4) $E_{weighted} = 1/2 \sum_{i=1}^{N} t_i \left(w^T \phi(x_i) - y_i\right)^2 = 1/2 \sum_{i=1}^{N} t_i\left(\left(w^T \phi_i(x_i)\right)^2 - 2 w^T \phi_i(x_i) y_i - y_i^2 \right)$

4 $\cancel{\frac{1}{2} \sum t_i}$

$$T = \begin{pmatrix} t_{11} & 0 & \cdots & 0 \\ 0 & t_{22} & & \vdots \\ \vdots & & \ddots & \\ 0 & \cdots & & t_{NN} \end{pmatrix}$$

$$\nabla_w \left(\frac{1}{2} (\Phi w - y)^T t (\Phi w - y) \right) \quad , \text{with } T \to \mathbb{R}^{N \times N}$$

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

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

By parts :

$$① = \nabla_w \left(w^T \underbrace{\Phi^T \cdot t \Phi}_{Z} \cdot w \right) \qquad Z = \Phi^T \cdot t \cdot \Phi$$

$$= \nabla_w (w^T \cdot Z \cdot w)$$

$$= (Z + Z^T) \cdot w$$

$$= \left(\Phi^T \cdot t \Phi + (\Phi^T t \cdot \Phi)^T \right) \cdot w$$

$$= \left(\Phi^T \cdot t \cdot \Phi + \Phi^T \cdot t^T \cdot \Phi^T \right) \cdot w$$

Since t^T is a diagonal matrix $t^T = t$

$$= 2 \cdot (\Phi^T \cdot t \cdot \Phi) \cdot w$$

$$② \ \nabla_w \left(w^T \underbrace{\Phi^T \cdot t \cdot y}_{Z} \right) \qquad Z = \Phi^T \cdot t \cdot y$$

$$= \Phi^T \cdot t \cdot y$$

$$③ \ \nabla_w (y^T \cdot t \cdot \Phi \cdot w) \qquad ④ \ \nabla_w (y^T t \, y) = \underline{0}$$

$$= \nabla_w \left((\Phi^T \cdot t^T \cdot y)^T \cdot w \right)$$

$$= \Phi^T \cdot t^T \cdot y$$

$$= \Phi^T \cdot t \cdot y$$

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

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

$$\nabla_w E_{weighted} = 0 \leftrightarrow \left(\Phi^T \cdot t \cdot \Phi \right) \cdot w = \Phi \cdot t \cdot y$$

$$\leftrightarrow w^* = (\Phi^T \cdot t \cdot \Phi)^{-1} \cdot \Phi \cdot t \cdot y$$

1) If we attribute weights to each datapoint our model gains flexibility. For example, we can assign less weight to outliers that follow no perceivable pattern, thus minimizing the noise in our data.

2) Repeated values bring no value to the data and thus can be ignored in the learning process by assigning them a 0 weighting factor.

5 $\hat{\Phi} = \begin{pmatrix} \Phi \\ \sqrt{\lambda} I_M \end{pmatrix}$, $\hat{y} = \begin{pmatrix} y \\ 0_M \end{pmatrix}$, $E_{LS}(w) = 1/2 \sum (w^t \phi(x_i) - y_i)^2$

$$\Rightarrow \sum_{i=1}^{m} (w^t \phi_i(x_i) - y_i)^2 = [w^t \Phi - y]^t [w^t \Phi - y]$$

$$\text{ridge} = [w^t \hat{\Phi} - \hat{y}]^t [w^t \hat{\Phi} - \hat{y}]$$

$$\hat{\Phi} = [\Phi \mid \sqrt{\lambda} I] \qquad \hat{y} = \begin{pmatrix} y \\ 0 \end{pmatrix}$$

$$\Rightarrow w^t \hat{\Phi} - \hat{y} = w^t [\Phi \mid \sqrt{\lambda} I] - \hat{y}$$

$$\Rightarrow w [\Phi \mid \sqrt{\lambda} I] - \hat{y} = \begin{bmatrix} w\Phi_1^t \\ \vdots \\ w\Phi_m^t \\ w_1\sqrt{\lambda} \\ \vdots \\ w_m\sqrt{\lambda} \end{bmatrix} - \underbrace{\begin{bmatrix} y_1 \\ \vdots \\ y_m \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{2m} = \underbrace{\begin{bmatrix} w\Phi_1^t - y_1 \\ \vdots \\ w\Phi_m^t - y_m \\ w_1\sqrt{\lambda} \\ \vdots \\ w_m\sqrt{\lambda} \end{bmatrix}}_{2m}$$

