EE2211 Tutorial 6

Dr Feng LIN

Q1

(Ridge Regression in Dual Form)

Derive the solution for linear ridge regression in dual form (see Lecture 6 notes page 15).

Ridge Regression in Dual Form (when $m < d$)

$(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})$ is invertible for λ > 0,

Learning: $\hat{\mathbf{w}} = \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})^{-1}\,\mathbf{y}$

Prediction: $\hat{\boldsymbol{f}}_{\mathbf{w}}\,(\mathbf{X}new) = \mathbf{X}new\;\hat{\mathbf{w}}$

Hint: start off with $(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})\mathbf{w} = \mathbf{X}^T\mathbf{y}$ and make use of $\mathbf{w} = \mathbf{X}^T\boldsymbol{a}$ and $\boldsymbol{a} = \lambda^{-1}\,(\mathbf{y} - \mathbf{X}\mathbf{w}), \lambda > 0$

Q1

$$(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})\mathbf{w} = \mathbf{X}^T\mathbf{y}$$

where:

- $\boldsymbol{X}$ is the data matrix (with rows as samples and columns as features).
- $\boldsymbol{y}$ is the vector of target values.
- $\boldsymbol{w}$ is the vector of weights we want to solve for.
- $\lambda > 0$ is the regularization parameter.
- $\boldsymbol{I}$ is the identity matrix.

Step 1: Rearranging and Simplifying the Equation

Expend left-hand side $\Rightarrow \mathbf{X}^T\mathbf{X}\mathbf{w} + \lambda\mathbf{w} = \mathbf{X}^T\mathbf{y}$

Isolatate $\lambda\boldsymbol{w}$ term $\Rightarrow \lambda\mathbf{w} = \mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\mathbf{w}$

$$\Rightarrow \mathbf{w} = \lambda^{-1}(\mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\mathbf{w})$$

Factor out λ and rewrite as $\Rightarrow \mathbf{w} = \lambda^{-1}\,\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w})$

Step 2: Introducing Dual Variable

$$\mathbf{w} = \mathbf{X}^T\boldsymbol{a}$$

where

$$\boldsymbol{a} = \lambda^{-1}(\mathbf{y} - \mathbf{X}\mathbf{w})$$

The key idea is to express the weight vector $\mathbf{w}$ in terms of $\boldsymbol{a}$.

Q1

Step 3: Simplifying the Dual Form

Now we substitute $\boldsymbol{w} = X^T\boldsymbol{a}$ into the expression for $\boldsymbol{a}$:

$$\boldsymbol{a} = \lambda^{-1}(\mathbf{y} - \mathbf{X}\mathbf{w})$$

Multiplying both sides by λ

$$\Rightarrow \lambda\boldsymbol{a} = (\mathbf{y} - \mathbf{X}\mathbf{w})$$

$$\Rightarrow \lambda\boldsymbol{a} = (\mathbf{y} - \mathbf{X}\mathbf{X}^T\boldsymbol{a})$$

Rearranging

$$\Rightarrow \mathbf{X}\mathbf{X}^T\boldsymbol{a} + \lambda\boldsymbol{a} = \mathbf{y}$$

$$\Rightarrow (\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})\boldsymbol{a} = \mathbf{y}$$

Solving for $\boldsymbol{a}$

$$\Rightarrow \boldsymbol{a} = (\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})^{-1}\mathbf{y}$$

Step 4: Solving for $\mathbf{w}$

using $\mathbf{w} = X^T\boldsymbol{a}$ again, we get the solution for $\mathbf{w}$

$$\mathbf{w} = \mathbf{X}^T\boldsymbol{a} = \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})^{-1}y$$

Q2

(Polynomial Regression, 1D data)

Given the following data pairs for training:

$$\{x = -10\} \rightarrow \{y = 5\}$$
$$\{x = -8\} \rightarrow \{y = 5\}$$
$$\{x = -3\} \rightarrow \{y = 4\}$$
$$\{x = -1\} \rightarrow \{y = 3\}$$
$$\{x = 2\} \rightarrow \{y = 2\}$$
$$\{x = 8\} \rightarrow \{y = 2\}$$

(a) Perform a 3rd-order polynomial regression and sketch the result of line fitting.

(b) Given a test point $\{ x=9 \}$ predict y using the polynomial model.

(c) Compare this prediction with that of a linear regression.

