E36

September 6, 2024

1 Exercises Week 36

1.1 Exercise 1

1.2 a)

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

In other words, we want to minimize the cost function

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

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

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

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

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

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

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

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

$$= -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) + 2\lambda\beta$$

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

$$
\begin{aligned}
-2\mathbf{X}^T(\mathbf{y}-\mathbf{X}\beta)+2\lambda\beta &= 0\\
-2\mathbf{X}^T\mathbf{y}+2\mathbf{X}^T\mathbf{X}\beta+2\lambda\beta &= 0\\
2\mathbf{X}^T\mathbf{X}\beta+2\lambda\beta &= 2\mathbf{X}^T\mathbf{y}\\
(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})\beta &= \mathbf{X}^T\mathbf{y}
\end{aligned}
$$

$$
\beta_{\text{Ridge}} = (\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}
$$

1.3 b)

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

$$
\begin{aligned}
\left((\mathbf{U}\Sigma\mathbf{V}^T)^T(\mathbf{U}\Sigma\mathbf{V}^T)\right)^{-1}(\mathbf{U}\Sigma\mathbf{V}^T)^T\mathbf{y} &= \left(\mathbf{V}\Sigma^T\underbrace{\mathbf{U}^T\mathbf{U}}_{=\mathbf{I}}\Sigma\mathbf{V}^T\right)^{-1}\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{y}\\
&= \left(\mathbf{V}\Sigma^T\Sigma\mathbf{V}^T\right)^{-1}\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{y}
\end{aligned}
$$

Since $\mathbf{V}$ is orthogonal and $\Sigma^T\Sigma$ is an symmetric matrix we exploit that

$$
\left(\mathbf{V}\Sigma^T\Sigma\mathbf{V}^T\right)^{-1} = \mathbf{V}(\Sigma^T\Sigma)^{-1}\mathbf{V}^T
$$

$$
\Rightarrow \mathbf{V}(\Sigma^T\Sigma)^{-1}\mathbf{V}^T\ \mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{y} = \mathbf{V}\frac{\Sigma^T}{(\Sigma^T\Sigma)}\underbrace{\mathbf{V}^T\mathbf{V}}_{=\mathbf{I}}\mathbf{U}^T\mathbf{y} = \mathbf{V}\Sigma^{-1}\mathbf{U}^T\mathbf{y}
$$

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

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

Where $\mathbf{u}_j$ are the columns (eigenvectors) of $\mathbf{U}$

1.4 c)

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

$$
\begin{aligned}
\hat{\beta}_{\text{Ridge}} = (\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} &= \left((\mathbf{U}\Sigma\mathbf{V}^T)^T(\mathbf{U}\Sigma\mathbf{V}^T)+\lambda\mathbf{I}\right)^{-1}(\mathbf{U}\Sigma\mathbf{V}^T)^T\mathbf{y}\\
&= \left(\mathbf{V}\Sigma^T\Sigma\mathbf{V}^T+\lambda\mathbf{I}\right)^{-1}(\mathbf{U}\Sigma\mathbf{V}^T)^T\mathbf{y}
\end{aligned}
$$

From here we will assign the diagonal matrix $\Sigma^T\Sigma$ as Σ^2

$$
\hat{\beta}_{\text{Ridge}} = \left(\mathbf{V}\Sigma^2\mathbf{V}^T+\lambda\mathbf{I}\right)^{-1}(\mathbf{U}\Sigma\mathbf{V}^T)^T\mathbf{y}
$$

We then plug this into equation for the solution

$$\tilde{y}_{\text{Ridge}} = \mathbf{X}\hat{\beta}_{\text{Ridge}} = \mathbf{U}\Sigma\mathbf{V}^T \left(\mathbf{V}\Sigma^2\mathbf{V}^T + \lambda\mathbf{I}\right)^{-1} (\mathbf{U}\Sigma\mathbf{V}^T)^T\mathbf{y}$$

