Linear Regression

Machine Learning - Prof. Dr. Stephan Günnemann

Leonardo Freiherr von Lerchenfeld

November 16, 2017

Contents

1 Least squares regression

1.1 Problem 1

Is at the ending of the document

1.2 Problem 2

$$
\begin{aligned}
E_{weighted}(w) &= \frac{1}{2}\sum_{i=1}^{N} t_i[w^T\Phi(x_i) - y_i]^2 \\
with & \quad T = diag(t_1 \ ... \ t_n) \\
E_{weighted}(w) &= \frac{1}{2}(Y - \Phi W)^T T(Y - \Phi W) \\
0 &= \frac{\partial}{\partial w}\frac{1}{2}(Y^T - W^T\Phi^T)T(Y - \Phi W) \\
0 &= \frac{\partial}{\partial w}\frac{1}{2}(Y^TTY - Y^TT\Phi W - W^T\Phi^T TY + W^T\Phi^T T\Phi W) \\
with & \quad 0 = \frac{\partial}{\partial w}\frac{1}{2}(Y^TTY) \\
0 &= \frac{\partial}{\partial w}\frac{1}{2}(-2Y^TT\Phi W + W^T\Phi^T T\Phi W) \\
0 &= -Y^TT\Phi + \frac{1}{2}(\Phi^T T\Phi + (\Phi^T T\Phi)^T)W \\
W &= (\Phi^T T\Phi)^{-1}\Phi^T TY
\end{aligned}
$$

The weighting factor t_i can be interpreted as a weight for the datapoints.
1) The variance of a particular datapoint is inversely proportional to t_i. So t_i is the precision of the distribution.
2) t_i could also be interpreted as a multiplicator of data points, e.g., for $t_i = 2$ the i-th datapoint has double influence, so the i-th datapoint is treated as it would be twice in the dataset.

2 Ridge regression

2.1 Problem 3

$$
\begin{aligned}
E_{LS} &= \frac{1}{2}(\Phi w - y)^T(\Phi w - y) \\
y = \begin{pmatrix} y \\ 0 \end{pmatrix} & \quad \Phi = \begin{pmatrix} \Phi \\ \sqrt{\lambda} I \end{pmatrix} \\
E_{LS} &= \frac{1}{2}\left(\begin{pmatrix} \Phi \\ \sqrt{\lambda} I \end{pmatrix} w - \begin{pmatrix} y \\ 0 \end{pmatrix}\right)^T \left(\begin{pmatrix} \Phi \\ \sqrt{\lambda} I \end{pmatrix} w - \begin{pmatrix} y \\ 0 \end{pmatrix}\right) \\
&= \frac{1}{2}((\Phi w - y)^T(\sqrt{\lambda} w)^T)\begin{pmatrix} \Phi w - y \\ \sqrt{\lambda} w - 0 \end{pmatrix} \\
&= \frac{1}{2}((\Phi w - y)^T(\Phi w - y) + (\sqrt{\lambda} w)^T(\sqrt{\lambda} w)) \\
E_{ridge} &= \frac{1}{2}((\Phi w - y)^T(\Phi w - y)) + \frac{\lambda}{2} w^T w
\end{aligned}
$$

3 Bayesian linear regression

3.1 Problem 4

Our likelihood is as follows: $p(\mathbf{y}|\mathbf{\Phi}, \mathbf{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}(y_i|\mathbf{w^T}\mathbf{\Phi}(\mathbf{x_i}), \beta^{-1})$
The conjugate prior for w and β is: $p(\mathbf{w}, \beta) = \mathcal{N}(\mathbf{w}|\mathbf{m_0}, \beta^{-1}\mathbf{S_0})Gamma(\beta|a_0, b_0)$

