SAN Assignment - regression

Martin Endler endlemar@fel.cvut.cz

Submission

Fill in your name above for clarity. To solve this homework, simply write your answers into this document and fill in the marked pieces of code. Submit your solution consisting of both this modified Rmd file and a knitted PDF document as an archive to the courseware BRUTE upload system for the SAN course. The deadline is specified there.

Initialization

Load the required libraries `gtools`, `caret` and `glmnet`, make sure you have those installed. We also fix the random seed for reproducibility convenience.

```
require(gtools);
```

```
## Loading required package: gtools
```

```
require(caret);
```

```
## Loading required package: caret
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
require(glmnet);
```

```
## Loading required package: glmnet
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-4
```

```
##
## Attaching package: 'glmnet'
```

```
## The following object is masked from 'package:gtools':
##
##     na.replace
```

```
set.seed(0)
```

Here, we define constants of the assignment. You may play with the values and observe what happens, but in your solution you should use the given values unchanged.

```
n.samples <- 256 # Total number of samples (training and testing together)
n.dimensions <- 100 # Number of n.dimensions, a.k.a. attributes or features
```

Model evaluation procedure

The function `learnAndTest` takes a matrix of independent variables values `X` and a corresponding vector of dependent variable (a.k.a. response) `Y` then trains and evaluates a model specified by the `modelType` parameter.

If you are interested in all possible `modelType` parameter values, refer to http://topepo.github.io/caret/train-models-by-tag.html. Here we will only be using three: `"lm"` for ordinary least squares and `"glmnet"` with parameter `alpha = 1` resp. `alpha = 0` for LASSO resp. Ridge. (There are also `lasso` and `ridge` methods you may try, but these have inconsistent API for passing lambda.) This function learns the model from the data and estimates its accuracy using cross validation. The results are then printed to output. For purpose of this assignment, consider only the RMSE (root-mean-square error) criterion.

```
learnAndTest <- function(X, Y, modelType, ...) {
    data <- data.frame(X, Y)

    train.control <- trainControl(method = "cv", number = 10) # 10-fold cross-validataion
    # Alternative to get more accurate, but slower evaluation:
    # train.control <- trainControl(method = "LOOCV")

    # Here we train the model using the versatile `train` function from the caret package
    train(Y ~ .,
          data = data,
          method = modelType,
          trControl = train.control,
          ...
    )

}
```

We will also precompute an array of candidate lambda values for LASSO and Ridge.

```
lambda_lasso <- expand.grid(lambda = 10^seq(10, -3, length = 10), alpha = 1)
lambda_ridge <- expand.grid(lambda = 10^seq(10, -3, length = 10), alpha = 0)
```

Initial data generation

Here, we generate some data.

```
# Generates independent variables by uniform i.i.d. sampling
X <- replicate(
    n.dimensions,
    runif(n.samples, min = -10, max = 10)
)

# Randomly generates the actual underlying coefficients of the linear dependency
coefs <- runif(n.dimensions, min = 1, max = 4)
intercept <- 0 # For simplicity

# Synthesizes dependent variable (observed values) by the given linear dependency plus noise
noise <- rnorm(n.samples, sd = 8, mean = 0) # Gaussian noise to be added to the response
Y <- (X %*% coefs) + intercept + noise # Note: (%*%) is the matrix multiplication operator
```

Testing the models

Now let us run the following tests:

```
print(learnAndTest(X, Y, "lm"))
```

```
## Linear Regression
##
## 256 samples
## 100 predictors
```

```
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 231, 231, 230, 231, 231, 229, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   10.22282  0.9954098  8.107877
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
print(learnAndTest(X, Y, "glmnet", tuneGrid = lambda_ridge))
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```
## glmnet
##
## 256 samples
## 100 predictors
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 229, 231, 231, 231, 228, 230, ...
## Resampling results across tuning parameters:
##
##   lambda        RMSE       Rsquared   MAE
##   1.000000e-03   13.58962  0.9933283   11.09131
##   2.782559e-02   13.58962  0.9933283   11.09131
##   7.742637e-01   13.58962  0.9933283   11.09131
##   2.154435e+01   34.01922  0.9740668   27.24431
##   5.994843e+02  120.12106  0.7891572   97.88858
##   1.668101e+04  143.78427  0.7190257  117.35115
##   4.641589e+05  144.96674        NaN  118.32455
##   1.291550e+07  144.96674        NaN  118.32455
##   3.593814e+08  144.96674        NaN  118.32455
##   1.000000e+10  144.96674        NaN  118.32455
##
## Tuning parameter 'alpha' was held constant at a value of 0
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 0 and lambda = 0.7742637.
```

```
print(learnAndTest(X, Y, "glmnet", tuneGrid = lambda_lasso))
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```
## glmnet
##
## 256 samples
## 100 predictors
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 230, 230, 230, 231, 228, 232, ...
## Resampling results across tuning parameters:
```

```
##
##   lambda        RMSE       Rsquared   MAE
##   1.000000e-03   10.43912  0.9950887    8.530893
##   2.782559e-02   10.43912  0.9950887    8.530893
##   7.742637e-01   21.50918  0.9851375   16.937733
##   2.154435e+01  135.67380  0.1978032  109.662368
##   5.994843e+02  145.51104        NaN  118.523560
##   1.668101e+04  145.51104        NaN  118.523560
##   4.641589e+05  145.51104        NaN  118.523560
##   1.291550e+07  145.51104        NaN  118.523560
##   3.593814e+08  145.51104        NaN  118.523560
##   1.000000e+10  145.51104        NaN  118.523560
##
## Tuning parameter 'alpha' was held constant at a value of 1
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 1 and lambda = 0.02782559.
```

