Exercise Sheet 04

CORBUCCI LUCA
03726968

Problem 1: Let's assume we have a dataset where each datapoint, $(\boldsymbol{x}_i, y_i)$ is weighted by a scalar factor which we will call t_i. We will assume that $t_i > 0$ for all i. This makes the sum of squares error function look like the following:

$$E_{\text{weighted}}(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N} t_i \left[\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_i) - y_i\right]^2$$

Find the equation for the value of $\boldsymbol{w}$ that minimizes this error function.

Furthermore, explain how this weighting factor, t_i, can be interpreted in terms of
1) the variance of the noise on the data and
2) data points for which there are exact copies in the dataset.

Minimize the error function w.r.t. w:

$$w^* = \text{argmin}\ E_{\text{weighted}}(w)$$

$$= \text{argmin}\ \frac{1}{2}\sum_{i=1}^{N} t_i \left[w^T \phi(x_i) - y_i\right]^2$$

I define the vector $T \in \mathbb{R}^N$ which contains $t_i \in \mathbb{R}\ \ \forall i\ \ 1 < i < N$.

$$= \text{argmin}\ \frac{1}{2}\left(\Phi w - y\right)^T T \left(\Phi w - y\right)$$

$$E_{\text{weighted}}(w) = \frac{1}{2}\left(\Phi w - y\right)^T T \left(\Phi w - y\right)$$

I compute the gradient:

$$\nabla_w E_{\text{weighted}}(w) = \nabla_w \frac{1}{2}\left(\Phi w - y\right)^T T \left(\Phi w - y\right)$$

$$= \nabla_w \frac{1}{2}\left(w^T \Phi^T - y^T\right) T \left(\Phi w - y\right)$$

$$= \frac{1}{2}\left(w^T \Phi^T T - y^T T\right)\left(\Phi w - y\right)$$

$$\nabla_w \frac{1}{2}\Big(w^T \Phi^T T \Phi w - w^T \Phi^T T y - y^T \Phi T w + y^T T y \Big)$$

Differentiate with respect to w:

$$\frac{1}{2} w^T \left(\Phi^T T \Phi + \Phi^T T \Phi\right) - \frac{1}{2}\left(2 \Phi^T T y\right) =$$

$$= w^T \left(\Phi T \Phi\right) - \left(\Phi^T T y\right) = 0$$

Set the gradient to ϕ to find w^*

$$w^* = \left(\Phi^T T \Phi\right)^{-1} \left(\Phi^T T y\right)$$

How can t_i can be interpreted in term of:

(a) the variance of the noise on the data

Using the weighting factor t_i and knowing the variance σ of the noise data we can balance the noise term setting $t_i = \frac{1}{\sigma^2}$. this leads to a more precise regression.

(b) Data points for which there are exact copies in the dataset

I think that if we have a data point which has some copies, the model will consider these data points as more important with respect to the others data points.
Maybe we can use the weight to balance the importance of these points so that the model will not consider them more important than the others.

Problem 2: Show that the following holds: The ridge regression estimates can be obtained by ordinary least squares regression on an augmented dataset: Augment the design matrix $\mathbf{\Phi} \in \mathbb{R}^{N \times M}$ with M additional rows $\sqrt{\lambda}\boldsymbol{I}_{M\times M}$ and augment $\boldsymbol{y}$ with M zeros.

$$E_{LS} = \frac{1}{2}\sum_{i=1}^{N}\left[\omega^T\phi(x_i) - y_i\right]^2 + \frac{1}{2}\sum_{i=N+1}^{N+M}\left[\omega^T\phi(x_i) - y_i\right]^2$$

if $N+1 < i < N+M$

$$\phi(x_i) = \begin{pmatrix}\sqrt{\lambda}\, I_{M\times M} \\ \vdots \\ \sqrt{\lambda} I_{M\times M}\end{pmatrix} = V$$

Consider the second part of the E_{LS}:

$$\frac{1}{2}\sum_{i=N+1}^{N+M}(\omega^T V - y_i)^T(\omega^T V - y_i)$$

equal to ϕ

$$= \frac{1}{2}\left(\omega_0\sqrt{\lambda} I_{M\times M} + \dots + \omega_M\sqrt{\lambda} I_{M\times M}\right)^2$$

$$= \frac{1}{2}\left(\sqrt{\lambda} I_{M\times M}\left(\omega_0 + \dots + \omega_M\right)\right)^2$$

$$= \frac{1}{2}\lambda\left(\omega_0 + \dots + \omega_M\right)^2$$

$$= \frac{\lambda}{2}\|\omega\|_2^2$$

Problem 3: Derive the closed form solution for ridge regression error function

$$E_{\text{ridge}}(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) - y_i)^2 + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}$$

