# 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 plain 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). This could mean that the Least squares method is not overfitting on our data.
- 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 ($\sigma^2$ 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
noise2 <- rbeta(n.samples, 1 / 7, 1 / 7)
# noise3 <- ((noise2 - mean(noise2))/sd(noise2) * 8)
noise3 <- ((seq(n.samples) - mean(seq(n.samples))) / sd(seq(n.samples))) * 8

noise2 <- runif(n.samples, min = -8, max = 8)
noise <- rnorm(n.samples, sd = 8, mean = 0)
mean(noise3)
```

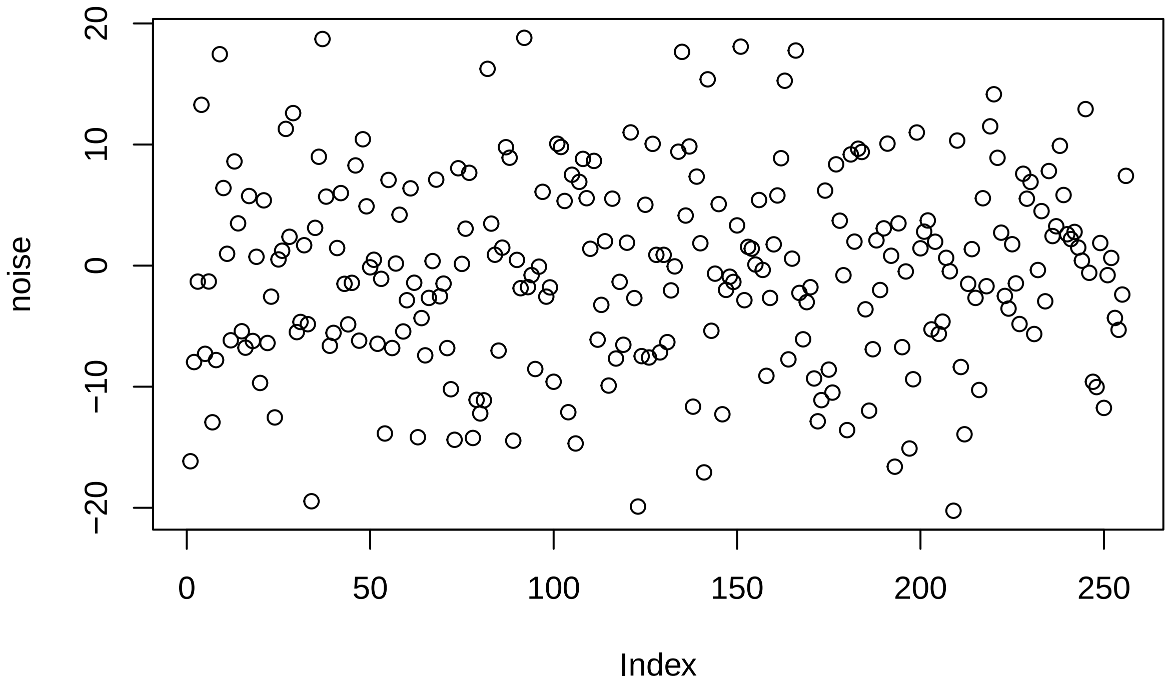
### Violate noise normality

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

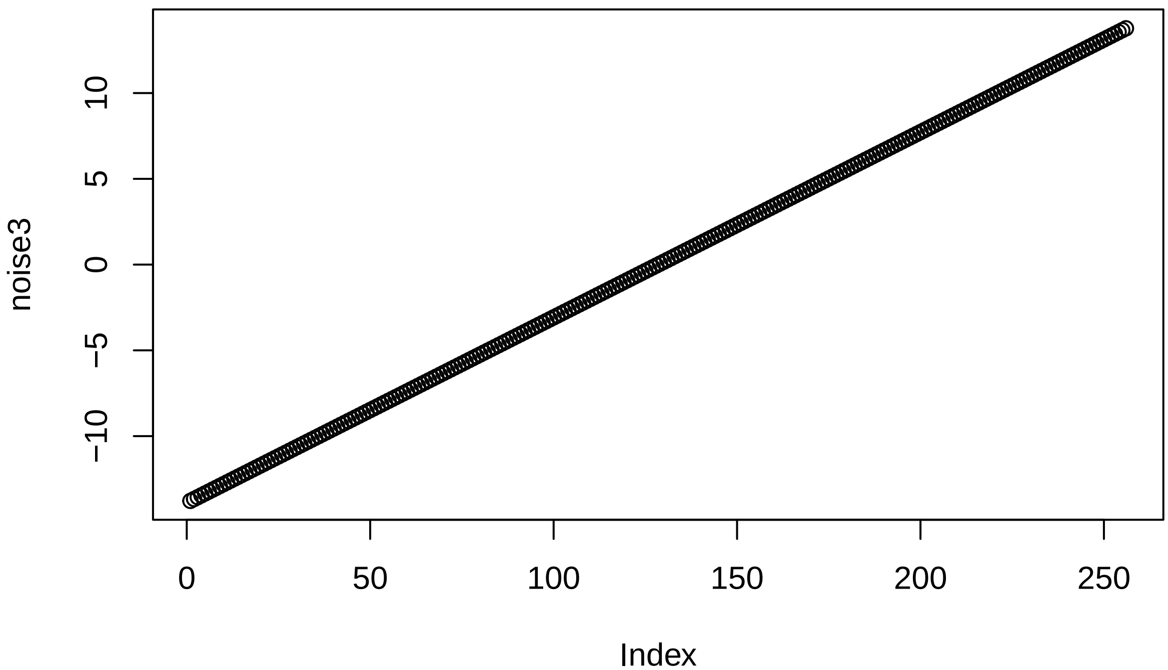
```r
var(noise3)
```

