike LASSO, does not bring down any parameter estimates to zero.

Upon running 3-fold cross-validation on the training dataset, we can see that CV score increases as λ increases.

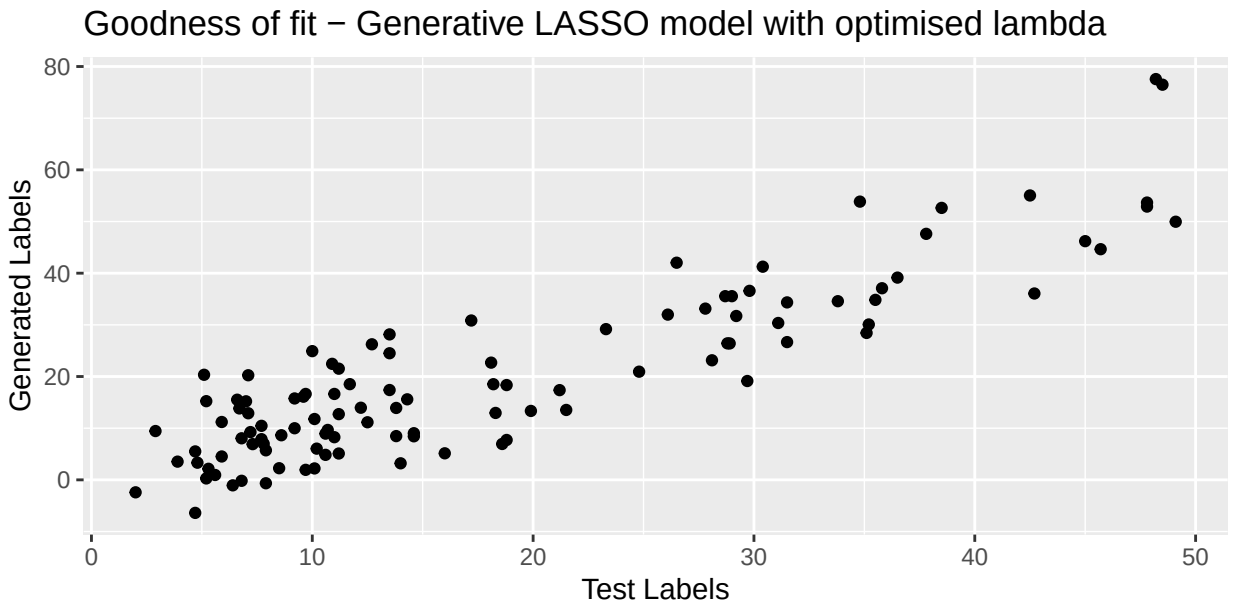



```
## Optimal Lambda is: 0.001

## Number of variables chosen in the model: 84

## The p-value for the test of null hypothesis that optimal lambda works similar to log(lambda) = -2 vs

## We, therefore, cannot conclude confidently that the optimal lambda works significantly better than lo
```



The LASSO regression model with optimised $\lambda = 10^-3$ produces good test predictions which are concentrated around the $y = x$ line in the plot of Predictions vs. Actual Labels.

The same, however, is not a good generative model, because of the assumption that the training dataset is deterministic. This can also be seen in the plot below, where points generated from the $\mathcal{N}(\mathbb{X}\hat{\beta}_{LASSO}, \hat{\sigma}^2_{MLE})$ are quite spread out around the $y = x$ line.