Task 1:

Answer the following questions:

- What change in the learned model would you anticipate if we changed the `mean` parameter value to a different constant in the noise generation? You may answer this either by talking about the coefficients or by giving a geometrical interpretation. **Answer:** From a geometric point of view, the predicted y values would be shifted up (for a positive mean), resp. down (for a negative mean). This shift would correspond to a change in the estimated intercept parameter. The other estimated coefficients would remain the same.
- In our example we generated samples (i.e. the independent variables `X`) from uniform distribution. The least squares method, on the other hand, has something called the "normality assumption". Have we violated that assumption? Justify. **Answer:** No, we haven't. The "normality assumption" requires that the residuals are normally distributed, not the data (our generated samples).
- Which method gave the best results? Is it the most common one to do so if you re-run the test several times? Why do you think it performs better the best? **Answer:** The LASSO gave the best result (smallest RMSE) most often while re-running the test multiple times. However, the ordinary Least squares method was also very good, with a small RMSE very close to the one from LASSO. From the very small lambda parameter in the best LASSO, we can see that the LASSO is almost the same as the Least squares method (the shrinkage penalty term is close to zero).
- Check the selected values for the lambda parameter for ridge and LASSO. Are they low or high? How does it relate to the above answer? **Answer:** They are both very low. That means that both ridge and LASSO are almost the same as the Least squares method (the shrinkage penalty term is close to zero).

Least squares assumptions

The data generation model assumed by the ordinary least squares method can be mathematically written as follows:

$$Y = \mathbf{X}^T \boldsymbol{\beta} + \beta_0 + G,\ G \sim \mathcal{N}(0, \sigma^2)$$

This formula implicitly expresses some of the assumptions about the data, required for the method to work reliably.

- The observed value Y is influenced by some Gaussian noise G.
- There truly exists an underlying linear dependency.
- The noise is homoscedastic (σ^2 is a constant).

Task 2:

First of all, make sure you understand how elements of this formula correspond to the code in the "data generation" section.

Your task is to violate each of these assumptions (one at a time) and **briefly** comment the changes in the learned model by statistically comparing it to model using the above data. (Coefficient summary below.) The catch here is that you are allowed to only modify the noise generation procedure to achieve that. Attempt to find a way of violating the assumptions to achieve a clear difference, but any solution that is technically correct will be awarded full points.