```
## [1] 64
```

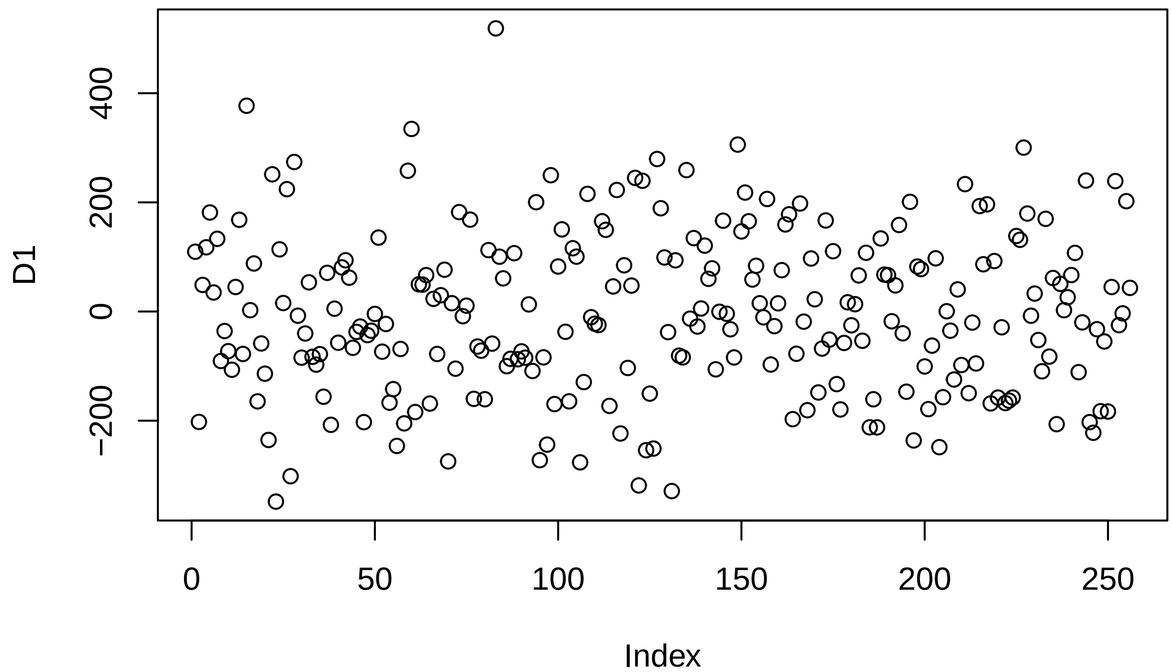
```r
plot(noise)
```



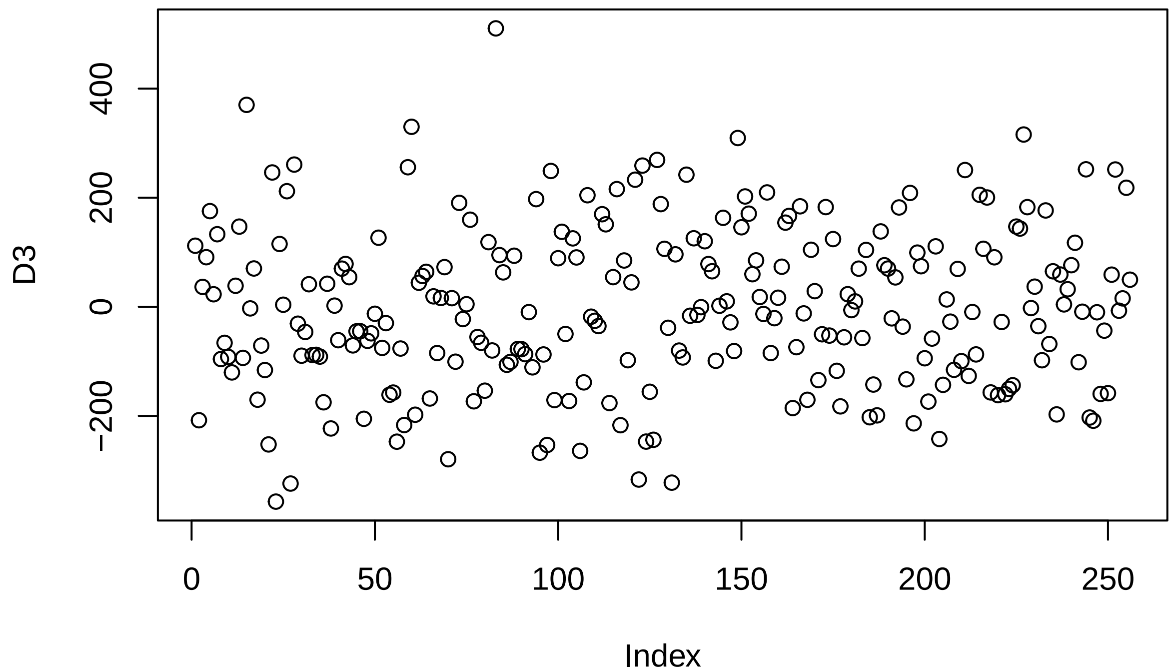