Appendix: Code

Question 2

```
# read in data
parkinsons <- read_csv("parkinsons.csv", col_types = cols())

# drop columns not to be used for prediction
parkinsons <- parkinsons[, -c(1, 2, 3, 4, 6)]

# bayesian model

## $$\mathbf{y} | \mathbb{X}, w_0, \mathbf{w} \sim N(w_0 + \mathbb{X}\mathbf{w}, \, \sigma^2 \mathbb{I}
## $\boldsymbol{w}$ is: $$\mathbf{w} \sim N(0, \, \frac{\sigma^2}{\lambda}\mathbb{I})$$

# create train/test partitions
all_indices <- 1:(nrow(parkinsons))
set.seed(12345)
train_indices <- sample(all_indices, ceiling(0.6*nrow(parkinsons)))
test_indices <- all_indices[!(all_indices %in% train_indices)]
train_data <- parkinsons[train_indices, ]
test_data <- parkinsons[test_indices, ]

# scale data -- see https://sebastianraschka.com/faq/docs/scale-training-test.html for why training mu/
n_col <- ncol(parkinsons)
train_mu <- sapply(1:n_col, function(x) mean(unlist(train_data[x])))
train_sigma <- sapply(1:n_col, function(x) sd(unlist(train_data[x])))

train_data <- as.matrix(as.data.frame(
  sapply(1:n_col, function(x) (train_data[x] - train_mu[x])/train_sigma[x]))) # as.matrix has no effect
test_data <- as.matrix(as.data.frame(
  sapply(1:n_col, function(x) (test_data[x] - train_mu[x])/train_sigma[x])))

# set up helper functions
```

```r
model_log_likelihood <- function(w_vector, sigma, X) { # built for no intercept models, as asked

  n <- nrow(X)
  term1 <- -n/2*log(2*pi*sigma^2)
  term2 <- -sum(sapply(1:n, function(x) X[x, 1] - X[x, -1] %*% w_vector)^2)/2/sigma^2 # assumes y = X[,

  return(term1 + term2)

}

ridge <- function(w_vector_sigma, X, lambda) { # built for no intercept models, as asked

  n <- ncol(X)
  sigma <- w_vector_sigma[n]
  w_vector <- w_vector_sigma[1:(n-1)]
  return(lambda*sum(w_vector^2) - model_log_likelihood(w_vector, sigma, X))

}

ridge_opt <- function(X, lambda) {

  n <- ncol(X)
  par <- rnorm(n-1) # initial parameter values (to seed optimisation); random
  par[n] <- 1 # initialise with positive value
  return(optim(par = par, fn = ridge, method = "BFGS", X = X, lambda = lambda)$par)

}

degrees_of_freedom <- function(X, num_samples, num_simulations, optimal_weigths_sigma) {

  n <- ncol(X)
  r <- nrow(X)
  optimal_weigths <- optimal_weigths_sigma[1:(n-1)] # assumes no intercept as asked
  optimal_sigma <- optimal_weigths_sigma[n]
  sum_cov = 0

  while (num_simulations > 0) {

    set.seed(num_simulations)
    sample_data <- X[sample(1:r, num_samples), ]
    sum_cov <- sum_cov + cov(sample_data[, 1], sample_data[, -1] %*% optimal_weigths)
    num_simulations = num_simulations - 1

  }

  return(as.numeric(sum_cov/optimal_sigma^2))

}

MSE <- function(X, optimal_weights) { # based on definition here: https://en.wikipedia.org/wiki/Mean_sq

  n <- nrow(X)
  return(sum((X[, 1] - X[, -1] %*% optimal_weights)^2)/n) # built for no-intercept models as asked
```

```
}

# model 1: lambda = 1
model1_optimal_weights_sigma <- ridge_opt(train_data, lambda = 1)
model1_optimal_weights <- model1_optimal_weights_sigma[1:(n_col - 1)]
model1_optimal_sigma <- model1_optimal_weights_sigma[n_col]
model1_train_MSE <- MSE(train_data, model1_optimal_weights)
model1_test_MSE <- MSE(test_data, model1_optimal_weights)
model1_AIC <- -2*model_log_likelihood(model1_optimal_weights, model1_optimal_sigma, as.matrix(train_data
  2*degrees_of_freedom(as.matrix(train_data), num_samples = 1000,
                       num_simulations = 50, model1_optimal_weights_sigma)