It is sufficient to look at the `summary` of the OLS model.

```
summary(learnAndTest(X, Y, "lm"))
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16.3220  -3.7072   0.1499   4.0184  16.0036
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.43023    0.63316  -0.679    0.498
## X1           3.32876    0.12014  27.708   <2e-16 ***
## X2           1.61195    0.10468  15.399   <2e-16 ***
## X3           2.42437    0.11129  21.784   <2e-16 ***
## X4           2.90811    0.09546  30.466   <2e-16 ***
## X5           1.15971    0.10603  10.937   <2e-16 ***
## X6           3.82874    0.10620  36.051   <2e-16 ***
## X7           2.37577    0.11010  21.578   <2e-16 ***
## X8           1.94196    0.10157  19.119   <2e-16 ***
## X9           1.26780    0.10836  11.700   <2e-16 ***
## X10          2.43546    0.11277  21.597   <2e-16 ***
## X11          2.79039    0.11020  25.322   <2e-16 ***
## X12          1.84899    0.09814  18.840   <2e-16 ***
## X13          3.31907    0.10670  31.105   <2e-16 ***
## X14          3.73201    0.10760  34.684   <2e-16 ***
## X15          2.85848    0.10641  26.863   <2e-16 ***
## X16          2.18332    0.10949  19.941   <2e-16 ***
## X17          2.98273    0.10435  28.583   <2e-16 ***
## X18          3.95243    0.11494  34.388   <2e-16 ***
## X19          2.72129    0.11400  23.870   <2e-16 ***
## X20          2.26276    0.10559  21.430   <2e-16 ***
## X21          1.45329    0.11139  13.047   <2e-16 ***
## X22          2.06841    0.10820  19.117   <2e-16 ***
## X23          2.65462    0.10546  25.172   <2e-16 ***
## X24          2.43848    0.11204  21.764   <2e-16 ***
## X25          3.39725    0.10541  32.230   <2e-16 ***
## X26          3.63669    0.10717  33.933   <2e-16 ***
## X27          4.08163    0.12419  32.867   <2e-16 ***
## X28          1.39751    0.10846  12.885   <2e-16 ***
## X29          1.21262    0.10735  11.296   <2e-16 ***
## X30          3.36342    0.10379  32.405   <2e-16 ***
```

```
## X31          1.98639    0.11157  17.804   <2e-16 ***
## X32          2.06385    0.10752  19.194   <2e-16 ***
## X33          3.41435    0.10517  32.465   <2e-16 ***
## X34          1.04075    0.10473   9.938   <2e-16 ***
## X35          2.57185    0.12316  20.883   <2e-16 ***
## X36          1.44023    0.10446  13.787   <2e-16 ***
## X37          3.28405    0.11017  29.809   <2e-16 ***
## X38          2.72006    0.10675  25.480   <2e-16 ***
## X39          2.26690    0.10584  21.418   <2e-16 ***
## X40          1.88773    0.10029  18.823   <2e-16 ***
## X41          2.95467    0.10517  28.094   <2e-16 ***
## X42          2.47410    0.12403  19.948   <2e-16 ***
## X43          3.85890    0.10951  35.236   <2e-16 ***
## X44          1.37370    0.10343  13.281   <2e-16 ***
## X45          1.31642    0.11187  11.767   <2e-16 ***
## X46          1.89909    0.11339  16.749   <2e-16 ***
## X47          1.35180    0.11030  12.255   <2e-16 ***
## X48          2.36427    0.10937  21.617   <2e-16 ***
## X49          3.06998    0.11053  27.776   <2e-16 ***
## X50          1.69993    0.10494  16.199   <2e-16 ***
## X51          2.66625    0.10163  26.236   <2e-16 ***
## X52          2.27966    0.10663  21.380   <2e-16 ***
## X53          2.46486    0.10828  22.763   <2e-16 ***
## X54          2.02121    0.10601  19.065   <2e-16 ***
## X55          4.06324    0.10403  39.058   <2e-16 ***
## X56          3.48045    0.11650  29.875   <2e-16 ***
## X57          3.82223    0.11656  32.792   <2e-16 ***
## X58          1.83125    0.10265  17.841   <2e-16 ***
## X59          2.07426    0.11293  18.368   <2e-16 ***
## X60          1.01282    0.10589   9.564   <2e-16 ***
## X61          1.88231    0.10576  17.798   <2e-16 ***
## X62          2.19831    0.11279  19.490   <2e-16 ***
## X63          3.36844    0.11272  29.884   <2e-16 ***
## X64          1.62348    0.10764  15.082   <2e-16 ***
## X65          2.63364    0.10530  25.010   <2e-16 ***
## X66          2.26839    0.10769  21.064   <2e-16 ***
## X67          3.15058    0.10654  29.572   <2e-16 ***
## X68          2.11758    0.10611  19.957   <2e-16 ***
## X69          3.48378    0.11229  31.026   <2e-16 ***
## X70          2.20811    0.10557  20.915   <2e-16 ***
## X71          2.45733    0.11585  21.211   <2e-16 ***
## X72          3.93237    0.10572  37.195   <2e-16 ***
## X73          3.03948    0.11007  27.615   <2e-16 ***
## X74          2.77817    0.10538  26.363   <2e-16 ***
## X75          2.04650    0.11603  17.638   <2e-16 ***
## X76          2.18996    0.11704  18.711   <2e-16 ***
## X77          1.84367    0.10785  17.095   <2e-16 ***
## X78          3.64003    0.10593  34.361   <2e-16 ***
## X79          2.09904    0.11699  17.943   <2e-16 ***
## X80          2.31115    0.10571  21.863   <2e-16 ***
## X81          1.60042    0.10789  14.833   <2e-16 ***
## X82          1.34889    0.11418  11.814   <2e-16 ***
## X83          2.51265    0.11094  22.649   <2e-16 ***
## X84          2.70763    0.10672  25.371   <2e-16 ***
```

```
## X85          1.17989    0.10689  11.038   <2e-16 ***
## X86          2.10964    0.10729  19.663   <2e-16 ***
## X87          2.16377    0.10097  21.431   <2e-16 ***
## X88          3.07060    0.11259  27.272   <2e-16 ***
## X89          2.72039    0.10465  25.995   <2e-16 ***
## X90          1.51493    0.12133  12.486   <2e-16 ***
## X91          2.63591    0.10543  25.002   <2e-16 ***
## X92          1.78357    0.10841  16.453   <2e-16 ***
## X93          1.66301    0.10155  16.377   <2e-16 ***
## X94          1.56552    0.10155  15.416   <2e-16 ***
## X95          1.47395    0.10822  13.620   <2e-16 ***
## X96          2.82689    0.11436  24.719   <2e-16 ***
## X97          2.07620    0.11338  18.311   <2e-16 ***
## X98          2.81395    0.10439  26.957   <2e-16 ***
## X99          1.97136    0.10571  18.649   <2e-16 ***
## X100         1.65748    0.10818  15.321   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.818 on 155 degrees of freedom
## Multiple R-squared:  0.9983, Adjusted R-squared:  0.9971
## F-statistic: 888.8 on 100 and 155 DF,  p-value: < 2.2e-16
```

```r
noise <- runif(n.samples, min = -8, max = 8)

# KEEP THE CODE BELOW
Y <- (X %*% coefs) + intercept + noise
summary(learnAndTest(X, Y, "lm"))
```

Violate noise normality

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.6597  -2.4261   0.1438   2.3403  10.0544
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.22102    0.36360   0.608    0.544
## X1           3.35392    0.06899  48.614   <2e-16 ***
## X2           1.75900    0.06012  29.260   <2e-16 ***
## X3           2.29553    0.06391  35.917   <2e-16 ***
## X4           2.79220    0.05482  50.936   <2e-16 ***
## X5           1.18499    0.06089  19.460   <2e-16 ***
## X6           3.86302    0.06099  63.339   <2e-16 ***
## X7           2.18476    0.06323  34.553   <2e-16 ***
## X8           1.95167    0.05833  33.459   <2e-16 ***
## X9           1.21902    0.06223  19.590   <2e-16 ***
## X10          2.60791    0.06476  40.270   <2e-16 ***
## X11          2.96699    0.06328  46.885   <2e-16 ***
## X12          1.91911    0.05636  34.050   <2e-16 ***
```