```
plot(noise3)
```




```
D3 <- (X %*% coefs) + intercept + noise3
D1 <- (X %*% coefs) + intercept + noise
plot(D1)
```




```
plot(D3)
```




```
# KEEP THE CODE BELOW
Y <- (X %*% coefs) + intercept + noise3
# summary(learnAndTest(X, Y, "lm"))
test_model <- lm(Y ~ X)
summary(test_model)
```

```
##
## Call:
## lm(formula = Y ~ X)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.8447  -4.6926   0.0893   4.4805  17.9361
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.18119    0.66154  -0.274    0.785
## X1           3.30541    0.12552  26.333   <2e-16 ***
## X2           1.84935    0.10937  16.909   <2e-16 ***
## X3           2.26219    0.11628  19.455   <2e-16 ***
## X4           2.77587    0.09974  27.832   <2e-16 ***
## X5           1.08395    0.11079   9.784   <2e-16 ***
## X6           3.83938    0.11096  34.600   <2e-16 ***
## X7           2.06626    0.11504  17.961   <2e-16 ***
## X8           1.89383    0.10613  17.845   <2e-16 ***
## X9           1.05184    0.11321   9.291   <2e-16 ***
```

```
## X10          2.55281    0.11782  21.666   <2e-16 ***
## X11          2.80752    0.11514  24.384   <2e-16 ***
## X12          1.74527    0.10254  17.020   <2e-16 ***
## X13          3.31791    0.11149  29.760   <2e-16 ***
## X14          3.77614    0.11242  33.589   <2e-16 ***
## X15          2.81804    0.11118  25.347   <2e-16 ***
## X16          2.44700    0.11440  21.390   <2e-16 ***
## X17          2.89316    0.10903  26.536   <2e-16 ***
## X18          3.74582    0.12009  31.192   <2e-16 ***
## X19          2.79024    0.11911  23.425   <2e-16 ***
## X20          2.16422    0.11032  19.618   <2e-16 ***
## X21          1.47295    0.11638  12.656   <2e-16 ***
## X22          2.03095    0.11305  17.966   <2e-16 ***
## X23          2.53931    0.11019  23.045   <2e-16 ***
## X24          2.58547    0.11706  22.086   <2e-16 ***
## X25          3.66204    0.11013  33.252   <2e-16 ***
## X26          3.76359    0.11198  33.611   <2e-16 ***
## X27          4.12068    0.12975  31.757   <2e-16 ***
## X28          1.43790    0.11332  12.688   <2e-16 ***
## X29          1.21979    0.11217  10.875   <2e-16 ***
## X30          3.41967    0.10845  31.534   <2e-16 ***
## X31          2.00793    0.11657  17.225   <2e-16 ***
## X32          2.16965    0.11235  19.312   <2e-16 ***
## X33          3.54981    0.10988  32.305   <2e-16 ***
## X34          1.19071    0.10942  10.882   <2e-16 ***
## X35          2.54635    0.12868  19.789   <2e-16 ***
## X36          1.29689    0.10915  11.882   <2e-16 ***
## X37          3.43179    0.11511  29.814   <2e-16 ***
## X38          2.63968    0.11154  23.666   <2e-16 ***
## X39          2.28773    0.11059  20.687   <2e-16 ***
## X40          1.95932    0.10478  18.699   <2e-16 ***
## X41          3.24824    0.10988  29.560   <2e-16 ***
## X42          2.65336    0.12959  20.476   <2e-16 ***
## X43          3.85008    0.11442  33.647   <2e-16 ***
## X44          1.56029    0.10807  14.438   <2e-16 ***
## X45          1.18534    0.11689  10.141   <2e-16 ***
## X46          1.81464    0.11847  15.317   <2e-16 ***
## X47          1.33129    0.11525  11.552   <2e-16 ***