$$\Rightarrow 1/2 \left([w[\Phi \mid \sqrt{\lambda} I] - \hat{y}]^t [w[\Phi \mid \sqrt{\lambda} I] - \hat{y}] \right) = 1/2 \left(\sum_{i=1}^{m} (w^t \phi_i(x_i) - y_i)^2 + \sum_{i=1}^{m} (w_i \sqrt{\lambda})^2 \right)$$

$$= 1/2 \sum_{i=1}^{m} (w^t \phi_i(x_i) - y_i)^2 + \frac{\lambda}{2} \|w\|_2^2 \quad //$$

6

John is allowing his polynomial regression model to go up to degree 50, thus losing all generality associated with allowing some error % to occur when using lower degrees.

By trying to minimize the error mindlessly, he happens to be overfitting his model to the training data and is learning a function that has a lot of variance in order to match all data points to the predicted values.

This could be solved by splitting the data into a training and validation set. Such that the minimization of the error function would occur in unseen data and actually be learning a model that encapsulates the behaviour of his data. Also, we could use K-fold cross validation and learn what degree has the least error on average regarding all folds and thus create the most generalizable model.

(7) likelihood $\Rightarrow p(y \mid \Phi, w, \beta) = \prod_{i=1}^{N} \mathcal{N}(y_i \mid w^t \phi(x_i), \beta^{-1})$

conjugate prior $\Rightarrow p(w, \beta) = \mathcal{N}(w \mid m_0, \beta^{-1} S_0)\ \mathrm{Gamma}(\beta \mid a_0, b_0)$

- $\frac{1}{\sigma\sqrt{2\pi}} e^{-1/2\left(\frac{x-\mu}{\sigma}\right)^2}$

$$\mathcal{N}(y_i \mid w^t \phi x_i, \beta^{-1}) \Rightarrow \prod_{i=1}^{N} \frac{\beta}{\sqrt{2\pi}} e^{-1/2\left(y_i - w^t \phi_i(x_i)\right)\beta)^2}, \quad \propto e^{-1/2\left[\sum_{i=1}^{N}\left((y_i - w^t \phi_i)\beta\right)^2\right]}$$

$$\mathcal{N}(w \mid m_0, \beta^{-1} S_0) \Rightarrow \frac{\beta}{S_0\sqrt{2\pi}} e^{-1/2\left(\frac{(w-m_0)\beta}{S_0}\right)^2}$$

$$\mathrm{Gamma}(\beta \mid a_0, b_0) \Rightarrow \frac{\Gamma(a_0+b_0)}{\Gamma(a_0)\Gamma(b_0)} \beta^{a_0-1} (1-\beta)^{b_0-1}$$

$$\rightarrow p(w, \beta \mid D) = \prod_{i=1}^{N} \mathcal{N}(y_i \mid w^t \phi(x_i), \beta^{-1}) \times \mathcal{N}(w \mid m_0, \beta^{-1} S_0)\ \mathrm{Gamma}(\beta \mid a_0, b_0)$$

$$= \left(\frac{\beta}{\sqrt{2\pi}}\right)^N e^{-1/2\left[\sum_{i=1}^{N}\left((y_i - w^t \phi_i)\beta\right)^2\right]} \left(\frac{\beta}{S_0\sqrt{2\pi}}\right) e^{-1/2\left(\frac{(w-m_0)\beta}{S_0}\right)^2} \frac{\Gamma(a_0+b_0)}{\Gamma(a_0)\Gamma(b_0)} \beta^{a_0-1} (1-\beta)^{b_0-1}$$

$$= e^{-\frac{\beta^2}{2}\left(\sum_{i=1}^{N}(y_i - w^t\phi_i(y_i))^2 + \left(\frac{w-m_0}{S_0}\right)^2\right)} \beta^{a_0-1} (1-\beta)^{b_0-1} \cdot \frac{1}{S_0}\left(\frac{\beta}{\sqrt{2\pi}}\right)^{N+1} \frac{\Gamma(a_0+b_0)}{\Gamma(a_0)\Gamma(b_0)}$$

$$= e^{-\frac{\beta^2}{2}\left(y^t y - 2y w^t \Phi - w^t \Phi \Phi w + \left(\frac{w-m_0}{S_0}\right)^2\right)} \beta^{a_0-1} (1-\beta)^{b_0-1} \frac{1}{S_0}\left(\frac{\beta}{\sqrt{2\pi}}\right)^{N+1} \frac{\Gamma(a_0+b_0)}{\Gamma(a_0)\Gamma(b_0)}$$