```
## X13          3.27055   0.06128  53.373   <2e-16 ***
## X14          3.59485   0.06179  58.177   <2e-16 ***
## X15          2.91142   0.06111  47.644   <2e-16 ***
## X16          2.32401   0.06288  36.961   <2e-16 ***
## X17          2.89123   0.05993  48.247   <2e-16 ***
## X18          3.80344   0.06600  57.624   <2e-16 ***
## X19          2.73710   0.06547  41.808   <2e-16 ***
## X20          2.29227   0.06064  37.804   <2e-16 ***
## X21          1.51705   0.06397  23.717   <2e-16 ***
## X22          1.99664   0.06213  32.134   <2e-16 ***
## X23          2.67939   0.06056  44.241   <2e-16 ***
## X24          2.47020   0.06434  38.391   <2e-16 ***
## X25          3.45721   0.06053  57.114   <2e-16 ***
## X26          3.71335   0.06155  60.335   <2e-16 ***
## X27          3.93587   0.07132  55.188   <2e-16 ***
## X28          1.32561   0.06229  21.283   <2e-16 ***
## X29          1.05586   0.06165  17.127   <2e-16 ***
## X30          3.18332   0.05961  53.407   <2e-16 ***
## X31          2.10471   0.06407  32.849   <2e-16 ***
## X32          2.15854   0.06175  34.957   <2e-16 ***
## X33          3.49628   0.06040  57.890   <2e-16 ***
## X34          1.13456   0.06014  18.865   <2e-16 ***
## X35          2.32519   0.07072  32.877   <2e-16 ***
## X36          1.43736   0.05999  23.960   <2e-16 ***
## X37          3.44222   0.06327  54.408   <2e-16 ***
## X38          2.57043   0.06131  41.928   <2e-16 ***
## X39          2.47086   0.06078  40.651   <2e-16 ***
## X40          1.83546   0.05759  31.870   <2e-16 ***
## X41          2.97487   0.06040  49.256   <2e-16 ***
## X42          2.61514   0.07122  36.717   <2e-16 ***
## X43          3.89452   0.06289  61.924   <2e-16 ***
## X44          1.65122   0.05940  27.799   <2e-16 ***
## X45          1.15451   0.06424  17.971   <2e-16 ***
## X46          1.81052   0.06512  27.805   <2e-16 ***
## X47          1.38634   0.06334  21.886   <2e-16 ***
## X48          2.31795   0.06281  36.906   <2e-16 ***
## X49          3.00236   0.06347  47.302   <2e-16 ***
## X50          1.73813   0.06027  28.841   <2e-16 ***
## X51          2.57682   0.05836  44.153   <2e-16 ***
## X52          2.32508   0.06123  37.972   <2e-16 ***
## X53          2.73581   0.06218  43.996   <2e-16 ***
## X54          1.94659   0.06088  31.974   <2e-16 ***
## X55          4.04786   0.05974  67.756   <2e-16 ***
## X56          3.28758   0.06690  49.140   <2e-16 ***
## X57          3.97055   0.06694  59.318   <2e-16 ***
## X58          1.76909   0.05895  30.012   <2e-16 ***
## X59          2.15947   0.06485  33.299   <2e-16 ***
## X60          1.21264   0.06081  19.941   <2e-16 ***
## X61          1.73856   0.06074  28.625   <2e-16 ***
## X62          2.18242   0.06477  33.693   <2e-16 ***
## X63          3.23326   0.06473  49.949   <2e-16 ***
## X64          1.74698   0.06181  28.262   <2e-16 ***
## X65          2.48855   0.06047  41.151   <2e-16 ***
## X66          2.34540   0.06184  37.925   <2e-16 ***
```

```
## X67          3.16775    0.06118  51.775   <2e-16 ***
## X68          2.07880    0.06094  34.115   <2e-16 ***
## X69          3.44976    0.06448  53.498   <2e-16 ***
## X70          2.28034    0.06063  37.611   <2e-16 ***
## X71          2.38348    0.06653  35.825   <2e-16 ***
## X72          3.96858    0.06071  65.365   <2e-16 ***
## X73          3.25039    0.06321  51.424   <2e-16 ***
## X74          2.73984    0.06052  45.274   <2e-16 ***
## X75          2.07016    0.06663  31.068   <2e-16 ***
## X76          2.17123    0.06721  32.304   <2e-16 ***
## X77          1.90020    0.06193  30.681   <2e-16 ***
## X78          3.48040    0.06083  57.211   <2e-16 ***
## X79          2.04801    0.06718  30.485   <2e-16 ***
## X80          2.37128    0.06071  39.061   <2e-16 ***
## X81          1.52090    0.06196  24.546   <2e-16 ***
## X82          1.39076    0.06557  21.211   <2e-16 ***
## X83          2.46487    0.06371  38.690   <2e-16 ***
## X84          2.47040    0.06129  40.309   <2e-16 ***
## X85          1.27911    0.06138  20.838   <2e-16 ***
## X86          1.98356    0.06161  32.193   <2e-16 ***
## X87          1.95006    0.05798  33.632   <2e-16 ***
## X88          3.11230    0.06466  48.135   <2e-16 ***
## X89          2.76429    0.06010  45.996   <2e-16 ***
## X90          1.51822    0.06968  21.789   <2e-16 ***
## X91          2.93123    0.06054  48.415   <2e-16 ***
## X92          1.71543    0.06225  27.555   <2e-16 ***
## X93          1.69781    0.05832  29.114   <2e-16 ***
## X94          1.72484    0.05832  29.576   <2e-16 ***
## X95          1.48116    0.06215  23.833   <2e-16 ***
## X96          2.89305    0.06567  44.052   <2e-16 ***
## X97          2.08861    0.06511  32.077   <2e-16 ***
## X98          2.91221    0.05995  48.581   <2e-16 ***
## X99          1.89725    0.06070  31.254   <2e-16 ***
## X100         1.47563    0.06213  23.752   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.49 on 155 degrees of freedom
## Multiple R-squared:  0.9994, Adjusted R-squared:  0.9991
## F-statistic:  2701 on 100 and 155 DF,  p-value: < 2.2e-16
```

Your comment: We violated noise normality by replacing the normal distribution with a uniform distribution. Because the model had a relatively large dimension compared to the number of samples, the model started to overfit (nearly-perfect R^2, very low RSE) when we replaced the normal-distributed noise with the uniform-distributed one.