## X48          2.27166    0.11427  19.879   <2e-16 ***
## X49          3.01522    0.11548  26.110   <2e-16 ***
## X50          1.66570    0.10965  15.191   <2e-16 ***
## X51          2.65017    0.10618  24.959   <2e-16 ***
## X52          2.50679    0.11141  22.502   <2e-16 ***
## X53          2.68830    0.11314  23.762   <2e-16 ***
## X54          2.09325    0.11077  18.898   <2e-16 ***
## X55          4.16219    0.10869  38.293   <2e-16 ***
## X56          3.37611    0.12172  27.736   <2e-16 ***
## X57          3.90781    0.12178  32.088   <2e-16 ***
## X58          1.81200    0.10725  16.896   <2e-16 ***
## X59          2.19546    0.11799  18.607   <2e-16 ***
## X60          1.02624    0.11064   9.275   <2e-16 ***
## X61          1.69194    0.11050  15.311   <2e-16 ***
## X62          2.24581    0.11785  19.057   <2e-16 ***
## X63          3.22959    0.11777  27.422   <2e-16 ***
```

```
## X64          1.67417    0.11247  14.886   <2e-16 ***
## X65          2.43306    0.11002  22.114   <2e-16 ***
## X66          2.31424    0.11252  20.568   <2e-16 ***
## X67          3.36971    0.11132  30.272   <2e-16 ***
## X68          1.86476    0.11087  16.820   <2e-16 ***
## X69          3.41243    0.11732  29.086   <2e-16 ***
## X70          2.25226    0.11031  20.418   <2e-16 ***
## X71          2.37516    0.12105  19.622   <2e-16 ***
## X72          3.90285    0.11046  35.332   <2e-16 ***
## X73          3.32479    0.11500  28.911   <2e-16 ***
## X74          2.77866    0.11010  25.237   <2e-16 ***
## X75          2.12503    0.12123  17.529   <2e-16 ***
## X76          2.27454    0.12229  18.600   <2e-16 ***
## X77          1.82027    0.11268  16.154   <2e-16 ***
## X78          3.55086    0.11068  32.081   <2e-16 ***
## X79          2.25025    0.12223  18.410   <2e-16 ***
## X80          2.38035    0.11045  21.552   <2e-16 ***
## X81          1.57821    0.11273  14.000   <2e-16 ***
## X82          1.31645    0.11929  11.035   <2e-16 ***
## X83          2.57145    0.11591  22.185   <2e-16 ***
## X84          2.44245    0.11150  21.905   <2e-16 ***
## X85          1.14263    0.11168  10.231   <2e-16 ***
## X86          1.74402    0.11210  15.557   <2e-16 ***
## X87          1.98537    0.10549  18.820   <2e-16 ***
## X88          3.16381    0.11764  26.895   <2e-16 ***
## X89          2.70078    0.10934  24.700   <2e-16 ***
## X90          1.51776    0.12677  11.972   <2e-16 ***
## X91          2.69703    0.11015  24.484   <2e-16 ***
## X92          1.52576    0.11327  13.471   <2e-16 ***
## X93          1.82118    0.10610  17.165   <2e-16 ***
## X94          1.50756    0.10611  14.208   <2e-16 ***
## X95          1.47727    0.11307  13.065   <2e-16 ***
## X96          2.96730    0.11949  24.834   <2e-16 ***
## X97          2.34247    0.11847  19.773   <2e-16 ***
## X98          2.76990    0.10907  25.397   <2e-16 ***
## X99          2.00603    0.11044  18.163   <2e-16 ***
## X100         1.33923    0.11304  11.848   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.169 on 155 degrees of freedom
## Multiple R-squared:  0.9981, Adjusted R-squared:  0.9969
## F-statistic: 816.9 on 100 and 155 DF,  p-value: < 2.2e-16
```

