Machine Learning Homework 04

Authors: Jesus Arturo Sol Navarro

1 Problem 5

Show that ridge regression with a design matrix $\Phi \in R^{N \times M}$ and regularization strength λ is equivalent to ordinary least squares (OLS) regression with an augmented design matrix and target vector:

$$\hat{\Phi} = \begin{pmatrix} \Phi \\ \sqrt{\lambda} I_M \end{pmatrix}, \quad \hat{y} = \begin{pmatrix} y \\ 0_M \end{pmatrix}.$$

The ridge regression objective function is:

$$E(w) = \frac{1}{2}\|\Phi w - y\|^2 + \frac{\lambda}{2}\|w\|^2.$$

For OLS with the augmented design matrix $\hat{\Phi}$ and target vector $\hat{y}$, the loss function becomes:

$$E(w) = \frac{1}{2}\|\hat{\Phi} w - \hat{y}\|^2.$$

Expanding the terms:

$$E(w) = \frac{1}{2}\left[\begin{pmatrix} \Phi w - y \\ \sqrt{\lambda} I w \end{pmatrix}^T \begin{pmatrix} \Phi w - y \\ \sqrt{\lambda} I w \end{pmatrix}\right].$$

$$E(w) = \frac{1}{2}(\Phi w - y)^T(\Phi w - y) + \frac{1}{2}(\sqrt{\lambda} w)^T(\sqrt{\lambda} w).$$

$$E(w) = \frac{1}{2}\|\Phi w - y\|^2 + \frac{\lambda}{2}\|w\|^2.$$

Ridge regression with the original Φ and y is mathematically equivalent to ordinary least squares applied to the augmented matrix $\hat{\Phi}$ and target $\hat{y}$.

2 Problem 6

John Doe is a data scientist, and he wants to t a polynomial regression model to his data. For this, he needs to choose the degree of the polynomial that works best for his problem.

Unfortunately, John has not attended IN2064, so he writes the following code for choosing the optimal degree of the polynomial:

```
 X, y = load_data()

best_error = -1
best_degree = None

for degree in range(1, 50):
    w = fit_polynomial_regression(X, y, degree)
    y_predicted = predict_polynomial_regression(X, w, degree)
    error = compute_mean_squared_error(y, y_predicted)

    if (error <= best_error) or (best_error == -1):
        best_error = error
        best_degree = degree

print("Best degree is " + str(best_degree))
```

Assume that the functions are implemented correctly and do what their name suggests.

a) Explain briefly why this code doesn't do what its supposed to do.

The code evaluates the model's performance using the same dataset for training and testing, which introduces over-fitting. Consequently, higher-degree polynomials, which are more complex, tend to have the lowest error, but this does not reflect true predictive performance.

b) Describe a possible way to fix the problem with this code.

To select the optimal polynomial degree and prevent over-fitting, split the dataset into training and validation sets. Train models of varying polynomial degrees on the training set and evaluate their performance on the validation set using Mean Squared Error. Choose the degree that minimizes validation error .

3 Problem 7

In the lecture, we made the assumption that we already knew the precision (inverse variance) for our Gaussian distributions. What about when we dont know the precision and we need to put a prior on that as well as our Gaussian prior that we already have on the weights of the model?

Task: It turns out that the conjugate prior for the situation where we place priors on both the weights and the precision is given by a Gaussian-Gamma distribution.This means that if our likelihood is as follows:

$$p(\mathbf{y} \mid \mathbf{\Phi}, \mathbf{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}(y_i \mid \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i), \beta^{-1})$$

Then the conjugate prior for both $\mathbf{w}$ and β is:

$$p(\mathbf{w}, \beta) = \mathcal{N}(\mathbf{w} \mid \mathbf{m}_0, \beta^{-1}\mathbf{S}_0)\,\text{Gamma}(\beta \mid a_0, b_0)$$

Show that the posterior distribution takes the same format as the prior, i.e.

$$p(\mathbf{w}, \beta \mid \mathcal{D}) = \mathcal{N}(\mathbf{w} \mid \mathbf{m}_N, \beta^{-1}\mathbf{S}_N)\,\text{Gamma}(\beta \mid a_N, b_N)$$

Also be sure to give the expressions for $\mathbf{m}_N$, $\mathbf{S}_N$, a_N, and b_N.