```r
noise <- X^4 %*% coefs

# KEEP THE CODE BELOW
Y <- (X %*% coefs) + intercept + noise
summary(learnAndTest(X, Y, "lm"))
```

Violate linearity

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -151753  -34912     959   34182  141637
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 487713.04    5359.69  90.997  < 2e-16 ***
## X1             -96.29    1016.96  -0.095  0.92469
## X2            1274.10     886.13   1.438  0.15250
## X3            -244.52     942.08  -0.260  0.79555
## X4           -1428.60     808.04  -1.768  0.07903 .
## X5             -11.70     897.58  -0.013  0.98961
## X6             433.64     899.01   0.482  0.63024
## X7           -1006.95     932.03  -1.080  0.28164
## X8             871.32     859.82   1.013  0.31246
## X9           -1030.54     917.25  -1.124  0.26296
## X10          -1323.38     954.59  -1.386  0.16764
## X11            285.02     932.82   0.306  0.76036
## X12            604.05     830.78   0.727  0.46827
## X13          -1645.18     903.25  -1.821  0.07047 .
## X14           -601.51     910.84  -0.660  0.50998
## X15           -170.18     900.76  -0.189  0.85039
## X16            111.43     926.84   0.120  0.90446
## X17            -26.49     883.34  -0.030  0.97611
## X18           -636.76     972.94  -0.654  0.51378
## X19           1398.92     965.04   1.450  0.14919
## X20           -560.23     893.80  -0.627  0.53172
## X21          -1654.20     942.88  -1.754  0.08134 .
## X22             20.03     915.89   0.022  0.98258
## X23           -576.12     892.72  -0.645  0.51965
## X24            874.30     948.44   0.922  0.35805
## X25           1045.78     892.26   1.172  0.24297
## X26            337.38     907.21   0.372  0.71048
## X27            -27.84    1051.25  -0.026  0.97890
## X28            -95.41     918.13  -0.104  0.91737
## X29            841.02     908.75   0.925  0.35616
## X30           1486.08     878.61   1.691  0.09277 .
## X31           1304.68     944.46   1.381  0.16914
## X32           -645.99     910.20  -0.710  0.47894
## X33            252.33     890.26   0.283  0.77722
## X34           -578.94     886.50  -0.653  0.51468
## X35          -1267.75    1042.52  -1.216  0.22582
## X36           -445.30     884.29  -0.504  0.61528
## X37           -294.38     932.58  -0.316  0.75268
## X38          -1027.43     903.68  -1.137  0.25732
## X39           -353.10     895.95  -0.394  0.69405
## X40           -639.90     848.93  -0.754  0.45213
## X41           -240.88     890.27  -0.271  0.78708
## X42           -175.78    1049.88  -0.167  0.86725
## X43           -688.43     927.05  -0.743  0.45884
```

```
## X44           -988.75     875.56  -1.129  0.26052
## X45           -478.85     946.99  -0.506  0.61382
## X46           -462.79     959.83  -0.482  0.63037
## X47            889.38     933.72   0.953  0.34232
## X48            677.21     925.81   0.731  0.46559
## X49           -245.53     935.60  -0.262  0.79334
## X50           1443.80     888.35   1.625  0.10614
## X51           1508.28     860.26   1.753  0.08153 .
## X52           -565.08     902.59  -0.626  0.53219
## X53           -303.38     916.61  -0.331  0.74111
## X54          -1407.78     897.41  -1.569  0.11876
## X55          -1536.11     880.61  -1.744  0.08308 .
## X56          -1574.51     986.17  -1.597  0.11239
## X57          -1245.00     986.67  -1.262  0.20891
## X58           -130.82     868.90  -0.151  0.88052
## X59          -1778.56     955.92  -1.861  0.06470 .
## X60          -1343.72     896.40  -1.499  0.13590
## X61           1145.08     895.26   1.279  0.20279
## X62            492.03     954.78   0.515  0.60705
## X63           -285.58     954.17  -0.299  0.76511
## X64           1640.78     911.18   1.801  0.07369 .
## X65          -1005.53     891.40  -1.128  0.26105
## X66           -949.71     911.59  -1.042  0.29912
## X67           -233.11     901.86  -0.258  0.79638
## X68            624.73     898.21   0.696  0.48777
## X69          -1274.39     950.52  -1.341  0.18197
## X70           -584.19     893.70  -0.654  0.51428
## X71           -866.49     980.70  -0.884  0.37831
## X72           -334.03     894.95  -0.373  0.70948
## X73           -498.92     931.71  -0.535  0.59308
## X74            571.28     892.04   0.640  0.52285
## X75          -1745.80     982.19  -1.777  0.07745 .
## X76           -297.69     990.74  -0.300  0.76422
## X77            118.28     912.94   0.130  0.89708
## X78           -657.16     896.73  -0.733  0.46476
## X79           2198.82     990.29   2.220  0.02784 *
## X80           -712.94     894.84  -0.797  0.42683
## X81           -809.20     913.33  -0.886  0.37699
## X82            190.85     966.50   0.197  0.84372
## X83           -602.26     939.08  -0.641  0.52226
## X84            449.68     903.39   0.498  0.61935
## X85            956.20     904.82   1.057  0.29225
## X86          -1210.24     908.23  -1.333  0.18464
## X87           -511.84     854.68  -0.599  0.55013
## X88             44.99     953.08   0.047  0.96241
## X89            612.37     885.88   0.691  0.49044
## X90           -817.55    1027.10  -0.796  0.42726
## X91           2611.04     892.44   2.926  0.00395 **
## X92            857.08     917.67   0.934  0.35177
## X93          -1015.76     859.61  -1.182  0.23915
## X94           -139.75     859.65  -0.163  0.87107
## X95           1520.92     916.09   1.660  0.09889 .
## X96           1703.29     968.06   1.759  0.08047 .
## X97            241.73     959.79   0.252  0.80149
```

```
## X98          -1342.92     883.63  -1.520  0.13060
## X99          -1526.63     894.81  -1.706  0.08999 .
## X100           116.39     915.79   0.127  0.89903
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 66180 on 155 degrees of freedom
## Multiple R-squared:  0.4049, Adjusted R-squared:  0.02105
## F-statistic: 1.055 on 100 and 155 DF,  p-value: 0.3793
```