$$= e^{-\frac{\beta^2}{2}\left(y^t y - 2y w^t \Phi - w^t \Phi \Phi w + \left(\frac{w-m_0}{S_0}\right)^2\right)} \beta^{a_0+N} (1-\beta)^{b_0-1} \frac{1}{S_0}\left(\frac{1}{\sqrt{2\pi}}\right)^{N+1} \frac{\Gamma(a_0+b_0)}{\Gamma(a_0)\Gamma(b_0)}$$

$$\xrightarrow{\ln} -\frac{\beta^2}{2}\left(y^t y - 2y w^t \Phi - w^t \Phi \Phi w + \left(\frac{w-m_0}{S_0}\right)^2\right) + (a_0+N)\ln\beta + (b_0-1)\ln(1-\beta) - \ln\left(S_0 \cdot (\sqrt{2\pi})^{N+1}\right) + \ln\left(\frac{\Gamma(a_0+b_0)}{\Gamma(a_0)\Gamma(b_0)}\right)$$

- $\mathcal{N}(w \mid m_N, \beta^{-1} S_N)\ \mathrm{Gamma}(\beta \mid a_N, b_N) = \frac{\beta}{S_N\sqrt{2\pi}} e^{-1/2\left(\frac{(w-m_N)\beta}{S_N}\right)^2} \cdot \frac{\Gamma(a_N, b_N)}{\Gamma(a_N)\Gamma(b_N)} \beta^{a_N-1} (1-\beta)^{b_N-1}$

$$\xRightarrow{\ln} \ln\left(\frac{\beta}{S_N\sqrt{2\pi}}\right) + -1/2\left(\frac{(w-m_N)\beta}{S_N}\right)^2 + \ln\left(\frac{\Gamma(a_N+b_N)}{\Gamma(a_N)\Gamma(b_N)}\right) + (a_N-1)\ln(\beta) + (b_N-)\ln(1-\beta) =$$

$$= -\ln\left(S_N\sqrt{2\pi}\right) - 1/2\left(\frac{(w-m_N)\beta}{S_N}\right)^2 + \ln\left(\frac{\Gamma(a_N+b_N)}{\Gamma(a_N)\Gamma(b_N)}\right) + (a_N)\ln(\beta) + (b_{N-1})\ln(1-\beta)$$

- $-\frac{\beta}{2}^2\left(y^t y - 2yw^t\Phi - w^t\Phi\Phi w + \left(\frac{w-m_0}{S_0}\right)^2\right) + (a_0+N)\ln\beta + (b_0-1)\ln(1-\beta) - \ln\left(S_0 \cdot (\sqrt{2\pi})^{N+1}\right) + \ln\left(\frac{\Gamma(a_0+b_0)}{\Gamma(a_0)\Gamma(b_0)}\right)$

- $-\ln\left(S_N\sqrt{2\pi}\right) - 1/2\left(\frac{(w-m_N)\beta}{S_N}\right)^2 + \ln\left(\frac{\Gamma(a_N+b_N)}{\Gamma(a_N)\Gamma(b_N)}\right) + (a_N)\ln(\beta) + (b_{N-1})\ln(1-\beta)$

$$\Rightarrow \begin{cases} S_N = S_0(\sqrt{2\pi})^N \\ m_N = * \text{ (found on the next page)} \\ a_N = a_0 + N \\ b_N = b_0 \end{cases}$$

8 we have $E_{ridge}(w) = \frac{1}{2}\sum_{i=1}^{N}(w^t\phi(x_i) - y_i)^2 + \frac{\lambda}{2}w^t w$

from lecture notes

$$\Rightarrow \nabla_w E_{ridge} = \nabla_w LS + \nabla_w(\|w\|_2^2) = (X^t X w - X^t y) + 2w$$

- $\|w\|_2^2 = \left(\left(\sum w_i^2\right)^{1/2}\right)^2 = \sum w_i^2$

$$\Rightarrow \frac{\partial}{\partial w_k}|w|_2^2 = \frac{\partial}{\partial w_k}\sum_{i=1}^{N} w_i^2 = \sum_{i=1}^{N}\frac{\partial}{\partial w_k}w_i^2 = 2w_i$$

$$\Rightarrow \nabla_w(|w|_2^2) = 2w$$

assuming $X^t X + \lambda I \neq 0$ $\Rightarrow$ det $\neq 0$

max $\Rightarrow X^t X w - X^t y + \lambda w = 0 \iff (X^t X + \lambda I)w = X^t y \iff w = \frac{X^t y}{X^t X + \lambda I} \iff w = X^t y (X^t X + \lambda I)^{-1}$

Since the amount of basis functions behaves like the degrees of freedom in the standard polynomial regression. If we were to have less data than basis functions, we would be encountering some form of overfitting in our models which is observed by the growing norm of the weight vector.