Additionally, discuss the scenario when the number of training samples N is smaller than the number of basis functions M. What computational issues arise in this case? How does regularization address them?

$$E_{ridge}(w) = \frac{1}{2}\sum_{i=1}^{N}\left(w^T\phi(x_i) - y_i\right)^T\left(w^T\phi(x_i) - y_i\right) + \frac{\lambda}{2}w^Tw$$

$$= \frac{1}{2}\left(\Phi w - y\right)^T\left(\Phi w - y\right) + \frac{\lambda}{2}w^Tw$$

Compute the gradient

$$\nabla_w \ \frac{1}{2}\left(\Phi w - y\right)^T\left(\Phi w - y\right) + \frac{\lambda}{2}w^Tw$$

$$\nabla_w \ \frac{1}{2}\left(w^T\Phi^T\Phi w - 2w^T\Phi^T y + y^Ty\right) + \frac{\lambda}{2}w^Tw$$

$$\nabla_w \ \frac{1}{2}\left(2\Phi^T\Phi w - 2\Phi^T y\right) + \frac{\lambda}{2}w^Tw$$

$$\Phi^T\Phi w - \Phi^T y + \frac{\lambda}{2}2w^T = 0$$

Put the gradient equal to ϕ to find w^*

$$\Phi^T \Phi w + \lambda w^T = \Phi^T y$$

$$w^* = \left(\Phi^T \Phi + \lambda I\right) \Phi^T y$$

↳ Identity matrix

Consider the case $N < M$

I think that we could have overfitting because our model will fit every data point that are in our dataset.

To fit every data points the coefficient w takes large positive values and large negative values.

the regularization (if we choose the right value for λ) solves this issue because it avoid that the coefficients can reach large positive or negative values

Problem 4: In class, we only considered functions of the form $f : \mathbb{R}^n \to \mathbb{R}$. What about the general case of $f : \mathbb{R}^n \to \mathbb{R}^m$? For linear regression with multiple outputs, write down the loglikelihood formulation and derive the MLE of the parameters.

Multi output: $f_w(x) = \mathbb{R}^n \to \mathbb{R}^M$

In this case we have

$$Y = W X + \mathcal{E}$$

(matrix)

Likelihood of the entire dataset:

$$P(y \mid X, w, \beta) = \prod_{i=1}^{N} P(y_i \mid f_w(x_i), \beta)$$

$$w_{ML}, \beta_{ML} = \underset{w, \beta}{\operatorname{argmax}}\, P(y \mid X, w, \beta)$$

$$= \underset{w, \beta}{\operatorname{argmin}} - \ln P(y \mid X, w, \beta)$$

$$E_{ML} = - \ln P(y \mid X, w, \beta)$$

$$= - \ln \left[\prod_{i=1}^{N} \mathcal{N}(y_i \mid f_w(x_i), \beta^{-1}) \right]$$

$$= -\ln\left[\prod_{i=1}^{N}\sqrt{\frac{\beta}{2\pi}}\exp\left(-\frac{\beta}{2}\left(wx_i - y_i\right)^2\right)\right]$$

$$= -\sum_{i=1}^{N}\ln\left(\sqrt{\frac{\beta}{2\pi}}\right) - \frac{\beta}{2}\left(wx_i - y_i\right)^2$$

$$= \frac{\beta}{2}\sum\left(wx_i - y_i\right)^2 - \frac{N}{2}\ln\beta + \frac{N}{2}\ln 2\pi$$

Derive w.r.t β and w

$$w_{ML} = \frac{\beta}{2}\sum\left(wx_i - y_i\right)^2$$

$$= \left(X^T X\right)^{-1} X^T y$$

$$\beta_{ML} = \operatorname{argmin}\left[\frac{\beta}{2}\sum\left(w_{ML}x_i - y_i\right)^2 - \frac{N}{2}\ln\beta + \frac{N}{2}\ln 2\pi\right]$$

set the derivative to ϕ

$$\frac{1}{2}\sum\left(w_{ML}x_i - y_i\right)^2 - \frac{N}{2\beta} = 0$$

$$\frac{1}{\beta_{ML}} = \frac{1}{2N}\sum\left(w_{ML}x_i - y_i\right)^2$$

Problem 5: We want to perform regression on a dataset consisting of N samples $\boldsymbol{x}_i \in \mathbb{R}^D$ with corresponding targets $y_i \in \mathbb{R}$ (represented compactly as $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ and $\boldsymbol{y} \in \mathbb{R}^N$).