Your comment: We violated the linearity by adding a non-linear term (the fourth power, `X^4 %*% coefs`). The F-statistic is almost equal to 1, which means that the model has no predictive capability (i.e., it is as good as an intercept-only model).

```r
noise <- rnorm(n.samples, sd = seq(from = 4, to = 32, length.out = n.samples), mean = 0)

# KEEP THE CODE BELOW
Y <- (X %*% coefs) + intercept + noise
summary(learnAndTest(X, Y, "lm"))
```

Violate homoscedasticity

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -45.860 -10.979  -0.396  11.107  40.398
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.3484     1.6338   0.213 0.831436
## X1            3.0153     0.3100   9.727  < 2e-16 ***
## X2            2.0975     0.2701   7.765 1.04e-12 ***
## X3            2.2721     0.2872   7.912 4.49e-13 ***
## X4            2.4563     0.2463   9.972  < 2e-16 ***
## X5            1.4821     0.2736   5.417 2.27e-07 ***
## X6            4.1105     0.2740  15.000  < 2e-16 ***
## X7            1.9203     0.2841   6.759 2.65e-10 ***
## X8            2.1342     0.2621   8.143 1.19e-13 ***
## X9            1.0557     0.2796   3.776 0.000227 ***
## X10           2.6741     0.2910   9.190 2.45e-16 ***
## X11           3.0767     0.2844  10.820  < 2e-16 ***
## X12           1.8402     0.2532   7.266 1.70e-11 ***
## X13           3.3236     0.2753  12.071  < 2e-16 ***
## X14           3.6122     0.2776  13.010  < 2e-16 ***
## X15           2.7170     0.2746   9.895  < 2e-16 ***
## X16           2.5454     0.2825   9.009 7.22e-16 ***
## X17           2.5162     0.2693   9.345  < 2e-16 ***
## X18           3.7894     0.2966  12.777  < 2e-16 ***
## X19           3.0005     0.2942  10.200  < 2e-16 ***
## X20           2.4748     0.2725   9.083 4.64e-16 ***
## X21           1.6190     0.2874   5.633 8.13e-08 ***
```

```
## X22            1.8612     0.2792   6.666 4.34e-10 ***
## X23            2.8197     0.2721  10.362  < 2e-16 ***
## X24            2.3143     0.2891   8.005 2.63e-13 ***
## X25            3.6321     0.2720  13.354  < 2e-16 ***
## X26            3.9784     0.2765  14.386  < 2e-16 ***
## X27            4.1854     0.3205  13.061  < 2e-16 ***
## X28            1.5043     0.2799   5.375 2.76e-07 ***
## X29            0.9787     0.2770   3.533 0.000541 ***
## X30            3.1974     0.2678  11.938  < 2e-16 ***
## X31            2.2308     0.2879   7.748 1.14e-12 ***
## X32            2.1960     0.2775   7.915 4.42e-13 ***
## X33            3.3093     0.2714  12.194  < 2e-16 ***
## X34            0.6634     0.2702   2.455 0.015190 *
## X35            2.4388     0.3178   7.674 1.74e-12 ***
## X36            1.2266     0.2696   4.551 1.07e-05 ***
## X37            3.5856     0.2843  12.613  < 2e-16 ***
## X38            2.6125     0.2755   9.484  < 2e-16 ***
## X39            2.0556     0.2731   7.526 3.99e-12 ***
## X40            1.8130     0.2588   7.006 7.04e-11 ***
## X41            3.2151     0.2714  11.847  < 2e-16 ***
## X42            3.0515     0.3200   9.535  < 2e-16 ***
## X43            4.1043     0.2826  14.524  < 2e-16 ***
## X44            1.3685     0.2669   5.128 8.64e-07 ***
## X45            0.9946     0.2887   3.445 0.000734 ***
## X46            2.1493     0.2926   7.346 1.09e-11 ***
## X47            1.5596     0.2846   5.480 1.69e-07 ***
## X48            2.0556     0.2822   7.284 1.54e-11 ***
## X49            2.7489     0.2852   9.638  < 2e-16 ***
## X50            2.3142     0.2708   8.546 1.13e-14 ***
## X51            2.3610     0.2622   9.004 7.48e-16 ***
## X52            1.9467     0.2751   7.075 4.83e-11 ***
## X53            2.7397     0.2794   9.805  < 2e-16 ***
## X54            1.8172     0.2736   6.643 4.92e-10 ***
## X55            4.2015     0.2684  15.652  < 2e-16 ***
## X56            3.4543     0.3006  11.491  < 2e-16 ***
## X57            3.6419     0.3008  12.109  < 2e-16 ***
## X58            1.7907     0.2649   6.761 2.63e-10 ***
## X59            2.6568     0.2914   9.117 3.78e-16 ***
## X60            1.2353     0.2733   4.521 1.22e-05 ***
## X61            1.6122     0.2729   5.908 2.12e-08 ***
## X62            2.0774     0.2910   7.138 3.43e-11 ***
## X63            2.8633     0.2909   9.844  < 2e-16 ***
## X64            1.3366     0.2778   4.812 3.51e-06 ***
## X65            2.4445     0.2717   8.996 7.81e-16 ***
## X66            2.3827     0.2779   8.574 9.56e-15 ***
## X67            3.5306     0.2749  12.843  < 2e-16 ***
## X68            2.0217     0.2738   7.384 8.84e-12 ***
## X69            3.5292     0.2897  12.180  < 2e-16 ***
## X70            2.4073     0.2724   8.836 2.03e-15 ***
## X71            2.9783     0.2989   9.962  < 2e-16 ***
## X72            4.3121     0.2728  15.806  < 2e-16 ***
## X73            3.3359     0.2840  11.745  < 2e-16 ***
## X74            2.5283     0.2719   9.298  < 2e-16 ***
## X75            2.0474     0.2994   6.838 1.74e-10 ***
```