```r
# plot(test_model)
```

**Your comment:** We violated noise normality by replacing the normal distribution with uniform distribution (with the 0 mean and

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

# KEEP THE CODE BELOW
Y <- (X %*% coefs) + intercept + noise
```

```r
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
## -51.091 -10.739   1.186  10.119  50.303
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.8570     1.7884   1.038 0.300700
## X1            2.8583     0.3393   8.423 2.32e-14 ***
## X2            1.2512     0.2957   4.232 3.96e-05 ***
## X3            2.2611     0.3143   7.193 2.54e-11 ***
## X4            3.0562     0.2696  11.335  < 2e-16 ***
## X5            1.4764     0.2995   4.929 2.10e-06 ***
## X6            3.5592     0.3000  11.865  < 2e-16 ***
## X7            2.4826     0.3110   7.983 2.99e-13 ***
## X8            2.2331     0.2869   7.784 9.33e-13 ***
## X9            1.3874     0.3061   4.533 1.16e-05 ***
## X10           2.5504     0.3185   8.007 2.60e-13 ***
## X11           3.0665     0.3113   9.852  < 2e-16 ***
## X12           1.7705     0.2772   6.387 1.88e-09 ***
## X13           3.1827     0.3014  10.560  < 2e-16 ***
## X14           3.9118     0.3039  12.871  < 2e-16 ***
## X15           3.2312     0.3006  10.751  < 2e-16 ***
## X16           2.6005     0.3093   8.409 2.53e-14 ***
## X17           2.6135     0.2947   8.867 1.69e-15 ***
```

```
## X18          4.3989     0.3246  13.550  < 2e-16 ***
## X19          2.6628     0.3220   8.269 5.71e-14 ***
## X20          2.4564     0.2982   8.236 6.91e-14 ***
## X21          1.3050     0.3146   4.148 5.51e-05 ***
## X22          1.9173     0.3056   6.274 3.36e-09 ***
## X23          2.5237     0.2979   8.472 1.74e-14 ***
## X24          2.1802     0.3165   6.889 1.32e-10 ***
## X25          3.3679     0.2977  11.312  < 2e-16 ***
## X26          3.7960     0.3027  12.540  < 2e-16 ***
## X27          3.9518     0.3508  11.266  < 2e-16 ***
## X28          1.0275     0.3064   3.354 0.001001 **
## X29          0.9684     0.3032   3.194 0.001702 **
## X30          3.0893     0.2932  10.538  < 2e-16 ***
## X31          1.8679     0.3151   5.927 1.93e-08 ***
## X32          2.6053     0.3037   8.578 9.35e-15 ***
## X33          3.2550     0.2971  10.958  < 2e-16 ***
## X34          0.8469     0.2958   2.863 0.004778 **
## X35          2.8325     0.3479   8.143 1.19e-13 ***
## X36          1.3446     0.2951   4.557 1.05e-05 ***
## X37          3.3069     0.3112  10.627  < 2e-16 ***
## X38          2.6245     0.3015   8.704 4.45e-15 ***
## X39          2.1381     0.2990   7.152 3.18e-11 ***
## X40          1.7628     0.2833   6.223 4.35e-09 ***
## X41          2.9588     0.2971   9.960  < 2e-16 ***
## X42          2.5143     0.3503   7.177 2.77e-11 ***
## X43          3.7129     0.3093  12.003  < 2e-16 ***
## X44          1.8393     0.2921   6.296 3.00e-09 ***
## X45          1.0386     0.3160   3.287 0.001253 **
## X46          1.9914     0.3203   6.218 4.47e-09 ***
## X47          1.3792     0.3116   4.427 1.79e-05 ***
## X48          2.0370     0.3089   6.594 