Machine Learning Homework Assignment 4

Akshay Aradhya

November 20, 2017

1 Problem 1

The notebook solution attached at the end.

2 Problem 2

Adding a weighting factor t_i to the datapoint (x_i, y_i) makes it:

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

Here T is a N-dimensional vector with the values of t_i
Hence, we have

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

Taking gradient of $E_w(\omega)$ w.r.t ω and equating it to 0 to find minimum, we get:

$$\Delta_w E_w = \Phi^TTT^T\Phi\omega - \Phi^TTT^Ty = 0$$

$$\omega^* = (\Phi^TTT^T\Phi)^{-1}\Phi^TTT^Ty$$

Each weighted scalar factor assigns an importance or weight to every datapoint. Hence, the weight for data points closer to the mean should be more and therefore the variance on the noise should reduce. Therefore more the variance of the noise, lesser the scalar factor and vice versa.
Similarly, the data points that have exact copies, will have more weightage on the model. Hence, higher the scaling factor.

3 Problem 3

The augmented matrix Φ - (N+M) x M is of the form:

$$\begin{bmatrix} \Phi_0(x_1) & \Phi_1(x_1) & \cdots & \Phi_M(x_1) \\ \Phi_0(x_2) & \Phi_1(x_2) & \cdots & \Phi_M(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \Phi_0(x_N) & \Phi_1(x_N) & \cdots & \Phi_M(x_N) \\ \sqrt{\lambda} & 0 & \cdots & 0 \\ 0 & \sqrt{\lambda} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sqrt{\lambda} \end{bmatrix}$$

Similarly the vector y - (N+M) x 1 with M additional zeroes looks like:

$$\begin{bmatrix} y_1 & y_2 & \cdots & y_N & 0 & 0 & \cdots & 0 \end{bmatrix}$$

Now, we know that ω^* for Least Squares and for Ridge is:

$$\omega_{\text{LS}} = (\Phi^T\Phi)^{-1}\Phi^T y \text{ and } \omega_R = (\lambda I_M + \Phi^T\Phi)^{-1}\Phi^T y$$

For the augmented matrices, the ω_{LS} can be expanded to be written as

$$\omega_{\text{LS}} = (\lambda I_M + \Phi'^T\Phi')^{-1}\Phi^T y \tag{1}$$

where Φ' is the N x M matrix with out the addition of M rows of $\sqrt{\lambda}I_M$. Also it follows Φy would result in $\Phi' y$ as there are M zeroes at the end and the matrix multiplication would result in 0. Therefore we can re-write eq (1) as:

$$\omega_{\text{LS}} = (\lambda I_M + \Phi'^T\Phi')^{-1}\Phi'^T y$$

Hence proved.

4 Problem 4

We know that,

$$P(\omega, \beta|D) \propto P(y|\Phi, \beta, \omega).P(\beta, \omega) \tag{2}$$

and

$$P(\omega, \beta) = N(\omega|m_o, \beta^{-1}S_o)Gamma(\beta|a_o, b_o)$$

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

Substituting these in equation (2) we get,

$$P(\omega, \beta|D) \propto \beta^{(a_o - 1 + N/2)}exp\Big(\frac{-\beta}{2}A - b_o\beta\Big) \tag{3}$$

where A is given by $A = (m_o - \omega)^T S_o^{-1}(m_o - w) + (\Phi\omega - y)^T(\Phi\omega - y)$

$$A = \omega^T S_o^{-1}\omega + \omega^T\Phi^T\Phi\omega - 2\omega^T S_o^{-1}m_o - 2\omega^T\Phi y + y^T y + m_o^T S_o^{-1}m_o \tag{4}$$

Considering only the ω terms first, we add and subtract the term: $m_N^T S_N^{-1} m_N$

$$A = \omega^T(S_o^{-1} + \Phi^T\Phi)\omega - 2\omega^T(S_o^{-1}m_o - \Phi y) + m_N^T S_N^{-1}m_N - m_N^T S_N^{-1}m_N$$

from this we can find the m_N and S_N terms:

$$S_N^{-1} = (S_o^{-1} + \Phi^T\Phi) \text{ and } m_N = S_N(S_o^{-1}m_o - \Phi y)$$

From equations (3) and (4) we have:

$$a_N = a_o + N/2 \text{ and } b_N = b_o + \frac{1}{2}(y^T y + m_o^T S_o^{-1}m_o - m_N^T S_N^{-1}m_N)$$

Hence the posterior is of the form,

$$P(\omega, \beta|D) = N(\omega|m_N, \beta^{-1}S_N)Gamma(\beta|a_N, b_N)$$

Programming assignment 4: Linear regression

In [3]:
```
import numpy as np

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

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.

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 (http://lib.stat.cmu.edu/datasets/boston)

In [4]:
```
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)
```

Task 1: Fit standard linear regression

In [22]:
```
from numpy.linalg import inv
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).

    """
    X_T = np.transpose(X)
    w=np.dot(np.dot(inv(np.dot(X_T,X)),X_T),y)

    # TODO
    return w
```

Task 2: Fit ridge regression

In [23]:
```
from numpy.linalg import inv
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).

    """
    X_T=np.transpose(X)
    I_D=np.identity(X.shape[1])
    lam_I=reg_strength*I_D
    w=np.dot(np.dot(inv(lam_I+np.dot(X_T,X)),X_T),y)

    # TODO
    return w
```

Task 3: Generate predictions for new data

In [24]:
```
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 = np.dot(X,w)
    # TODO
    return y
```

Task 4: Mean squared error

In [28]:
```
def mean_squared_error(y_true, y_pred):
    """Compute mean squared error between true and predicted regression target
s.

    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.

    """
    sum=0
    for i in range(0,len(y_true)):
        sum+=(y_true[i]-y_pred[i])**2

    mse = sum/(len(y_true))
    # TODO
    return mse
```

Compare the two models

The reference implementation produces

- MSE for Least squares $\approx$ **23.98**
- MSE for Ridge regression $\approx$ **21.05**

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

In [29]:
```
# 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 = 23.984307611781773
MSE for Ridge regression = 21.051487033772275
```