With regularization, since the norm is taken into account in the minimization portion of the training, it is expected that the level of overfitting should be drastically reduced.

$$\sum_{i=1}^{N}\left(y_i - w^t \phi_i(x_i)\right)^2 + \left(\frac{w - m_0}{2}\right)^2 = \left(\frac{w - m_N}{s_N}\right)^2 = \left(\frac{w - m_N}{s_0 \left(\sqrt{2\pi}\right)^N}\right)^2 \Leftrightarrow$$

$$\Leftrightarrow \left(\sum_{i=1}^{N}\left(y_i - w^t \phi_i(x_i)\right)^2 + \left(\frac{w - m_0}{2}\right)^2\right) s_0^2 (2\pi)^N = (w - m_N)^2 \Leftrightarrow$$

$$\Leftrightarrow m_N = w - \left[s_0^2 2\pi^N \left(\sum_{i=1}^{N}\left(y_i - w^t \phi_i(x_i)\right)^2 + \left(\frac{w - m_0}{s_0}\right)^2\right)\right]^{1/2}$$

$\Rightarrow$ note that this is likely wrong since m_N shouldn't depend on w

9

a) prediction is $f(X) = w^t X$

$\rightarrow$ find (w_{new}) such that $w_{new}^t X_{new} = w^t X \Leftrightarrow$

$\Leftrightarrow w_{new}^t a X = w^t X \Leftrightarrow$

$\Leftrightarrow w_{new}^t = \frac{1}{a} w^t \Leftrightarrow w_{new} = 1/a \, w$

b) from problem 8 we have that $w = X^t y (X^t X + \lambda I)^{-1}$ in the original ridge regression problem

$\Rightarrow X_{new}^t y (X_{new}^t X_{new} + \lambda_{new} I)^{-1} = X^t y (X^t X + \lambda I)^{-1} \Leftrightarrow$

$\Leftrightarrow a \cancel{X^t y} (a^2 X^t X + \lambda_{new} I)^{-1} = \cancel{X^t y} (X^t X + \lambda I)^{-1} \Leftrightarrow$

$\Leftrightarrow a (a^2 X^t X + \lambda_{new} I)^{-1} = (X^t X + \lambda I)^{-1} \Leftrightarrow$

$\Leftrightarrow a (X^t X + \lambda I) = a^2 X^t X + \lambda_{new} I \Leftrightarrow$

$\Leftrightarrow \lambda_{new} = (a - a^2) X^t X + \lambda I$

code

November 16, 2021

1 Programming Task: Linear Regression

```
[ ]: import numpy as np

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

1.1 Your task

This notebook provides a code skeleton for performing linear regression. 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.

In the beginning of every function there is docstring which specifies the input and and expected output. Write your code in a way that adheres to it. You may only use plain python and anything that we imported for you above such as numpy functions (i.e. no scikit-learn classifiers).

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 (`Kernel -> Restart & Run All`) 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 Linux 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

In 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

```
[ ]: 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

[ ]:
```
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).

    """

    return np.linalg.inv(X.transpose().dot(X)).dot(X.transpose()).dot(y)
```

1.5 Task 2: Fit ridge regression

[ ]:
```
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).

    """

    num_cols, num_rows = np.shape(X)

    return X.transpose().dot(y).dot(np.linalg.inv(X.transpose().
 ↪dot(X)+reg_strength*np.identity(num_rows)))
```

1.6 Task 3: Generate predictions for new data

[ ]:
```
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)
```

1.7 Task 4: Mean squared error

[ ]:
```
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 ((y_pred - y_true)**2).mean(axis=0)
```

1.8 Compare the two models

The reference implementation produces * MSE for Least squares ≈ **23.96** * MSE for Ridge regression ≈ **21.03**

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

[ ]:
```
# 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 = 23.96457138495312
MSE for Ridge regression = 21.034931215917556
```