6.36e-10 ***
## X49          3.0155     0.3122   9.659  < 2e-16 ***
## X50          1.6025     0.2964   5.406 2.39e-07 ***
## X51          2.3854     0.2870   8.310 4.49e-14 ***
## X52          2.2253     0.3012   7.389 8.59e-12 ***
## X53          2.2475     0.3058   7.348 1.08e-11 ***
## X54          2.0225     0.2994   6.754 2.72e-10 ***
## X55          4.3913     0.2938  14.945  < 2e-16 ***
## X56          3.5769     0.3291  10.870  < 2e-16 ***
## X57          3.9237     0.3292  11.918  < 2e-16 ***
## X58          1.6454     0.2899   5.675 6.62e-08 ***
## X59          2.3402     0.3190   7.337 1.15e-11 ***
## X60          1.4962     0.2991   5.002 1.52e-06 ***
## X61          2.0669     0.2987   6.919 1.13e-10 ***
## X62          1.8844     0.3186   5.915 2.05e-08 ***
## X63          2.9723     0.3184   9.336  < 2e-16 ***
## X64          2.0512     0.3040   6.747 2.84e-10 ***
## X65          2.9233     0.2974   9.828  < 2e-16 ***
## X66          1.9337     0.3042   6.357 2.19e-09 ***
## X67          2.9563     0.3009   9.824  < 2e-16 ***
## X68          2.3102     0.2997   7.708 1.43e-12 ***
## X69          3.4407     0.3172  10.849  < 2e-16 ***
## X70          2.1576     0.2982   7.235 2.01e-11 ***
## X71          2.9113     0.3272   8.897 1.42e-15 ***
```

```
## X72           4.2092     0.2986  14.096  < 2e-16 ***
## X73           2.8380     0.3109   9.129 3.53e-16 ***
## X74           2.3955     0.2976   8.048 2.05e-13 ***
## X75           2.3246     0.3277   7.093 4.39e-11 ***
## X76           2.1763     0.3306   6.583 6.74e-10 ***
## X77           1.6540     0.3046   5.430 2.14e-07 ***
## X78           3.6930     0.2992  12.342  < 2e-16 ***
## X79           1.7106     0.3304   5.177 6.91e-07 ***
## X80           2.0892     0.2986   6.997 7.39e-11 ***
## X81           1.9015     0.3047   6.240 4.00e-09 ***
## X82           1.7878     0.3225   5.544 1.25e-07 ***
## X83           2.7165     0.3133   8.669 5.46e-15 ***
## X84           2.5323     0.3014   8.401 2.65e-14 ***
## X85           1.2087     0.3019   4.003 9.66e-05 ***
## X86           2.0173     0.3030   6.657 4.58e-10 ***
## X87           2.4892     0.2852   8.728 3.85e-15 ***
## X88           3.5887     0.3180  11.285  < 2e-16 ***
## X89           2.9519     0.2956   9.986  < 2e-16 ***
## X90           1.1933     0.3427   3.482 0.000647 ***
## X91           2.5278     0.2978   8.489 1.58e-14 ***
## X92           1.6659     0.3062   5.441 2.03e-07 ***
## X93           1.7709     0.2868   6.174 5.59e-09 ***
## X94           1.6002     0.2868   5.579 1.06e-07 ***
## X95           1.4424     0.3057   4.719 5.26e-06 ***
## X96           3.0518     0.3230   9.448  < 2e-16 ***
## X97           2.2871     0.3203   7.141 3.37e-11 ***
## X98           2.9399     0.2948   9.971  < 2e-16 ***
## X99           2.2347     0.2986   7.485 5.04e-12 ***
## X100          1.4744     0.3056   4.825 3.32e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 22.08 on 155 degrees of freedom
## Multiple R-squared:  0.9863, Adjusted R-squared:  0.9775
## F-statistic: 111.8 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:

```
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.
```

### 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.

```
M1 <- diag(n.dimensions)
# M1 <- diag(x = rnorm(n.dimensions, sd = 0.2), n.dimensions)

# M1.fraction <- 0.1
M1.selection <- sample(n.dimensions, replace = FALSE, size = 10)
M1.idxs <- M1.selection


#1 <- matrix(
#   # runif(n.dimensions^2, min = 0, max = 2),
#   # prob = c(0.7, 0.3)
#   # sample(c(0, 1, 2, 3, 4, 5), replace = TRUE, size = n.dimensions^2),
#   1,
#   nrow = n.dimensions,
#   ncol = n.dimensions,
#

# M1[row(M1) == col(M1)] <- 0

#
# M1.tmp <- sample(c(0, 1), replace = TRUE, size = n.dimensions * length(M1.idxs), prob = c(0.7, 0.3))
#
# M1[, M1.idxs] <- 0
M1[row(M1) == col(M1)][M1.idxs] <- 100

# M1 <- matrix(
#   # runif(n.dimensions^2, min = 0, max = 2),
#   sample(c(0, 1), replace = TRUE, size = n.dimensions^2, prob = c(0.7, 0.3)),
#   nrow = n.dimensions,
#   ncol = n.dimensions,
# )
# M1 <- diag(n.dimensions)
# M1.ur <- sample(100, replace = FALSE, size = 50)
# M1.uc <- sample(100, replace = FALSE, size = 50)
# M1[M1.ur, M1.uc] <- sample(coefs, replace = TRUE, size = 50^2)
# M1[col(M1) == row(M1)] <- 1
# M1 <- matrix(
#   1,
#   nrow = n.dimensions,
#   ncol = n.dimensions,
# )
# M1.half <- (n.dimensions / 2)
# M1[, (M1.half + 1):n.dimensions] <- 0
```

```r
# M1[1:M1.half, 1:M1.half] <- 0
# M1[col(M1) == row(M1)] <- 0.05
# M1[
#   col(M1) > M1.half &
#       row(M1) > M1.half &
#       col(M1) == row(M1)
# ] <- 1
# M1[1, 1] <- 1

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

M1.c <- round(cor(X1), 2) > 0.9
test.pca <- prcomp(X1)

summary(test.pca)
```

**Ridge**

```
## Importance of components:
##                            PC1      PC2      PC3      PC4      PC5      PC6
## Standard deviation     669.975 635.8636 618.1497 612.6362 598.6935 579.4355
## Proportion of Variance   0.132   0.1189   0.1123   0.1103   0.1054   0.0987
## Cumulative Proportion    0.132   0.2508   0.3631   0.4735   0.5788   0.6775
##                              PC7       PC8       PC9      PC10    PC11    PC12
## Standard deviation     555.94339 529.23575 515.19477 489.35120 8.90688 8.69171
## Proportion of Variance   0.09086   0.08234   0.07803   0.07039 0.00002 0.00002
## Cumulative Proportion    0.76839   0.85073   0.92875   0.99915 0.99917 0.99919
##                           PC13    PC14    PC15    PC16    PC17    PC18    PC19
## Standard deviation     8.49953 8.41474 8.34254 8.29512 8.13161 7.92884 7.83090
## Proportion of Variance 0.00002 0.00002 0.00002 0.00002 0.00002 0.00002 0.00002
## Cumulative Proportion  0.99922 0.99924 0.99926 0.99928 0.99930 0.99932 0.99933
##                           PC20    PC21    PC22    PC23    PC24    PC25    PC26
## Standard deviation     7.81935 7.66864 7.65636 7.58187 7.45721 7.40331 7.32942
## Proportion of Variance 0.00002 0.00002 0.00002 0.00002 0.00002 0.00002 0.00002
## Cumulative Proportion  0.99935 0.99937 0.99939 0.99940 0.99942 0.99944 0.99945
##                           PC27    PC28    PC29    PC30    PC31    PC32    PC33
## Standard deviation     7.17902 7.17049 7.07414 7.00771 6.94535 6.89184 6.85866
## Proportion of Variance 0.00002 0.00002 0.00001 0.00001 0.00001 0.00001 0.00001
## Cumulative Proportion  0.99947 0.99948 0.99950 0.99951 0.99952 0.99954 0.99955
##                           PC34    PC35    PC36    PC37    PC38    PC39    PC40
## Standard deviation     6.74447 6.67259 6.54965 6.52510 6.45375 6.41294 6.33012
## Proportion of Variance 0.00001 0.00001 0.00001 0.00001 0.00001 0.00001 0.00001
## Cumulative Proportion  0.99957 0.99958 0.99959 0.99960 0.99962 0.99963 0.99964
##                           PC41    PC42    PC43    PC44    PC45    PC46    PC47
## Standard deviation     6.31135 6.16821 6.09280 6.07045 5.99003 5.89599 5.81212
## Proportion of Variance 0.00001 0.00001 0.00001 0.00001 0.00001 0.00001 0.00001
## Cumulative Proportion  0.99965 0.99966 0.99967 0.99968 0.99970 0.99971 0.99972
##                           PC48    PC49    PC50    PC51    PC52    PC53    PC54
## Standard deviation     5.72234 5.70523 5.59733 5.56984 5.43994 5.39251 5.35795
## Proportion of Variance 0.00001 0.00001 0.00001 0.00001 0.00001 0.00001 0.00001
## Cumulative Proportion  0.99973 0.99973 0.99974 0.99975 0.99976 0.99977 0.99978
```

