

# Exercise

04

TUM Department of Informatics

<b>Supervised by</b>	Prof. Dr. Stephan Günnemann Informatics 3 - Professorship of Data Mining and Analytics
<b>Submitted by</b>	Marcel Bruckner (03674122) Julian Hohenadel (03673879) Kevin Bein (03707775)
<b>Submission date</b>	Munich, November 2, 2019

## Least squares regression

### Problem 1:

Using  $T = \text{diag}(t_i)$ , the derivative can be calculated similar to the lecture:

$$\begin{aligned}\nabla_w E_{\text{weighted}}(w) &= \nabla_w \frac{1}{2} \sum_{i=1}^N t_i [w^T \phi(x_i) - y_i]^2 \\&= \nabla_w \frac{1}{2} (\Phi w - y)^T T (\Phi w - y) \\&= \nabla_w \frac{1}{2} (w^T \Phi^T - y^T) (T \Phi w - T y) \\&= \nabla_w \frac{1}{2} (w^T \Phi^T T \Phi w - 2w^T \Phi^T T y + y^T T y) \\&= \nabla_w \left( \frac{1}{2} w^T \Phi^T T \Phi w - w^T \Phi^T T y + \frac{1}{2} y^T T y \right) \\&= \Phi^T T \Phi w - \Phi^T T y \\&\stackrel{!}{=} 0 \\&\Rightarrow \Phi^T T y = \Phi^T T \Phi w \\&\Rightarrow w = \Phi^T T y (\Phi^T T \Phi)^{-1}\end{aligned}$$

*Missing explanation for 1) and 2)*

## **Ridge regression**

**Problem 2:**

**Problem 3:**

## Multi-output linear regression

Problem 4:

## Comparison of linear regression models

Problem 5:

## Programming Task

### Problem 6:

# exercise\_04\_notebook

November 2, 2019

## 1 Programming assignment 2: Linear regression

```
In [1]: import numpy as np
```

```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
```

### 1.1 Your task

In this notebook code skeleton for performing linear regression is given. Your task is to complete the functions where required. You are only allowed to use built-in Python functions, as well as any numpy functions. No other libraries / imports are allowed.

### 1.2 Exporting the results to PDF

Once you complete the assignments, export the entire notebook as PDF and attach it to your homework solutions. The best way of doing that is 1. Run all the cells of the notebook. 2. Export/download the notebook as PDF (File -> Download as -> PDF via LaTeX (.pdf)). 3. Concatenate your solutions for other tasks with the output of Step 2. On a Linux machine you can simply use `pdffunite`, there are similar tools for other platforms too. You can only upload a single PDF file to Moodle.

Make sure you are using `nbconvert` Version 5.5 or later by running `jupyter nbconvert --version`. Older versions clip lines that exceed page width, which makes your code harder to grade.

### 1.3 Load and preprocess the data

In this assignment we will work with the Boston Housing Dataset. The data consists of 506 samples. Each sample represents a district in the city of Boston and has 13 features, such as crime rate or taxation level. The regression target is the median house price in the given district (in \$1000's).

More details can be found here: <http://lib.stat.cmu.edu/datasets/boston>

```
In [2]: X , y = load_boston(return_X_y=True)
```

```
# Add a vector of ones to the data matrix to absorb the bias term
# (Recall slide #7 from the lecture)
X = np.hstack([np.ones([X.shape[0], 1]), X])
# From now on, D refers to the number of features in the AUGMENTED dataset
```

```

# (i.e. including the dummy '1' feature for the absorbed bias term)

# Split into train and test
test_size = 0.2
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)

```

## 1.4 Task 1: Fit standard linear regression

```

In [3]: def fit_least_squares(X, y):
        """Fit ordinary least squares model to the data.

        Parameters
        -----
        X : array, shape [N, D]
            (Augmented) feature matrix.
        y : array, shape [N]
            Regression targets.

        Returns
        -----
        w : array, shape [D]
            Optimal regression coefficients (w[0] is the bias term).

        """

        w = np.linalg.pinv(X) @ y
        return w

```

## 1.5 Task 2: Fit ridge regression

```

In [4]: def fit_ridge(X, y, reg_strength):
        """Fit ridge regression model to the data.

        Parameters
        -----
        X : array, shape [N, D]
            (Augmented) feature matrix.
        y : array, shape [N]
            Regression targets.
        reg_strength : float
            L2 regularization strength (denoted by lambda in the lecture)

        Returns
        -----
        w : array, shape [D]
            Optimal regression coefficients (w[0] is the bias term).

        """

```



```

# https://stats.stackexchange.com/questions/69205/how-to-derive-the-ridge-regressi
N = X.shape[0]
#regulator = reg_strength * np.identity(N)
#XTX = X @ X.T
#inv = np.linalg.inv((X @ X.T) + regulator)
#w = (inv.T @ X).T @ y
w = (np.linalg.inv((X @ X.T) + (reg_strength * np.identity(N)))) .T @ X).T @ y

return w

```

## 1.6 Task 3: Generate predictions for new data

```

In [5]: def predict_linear_model(X, w):
        """Generate predictions for the given samples.

        Parameters
        -----
        X : array, shape [N, D]
            (Augmented) feature matrix.
        w : array, shape [D]
            Regression coefficients.

        Returns
        -----
        y_pred : array, shape [N]
            Predicted regression targets for the input data.

        """

        y_pred = w @ X.T
        return y_pred

```

## 1.7 Task 4: Mean squared error

```

In [6]: def mean_squared_error(y_true, y_pred):
        """Compute mean squared error between true and predicted regression targets.

        Reference: `https://en.wikipedia.org/wiki/Mean_squared_error`

        Parameters
        -----
        y_true : array
            True regression targets.
        y_pred : array
            Predicted regression targets.

        Returns
        -----


```

```

mse : float
    Mean squared error.

"""

mse = (1/len(y_true)) * (np.sum((y_true - y_pred)**2))
#from sklearn.metrics import mean_squared_error
#return mean_squared_error(y_true, y_pred)
return mse

```

## 1.8 Compare the two models

The reference implementation produces \* MSE for Least squares  $\approx 23.98$  \* MSE for Ridge regression  $\approx 21.05$

Your results might be slightly (i.e.  $\pm 1\%$ ) different from the reference solution due to numerical reasons.

```

In [7]: # Load the data
np.random.seed(1234)
X , y = load_boston(return_X_y=True)
X = np.hstack([np.ones([X.shape[0], 1]), X])
test_size = 0.2
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)

# Ordinary least squares regression
w_ls = fit_least_squares(X_train, y_train)
y_pred_ls = predict_linear_model(X_test, w_ls)
mse_ls = mean_squared_error(y_test, y_pred_ls)
print('MSE for Least squares = {0}'.format(mse_ls))

# Ridge regression
reg_strength = 1
w_ridge = fit_ridge(X_train, y_train, reg_strength)
y_pred_ridge = predict_linear_model(X_test, w_ridge)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
print('MSE for Ridge regression = {0}'.format(mse_ridge))

```

MSE for Least squares = 23.964571384956837

MSE for Ridge regression = 21.034931232092767

In [ ]:

# Appendix

We confirm that the submitted solution is original work and was written by us without further assistance.  
Appropriate credit has been given where reference has been made to the work of others.

---

Munich, November 2, 2019, Signature Marcel Bruckner (03674122)

---

Munich, November 2, 2019, Signature Julian Hohenadel (03673879)

---

Munich, November 2, 2019, Signature Kevin Bein (03707775)