# model 2: lambda = 100
model2_optimal_weights_sigma <- ridge_opt(train_data, lambda = 100)
model2_optimal_weights <- model2_optimal_weights_sigma[1:(n_col - 1)]
model2_optimal_sigma <- model2_optimal_weights_sigma[n_col]
model2_train_MSE <- MSE(train_data, model2_optimal_weights)
model2_test_MSE <- MSE(test_data, model2_optimal_weights)
model2_AIC <- -2*model_log_likelihood(model2_optimal_weights, model2_optimal_sigma, as.matrix(train_data
  2*degrees_of_freedom(as.matrix(train_data), num_samples = 1000,
                       num_simulations = 50, model2_optimal_weights_sigma)

# model 3: lambda = 1000
model3_optimal_weights_sigma <- ridge_opt(train_data, lambda = 1000)
model3_optimal_weights <- model3_optimal_weights_sigma[1:(n_col - 1)]
model3_optimal_sigma <- model3_optimal_weights_sigma[n_col]
model3_train_MSE <- MSE(train_data, model3_optimal_weights)
model3_test_MSE <- MSE(test_data, model3_optimal_weights)
model3_AIC <- -2*model_log_likelihood(model3_optimal_weights, model3_optimal_sigma, as.matrix(train_data
  2*degrees_of_freedom(as.matrix(train_data), num_samples = 1000,
                       num_simulations = 50, model3_optimal_weights_sigma)

# plot performance
MSE_data <- data.frame(log_10_lambda = c(0, 0, 2, 2, 3, 3), Legend = rep(c("Training", "Test"), 3),
                       value = c(model1_train_MSE, model1_test_MSE, model2_train_MSE,
                                 model2_test_MSE, model3_train_MSE, model3_test_MSE))

ggplot(MSE_data) + geom_line(aes(log_10_lambda, value, colour = Legend)) + theme_bw() +
  geom_vline(xintercept = 2, linetype = "dotted") +
  xlab("Log (base-10) of Hyper-Parameter 'lambda'") + ylab("Mean Squared Error") +
  ggtitle("Finding the Optimal Hyper-Parameter")

AIC_data <- data.frame(log_10_lambda = c(0, 2, 3), value = c(model1_AIC, model2_AIC, model3_AIC))

ggplot(AIC_data) + geom_line(aes(log_10_lambda, value)) + theme_bw() +
  geom_vline(xintercept = 2, linetype = "dotted") +
  xlab("Log (base-10) of Hyper-Parameter 'lambda'") + ylab("Model AIC") +
  ggtitle("Finding the Optimal Hyper-Parameter")
```

Question 3

```
# read in data
tecator <- read_csv("tecator.csv", col_types = cols())
```

```r
# drop columns not to be used for prediction
tecator <- tecator[, -c(1, 103, 104)]

# split into training/test
all_indices <- 1:(nrow(tecator))
set.seed(12345)
train_indices <- sample(all_indices, ceiling(0.5*nrow(tecator)))
test_indices <- all_indices[!(all_indices %in% train_indices)]
train_data <- tecator[train_indices, ]
test_data <- tecator[test_indices, ]

## linear regression

# model

# $$\mathbf{y} \sim N(\boldsymbol{\beta}\mathbb{X}, \, \sigma^2\mathbb{I})$$

# fit
linreg <- lm(Fat~., train_data)
y_train_pred <- predict(linreg)
y_test_pred <- predict(linreg, test_data)
train_mse <- sum((train_data$Fat - y_train_pred)^2)/nrow(train_data)
test_mse <- sum((test_data$Fat - y_test_pred)^2)/nrow(test_data)

cat("Training MSE is:", train_mse)
cat("\nTest MSE is: ", test_mse)

## lasso regression

# objective function

## $$\sum\limits_{i=1}^N{(y_i - \boldsymbol{\beta}\mathbb{X}_i)^2} + \lambda\sum\limits_{j=1}^p{|\beta_

# helper function: degrees of freedom
lasso_degrees_of_freedom <- function(lasso_model, y, X, num_simulations, num_samples) {

  X <- as.matrix(X)
  y <- unlist(y)
  y_pred <- as.numeric(predict(lasso, X))
  r <- nrow(X)
  sum_cov = 0

  while (num_simulations > 0) {

    set.seed(num_simulations)
