Programming Task: Linear Regression

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

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

Your task

This notebook provides a code skeleton for performing linear regression. 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.

In the beginning of every function there is docstring which specifies the input and and expected output. Write your code in a way that adheres to it. You may only use plain python and anything that we imported for you above such as numpy functions (i.e. no scikit-learn classifiers).

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 = fetch_california_housing(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. in

# Split into train and test
test_size = 0.9
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).

    """
    ### YOUR CODE HERE ###
    return np.linalg.pinv(X) @ y
```

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).

    """
    ### YOUR CODE HERE ###
    return np.linalg.inv(X.T @ X + reg_strength * np.eye(X.shape[1])) @ X.T @ y
```

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.

    """
    ### YOUR CODE HERE ###
    return X @ w
```

Task 4: Mean squared error

In [6]:
```
import numpy as np
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.

    """
    ### YOUR CODE HERE ###
    return np.mean((y_true - y_pred)**2)
```

Compare the two models

The reference implementation produces

- MSE for Least squares $\approx$ **0.5347**
- MSE for Ridge regression $\approx$ **0.5331**

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

In [7]:
```
# Load the data
np.random.seed(1234)
X , y = fetch_california_housing(return_X_y=True)
X = np.hstack([np.ones([X.shape[0], 1]), X])
test_size = 0.9
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 = 0.5347102426013359
MSE for Ridge regression = 0.5912098054500012
```