# Least squares regression

## Q1:

The pdf of Jupyter Notebook homework is attached at the end of the file.

## Q2:

The weighted error function:

$$E_{weighted}(\boldsymbol{w}) = \frac{1}{2}\sum_{t_i}^{N} t_i[\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) - y_i]^2 \tag{1}$$

can be re-write in matrix form

$$\begin{aligned} E_{weighted}(\boldsymbol{w}) =& \frac{1}{2}\sum_{t_i}^{N} t_i[\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) - y_i]^2 \\ =& \frac{1}{2}(\boldsymbol{\phi}\boldsymbol{w} - \boldsymbol{y})^T\boldsymbol{T}(\boldsymbol{\phi}\boldsymbol{w} - \boldsymbol{y}) \\ =& \frac{1}{2}(\boldsymbol{w}^T\boldsymbol{\phi}^T\boldsymbol{T} - \boldsymbol{y}^T\boldsymbol{T})(\boldsymbol{\phi}\boldsymbol{w} - \boldsymbol{y}) \\ =& \frac{1}{2}(\boldsymbol{w}^T\boldsymbol{\phi}^T\boldsymbol{T}\boldsymbol{\phi}\boldsymbol{w} - \boldsymbol{y}^T\boldsymbol{T}\boldsymbol{\phi}\boldsymbol{w} - \boldsymbol{w}^T\boldsymbol{\phi}^T\boldsymbol{T}\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{T}\boldsymbol{y}) \end{aligned} \tag{2}$$

To find the optimal $\mathbf{w}^*$:

$$\boldsymbol{w}^* = arg\max_{\boldsymbol{w}} \frac{1}{2}(\boldsymbol{w}^T\boldsymbol{\phi}^T\boldsymbol{T}\boldsymbol{\phi}\boldsymbol{w} - \boldsymbol{y}^T\boldsymbol{T}\boldsymbol{\phi}\boldsymbol{w} - \boldsymbol{w}^T\boldsymbol{\phi}^T\boldsymbol{T}\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{T}\boldsymbol{y}) \tag{3}$$

Compute the gradient to find the minimum:

$$\begin{aligned} \nabla_{\boldsymbol{w}}E_{weighted}(\boldsymbol{w}) =& \nabla_{\boldsymbol{w}}\frac{1}{2}(\boldsymbol{w}^T\boldsymbol{\phi}^T\boldsymbol{T}\boldsymbol{\phi}\boldsymbol{w} - \boldsymbol{y}^T\boldsymbol{T}\boldsymbol{\phi}\boldsymbol{w} - \boldsymbol{w}^T\boldsymbol{\phi}^T\boldsymbol{T}\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{T}\boldsymbol{y}) \\ =& \boldsymbol{\phi}^T\boldsymbol{T}\boldsymbol{\phi}\boldsymbol{w} - \frac{1}{2}\boldsymbol{y}^T\boldsymbol{T}\boldsymbol{\phi} - \frac{1}{2}\boldsymbol{\phi}^T\boldsymbol{T}\boldsymbol{y} \end{aligned} \tag{4}$$

Set the gradient to zero:

$$\boldsymbol{\phi}^T\boldsymbol{T}\boldsymbol{\phi}\boldsymbol{w} - \frac{1}{2}\boldsymbol{y}^T\boldsymbol{T}\boldsymbol{\phi} - \frac{1}{2}\boldsymbol{\phi}^T\boldsymbol{T}\boldsymbol{y} \overset{!}{=} 0 \tag{5}$$

$$\boldsymbol{w}^* = \frac{1}{2}(\boldsymbol{\phi}^T\boldsymbol{T}\boldsymbol{\phi})^{-1}(\boldsymbol{y}^T\boldsymbol{T}\boldsymbol{\phi} + \boldsymbol{\phi}^T\boldsymbol{T}\boldsymbol{y})$$

$$\boldsymbol{w}^* = (\boldsymbol{\phi}^T\boldsymbol{T}\boldsymbol{\phi})^{-1}\boldsymbol{\phi}^T\boldsymbol{T}\boldsymbol{y}$$

a) According to the last equation, we notice that the optimal solution of $\boldsymbol{w}^*$ relies on the factor $\boldsymbol{T}$ ($\boldsymbol{T}$ is a diagonal matrix). if the element inside this matrix is set to 1, then the equation is identical to the equation derived from non-weighted least square error function. This means that each weighted factor has an impact on the correspond coefficient of $\boldsymbol{\phi}$ (row-wise, namely each sample set with full features is scaled with a positive factor). Thus, we can interpret it as the variance of the noise on the data.
b) The data point will be weighted by its number of occurrences.