The posterior distribution should be $p(\mathbf{w}, \beta, |\mathcal{D}) = \mathcal{N}(\mathbf{w}|\mathbf{m_N}, \beta^{-1}\mathbf{S_N})Gamma(\beta|a_N, b_N)$

$$
\begin{aligned}
p(w, \beta|\mathcal{D}) &= \frac{\beta^{\frac{1}{2}}}{S_N^{\frac{1}{2}}\sqrt{2\pi}} exp(-\frac{(w - m_N)^2}{2\beta^{-1}S_N}) \frac{b_N^{a_N}}{\Gamma(a_N)} \beta^{a_n - 1} exp(-b_N\beta) \\
ln(p(w, \beta|\mathcal{D})) &= \frac{1}{2} ln\ \beta - \frac{1}{2} ln\ S_N - \beta(w - m_N)^2 - 2S_N + a_N ln\ b_N + (a_N - 1) ln\ \beta - b_N\beta + const \\
posterior &\propto likelihood * prior \\
&= p(\mathbf{y}|\mathbf{\Phi}, \mathbf{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}(y_i|\mathbf{w^T}\mathbf{\Phi}(\mathbf{x_i}), \beta^{-1}) p(\mathbf{w}, \beta) = \mathcal{N}(\mathbf{w}|\mathbf{m_0}, \beta^{-1}\mathbf{S_0})Gamma(\beta|a_0, b_0) \\
&= \prod_{i=1}^{N} \frac{\beta^{\frac{1}{2}}}{\sqrt{2\pi}} exp(-\frac{(y_i - w^T\Phi_i)^2}{2\beta^{-1}}) \frac{\beta^{\frac{1}{2}}}{S_0^{\frac{1}{2}}\sqrt{2\pi}} exp(-\frac{(w - m_0)^2}{2\beta^{-1}S_0}) \frac{b_0^{a_0}}{\Gamma(a_0)} \beta^{a_0 - 1} exp(-b_0\beta) \\
ln\ (...) &= \frac{N}{2} ln\ \beta - \frac{\beta}{2} \sum_{i=1}^{N} (y_i - w^T\Phi)^2 + \frac{1}{2} ln\ \beta - \frac{1}{2} ln\ S_0 - \beta(w - m_0)^2 - 2S_0 + a_0 ln\ b_0 \\
&+ (a_0 - 1) ln\ \beta - b_0\beta + const
\end{aligned}
$$

When we look at all the terms with $ln\ \beta$ we get

$$
\begin{aligned}
a_N - 1 &= a_0 - 1 + \frac{N}{2} \\
a_N &= a_0 + \frac{N}{2}
\end{aligned}
$$

4 Appendix

4.1 Jupyter Notebook

04_homework_linear_regression

November 14, 2017

1 Programming assignment 4: Linear regression

```
In [33]: 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 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

```
In [34]: 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

         # 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_s
```

1.3 Task 1: Fit standard linear regression

```
In [35]: 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
    XTX = np.matmul(np.transpose(X),X)
    Pseudoinv = np.matmul(np.linalg.inv(XTX),np.transpose(X))
    w = np.matmul(Pseudoinv,y)
    return w
```

1.4 Task 2: Fit ridge regression

```
In [36]: 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
    D = np.size(X,1)
    XTX = np.matmul(np.transpose(X),X)
    xtxlagrange = XTX + reg_strength*np.identity(D)
    Pseudoinv = np.matmul(np.linalg.inv(xtxlagrange),np.transpose(X))
    w = np.matmul(Pseudoinv,y)
    return w
```

1.5 Task 3: Generate predictions for new data

In [37]:
```
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 = np.matmul(X,w)
    return y_pred
```

1.6 Task 4: Mean squared error

In [41]:
```
def mean_squared_error(y_true, y_pred):
    """Compute mean squared error between true and predicted regression ta

    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.mean((y_true - y_pred)**2)
    return mse
```

1.7 Compare the two models

The reference implementation produces * MSE for Least squares ≈ **23.98** * MSE for Ridge regression ≈ **21.05**

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

```
In [42]: # 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_s

         # 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 = 23.984307611781773
MSE for Ridge regression = 21.051487033772275
```

```
In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```