Machine learning Homework- Linear regression

Abinav Ravi Venkatakrishnan - 03694216 and Abhijeet Parida - 03679676

November 11, 2018

1 Least Square Regression

Problem 2

Given: The error function

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

We know that taking the first derivative of the cost function gives the minimum of the error function.

$$\nabla E_{weighted}(w) = \nabla_w \frac{1}{2}\sum_{i=1}^{N} t_i[w^T\phi(x_i) - y_i]^2 = 0 \tag{2}$$

Now the RHS can be written as a matrix equation with T as the diagonal matrix.

$$\nabla_w E_{weighted} = \nabla_w \frac{1}{2}[\Phi^T \mathbf{T} w - y]^T[\Phi^T \mathbf{T} w - y] \tag{3}$$

If T = I(Identity Matrix) hence giving the solution

$$w_* = (\Phi^T\Phi)^{-1}\Phi^T y \tag{4}$$

t_i acts as a multiplication factor for variance in y_i due to x_i

when the data points for which there are exact copies in the dataset.The error contribution due to these repeated points are increased.

2 Ridge regression

Problem 3

Before Augmentation,

$$E_{ridge} = \frac{1}{2}\sum_{i=1}^{N}(w_N^T\phi(x_i) - y_i)^2 + \frac{\lambda}{2}w_N^T w_N \tag{5}$$

After Augmentation,

$$E_{ridge} = \frac{1}{2}\sum_{i=1}^{N+M}(w_{N+M}^T\phi(x_i) - y_i)^2 + \frac{\lambda}{2}w_{N+M}^T w_{N+M}$$

This can be split as,

$$\begin{aligned} E_{ridge} &= \tfrac{1}{2}\textstyle\sum_{i=1}^{N+M}(w_{N+M}^T\phi(x_i) - y_i)^2 + \tfrac{\lambda}{2}w_{N+M}^T w_{N+M} && (6)\\ &= \tfrac{1}{2}\textstyle\sum_{i=1}^{N}(w_N^T\phi(x_i) - y_i)^2 + \tfrac{\lambda}{2}w_N^T w_N + \tfrac{1}{2}\textstyle\sum_{i=N}^{N+M}(w_{N:N+M}^T\phi(x_i) - y_i)^2 + \tfrac{\lambda}{2}w_{N:N+M}^T w_{N:N+M} && (7)\\ & && (8) \end{aligned}$$

In 8, the term $w_{N:N+M} = 0$, because

$$w_{N:N+M} = (X_{N:N+M}^T X_{N:N+M})^{-1} X_{N:N+M}^T y_{N:N+M}$$

because

$$y_{N:N+M} = 0$$

Therefore 8 is equal to 5

3 Bayesian linear regression

Problem 4

We know that
posterior $\propto$likelyhood x prior
we will work in the log scale to easily deal with the $\prod$

$$\begin{aligned} log(\mathcal{N}(\mathbf{w}|\mathbf{m}_N, \beta^{-1}\mathbf{S}_N)) + log(Gamma(\beta|a_N, b_N)) = \quad & \textstyle\sum_{i=1}^{N} log(\mathcal{N}(y_i|\mathbf{w}^T\phi(x_i), \beta^{-1})) \\ & +log(\mathcal{N}(\mathbf{w}|\mathbf{m}_0, \beta^{-1}\mathbf{S}_0)) \\ & +log(Gamma(\beta|a_0, b_0)) \end{aligned}$$

By simple pattern matching after resolution we can easily find the unknowns and will hence prove the validity of the claim.

homework_03_notebook

November 9, 2018

1 Programming assignment 2: Linear regression

```
In [1]: 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 [2]: 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)
```

1.3 Task 1: Fit standard linear regression

```
In [3]: 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
    w = np.zeros_like(X.shape[1])
    X_transpose = X.transpose((1,0))
    X_star = X_transpose.dot(X)
    X_inverse = np.linalg.inv(X_star)
    w = X_inverse.dot(X_transpose).dot(y)
    return w
```

1.4 Task 2: Fit ridge regression

```
In [10]: 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
             w = np.zeros_like(X.shape[1])
             X_transpose = X.transpose((1,0))
             X_star = X_transpose.dot(X)+ reg_strength*np.eye(X.shape[1])
             X_inverse = np.linalg.inv(X_star)
             w = X_inverse.dot(X_transpose).dot(y)
             return w
```

1.5 Task 3: Generate predictions for new data

In [11]:
```python
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.dot(X,w)
    return y_pred
```

1.6 Task 4: Mean squared error

In [6]:
```python
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 = np.mean((y_pred-y_true)**2)
    return mse
```

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 [12]:
```python
# 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
```