```
## X76            2.7044     0.3020   8.955 1.00e-15 ***
## X77            2.4624     0.2783   8.848 1.89e-15 ***
## X78            3.2735     0.2734  11.975  < 2e-16 ***
## X79            2.3339     0.3019   7.732 1.26e-12 ***
## X80            2.5439     0.2728   9.326  < 2e-16 ***
## X81            1.2723     0.2784   4.570 9.91e-06 ***
## X82            1.4678     0.2946   4.982 1.66e-06 ***
## X83            2.2225     0.2863   7.764 1.04e-12 ***
## X84            2.4679     0.2754   8.962 9.61e-16 ***
## X85            1.5728     0.2758   5.702 5.81e-08 ***
## X86            1.9681     0.2769   7.109 4.02e-11 ***
## X87            1.8789     0.2605   7.212 2.29e-11 ***
## X88            3.5052     0.2905  12.065  < 2e-16 ***
## X89            2.8293     0.2700  10.477  < 2e-16 ***
## X90            1.7530     0.3131   5.599 9.57e-08 ***
## X91            2.8495     0.2720  10.475  < 2e-16 ***
## X92            1.5911     0.2797   5.688 6.23e-08 ***
## X93            1.5468     0.2620   5.903 2.17e-08 ***
## X94            1.7570     0.2620   6.705 3.55e-10 ***
## X95            0.9938     0.2792   3.559 0.000495 ***
## X96            2.7859     0.2951   9.441  < 2e-16 ***
## X97            2.6419     0.2926   9.030 6.39e-16 ***
## X98            2.3278     0.2694   8.642 6.42e-15 ***
## X99            2.2463     0.2728   8.235 6.96e-14 ***
## X100           1.4841     0.2792   5.316 3.63e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 20.17 on 155 degrees of freedom
## Multiple R-squared:  0.9887, Adjusted R-squared:  0.9813
## F-statistic:   135 on 100 and 155 DF,  p-value: < 2.2e-16
```

Your comment: We violated the homoscedasticity by making the variance non-constant. Compared to the model trained on the original data, this model has much lower F-statistic and bigger RSE.

Understanding the advantages of shrinkage methods

In this part, we will be modifying the dependent variables `X` from task 1 by a linear transformation, represented by a square matrix of `n.dimensions` sides. For demonstration, consider that the identity function represented by the identity matrix:

```r
M0 <- diag(n.dimensions) # Makes an identity matrix
X0 <- X %*% M0 # You can check that X0 == X
noise <- rnorm(n.samples, sd = 8, mean = 0)
Y0 <- (X0 %*% coefs) + noise # We can reuse the noise as it's independent of X.

# to see the original case
# print(learnAndTest(X0, Y0, "lm"))
# print(learnAndTest(X0, Y0, "ridge"))
# print(learnAndTest(X0, Y0, "lasso"))
```

Task 3

In this task, you will show your understanding of the advantages of Ridge by synthesizing data on which they perform the best. You are supposed do this by linearly transforming the dataset `X` as in the example above.

In other words, you should construct a matrix which if used in place of `M0` in the above example would make Ridge perform better than the other two methods. You should not resort to degenerate cases where you would get a warning about using a rank-deficient matrix. Justify your method.

Scoring note: The difference in the RMSE criterion doesn't need to be large or does not need to be present if you are certain it's just an statistical artifact (which you could verify by re-running the tests multiple times or using the LOOCV in `learnAndTest` function). Your design and justification is what matters for the assignment evaluation and the measurements are here only to guide you. We expect this task to be challenging to students, but once you get the right idea, it is possible to implement it with very small amount of code.

```
# 1. Start with the identity transformation.
M1 <- diag(n.dimensions)
M1.fmin <- 0.2
M1.fmax <- 4
# 2. Introduce multicollinearity
# x_i = x_i + ki_1*x_i-1 + ki_2*x_i-2 + ki_3*x_i-3
M1[row(M1) + 1 == col(M1)] <- seq(M1.fmin, M1.fmax, length = 99)
M1[row(M1) + 2 == col(M1)] <- seq(M1.fmax, M1.fmin, length = 98)
M1[row(M1) + 3 == col(M1)] <- seq(M1.fmin, M1.fmax, length = 97)
# do not change variable 1
M1[, 1] <- 0
M1[1, 1] <- 1

# KEEP THE CODE BELOW
noise <- rnorm(n.samples, sd = 8, mean = 0)
X1 <- X %*% M1
Y1 <- (X1 %*% coefs) + noise

# additional test code to see what M1 actually did
# X1.cov <- cov(X1)
# X1.cor <- round(cor(X1), 2)
# X1.cor_ <- X1.cor
# X1.cor_[X1.cor_ > 0.4] <- 1
# X1.cor_[X1.cor_ < 1] <- 0
# X1.cor__ <- colSums(X1.cor_)
# X1.pca <- prcomp(X1, scale. = TRUE)
# summary(X1.pca)
```

Ridge Running these test should now make Ridge perform the best.