Where we again exploit the orthogonality of $\mathbf{V}$

$$= \mathbf{U}\Sigma\mathbf{V}^T \ \mathbf{V}(\Sigma^2 + \lambda\mathbf{I})^{-1}\mathbf{V}^T \ (\mathbf{U}\Sigma\mathbf{V}^T)^T\mathbf{y}$$

$$= \mathbf{U}\Sigma \underbrace{\mathbf{V}^T\mathbf{V}}_{=\mathbf{I}} (\Sigma^2 + \lambda\mathbf{I})^{-1} \underbrace{\mathbf{V}^T\mathbf{V}}_{=\mathbf{I}} \Sigma^T\mathbf{U}^T\mathbf{y}$$

$$= \mathbf{U}\Sigma(\Sigma^2 + \lambda\mathbf{I})^{-1}\Sigma^T\mathbf{U}^T\mathbf{y}$$

The term $\Sigma(\Sigma^2 + \lambda\mathbf{I})^{-1}\Sigma^T$ can be broken down elementwise diagonally:

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

$$\text{The } j\text{-}th \text{ diagonal element of } (\Sigma^2 + \lambda\mathbf{I})^{-1} : \frac{1}{\sigma_j^2 + \lambda}$$

Which plugged back into the expression

$$= \mathbf{U}\frac{\sigma^2 j}{\sigma_j^2 + \lambda}\mathbf{U}^T\mathbf{y} = \mathbf{U}\mathbf{U}^T\frac{\sigma^2 j}{\sigma_j^2 + \lambda}\mathbf{y} = \sum_{j=0}^{p-1}\mathbf{u}_j\mathbf{u}_j^T\frac{\sigma^2 j}{\sigma_j^2 + \lambda}\mathbf{y}$$

1.5 Exercise 2

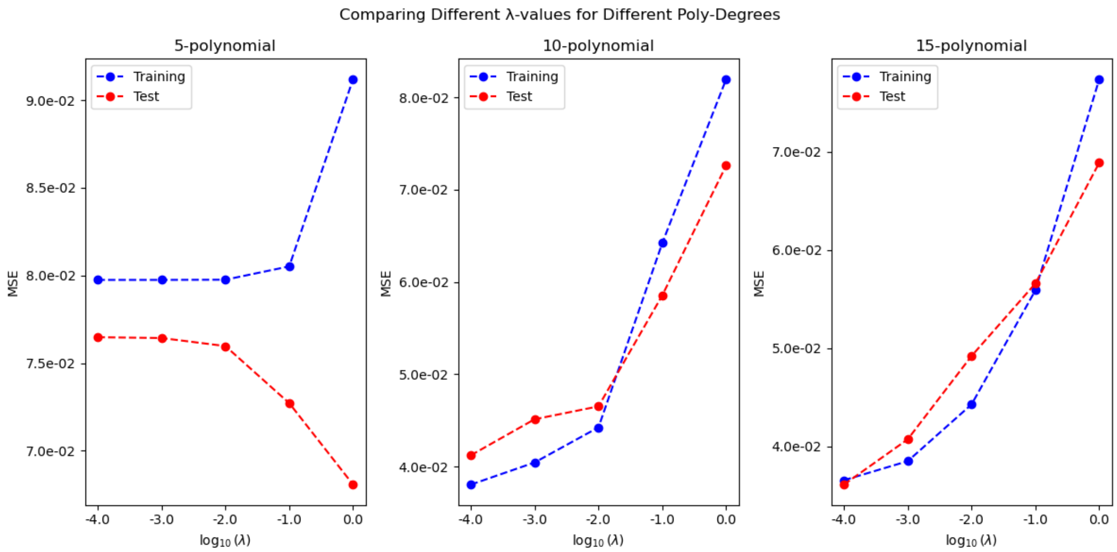
[1]:
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
```

[2]:
```
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
```

[3]:
```
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}
```

```python
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  -parameter
    for   in lambdas:
          = np.linalg.inv(X_train_scaled.T @ X_train_scaled +  *I) @ X_train_scaled.T @ y_train_scaled

        y_tilde = X_train_scaled @
        y_pred = X_test_scaled @

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

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

```
plt.show()
```



1.5.1 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.