Machine Learning Assignment 4

Shivangi Aneja

20-November-2017

04_homework_linear_regression

November 19, 2017

1 Programming assignment 4: Linear regression

```
In [180]: 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 [181]: 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. inclu

          # 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 [182]: 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

    X_trans = np.transpose(X)
    X_trans_X_inv = np.linalg.inv(np.dot(X_trans,X))
    X_trans_y = np.dot(X_trans,y)
    weight_array = np.dot(X_trans_X_inv,X_trans_y)

    return weight_array
```

1.4 Task 2: Fit ridge regression

```
In [196]: 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
              X_trans = np.transpose(X)
              X_trans_X = np.dot(X_trans,X)
              iden = np.identity(X_trans_X.shape[0])
              iden_rg = np.multiply(iden,reg_strength)
              inv = np.linalg.inv(np.add(X_trans_X,iden_rg))
              X_trans_y = np.dot(X_trans,y)
```

```
        weight_array = np.dot(inv,X_trans_y)

        return weight_array
```

1.5 Task 3: Generate predictions for new data

```
In [172]: 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
              X_weighted = X * w
              Y_predicted = X_weighted.sum(axis=1)

              return Y_predicted
```

1.6 Task 4: Mean squared error

```
In [173]: 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
    size = y_true.shape[0]
    y_diff_sqr = np.square(np.subtract(y_true,y_pred))
    y_sum_diff_sqr = np.sum(y_diff_sqr,axis=0)
    return np.divide(y_sum_diff_sqr,size)
```

1.7 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 [197]: # 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 = 23.9843076118
MSE for Ridge regression = 21.0514870338
```

Problem 2

Datapoint x_i, y_i weighted by a scalar $t_i > 0$
$E_{weighted}(w) = \frac{1}{2}\sum_{i=1}^{N} t_i[w^T\phi(x_i) - y_i]^2 = \frac{1}{2}\sum_{i=1}^{N}[\sqrt{t_i}(w^T\phi(x_i) - y_i)]^2$

Converting to vector form , we have
$\Rightarrow \frac{1}{2}\boldsymbol{(\phi w - y)^T t(\phi w - y)} = \frac{1}{2}\boldsymbol{(w^T\phi^T - y^T)t(\phi w - y)} = \frac{1}{2}\boldsymbol{(w^T\phi^T t - y^T t)(\phi w - y)}$
$\Rightarrow \frac{1}{2}\boldsymbol{[w^T\phi^T t\phi w - y^T t\phi w - w^T\phi^T t y + y^T t y]}$
$\nabla_w = \boldsymbol{\phi^T t\phi w - \phi^T t y}$

To find the minimum value, put $\nabla_w = 0$, we have,
$w^* = \boldsymbol{(\phi^T t\phi)^{-1}\phi^T t y}$

if $\boldsymbol{t = I}$, then we have, $w_{ML} = \boldsymbol{(\phi^T\phi)^{-1}\phi^T y}$

(1) When we compare w^* to w_{ML} as discusses in the likelihood function of the lecture , then $\boldsymbol{t}$ acts as a precision / inverse Variance for the data point $(\boldsymbol{x_i}, y_i)$
(2) If $\boldsymbol{t} > 0$ and takes positive Integral values , then more priority to data point with high scaling factor is given for duplicate / replicated data point.

Problem 3

$E_{ridge}(w) = \frac{1}{2}\sum_{i=1}^{N}[w^T\phi(x_i) - y_i]^2 + \frac{\lambda}{2}||w||_2^2$

$\phi \in R^{N\times M} \Rightarrow \phi'_{(N+M)\times M} = \begin{bmatrix}\phi \\ \sqrt{\lambda}I_{M\times M}\end{bmatrix}$

Similarly,

$y' = \begin{bmatrix} y \\ 0_{M\times 1}\end{bmatrix}$

We know,
$w^* = (\phi'^T\phi')^{-1}\phi'^T y$ ......(1)
$\phi'^T\phi' = \phi^T\phi + \lambda I$.......(2)
$\phi'^T y' = \phi^T y$...........(3)

Using (1),(2) and (3) we have,
$E_{ridge} = \boldsymbol{(\phi^T \phi + \lambda I)^{-1} \phi^T y}$

Hence Proved

Problem 4

We have,

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

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

Posterior $\propto$ Prior $\times$ likelihood

$$p(w, \beta|y) \propto p(y|\phi, w, \beta)p(w|\beta)$$

Taking log on both sides, we have ,
$ln\ p(w, \beta|y) = ln[p(y|\phi, w, \beta) + ln[p(w|\beta)]$
$\Rightarrow ln[\prod_{i=1}^{N} \mathcal{N}(y_i|w^T\phi(x_i), \beta^{-1})] + ln[\mathcal{N}(w|m_0, \beta^{-1}S_0)Gamma(\beta|a_0, b_0)]$

$$\Rightarrow \sum_{i=1}^{N} ln[\frac{1}{\sqrt{2\pi\beta^{-1}}}e^{-\frac{(y_i - w^T\phi(x_i))^2}{2\beta^{-1}}}] + ln[\frac{1}{\sqrt{2\pi\beta^{-1}S_0}}e^{-\frac{(w-m_0)^2}{2\beta^{-1}S_0}}] + log[\frac{b^{a_0}\beta^{a_0-1}e^{-b_0\beta}}{\Gamma(a_0)}]$$

$$\Rightarrow \sum_{i=1}^{N}(\frac{-1}{2}ln(2\pi\beta^{-1}) - \frac{(w^T\phi(x_i) - y_i)^2}{x\beta^{-1}}) - \frac{1}{2}ln(2\pi\beta^{-1}S_0) - \frac{(w-m_0)^2}{2\beta^{-1}S_0} + (a_0 - 1)ln(\beta) - b_0\beta$$

$$\Rightarrow \frac{N}{2}ln(\beta) - \frac{\beta}{2}\sum_{i=1}^{N}(w^T\phi(x_i) - y_i)^2 + (a_0 - 1)ln(\beta) - -b_0\beta - \frac{\beta}{2}(w - m_0)^T S_0^{-1}(w - m_0) - \frac{1}{2}ln|S_0| + \frac{m}{2}ln(\beta)$$

Using Product Rule, we have ,

$$p(w, \beta|y) = p(w|\beta, y)p(\beta|y)$$

$$ln(p(w|\beta, y)) = \frac{-\beta}{2}w^T(\phi^T\phi + S_0^{-1})w + w^T[\beta S_0^{-1}m_0 + \beta\phi^T y] + constt.$$

Thus $p(w|\beta, y)$ is a Gaussian distribution with following mean and covariance
$m_N = S_N[S_0^{-1}m_0 + \phi^T y]$
$S_N^{-1} = \beta(S_0^{-1} + \phi^T \phi)$

To find $p(\beta|y)$ we need all the terms involvig β and discard terms independent of β

$$ln(p(\beta|y)) = \frac{-\beta}{2}m_0^T S_0^{-1}m_0 + \frac{\beta}{2}m_N^T S_N^{-1}m_N + \frac{N}{2}ln(\beta) - b_0\beta + (a_0 - 1)ln(\beta) - \frac{\beta}{2}\sum_{i=1}^{N} y_i^2 + constt.$$

Thus $p(\beta|y)$ is a Gamma distribution with following parameters
$a_N = a_0 + \frac{N}{2}$
$b_N = b_0 + \frac{1}{2}(m_0^T S_0^{-1}m_0 - m_N^T S_N^{-1}m_N + \sum_{i=1}^{N} y_i^2)$