Exercises Week 36

Exercise 1

a)

> Show that the optimal parameters for Ridge regression are given by
> $\hat{\beta}_{\text{Ridge}} = \left(X^T X + \lambda I\right)^{-1} X^T y$

In other words, we want to minimize the cost function

$$C(X, \beta) = \frac{1}{n} \|y - X\beta\|_2^2 + \lambda \|\beta\|_2^2$$

And we would like to minimize with respect to β (while also dropping the $\frac{1}{n}$ parameter)

$$\frac{\delta}{\delta \beta} \|y - X\beta\|_2^2 + \lambda \|\beta\|_2^2 = 0$$

To do so we first need to expand the expression for the cost function

$$\|y - X\beta\|_2^2 + \lambda \|\beta\|_2^2 = (y - X\beta)^T (y - X\beta) + \lambda \beta^T \beta$$

We then need to find the derivate of the cost function with respect to β

$$\frac{\delta}{\delta \beta} C(X, \beta) = (y - X\beta)^T (y - X\beta) + \lambda \beta^T \beta$$

From weekly exercises 35 we proved the derivation for the first term, allowing us to easily find the full expression

$$\text{¿} -2X^T (y - X\beta) + 2\lambda\beta$$

This expression is then set to zero and allows us to then find the solution

$$\begin{aligned}
-2X^T (y - X\beta) + 2\lambda\beta \quad & \text{¿}\, 0 \\
-2X^T y + 2X^T X\beta + 2\lambda\beta \quad & \text{¿}\, 0 \\
2X^T X\beta + 2\lambda\beta \quad & \text{¿}\, 2X^T y \\
\left(X^T X + \lambda I\right)\beta \quad & \text{¿}\, X^T y \\
\text{¿}\, \beta_{\text{Ridge}} \quad & \text{¿} \left(X^T X + \lambda I\right)^{-1} X^T y
\end{aligned}$$

b)

Show that you can write the OLS solutions in terms of the eigenvectors (the columns) of the orthogonal matrix U The OLS results is given as $\hat{\beta}_{\text{OLS}} = \left(X^T X\right)^{-1} X^T y$. We substitute the SVD for X into the expression

¿

To find the solution we just plug in $\tilde{y}_{\text{OLS}} = X \hat{\beta}_{\text{OLS}}$

$$\tilde{y}_{\text{OLS}} = U \Sigma V^T V \Sigma^{-1} U^T y = U U^T y = \sum_{j=0}^{p-1} u_j u_j^T y$$

Where u_j are the columns (eigenvectors) of U

c)

Show a likewise expression for Ridge Regression The methods is similar as for OLS, but we instead insert $X = U \Sigma V^T$ into the expression we proved in the first task

$$\hat{\beta}_{\text{Ridge}} = \left(X^T X + \lambda I\right)^{-1} X^T y \quad ¿$$

We then plug this into equation for the solution

$$\tilde{y}_{\text{Ridge}} = X \hat{\beta}_{\text{Ridge}} \quad ¿$$

The term Σ ¿ can be broken down elementwise diagonally:

$$\text{The j-th diagonal element of } \Sigma \Sigma^T \quad : \sigma_j^2$$

¿ ¿

Which plugged back into the expression

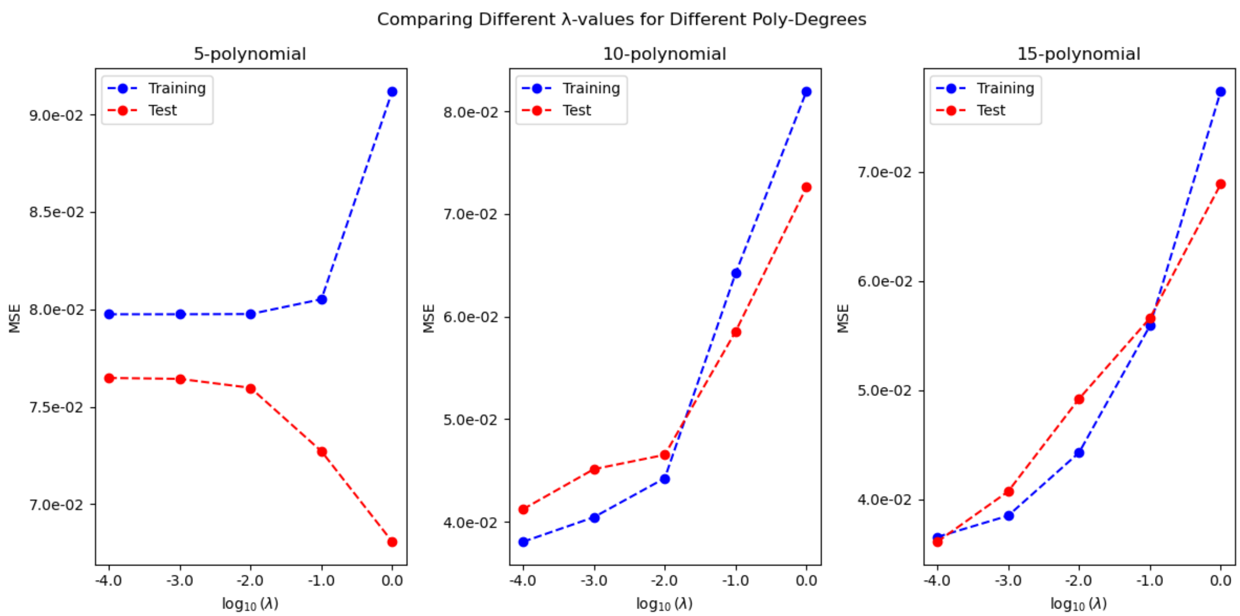
$$¿\, U \frac{\sigma^2 j}{\sigma_j^2 + \lambda} U^T y = U U^T \frac{\sigma^2 j}{\sigma_j^2 + \lambda} y = \sum_{j=0}^{p-1} u_j u_j^T \frac{\sigma^2 j}{\sigma_j^2 + \lambda} y$$

Exercise 2

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score

def design_matrix(x, degree):
    X = np.zeros((len(x), degree + 1))
    for i in range(degree + 1):
```

```
        X[:,i] = x.flatten()**i
    return X

seed = 2021
np.random.seed(seed)
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)

lambdas   = np.logspace(-4, 0, 5)
degrees   = [5, 10, 15]
MSE_train = {deg : [] for deg in degrees}
MSE_test  = {deg : [] for deg in degrees}

plt.figure(figsize=(12, 6))

for idx, deg in enumerate(degrees):
    # Creating Design matrix and splitting data
    X  = design_matrix(x, deg)
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=seed)

    # Scaling
    scaler_X = StandardScaler().fit(X_train)
    scaler_y = StandardScaler().fit(y_train)

    X_train_scaled = scaler_X.transform(X_train)
    X_test_scaled = scaler_X.transform(X_test)

    y_train_scaled = scaler_y.transform(y_train)
    y_test_scaled = scaler_y.transform(y_test)

    I = np.eye(deg+1)
    # Varying the λ-parameter
    for λ in lambdas:
        β = np.linalg.inv(X_train_scaled.T @ X_train_scaled + λ*I) @
X_train_scaled.T @ y_train_scaled

        y_tilde = X_train_scaled @ β
        y_pred = X_test_scaled @ β

        MSE_train[deg].append(mean_squared_error(y_train_scaled,
y_tilde))
        MSE_test[deg].append(mean_squared_error(y_test_scaled,
y_pred))

    p = plt.subplot(1, 3, idx+1)
    p.yaxis.set_major_formatter(plt.FormatStrFormatter("%.1e"))
```

```
    plt.plot(np.log10(lambdas), MSE_train[deg], 'b--o', 
label='Training')
    plt.plot(np.log10(lambdas), MSE_test[deg], 'r--o', label='Test')

    plt.xlabel("$\log_{10}(λ)$")
    plt.ylabel("MSE")
    plt.title(f'{deg}-polynomial')
    plt.legend()
    plt.xticks([np.log10(λ) for λ in lambdas], [str(np.log10(λ)) for λ 
in lambdas])

plt.suptitle("Comparing Different λ-values for Different Poly-
Degrees")
plt.tight_layout()
plt.show()
```



Conclusions

- I have a hard time interpreting the results.
- Only for the higher polynomial degrees, do we see a clear benefit of smaller λ-values.
- For the lower polynomial degree, the error diverges for larger λ-values, for the test and training data.