    sample_indices <- sample(1:r, num_samples)
    sample_X <- X[sample_indices, ]
    sample_y <- y[sample_indices]
    sample_y_pred <- y_pred[sample_indices]
    sum_cov <- sum_cov + cov(sample_y, sample_y_pred)
    num_simulations = num_simulations - 1

  }
```

```r
  sigma_hat_square <- sum((y - y_pred)^2)/num_samples

  return(sum_cov/sigma_hat_square)

}

# fit
lambda_choices <- c(10^-3, 10^-2, 10^-1, 1, 10, 100, 1000)
zero_coeff <- c()
sum_abs_val_params <- c()
lasso_deg_freedoms <- c()
for (lam in lambda_choices){

  lasso <- glmnet(x = as.matrix(subset(train_data, select = -Fat)),
                  y = unlist(train_data[, "Fat"]), alpha = 1, lambda = lam)
  sum_abs_val_params <- c(sum_abs_val_params, sum(abs(lasso$beta)))
  zero_coeff <- c(zero_coeff, sum(lasso$beta == 0))
  lasso_deg_freedoms <- c(lasso_deg_freedoms,
                          lasso_degrees_of_freedom(
                            lasso, tecator[, "Fat"], subset(tecator, select = -Fat),
                            num_simulations = 50, num_samples = 100)) # using full dataset rather than

}

# plot results
lasso_results <- data.frame(log_10_lambda = rep(-3:3, 2),
                            Legend = c(rep("# Parameters with Coefficients = 0", 7),
                                       rep("Sum of absolute values of Coefficients", 7)),
                            value = c(zero_coeff, sum_abs_val_params))

ggplot(lasso_results) + geom_line(aes(log_10_lambda, value, colour = Legend)) + theme_bw() +
  scale_x_continuous(breaks = -3:3) + theme(legend.position="bottom") +
  xlab("Log (base-10) of LASSO Hyper-Parameter 'lambda'") + ylab("Parameter Attributes") +
  ggtitle("Relationship between LASSO Lambda and parameter attributes")

lasso_deg_freedom <- data.frame(log_10_lambda = rep(-3:3, 2), value = lasso_deg_freedoms)

ggplot(lasso_deg_freedom) + geom_line(aes(log_10_lambda, value)) + theme_bw() +
  scale_x_continuous(breaks = -3:3) + xlab("Log (base-10) of LASSO Hyper-Parameter 'lambda'") +
  ylab("Model Degrees of Freedom") + ggtitle("Relationship between LASSO Lambda and Model Degrees of Fr

## ridge regression

# fit
zero_coeff <- c()
sum_params_squared <- c()
for (lam in lambda_choices){

  ridge <- glmnet(x = as.matrix(subset(train_data, select = -Fat)),
                  y = unlist(train_data[, "Fat"]), alpha = 0, lambda = lam)
  sum_params_squared <- c(sum_params_squared, sum(ridge$beta^2))
  zero_coeff <- c(zero_coeff, sum(ridge$beta == 0))
```

```r
}

# plot results
ridge_results <- data.frame(log_10_lambda = rep(-3:3, 2),
                            Legend = c(rep("# Parameters with Coefficients = 0", 7),
                                       rep("Sum of squared values of Coefficients", 7)),
                            value = c(zero_coeff, sum_params_squared))

ggplot(ridge_results) + geom_line(aes(log_10_lambda, value, colour = Legend)) + theme_bw() +
  scale_x_continuous(breaks = -3:3) + theme(legend.position="bottom") +
  xlab("Log (base-10) of Ridge Hyper-Parameter 'lambda'") + ylab("Parameter Attributes") +
  ggtitle("Relationship between Ridge Lambda and parameter attributes")

## lasso with CV

# fit
cv_means <- c()
cv_min_lambda <- c()
for (i in 1:100) {

  cv_lasso <- cv.glmnet(x = as.matrix(subset(train_data, select = -Fat)),
                        y = unlist(train_data[, "Fat"]), alpha = 1,
                        lambda = lambda_choices, nfolds = 3)
  cv_means <- cbind(cv_means, cv_lasso$cvm)
