Machine Learning Worksheet Solution 4

Julius Jankowski

1 Least squares regression

Problem 1:

See attached pages.

Problem 2:

Starting from the error sum formulation:

$$E_{weighted} = \frac{1}{2}\sum_{i=1}^{N} t_i(\mathbf{w}^T\mathbf{\Phi}(\mathbf{x}_i) - y_i)^2 \tag{1.1}$$

We can reformulate this problem by stacking all summation terms in a matrix vector style. The t_is will be represented by an eye matrix $\widetilde{\mathbf{T}}$ with the corresponding t_i in row i:

$$E_{weighted} = \frac{1}{2}(\widetilde{\mathbf{\Phi}} \cdot \mathbf{w} - \mathbf{y})^T \cdot \widetilde{\mathbf{T}} \cdot (\widetilde{\mathbf{\Phi}} \cdot \mathbf{w} - \mathbf{y}) \tag{1.2}$$

By resolving the brackets, we obtain ($\widetilde{\mathbf{T}}^T = \widetilde{\mathbf{T}}$):

$$E_{weighted} = \frac{1}{2}(\mathbf{w}^T \cdot \widetilde{\mathbf{\Phi}}^T \cdot \widetilde{\mathbf{T}} \cdot \widetilde{\mathbf{\Phi}} \cdot \mathbf{w} - 2\mathbf{y}^T \cdot \widetilde{\mathbf{T}} \cdot \widetilde{\mathbf{\Phi}} \cdot \mathbf{w} + \mathbf{y}^T \cdot \widetilde{\mathbf{T}} \cdot \mathbf{y}) \tag{1.3}$$

and now we can easily compute the derivative w.r.t. the weights $\mathbf{w}$ in order to get the optimizer $\mathbf{w}^*$:

$$\frac{\partial E_{weighted}}{\partial \mathbf{w}} = \mathbf{w}^T \cdot \widetilde{\mathbf{\Phi}}^T \cdot \widetilde{\mathbf{T}} \cdot \widetilde{\mathbf{\Phi}} - \mathbf{y}^T \cdot \widetilde{\mathbf{T}} \cdot \widetilde{\mathbf{\Phi}} = 0 \tag{1.4}$$

$$\mathbf{w}^* = (\widetilde{\mathbf{\Phi}}^T \cdot \widetilde{\mathbf{T}} \cdot \widetilde{\mathbf{\Phi}})^{-1} \cdot \widetilde{\mathbf{\Phi}}^T \cdot \widetilde{\mathbf{T}} \cdot \mathbf{y} \tag{1.5}$$

Here we can see that there is a new version of a pseudo-inverse matrix $\widetilde{\mathbf{\Phi}}^+ = (\widetilde{\mathbf{\Phi}}^T \cdot \widetilde{\mathbf{T}} \cdot \widetilde{\mathbf{\Phi}})^{-1} \cdot \widetilde{\mathbf{\Phi}}^T \cdot \widetilde{\mathbf{T}}$ which fits the general condition for a inverse matrix $\widetilde{\mathbf{\Phi}}^+ \cdot \widetilde{\mathbf{\Phi}} = \widetilde{\mathbf{I}}$. Hence the t_is can be interpreted as metric for the feature matrix $\widetilde{\mathbf{\Phi}}$, which weights each feature independtly. From a machine learning perspective, the t_is can be used for:

1) giving samples with less variance (hence more reliable) a higher impact in learning the $\mathbf{w}^*$.
2) replacing two exact copies of one sample with one sample with a weighting factor of 2.

2 Ridge regression

Problem 3:

Since we can reformulate the error equation between sum style and matrix-vector style as follows:

$$E = \frac{1}{2}(\widetilde{\mathbf{\Phi}} \cdot \mathbf{w} - \mathbf{y})^T \cdot (\widetilde{\mathbf{\Phi}} \cdot \mathbf{w} - \mathbf{y}) = \frac{1}{2}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{\Phi}_i - y_i)^2 \tag{2.1}$$

where $\mathbf{\Phi}_i$ corresponds to the i-th row vector of the design matrix. Hence we can also reformulate the specification stated in the task description:

$$E = \frac{1}{2}\sum_{i=1}^{N+M}(\mathbf{w}^T\mathbf{\Phi}_i - y_i)^2 \tag{2.2}$$

and now split up the sum after N samples:

$$E = \frac{1}{2}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{\Phi}_i - y_i)^2 + \frac{1}{2}\sum_{i=N+1}^{N+M}(\sqrt{\lambda}w_{i-N})^2 = \frac{1}{2}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{\Phi}_i - y_i)^2 + \frac{\lambda}{2}\sum_{i=1}^{M}w_i^2 \tag{2.3}$$

which is equal to the ridge regression formulation.

3 Bayesian linear regression

Problem 4:

04_homework_linear_regression

November 16, 2017

1 Programming assignment 4: Linear regression

```
In [5]: 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 [6]: 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 [28]: 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.inv(X.transpose().dot(X)).dot(X.transpose()).dot(y)

    return w
```

1.4 Task 2: Fit ridge regression

```
In [32]: 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).

    """

    D = X.shape[1]
    lambda_matrix = reg_strength*np.eye(D)
    w = np.linalg.inv(lambda_matrix + X.transpose().dot(X)).dot(X.transpose()).dot(y)

    return w
```

1.5 Task 3: Generate predictions for new data

```
In [25]: 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 = X.dot(w)

    return y_pred
```

1.6 Task 4: Mean squared error

```
In [30]: 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.

    """

    n = len(y_true)
    E = 0
    for i in range(n):
        E = E + np.dot((y_true[i] - y_pred[i]),(y_true[i] - y_pred[i]))

    return E / n
```

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 [33]: # 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 = 23.984307611774053
MSE for Ridge regression = 21.051487033771537
```