# Ridge regression

## Q3:

The normal linear regression has the following form:

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}[\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) - y_i]^2 \tag{6}$$

Assume:

$$\boldsymbol{\Phi} \in \mathbb{R}^{(N+M)\times M}$$

$$\boldsymbol{y} \in \mathbb{R}^{(N+M)\times 1}$$

and the last $M$ rows is a matrix $\sqrt{\lambda}\boldsymbol{I}_{M\times M}$ in $\boldsymbol{\Phi}$, and the last $M$ rows of $\boldsymbol{y}$ are zeros. Insert the new matrix into the right side of the normal linear regression.

$$\begin{aligned}
E_{new} =& \frac{1}{2}\sum_{i=1}^{N+M}[\boldsymbol{w}^T\boldsymbol{\phi_{new}}(\boldsymbol{x}_i) - y_i]^2\\
=& \frac{1}{2}\sum_{i=1}^{N}[\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) - y_i]^2 + \frac{1}{2}\sum_{i=N+1}^{N+M}[\boldsymbol{w}^T\boldsymbol{\phi_{rest}}(\boldsymbol{x}_i) - y_i]^2\\
=& \frac{1}{2}\sum_{i=1}^{N}[\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) - y_i]^2 + \frac{1}{2}\sum_{i=N+1}^{N+M}[\boldsymbol{w}^T\boldsymbol{\phi_{rest}}(\boldsymbol{x}_i)]^2\\
=& \frac{1}{2}\sum_{i=1}^{N}[\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) - y_i]^2 + \frac{1}{2}\sum_{i=N+1}^{N+M}[\boldsymbol{w}^T\sqrt{\lambda}\boldsymbol{I}]^2\\
=& \frac{1}{2}\sum_{i=1}^{N}[\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) - y_i]^2 + \frac{\lambda}{2}\sum_{i=1}^{M}[\boldsymbol{w}^T]^2\\
=& \frac{1}{2}\sum_{i=1}^{N}[\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) - y_i]^2 + \frac{\lambda}{2}\left\|\boldsymbol{w}\right\|_2^2 = E_{ridge}
\end{aligned} \tag{7}$$

# Bayesian linear regression

## Q4:

The prior:

$$\begin{aligned}
p(\boldsymbol{w},\beta) =& \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_0, \beta^{-1}\boldsymbol{S}_0)Gamma(\beta|a_0,b_0)\\
=& \frac{1}{\sqrt{2\pi\beta^{-1}\boldsymbol{S_0}}}\exp\left(-\frac{(\boldsymbol{w}-\boldsymbol{m_0})^2}{2\beta^{-1}\boldsymbol{S_0}}\right)\frac{b_0^{a_0}}{\Gamma(a_0)}\beta^{a_0-1}\exp\left(-b_0\beta\right)\\
\propto& \beta^{a_0-1+\frac{1}{2}}\exp\left(-\frac{1}{2}\beta\boldsymbol{S}_0^{-1}(\boldsymbol{w}-\boldsymbol{m}_0)^2 - b_0\beta\right)\\
=& \beta^{a_0-\frac{1}{2}}\exp\left(-\frac{1}{2}\beta(\boldsymbol{S}_0^{-1}\boldsymbol{w}^T\boldsymbol{w} - 2\boldsymbol{S}_0^{-1}\boldsymbol{m}_0^T\boldsymbol{w} + \boldsymbol{S}_0^{-1}\boldsymbol{m}_0^T\boldsymbol{m}_0 + 2b_0\right)
\end{aligned} \tag{8}$$

The likelihood times prior (our posterior):