```
##                           PC55    PC56    PC57    PC58    PC59    PC60    PC61
## Standard deviation     5.26415 5.23008 5.18011 5.11853 5.06531 4.97547 4.91258
## Proportion of Variance 0.00001 0.00001 0.00001 0.00001 0.00001 0.00001 0.00001
## Cumulative Proportion  0.99979 0.99980 0.99980 0.99981 0.99982 0.99983 0.99983
##                           PC62    PC63    PC64    PC65    PC66    PC67    PC68
## Standard deviation     4.89528 4.81930 4.78540 4.75813 4.73132 4.62681 4.55507
## Proportion of Variance 0.00001 0.00001 0.00001 0.00001 0.00001 0.00001 0.00001
## Cumulative Proportion  0.99984 0.99985 0.99985 0.99986 0.99987 0.99987 0.99988
##                           PC69    PC70    PC71    PC72    PC73    PC74    PC75
## Standard deviation     4.50562 4.43473 4.38530 4.34265 4.26539 4.25630 4.19277
## Proportion of Variance 0.00001 0.00001 0.00001 0.00001 0.00001 0.00001 0.00001
## Cumulative Proportion  0.99988 0.99989 0.99990 0.99990 0.99991 0.99991 0.99992
##                           PC76   PC77   PC78   PC79   PC80  PC81  PC82  PC83
## Standard deviation     4.13556 4.1026 3.9977 3.9640 3.9326 3.853 3.789 3.733
## Proportion of Variance 0.00001 0.0000 0.0000 0.0000 0.0000 0.000 0.000 0.000
## Cumulative Proportion  0.99992 0.9999 0.9999 0.9999 0.9999 1.000 1.000 1.000
##                         PC84  PC85  PC86 PC87  PC88 PC89  PC90  PC91  PC92
## Standard deviation     3.634 3.572 3.526 3.38 3.337 3.32 3.217 3.116 3.015
## Proportion of Variance 0.000 0.000 0.000 0.00 0.000 0.00 0.000 0.000 0.000
## Cumulative Proportion  1.000 1.000 1.000 1.00 1.000 1.00 1.000 1.000 1.000
##                         PC93  PC94  PC95  PC96  PC97  PC98  PC99 PC100
## Standard deviation     2.856 2.821 2.792 2.668 2.626 2.566 2.384 2.216
## Proportion of Variance 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
## Cumulative Proportion  1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
```

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: 232, 231, 230, 231, 229, 228, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   10.56176  0.9999935  8.405541
##
## 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, 230, 231, 230, 231, 229, ...
## Resampling results across tuning parameters:
##
```

```
##   lambda  RMSE       Rsquared   MAE
##   0e+00    10.15353  0.9999934    8.182463
##   1e-04    10.19487  0.9999933    8.215248
##   1e-01   460.77793  0.9883638  371.674820
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was lambda = 0.
```

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

```
## The lasso
##
## 256 samples
## 100 predictors
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 231, 232, 231, 230, 231, 228, ...
## Resampling results across tuning parameters:
##
##   fraction  RMSE       Rsquared   MAE
##   0.1       3541.6149  0.4948989  2866.9125
##   0.5       1786.2961  0.9538453  1469.3268
##   0.9        146.2663  0.9990000   118.3576
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was fraction = 0.9.
```

**Justification:** Ridge works better than OLS and LASSO when there is multicollinearity in the features.

**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.

```r
# 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.

```
t31 <- learnAndTest(X2, Y2, "lm")
t32 <- learnAndTest(X2, Y2, "ridge")
t33 <- learnAndTest(X2, Y2, "lasso")
t32_c <- as.matrix(predict(t32$finalModel, s = t32$bestTune[1, "fraction"], type = "coef", mode = "frac
t33_c <- as.matrix(predict(t33$finalModel, s = t33$bestTune[1, "fraction"], type = "coef", mode = "frac
```

**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).