  cv_min_lambda <- c(cv_min_lambda, cv_lasso$lambda.min)
}

# plot
ggplot(data.frame(log_10_lambda = -3:3, cv_score = rev(rowMeans(cv_means)))) +
  geom_line(aes(log_10_lambda, cv_score)) + scale_x_continuous(breaks = -3:3) +
  geom_vline(xintercept = -3, linetype = "dotted") +
  xlab("Log (base-10) of LASSO Hyper-Parameter 'lambda'") + ylab("Mean squared error across CV folds") +
  ggtitle("Relationship between LASSO Lambda and CV Scores")

opt_lambda <- cv_lasso$lambda.min
cat("Optimal Lambda is:", opt_lambda)

model_optimal_lambda <- glmnet(x = as.matrix(subset(train_data, select = -Fat)),
                               y = unlist(train_data[, "Fat"]), alpha = 1, lambda = opt_lambda)
num_non_zero_params_opt_lambda <- sum(model_optimal_lambda$beta != 0)
cat("Number of variables chosen in the model:", num_non_zero_params_opt_lambda)

# comparing LASSO regression model with optimal lambda and another with log (base-10) lambda = -2
model_log_lambda_minus_2 <- glmnet(x = as.matrix(subset(train_data, select = -Fat)),
                                   y = unlist(train_data[, "Fat"]), alpha = 1, lambda = 10^-2)
y_test <- unlist(test_data[, "Fat"])
y_test_pred_opt_lambda <- predict(model_optimal_lambda, as.matrix(subset(test_data, select = -Fat)))
y_test_log_lambda_minus_2 <- predict(model_log_lambda_minus_2, as.matrix(subset(test_data, select = -Fa

num_non_zero_params_log_lambda_minus_2 <- sum(model_log_lambda_minus_2$beta != 0)

opt_lambda_SSE <- sum((y_test - y_test_pred_opt_lambda)^2)
log_lambda_minus_2_SSE <- sum((y_test - y_test_log_lambda_minus_2)^2)
```

```r
opt_lambda_deg_freedom <- nrow(test_data) - num_non_zero_params_opt_lambda - 1
log_lambda_minus_2_deg_freedom <- nrow(test_data) - num_non_zero_params_log_lambda_minus_2 - 1

opt_lambda_MSE <- opt_lambda_SSE / opt_lambda_deg_freedom
log_lambda_minus_2_MSE <- log_lambda_minus_2_SSE / log_lambda_minus_2_deg_freedom
test_statistic <- opt_lambda_MSE / log_lambda_minus_2_MSE

# Null Hypothesis: opt_lambda_MSE = log_lambda_minus_2_MSE
# Alternative Hypothesis: opt_lambda_MSE < log_lambda_minus_2_MSE

p_value <- pf(q = test_statistic,
              df1 = opt_lambda_deg_freedom,
              df2 = log_lambda_minus_2_deg_freedom)

cat("The p-value for the test of null hypothesis that optimal lambda works similar to log(lambda) = -2"
    "vs. the alternative that the former is better is:", p_value)
cat("We, therefore, cannot conclude confidently that the optimal",
    "lambda works significantly better than log(lambda) = -2")

# plot predictions
ggplot(data.frame(y_test, y_test_pred_opt_lambda)) +
  geom_point(aes(y_test, y_test_pred_opt_lambda)) +
  xlab("Test Labels") + ylab("Test Predictions") +
  ggtitle("Goodness of fit - LASSO model with optimised lambda")

## generative model

# estimate sigma: use MLE now ;-)
y_train <- unlist(train_data[, "Fat"])
y_train_pred <- predict(model_optimal_lambda, as.matrix(subset(train_data, select = -Fat)))
MLE_sigma_estimate <- sum((y_train - y_train_pred)^2)/nrow(train_data)

# generate labels from distribution N(y_train_pred, MLE_sigma_estimate*I)
y_generated <- rnorm(length(y_test), y_test_pred_opt_lambda, MLE_sigma_estimate)

# plot generated labels vs. original labels
ggplot(data.frame(y_test, y_generated)) +
  geom_point(aes(y_test, y_generated)) +
  xlab("Test Labels") + ylab("Generated Labels") +
  ggtitle("Goodness of fit - Generative LASSO model with optimised lambda")
```