Assume that we have fitted an L_2-regularized linear regression model and obtained the optimal weight vector $\boldsymbol{w}^* \in \mathbb{R}^D$ as

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \frac{1}{2} \sum_{i=1}^{N} (\boldsymbol{w}^T \boldsymbol{x}_i - y_i)^2 + \frac{\lambda}{2} \boldsymbol{w}^T \boldsymbol{w}$$

Note that there is no bias term.

Now, assume that we obtained a new data matrix $\boldsymbol{X}_{new}$ by scaling all samples by the same positive factor $a \in (0, \infty)$. That is, $\boldsymbol{X}_{new} = a\boldsymbol{X}$ (and respectively $\boldsymbol{x}_i^{new} = a\boldsymbol{x}_i$).

a) Find the weight vector $\boldsymbol{w}_{new}$ that will produce the same predictions on $\boldsymbol{X}_{new}$ as $\boldsymbol{w}^*$ produces on $\boldsymbol{X}$.

b) Find the regularization factor $\lambda_{new} \in \mathbb{R}$, such that the solution $\boldsymbol{w}_{new}^*$ of the new L_2-regularized linear regression problem

$$\boldsymbol{w}_{new}^* = \arg\min_{\boldsymbol{w}} \frac{1}{2} \sum_{i=1}^{N} (\boldsymbol{w}^T \boldsymbol{X}_i^{new} - y_i)^2 + \frac{\lambda_{new}}{2} \boldsymbol{w}^T \boldsymbol{w}$$

will produce the same predictions on $\boldsymbol{X}_{new}$ as $\boldsymbol{w}^*$ produces on $\boldsymbol{X}$.

Provide a mathematical justification for your answer.

I want a w_{new} that produces the same predictions on x_{new} as w^*:

$$f_{w^*}(x) = f_{w_{new}}(x)$$

$$w_0^* + w_1^* x_1 + \dots + w_D x_D = w_{0\,new} + w_{1new} a x_1 + \dots + w_{D\,new} a x_{D\,new}$$

$$w_0 + w^{*T} x = w_{new}^T \cdot a \cdot x$$

$$w_{new}^* = \frac{1}{a} \cdot w^*$$

(b)

$$w^*_{new} = \underset{w}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} \left(w^T x_i^{new} - y_i \right)^2 + \frac{\lambda_{new}}{2} w^T w$$

From the Problem 3 we know that the closed form solution for ridge regression is:

$$w^*_{new} = \left(X^T X + \lambda_{new} I \right)^{-1} X^T y$$

We know that $w^*_{new} = \frac{1}{a} w^*$

$$\frac{1}{a} w^* = \left(X^T X + \lambda_{new} I \right)^{-1} X^T y$$

$$\frac{1}{a} \left(X^T X + \lambda_{new} I \right) w^* = X^T y$$

$$\frac{1}{a} X^T w^* X + \frac{1}{a} \lambda_{new} w^* = X^T y$$

$$\frac{1}{a} \lambda_{new} w^* = X^T y - \frac{1}{a} X^T w^* X$$

$$\lambda_{new} w^* = a X^T y - X^T w^* X$$

exercise_04_notebook

November 10, 2019

1 Programming assignment 2: Linear regression

```
In [10]: 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 Exporting the results to PDF

Once you complete the assignments, export the entire notebook as PDF and attach it to your homework solutions. The best way of doing that is 1. Run all the cells of the notebook. 2. Export/download the notebook as PDF (File -> Download as -> PDF via LaTeX (.pdf)). 3. Concatenate your solutions for other tasks with the output of Step 2. On a Linux machine you can simply use `pdfunite`, there are similar tools for other platforms too. You can only upload a single PDF file to Moodle.

Make sure you are using `nbconvert` Version 5.5 or later by running `jupyter nbconvert --version`. Older versions clip lines that exceed page width, which makes your code harder to grade.

1.3 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 [11]: 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)
```

1.4 Task 1: Fit standard linear regression

```
In [12]: def fit_least_squares(X, y):
             """Fit ordinary least squares model to the data.
             D is the number of feature in the augmented dataset
             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
             XT = X.T
             w = (np.linalg.inv(XT.dot(X))).dot(XT.dot(y))
             return w
```

1.5 Task 2: Fit ridge regression

```
In [13]: 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
    XT = X.T
    w = (np.linalg.inv(np.dot(XT, X) + (reg_strength)*np.identity(len(XT))))
            .dot(XT.dot(y))
    return w
```

1.6 Task 3: Generate predictions for new data

```
In [14]: 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 = [w.dot(x) for x in X ]
    return y_pred
```

1.7 Task 4: Mean squared error

```
In [15]: 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.square(np.subtract(y_true, y_pred)).mean()
    return mse
```

1.8 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 [16]:
```
# 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 = 23.964571384952194
MSE for Ridge regression = 21.034931215919112
```