Solution: Expanding loglikelihood.

$$\begin{aligned}\log p(y \mid \Phi, w, \beta) = \sum_{i=1}^{N} \log \mathcal{N}\left(y_i \mid w^T\phi(x_i), \beta^{-1}\right) &= -\frac{N}{2}\log(2\pi) + \frac{N}{2}\log\beta - \frac{\beta}{2}\sum_{i=1}^{N}(w^T\phi(x_i) - y_i)^2 \\ &= \frac{N}{2}\log\beta - \frac{\beta}{2}y^Ty + \beta w^T\Phi^T y - \frac{\beta}{2}w^T\Phi^T\Phi w + const.\end{aligned}$$

Expanding logprior .

$$\begin{aligned}\log p(w, \beta) &= \log\left[\left(\frac{\beta}{2\pi}\right)^{\frac{M}{2}} \exp\left(-\frac{\beta}{2}(\mathbf{m}_0 - w)^T\mathbf{S}_0^{-1}(\mathbf{m}_0 - w)\right)\right] + log\left[\left(\frac{b^a}{\Gamma(a)}\beta^{a-1}e^{-b\beta}\right)\right] \\ &= \frac{M}{2}\log\beta - \frac{\beta}{2}\mathbf{m}_0^T\mathbf{S}_0^{-1}\mathbf{m}_0 + \beta\mathbf{m}_0^T\mathbf{S}_0^{-1}w - \frac{\beta}{2}w^T\mathbf{S}_0^{-1}w + (a_0 - 1)\log\beta - b_0\beta + const.\end{aligned}$$

Expanding logposterior.

$$\begin{aligned}\log \mathrm{p}(\mathbf{w}, \beta \mid \mathcal{D}) &\propto \log p(y \mid \Phi, w, \beta) + \log p(w, \beta) \\ \propto \left(\frac{M}{2} + \frac{N}{2} + (a_0 - 1)\right)\log\beta - \frac{\beta}{2}w^T\left(\phi^T\phi + \mathbf{S}_0^{-1}\right)w + \beta\left(y^T\Phi + m_0^T\mathbf{S}_0^{-1}\right)w - \beta\left(b_0 + \frac{1}{2}y^Ty + \frac{1}{2}m_0^T\mathbf{S}_0^{-1}m_0\right)\end{aligned}$$

Expanding presumed logposterior.

$$\begin{aligned}\log \mathrm{p}(\mathbf{w}, \beta \mid \mathcal{D}) &\propto \log p(y \mid \Phi, w, \beta) + \log p(w, \beta) \\ \propto \left(\frac{M}{2} + (a_N - 1)\right)\log\beta - \frac{\beta}{2}w^T\left(\mathbf{S}_N^{-1}\right)w + \beta\left(m_N^T\mathbf{S}_N^{-1}\right)w - \beta\left(b_N + \frac{1}{2}m_N^T\mathbf{S}_N^{-1}m_N\right)\end{aligned}$$

Matching terms.

$$a_N = a_0 + \frac{N}{2}$$

$$b_N = b_0 + \frac{1}{2}\left(y^\top y + m_0^\top S_0^{-1} m_0 - m_N^\top S_N^{-1} m_N\right)$$

$$S_N^{-1} = S_0^{-1} + \Phi^\top \Phi$$

$$m_N = S_N\left(S_0^{-1} m_0 + \Phi^\top y\right)$$

4 Problem 8

Derive the closed form solution for ridge regression error function:

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

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?

Expanding ridge regression error function.

$$E_{\text{ridge}}(w) = \frac{1}{2}(\Phi w - y)^2 + \lambda\|w\|^2 = \frac{1}{2}\left[w^\top\Phi^\top\Phi w - 2y^\top\Phi w + y^\top y + \lambda w^\top w\right]$$

Differentiate w.r.t w.

$$\frac{\partial E_{\text{ridge}}}{\partial w} = \Phi^\top\Phi w - y^\top\Phi + \lambda w$$

Solving for w.

$$w^* = (\Phi^\top\Phi + \lambda I)^{-1}\Phi^\top y$$

When $N < M$, the matrix $\Phi^T\Phi$ becomes rank-deficient and therefore not invertible. This leads to failure in finding a solution. Regularization, with $\lambda > 0$, addresses this problem by ensuring that the matrix $\Phi^T\Phi + \lambda I$ is positive definite and invertible, even when $\Phi^T\Phi$ is rank-deficient.

