Machine Learning homework 3 solution
Linear Regression

Wiktor Jurasz - M.Nr. 03709419

November 10, 2018

1 Problem 1

1.1 Programming assignment 2: Linear regression

```
In [2]: import numpy as np

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

1.1.1 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.

1.1.2 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. 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)
```

1.1.3 Task 1: Fit standard linear regression

```
In [18]: 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).

"""
# np.linalg.pinv(X_train).dot(y_train)
return np.linalg.solve(X.T.dot(X), X.T.dot(y))
```

1.1.4 Task 2: Fit ridge regression

```
In [56]: 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).

             """
             return np.linalg.solve(X.T.dot(X) + reg_strength
                                    * np.identity(X.shape[1]), X.T.dot(y))
```

1.1.5 Task 3: Generate predictions for new data

```
In [60]: 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.

             """
```

```
            return X.dot(w)
```

1.1.6 Task 4: Mean squared error

```
In [50]: 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.

            """
            return (np.square(y_true - y_pred)).mean()
```

1.1.7 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 [64]: # 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 = 23.96457138495054
MSE for Ridge regression = 21.03493121591798
```

2 Problem 2

2.1 Minimizing Error

$$E_w(\omega) = \frac{1}{2}\sum_{i=1}^{N} t_i[\omega^T\phi(x_i) - y_i] = \frac{1}{2}[\Phi\omega - \mathbf{y}]^T T[\Phi\omega - \mathbf{y}] \tag{1}$$

Where T is diagonal matrix NxN with $T_{i,i} = t_i$
Expading this equation we get:

$$E_w(\omega) = \frac{1}{2}[\omega^T\Phi^T T\Phi\omega - \omega^T\Phi^T T\mathbf{y} - \mathbf{y}^T T\Phi\omega + \mathbf{y}^T T\mathbf{y}] \tag{2}$$

To find ω that minimize the value we caluclate $\nabla_\omega E_w(\omega)$:

$$\nabla_\omega(\omega^T\Phi^T T\Phi\omega) = \omega^T(\Phi^T T\Phi + (\Phi^T T\Phi)^T) = \omega^T(\Phi^T T\Phi + \Phi^T T\Phi) = 2\omega^T\Phi^T T\Phi \tag{3}$$

$$\nabla_\omega(\omega^T\Phi^T T\mathbf{y}) = (\Phi^T T\mathbf{y})^T = \mathbf{y}^T T\Phi \tag{4}$$

$$\nabla_\omega(\mathbf{y}^T T\Phi\omega) = \mathbf{y}^T T\Phi \tag{5}$$

$$\nabla_\omega(\mathbf{y}^T T\mathbf{y}) = 0 \tag{6}$$

Using: T is diagonal $\Rightarrow$ T is simmetric $\Rightarrow$ $T^T = T$
From above equations we have:

$$\nabla_\omega E_w(\omega) = \omega^T\Phi^T T\Phi - \mathbf{y}^T T\Phi \tag{7}$$

Now comparing it to 0 and Transposing we have finall solution:

$$\Phi^T T\Phi\omega = \Phi^T T\mathbf{y} \Rightarrow \omega = (\Phi^T T\Phi)^1\Phi^T T\mathbf{y} \tag{8}$$

Since it was said on the lecture that $\nabla_\omega\nabla_\omega E(\omega) = \Phi^T\Phi$ is positive (semi)definite and we have by assumption that all values of T are positive, thus $\nabla_\omega\nabla_\omega E_w(\omega) = \Phi^T T\Phi$ is also positive (semi)definite, so equation (8) is minimal solution for ω

2.2 Variance of the noise

It might be that the variance of noise is not constant (data are heteroskedast). It means that some obervations have higher noise variance then others, so we cannot predict with the same accuracy for all points. We can use T to put more emphasis on points with smaller variance by setting $t_i = \frac{1}{\sigma_i^2}$

2.3 Exact Copies

If our dataset contains duplicated, regression is biased in their direction (the duplicated points have higher impact on regression line). However we might want to eliminate duplicates (or minimize their impact) by setting lower weight t on such cases (e.g $t = \frac{1}{|duplicates|}$ for all subsets of observations that have the same X and y).

3 Problem 3

3.1 Notation

A denotes augmented design matrix and $\mathbf{U}$ denotes augmented values vector.
Let's also define a function f:

$$f(i,j,\lambda) = \begin{cases} 0 & i \neq j \\ \lambda & i = j \end{cases}$$

Matrix form of ridge regression:

$$w_r^* = (\lambda I_D + \Phi^T \Phi)^{-1} \Phi^T \mathbf{y} \tag{9}$$

Matrix form of augmented regression:

$$w_a^* = (A^T A)^{-1} A^T U \tag{10}$$

3.2 Proof

Show that $w_r^* = w_a^*$

3.2.1 Part 1

First we show that $\lambda I_D + \Phi^T \Phi = L$ is the same as $A^T A = D$.

Single element of L:

$$l_{i,j} = \sum_{k=1}^{N} \phi_{i,k}^T \phi_{k,j} + f(i,j,\lambda) \tag{11}$$

The (i,j) element of D is equal to

$$d_{i,j} = \sum_{k=1}^{N+M} a_{i,k}^T a_{k,j} \tag{12}$$

Now we make following obervation $\forall k > N$:
if $i \neq j$ $a_{i,k}^T a_{k,j} = 0$ else $a_{i,k}^T a_{k,j} = \sqrt{\lambda} * \sqrt{\lambda} = \lambda$. Thus every element of D can be expressed as follows:

$$d_{i,j} = \sum_{k=1}^{N} [a_{i,k}^T a_{k,j}] + f(i,j,\lambda) \tag{13}$$

From problem description we also know that $\forall i \leq N$, $i-th$ row of A is equal to $i-th$ row of Φ (The same for columns for transposed Φ and A). So we can rewrite (13) equation in to following form:

$$d_{i,j} = \sum_{k=1}^{N} [\phi_{i,k}^T \phi_{k,j}] + f(i,j,\lambda) \tag{14}$$

From above:

$$L = D \tag{15}$$

3.2.2 Part 2

Now we show that $\Phi^T \mathbf{y} = L$ is the same as $A^T U = D$.

Single element of L:

$$l_{i,j} = \sum_{k=1}^{N} \phi_{i,k}^T \mathbf{y}_k \tag{16}$$

The (i,j) element of D is equal to

$$d_{i,j} = \sum_{k=1}^{N+M} a_{i,k}^T u_k \tag{17}$$

From problem descriptoin we know that $\forall k > N$, $u_k = 0$ Thus every element of D can be expressed as follows:

$$d_{i,j} = \sum_{k=1}^{N} a_{i,k}^T u_k \tag{18}$$

From problem description we also know that $\forall i \leq N$, $i-th$ row of A is equal to $i-th$ row of Φ (The same for columns for transposed Φ and A), and $i-th$ row of U is equal to $i-th$ row of $\mathbf{y}$ So we can rewrite (18) equation in to following form:

$$d_{i,j} = \sum_{k=1}^{N} \phi_{i,k}^{T} \mathbf{y}_k \tag{19}$$

From above:

$$L = D \tag{20}$$

3.2.3 Part 3

From part 1:

$$\lambda I_D + \Phi^T \Phi = A^T A \tag{21}$$

From part 2:

$$\Phi^T \mathbf{y} = A^T U \tag{22}$$

Thus $w_r^* = w_a^*$ Q.E.D.