Exercises week 36

Python library import

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

def design_poly_n(x, n):
    X = np.zeros((len(x), n))
    X[:,0] = 1
    for i in range(1, n):
        X[:,i] = (x**i).T
    return X
```

R library import

```
library(tidyverse)
library(reticulate)
```

We start by generating data using the same code as last week.

```
np.random.seed(8392)
n = 100
# Make data set.
x = np.linspace(-3, 3, n).reshape(-1, 1)
y = np.exp(-x**2) + 1.5 * np.exp(-(x-2)**2)+ np.random.normal(0, 0.1, x.shape)

X = design_poly_n(x, 5)
np.random.seed(524)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Then we scale input data by subtracting the mean of each column.

```
X_colmeans = np.mean(X_train, 0)[1:]

# without intercept
X_train_scaled = X_train[:, 1:] - X_colmeans
X_test_scaled = X_test[:, 1:] - X_colmeans
y_scaler = np.mean(y_train)
y_train_scaled = y_train - y_scaler
```

First we fit with ordinary least squares

```
Bhat = np.linalg.inv(X_train_scaled.T @ X_train_scaled) @ X_train_scaled.T @ y_train_scale

print("Beta:", Bhat.T)
```

```
Beta: [[ 0.44343093 -0.00864978 -0.0346039  -0.00467165]]
```

```
ytilde_train = X_train_scaled @ Bhat + y_scaler
intercept = y_scaler - X_colmeans @ Bhat

MSE_OLS_train = sum((y_train - ytilde_train)**2) / len(y_train)
print("training data MSE:", MSE_OLS_train)
```

```
training data MSE: [0.03319703]
```

```
ytilde_test = X_test_scaled @ Bhat + y_scaler
MSE_OLS_test = sum((y_test - ytilde_test)**2) / len(y_test)
print("test data MSE:", MSE_OLS_test)
```

```
test data MSE: [0.03824773]
```

And then fit the same data with ridge regression

```
# define some functions for convenience
def get_ridge_beta(X, y, lmbda):
    p = X.shape[1]
    I = np.eye(p,p)

    return np.linalg.pinv(X.T @ X + (lmbda * I)) @ X.T @ y
```

```python
def predict(X, Beta, scaler = 0):
    return X @ Beta + scaler

def get_intercept(X, Beta, scaler):
    return scaler - np.mean(X, 0) @ Beta

def MSE(y, ytilde):
    return sum((y - ytilde)**2) / len(y)
```

```python
lambdas = 10**np.arange(-4, 1, dtype = float)
MSE_train = np.zeros(len(lambdas))
MSE_test = np.zeros(len(lambdas))

for i in range(len(lambdas)):
    ridge_beta = get_ridge_beta(X_train_scaled, y_train_scaled,
                                lambdas.item(i))

    y_ridge_train = predict(X_train_scaled, ridge_beta, y_scaler)
    y_ridge_test = predict(X_test_scaled, ridge_beta, y_scaler)

    MSE_tr = MSE(y_train, y_ridge_train)
    MSE_te = MSE(y_test, y_ridge_test)
    MSE_train[i] = MSE_tr
    MSE_test[i] = MSE_te

MSE_train
```

```
array([0.03319703, 0.03319703, 0.03319704, 0.03319765, 0.03325636])
```

```python
MSE_test
```

```
array([0.03824774, 0.03824787, 0.03824919, 0.03826286, 0.03844413])
```

```python
plt.plot(lambdas, MSE_train, "b-", label = "training data")
plt.plot(lambdas, MSE_test, "b-.", label = "test data")
plt.xlabel("lambda")
plt.ylabel("MSE")
plt.legend()
```

```
plt.show()
```

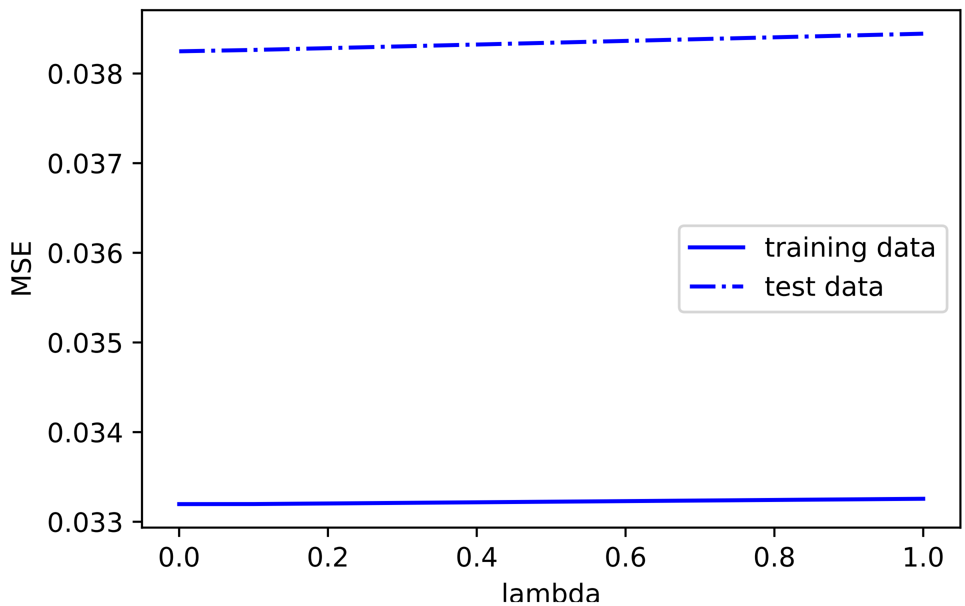

Figure 1: MSE as function of lambda for a ridge regression with predictors up to a 5th degree polynomial.