5 Problem 9

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

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

$$w^* = \arg\min_w \frac{1}{2}\sum_{i=1}^{N}\left(w^T x_i - y_i\right)^2 + \frac{\lambda}{2}w^T w$$

Note that there is no bias term.

Now, assume that we obtained a new data matrix X_{new} by scaling all samples by the same positive factor $a \in (0, \infty)$. That is, $X_{\text{new}} = aX$ (and respectively $x_i^{\text{new}} = ax_i$).

(a) Find the weight vector w_{new} that will produce the same predictions on X_{new} as w^* produces on X.

To guarantee same predictions $a\mathbf{X}\mathbf{w}_{\text{new}} = \mathbf{X}\mathbf{w}^*$ and then solving for $w_{\text{new}} = \frac{1}{a}\mathbf{w}^*$

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

$$w^*_{\text{new}} = \arg\min_w \frac{1}{2}\sum_{i=1}^{N}\left(w^T x_i^{\text{new}} - y_i\right)^2 + \frac{\lambda_{\text{new}}}{2}w^T w$$

From $\mathbf{w}^*_{\text{new}} = \left(a^2\mathbf{X}^\top\mathbf{X} + \lambda_{\text{new}}\mathbf{I}\right)^{-1}\left(a\mathbf{X}^\top\mathbf{y}\right) = \frac{1}{a}\mathbf{w}^* = \frac{1}{a}\left(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^\top\mathbf{y}$, it follows:

$$\frac{a}{a^2}\left(\mathbf{X}^\top\mathbf{X} + \frac{1}{a^2}\lambda_{\text{new}}\mathbf{I}\right)^{-1}\left(\mathbf{X}^\top\mathbf{y}\right) = \frac{1}{a}\left(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^\top\mathbf{y}$$

$$\left(\mathbf{X}^\top\mathbf{X} + \frac{1}{a^2}\lambda_{\text{new}}\mathbf{I}\right)^{-1} = \left(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}\right)^{-1}$$

$$\frac{1}{a^2}\lambda_{\text{new}}\mathbf{I} = \lambda\mathbf{I}$$

$$\lambda_{\text{new}} = a^2\lambda$$

6 Problem 10

6.1 Load and preprocess the data

```
X , y = fetch_california_housing(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 = 0.9
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)
```

6.2 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).

"""
### YOUR CODE HERE ###

# Compute X^T X
XTX = np.dot(X.T, X)

# Compute inverse of X^T X
XTX_inv = np.linalg.inv(XTX)
# Compute X^T y
XTy = np.dot(X.T, y)

# Compute optimal weights: w = (X^T X)^(-1) X^T y
w = np.dot(XTX_inv, XTy)

return w
```

6.3 Fit ridge regression

```python
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).

"""
### YOUR CODE HERE ###

# Get number of features (including bias term)
D = X.shape[1]

# Compute X^T X
XTX = np.dot(X.T, X)

# Create identity matrix for regularization
I = np.eye(D)

# Create regularization matrix with zero for bias term
# This prevents regularizing the bias term
I[0, 0] = 0

# Compute (X^T X +  I )
reg_matrix = XTX + reg_strength * I

# Compute inverse of regularized matrix
reg_matrix_inv = np.linalg.inv(reg_matrix)

# Compute X^T y
XTy = np.dot(X.T, y)

# Compute optimal weights: w = (X^T X +  I )^(-1) X^T y
w = np.dot(reg_matrix_inv, XTy)

return w
```

6.4 Generate predictions for new data

```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.

"""
### YOUR CODE HERE ###

# Compute predictions: y_pred = X * w
y_pred = np.dot(X, w)

return y_pred
```

6.5 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.

"""
### YOUR CODE HERE ###

# Calculate differences between true and predicted values
errors = y_true - y_pred

# Square the errors
squared_errors = errors ** 2

# Calculate mean of squared errors
mse = np.mean(squared_errors)

return mse
```

6.6 Compare the two models

```
# Load the data
np.random.seed(1234)
X , y = fetch_california_housing(return_X_y=True)
X = np.hstack([np.ones([X.shape[0], 1]), X])
test_size = 0.9
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 = 0.534710242602781
MSE for Ridge regression = 0.5341685503394193