$$
\begin{aligned}
p(\boldsymbol{y}|\boldsymbol{\Phi}, \boldsymbol{w}, \beta)p(\boldsymbol{w}, \beta) &= \prod_{i=0}^{N} \mathcal{N}(y_i|\boldsymbol{w}^T\Phi(\boldsymbol{x}_i), \beta^{-1})\mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_0, \beta^{-1}\boldsymbol{S}_0)Gamma(\beta|a_0, b_0) \\
&= \prod_{i=0}^{N}(\frac{1}{\sqrt{2\pi\beta^{-1}}}\exp{(-\frac{(y_i - \boldsymbol{w}^T\Phi(\boldsymbol{x}_i))^2}{2\beta^{-1}})}) \\
&\quad \frac{1}{\sqrt{2\pi\beta^{-1}\boldsymbol{S_0}}}\exp{(-\frac{(\boldsymbol{w}-\boldsymbol{m_0})^2}{2\beta^{-1}\boldsymbol{S_0}})}\frac{b_0^{a_0}}{\Gamma(a_0)}\beta^{a_0-1}\exp{(-b_0\beta)} \\
&\propto \beta^{\frac{N}{2}}\exp{(-\frac{1}{2}\beta\sum_{i=1}^{N}(y_i - \boldsymbol{w}^T\Phi(\boldsymbol{x}_i))^2)} \\
&\quad \beta^{a_0-\frac{1}{2}}\exp{(-\frac{1}{2}\beta(\boldsymbol{S}_0^{-1}\boldsymbol{w}^T\boldsymbol{w} - 2\boldsymbol{S}_0^{-1}\boldsymbol{m}_0^T\boldsymbol{w} + \boldsymbol{S}_0^{-1}\boldsymbol{m}_0^T\boldsymbol{m}_0 + 2b_0))} \\
&= \beta^{\frac{N+2a_0-1}{2}}\exp{(-\frac{1}{2}\beta(\boldsymbol{\Phi w} - \boldsymbol{y})^T(\boldsymbol{\Phi w} - \boldsymbol{y}))} \\
&\quad \exp{(-\frac{1}{2}\beta(\boldsymbol{S}_0^{-1}\boldsymbol{w}^T\boldsymbol{w} - 2\boldsymbol{S}_0^{-1}\boldsymbol{m}_0^T\boldsymbol{w} + \boldsymbol{S}_0^{-1}\boldsymbol{m}_0^T\boldsymbol{m}_0 + 2b_0))} \\
&= \beta^{\frac{N+2a_0-1}{2}}\exp{(-\frac{1}{2}\beta(\boldsymbol{w}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi w} - 2\boldsymbol{\Phi}^T\boldsymbol{w}^T\boldsymbol{y} + \boldsymbol{y}^T\boldsymbol{y}))} \\
&\quad \exp{(-\frac{1}{2}\beta(\boldsymbol{S}_0^{-1}\boldsymbol{w}^T\boldsymbol{w} - 2\boldsymbol{S}_0^{-1}\boldsymbol{m}_0^T\boldsymbol{w} + \boldsymbol{S}_0^{-1}\boldsymbol{m}_0^T\boldsymbol{m}_0 + 2b_0))} \\
&= \beta^{\frac{N+2a_0-1}{2}}\exp{(-\frac{1}{2}\beta(\sum_{i=1}^{N}\boldsymbol{\phi}(\boldsymbol{x_i})\boldsymbol{w} - 2\boldsymbol{\Phi}^T\boldsymbol{w}^T\boldsymbol{y} + \sum_{i=1}^{N}y_i^2))} \\
&\quad \exp{(-\frac{1}{2}\beta(\boldsymbol{S}_0^{-1}\boldsymbol{w}^T\boldsymbol{w} - 2\boldsymbol{S}_0^{-1}\boldsymbol{m}_0^T\boldsymbol{w} + \boldsymbol{S}_0^{-1}\boldsymbol{m}_0^T\boldsymbol{m}_0 + 2b_0))} \\
&= \beta^{\frac{N+2a_0-1}{2}}\exp{(-\frac{1}{2}\beta(((\sum_{i=1}^{N}\boldsymbol{\phi}(\boldsymbol{x_i})) + \boldsymbol{S}_0)\boldsymbol{w}^T\boldsymbol{w} - (2\boldsymbol{y}^T + 2\boldsymbol{S}_0\boldsymbol{m}_0^T)\boldsymbol{w}} \\
&\quad + \sum_{i=1}^{N}{y_i}^2 + \boldsymbol{S}_0\boldsymbol{m}_0^T\boldsymbol{m}_0 + 2b_0))
\end{aligned}
\tag{9}
$$

Compare to the prior we got above, we can easily see that:

$$
\begin{aligned}
\boldsymbol{S}_N &= (\sum_{i=1}^{N}\boldsymbol{\phi}(\boldsymbol{x}_i) + \boldsymbol{S}_0^{-1})^{-1} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \boldsymbol{S}_0^{-1})^{-1} \\
\boldsymbol{m}_N &= \boldsymbol{m}_0 + \boldsymbol{y}\boldsymbol{S}_N^T \\
a_N &= a_0 + \frac{N}{2} \\
b_N &= b_0 + \frac{1}{2}(\sum_{i=1}^{N}y_i^2 + \boldsymbol{S}_0^{-1}\boldsymbol{m}_0\boldsymbol{m}_0 - \boldsymbol{S}_N^{-1}\boldsymbol{m}_N^T\boldsymbol{m}_N)
\end{aligned}
\tag{10}
$$

# 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
```

## 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.

## Load and preprocess the data

I 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 (http://lib.stat.cmu.edu/datasets/boston)](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)
```

## 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).

    """
    # TODO
    for i in range(len(X)):
        np.insert(X[i],0,1)
    w = np.dot(np.dot(np.linalg.inv(np.dot(np.transpose(X),X)),np.transpose(X)),y)
    return w
```

## 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).

    """
    # TODO
    for i in range(len(X)):
        np.insert(X[i],0,1)
    w = np.dot(np.dot(np.linalg.inv(np.dot(np.transpose(X),X)+reg_strength),np.transpose(X)),y)
    return w
```

## Task 3: Generate predictions for new data

In [11]:

```
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.

    """
    # TODO
    y_pred = []
    for i in range(len(X)):
        y_pred.append(np.dot(X[i],w))
    y_pred = np.array(y_pred)
    return y_pred
```

## Task 4: Mean squared error

In [12]:

```
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.

    """
    # TODO
    mse = (np.square(y_true - y_pred)).mean()
    return mse
```

## Compare the two models ¶

The reference implementation produces

- MSE for Least squares $\approx$ **23.98**
- MSE for Ridge regression $\approx$ **21.05**

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

In [13]:

```
# 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 = 23.964571384953114
MSE for Ridge regression = 22.25443747761982
```