We can see that the MSE increases with increasing λ for both training and test data.

Finally, we do the same for multiple maximum polynomial degrees: 5, 10 and 15.

```
def ridge_test_lambdas(X, y, lambdas):

    ## Split and scale data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    X_colmeans = np.mean(X_train, 0)[1:]

    # without intercept
    X_train_scaled = X_train[:, 1:] - X_colmeans
    X_test_scaled = X_test[:, 1:] - X_colmeans
    y_scaler = np.mean(y_train)
    y_train_scaled = y_train - y_scaler

    ## Fit models for different lambdas
```

```
    MSE_train = np.zeros(len(lambdas))
    MSE_test = np.zeros(len(lambdas))

    for i in range(len(lambdas)):
        ridge_beta = get_ridge_beta(X_train_scaled, y_train_scaled,
                                    lambdas.item(i))

        y_ridge_train = predict(X_train_scaled, ridge_beta, y_scaler)
        y_ridge_test = predict(X_test_scaled, ridge_beta, y_scaler)

        MSE_tr = MSE(y_train, y_ridge_train)
        MSE_te = MSE(y_test, y_ridge_test)
        MSE_train[i] = MSE_tr
        MSE_test[i] = MSE_te

    return [MSE_train, MSE_test]
```

```
polys = [5, 10, 15]
lambdas = 10**np.arange(-4, 1, dtype = float)
MSE_train_p = np.zeros((len(lambdas), len(polys)))
MSE_test_p = np.zeros((len(lambdas), len(polys)))

for j in range(len(polys)):
    X = design_poly_n(x, polys[j])
    np.random.seed(524)
    MSE_train, MSE_test = ridge_test_lambdas(X, y, lambdas)
    MSE_train_p[:,j] = MSE_train
    MSE_test_p[:,j] = MSE_test
```

Hand over to R

```
train_data <- as.data.frame(py$MSE_train_p)
test_data <- as.data.frame(py$MSE_test_p)
names(train_data) <- names(test_data) <- paste0(py$polys)

train_data$dat <- "training"
test_data$dat <- "test"

rbind(train_data, test_data) %>%
    cbind(lmb = py$lambdas) %>%
    pivot_longer(-c(dat, lmb)) %>%
```

```
ggplot(aes(lmb, value, lty = dat)) +
geom_line() +
facet_wrap(~as.numeric(name),nrow = 3, scales = "free_y") +
labs(x = expression(paste(lambda)),
     y = "MSE",
     linetype = "data") +
theme_bw() +
scale_x_log10()
```

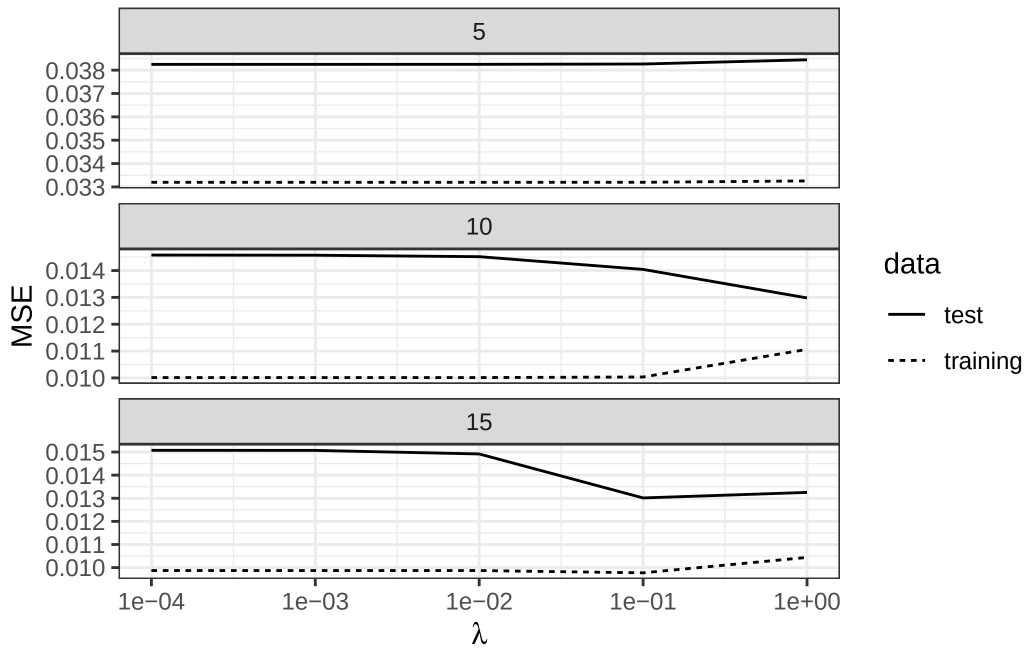

Figure 2: Mean square error for different as a function of λ for fits with predictors of polynomial degree up to 5, 10 and 15. Note individual Y-axis on each plot to emphasize patterns in the data.

In general, training data MSE increases with increasing λ. It seems that the model performs best on the test data when $\lambda \geq 0.1$ for the higher polynomial degrees, showing that the ridge regression could reduce overfitting for this data.