Machine Learning: Homework #4

Due on November 20, 2017 at 09:59am

Professor Dr. Stephan Guennemann

Marc Meissner - 03691673

Problem 1

Load the notebook 04_*homework_linear_regression.ipynb* from Piazza. Fill in the missing code and run the notebook. Convert the evaluated notebook to pdf and add it to the printout of your homework.

Solution
PDF added to the end of homework.

Problem 2

Let's assume we have a dataset where each datapoint, (x_i, y_i) is weighted by a scalar factor which we will call t_i. We will assume that $t_i > 0$ for all i. This makes the sum of squares error function look like the following:

$$E_{\text{weighted}}(w) = \frac{1}{2} \sum_{i=1}^{N} t_i [w^T \Phi(x_i) - y_i]^2$$

Find the equation for the value of w that minimizes this error function. Furthermore, explain how this weighting factor, t_i, can be interpreted in terms of
1) the variance of the noise on the data and
2) data points for which there are exact copies in the dataset.

Solution
Writing the equation above in matrix form and setting the derivative to zero:

$$E_{\text{weighted}}(W) = \frac{1}{2}(Y - \Phi W)^T T (Y - \Phi W)$$

$$\begin{aligned} 0 &= \frac{\partial}{\partial w} \frac{1}{2}(Y^T - W^T \Phi^T) T (Y - \Phi W) \\ &= \frac{\partial}{\partial w} \frac{1}{2}(Y^T T Y - Y^T T \Phi W - W^T \Phi^T T Y + W^T \Phi^T T \Phi W) \\ &= \frac{\partial}{\partial w} \frac{1}{2}(-2 Y^T T \Phi W + W^T \Phi^T T \Phi W) \\ &= -Y^T T \Phi + \frac{1}{2}(\Phi^T T \Phi + (\Phi^T T \Phi)^T) W \end{aligned}$$

$$W = (\Phi^T T \Phi)^{-1} \Phi^T T Y$$

t_i can be interpreted in two ways:

1) The higher t_i, the more trust we put into the data point (x_i, y_i). Therefore, it can be seen as an inverse variance parameter that models the uncertainty of individual data points.
2) Imagine $t_i = 2$ for a certain i. The result of the error function would be the same as if we had the i'th data point twice. Consequently, t_i can be seen as the number of identical copies of the i'th data point.

Problem 3

Show that the following holds: The ridge regression estimates can be obtained by ordinary least squares regression on an augmented dataset: Augment the design matrix $\Phi \in \mathbb{R}^{NxM}$ with M additional rows $\sqrt{\lambda}I_{MxM}$ and augment y with M zeros.

Solution

$E_{\text{normal}}(w) = \frac{1}{2}(\Phi w - y)^T(\Phi w - y)$

With augmented matrices:

$$
\begin{aligned}
&= \frac{1}{2}\left[\begin{pmatrix} \Phi \\ \hline \begin{matrix} \sqrt{\lambda} & & \\ & \ddots & \\ & & \sqrt{\lambda} \end{matrix} \end{pmatrix} w - \begin{pmatrix} y \\ 0 \end{pmatrix}\right]^T \left[\begin{pmatrix} \Phi \\ \hline \begin{matrix} \sqrt{\lambda} & & \\ & \ddots & \\ & & \sqrt{\lambda} \end{matrix} \end{pmatrix} w - \begin{pmatrix} y \\ 0 \end{pmatrix}\right] \\
&= \frac{1}{2}\left[\begin{pmatrix} \Phi w - y \\ \sqrt{\lambda} w \end{pmatrix}\right]^T \left[\begin{pmatrix} \Phi w - y \\ \sqrt{\lambda} w \end{pmatrix}\right] \\
&= \frac{1}{2}\left[\left((\Phi w - y)^T | (\sqrt{\lambda} w)^T\right)\right] \left[\begin{pmatrix} \Phi w - y \\ \sqrt{\lambda} w \end{pmatrix}\right] \\
&= \frac{1}{2}\left[(\Phi w - y)^T(\Phi w - y) + (\sqrt{\lambda} w)^T(\sqrt{\lambda} w)\right] \\
&= \frac{1}{2}\left[(\Phi w - y)^T(\Phi w - y) + \lambda w^T w\right] \\
&= E_{\text{ridge}}(W)
\end{aligned}
$$

Problem 4

It turns out that the conjugate prior for the situation when we have an unknown mean and unknown precision is a normal-gamma distribution (See section 2.3.6 in Bishop). This is also true when we have a conditional Gaussian distribution of the linear regression model. This means that if our likelihood is as follows:

$$p(y|\Phi, w, \beta) = \prod_{i=1}^{N} \mathcal{N}(y_i | w^T \Phi(x_i), \beta^{-1})$$

Then the conjugate prior for both w and β is

$$p(w, \beta) = \mathcal{N}(w | m_0, \beta^{-1} S_0)\text{Gamma}(\beta | a_0, b_0)$$

Show that the posterior distribution takes the same form as the prior, i.e.

$$p(w, \beta | D) = \mathcal{N}(w | m_N, \beta^{-1} S_N)\text{Gamma}(\beta | a_N, b_N)$$

Also be sure to give the expressions for m_N , S_N , a_N , and b_N.

Problem 4 continued on next page...

Solution
Solution

04_homework_linear_regression

November 16, 2017

1 Programming assignment 4: Linear regression

```
In [2]: 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 [3]: 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. includi

        # 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)
```

1.3 Task 1: Fit standard linear regression

```
In [4]: 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

    h1 = np.matmul(np.transpose(X),X)
    h2 = np.linalg.inv(h1)
    w = np.matmul(np.matmul(h2,np.transpose(X)),y)

    return w
```

1.4 Task 2: Fit ridge regression

```
In [5]: 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)
    w1 = np.linalg.inv(np.matmul(np.transpose(X),X) + reg_strength*np.identity(D))
    w = np.matmul(np.matmul(w1,np.transpose(X)),y)

    return w
```

1.5 Task 3: Generate predictions for new data

```
In [6]: 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 [7]: 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 = ((y_true-y_pred) ** 2).mean(axis=None)
            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 [8]:
```
# 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 = 23.984307611774046
MSE for Ridge regression = 21.051487033771537
```