```
print(learnAndTest(X1, Y1, "lm"))
```

```
## Linear Regression
##
## 256 samples
## 100 predictors
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 231, 230, 230, 230, 232, 231, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
```

```
##   9.520415  0.9999154  7.543669
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
print(learnAndTest(X1, Y1, "ridge"))
```

```
## Ridge Regression
##
## 256 samples
## 100 predictors
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 230, 232, 228, 231, 229, 232, ...
## Resampling results across tuning parameters:
##
##   lambda  RMSE          Rsquared   MAE
##   0e+00   1.537816e+09  0.6204258  1.136388e+09
##   1e-04   1.015722e+01  0.9999060  7.968957e+00
##   1e-01   6.507600e+01  0.9982386  5.249055e+01
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was lambda = 1e-04.
```

```
print(learnAndTest(X1, Y1, "lasso"))
```

```
## The lasso
##
## 256 samples
## 100 predictors
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 232, 231, 231, 231, 232, 230, ...
## Resampling results across tuning parameters:
##
##   fraction  RMSE      Rsquared   MAE
##   0.1        8517813  0.6063046   6467118
##   0.5       42590048  0.6063004  32336324
##   0.9       76662282  0.6063001  58205530
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was fraction = 0.1.
```

Justification: Ridge works better than OLS when there is multicollinearity in the features (independent variables). We construct a transformation matrix such that it adds linear combination of variables x_{i-3}, x_{i-2} and x_{i-1} to every variable x_i (except x_1, although it shouldn't matter). **We ran the test multiple times.** The way we transformed the data made LASSO perform visibly worse than Ridge and OLS (in terms of RMSE and R^2). Ridge and OLS gave similarly good results. However, **in most cases, Ridge performed best** (in terms of RMSE and R^2).

LASSO (OPTIONAL CHALLENGE) This part of the homework is purely optional, but we are eager to see students capable of solving this. Here you should do the same thing as above, but to make perform LASSO the best of the three methods. Although similar in nature, we consider this even more challenging than the above since being unable to modify the underlying coefficients, this may require some deeper considerations to justify the transformation method. It can still be implemented with a few short lines of code, though.

```
# 1. Start with the identity transformation.
M2 <- diag(n.dimensions)
# 2. Let's randomly select 80 % of the features (which we'll make unimportant by scaling the values).
M2.unimportant_fraction <- 0.8
M2.selection <- sample(n.dimensions, replace = FALSE, size = round(M2.unimportant_fraction * n.dimensio
M2.unimportant_idxs <- M2.selection
# 3. Update the transformation matrix so that it will scale down the selected features when applied.
# Note: We know the magnitude of the coefficients in our case, so we can choose correct scaling factor.
# We can also add the term `* (1/coefs[M2.unimportant_idxs])` (assumming abs(coefs) > 1)
# to further normalize the scaling factor.
M2[, M2.unimportant_idxs] <- M2[, M2.unimportant_idxs] * 0.001 * (1 / coefs[M2.unimportant_idxs])
# M2[,M2.unimportant_idxs] <- M2[,M2.unimportant_idxs] * 0.001 # also works great

# KEEP THE CODE BELOW
noise <- rnorm(n.samples, sd = 8, mean = 0)
X2 <- X %*% M2
Y2 <- (X2 %*% coefs) + noise
```

Running these test should now make LASSO perform the best.

```
print(learnAndTest(X2, Y2, "lm"))
```

```
## Linear Regression
##
## 256 samples
## 100 predictors
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 230, 228, 230, 231, 231, 231, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   10.51489  0.9738668  8.722386
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
print(learnAndTest(X2, Y2, "ridge"))
```

```
## Ridge Regression
##
## 256 samples
## 100 predictors
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 231, 231, 231, 230, 231, 232, ...
## Resampling results across tuning parameters:
##
##   lambda  RMSE      Rsquared   MAE
##   0e+00   10.48616  0.9740718   8.461986
##   1e-04   10.48573  0.9740723   8.460979
##   1e-01   12.73543  0.9629104  10.092765
##
```

```
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was lambda = 1e-04.
```

```
print(learnAndTest(X2, Y2, "lasso"))
```

```
## The lasso
##
## 256 samples
## 100 predictors
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 230, 230, 229, 231, 231, 231, ...
## Resampling results across tuning parameters:
##
##   fraction  RMSE       Rsquared   MAE
##   0.1       55.990259  0.5183220  45.212815
##   0.5       26.994841  0.9195134  21.729017
##   0.9        9.185182  0.9802712   7.298981
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was fraction = 0.9.
```

Justification: Generally, LASSO performs better when the response is a function of only a relatively small number of predictors. We can adjust the data by a linear transformation to make it the case. We create a scaling matrix that scales down a selected subset of features (making them unimportant) which also affects the linear regression coefficients. This in turn pushes LASSO to perform variable selection (forcing some of the coefficient estimates to be exactly equal to zero when lambda is sufficiently large). We ran the tests multiple times for our transformation and in all instances, the LASSO was the best (lowest RMSE, with a difference of around 1).