Q2 (a) Perform a 3rd-order polynomial regression

Step 1: Data Preparation: Collect the given data points:

$$\{(x = -10, y = 5), (x = -8, y = 5), (x = -3, y = 4), (x = -1, y = 3), (x = 2, y = 2), (x = 8, y = 2)\}$$

Step 2: Polynomial Regression:

- We'll use a 3rd-order polynomial, meaning the model will have the form:

$$f(\mathbf{x}) = w_0 + w_1 x + w_2 x^2 + w_3 x^3$$

- Use the given data to fit this model by solving for the coefficients:

$$\mathbf{P} = \begin{bmatrix} 1 & -10 & 100 & -1000 \\ 1 & -8 & 64 & -512 \\ 1 & -3 & 9 & -27 \\ 1 & -1 & 1 & -1 \\ 1 & 2 & 4 & 8 \\ 1 & 8 & 64 & 512 \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} 5 \\ 5 \\ 4 \\ 3 \\ 2 \\ 2 \end{bmatrix}.$$

- Polynomial regression results:

$$\hat{\mathbf{w}} = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T\mathbf{y}$$

$$= \begin{bmatrix} 6 & -12 & 242 & -1020 \\ -12 & 242 & -1020 & 18290 \\ 242 & -1020 & 18290 & -100212 \\ -1020 & 18290 & -100212 & 1525082 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ -10 & -8 & -3 & -1 & 2 & 8 \\ 100 & 64 & 9 & 1 & 4 & 64 \\ -1000 & -512 & -27 & -1 & 8 & 512 \end{bmatrix} \begin{bmatrix} 5 \\ 5 \\ 4 \\ 3 \\ 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 2.6894 \\ -0.3772 \\ 0.0134 \\ 0.0029 \end{bmatrix}$$

Q2(a) Perform a 3rd-order polynomial regression

```
# Given data
x = np.array([-10, -8, -3, -1, 2, 8]).reshape(-1, 1)
y = np.array([5, 5, 4, 3, 2, 2])

# (a) 3rd-order polynomial regression
poly_features = PolynomialFeatures(degree=3)
x_poly = poly_features.fit_transform(x)

# Method I: using primal form
P = x_poly
w = inv(P.T @ P) @ P.T @ y
print('w')
print(w)

Xt = np.array([[9]])
Pt = poly_features.fit_transform(Xt)
y_predict = Pt @ w
print('y_predict')
print(y_predict)

# Method II
# Fit the polynomial regression model
poly_model = LinearRegression()
poly_model.fit(x_poly, y)
print('ploy_model')
print(poly_model.coef_)
print(poly_model.intercept_)

# Predictions for the polynomial model
x_new = np.linspace(-11, 10, 100).reshape(-1, 1)
x_new_poly = poly_features.transform(x_new)
y_poly_pred = poly_model.predict(x_new_poly)
```

- PolynomialFeatures(): generate a new feature matrix consisting of all polynomial combinations of the features
- fit_transform(): Compute and return the polynomial features for the input data X

Q2(a) Perform a 3rd-order polynomial regression

class sklearn.preprocessing.PolynomialFeatures(degree=2, *, interaction_only=False, include_bias=True, order='C')

- Generate a new feature matrix consisting of all polynomial combinations of the features with degree less than or equal to the specified degree

Simple example. If your input feature vector X is $[x_1, x_2]$ and you choose degree 2, PolynomialFeatures will create the following new features:

Degree 1 Features (original features):

- x_1
- x_2

Degree 2 Features (squares of individual features):

- x_1^2
- x_2^2

Interaction Term (products of different features):

- $x_1 \times x_2$

So, the transformed feature set becomes:

$$[1, x_1, x_2, x_1^2, x_1 \times x_2, x_2^2]$$

Q2(a) Perform a 3rd-order polynomial regression

Method: PolynomialFeatures.fit_transform()

This method computes and returns the polynomial features for the input data X.

- fit_transform() takes in input features X (an array or matrix) and then computes the polynomial and interaction terms according to the degree specified.
- It returns the transformed data matrix where each row contains the polynomial terms for the corresponding input sample.

```
from sklearn.preprocessing import PolynomialFeatures

# Example data with two features
X = [[2, 3], [3, 4], [4, 5]]

# Initialize the transformer for degree 2
poly = PolynomialFeatures(degree=2)

# Transform the input data
X_poly = poly.fit_transform(X)

print(X_poly)
```

```
[[ 1.   2.   3.   4.   6.   9.]
 [ 1.   3.   4.   9.  12.  16.]
 [ 1.   4.   5.  16.  20.  25.]]
```

Q2(b) Given a test point $\{x = 9\}$ predict y using the polynomial model

Once the model is fitted, we can plug in $x = 9$ into the 3rd-order polynomial equation to predict the corresponding y.

- Predicted y for $x = 9$ using the 3rd-order polynomial model: 2.466097711361895
- Predicted y for $x = 9$ using the linear regression model: 1.3302752293577975

```
# (b) Predict y for x=9 using the polynomial model
x_test = np.array([[9]])
x_test_poly = poly_features.transform(x_test)
y_pred_poly = poly_model.predict(x_test_poly)

print(f"Predicted y for x=9 using the 3rd-order polynomial model: {y_pred_poly[0]}")

# (c) Linear regression for comparison
linear_model = LinearRegression()
linear_model.fit(x, y)
y_pred_linear = linear_model.predict(np.array([[9]]))

print(f"Predicted y for x=9 using the linear regression model: {y_pred_linear[0]}")
```

Q2 (c) Compare this prediction with that of a linear regression.

For comparison, we also perform linear regression, which assumes a model of the form:

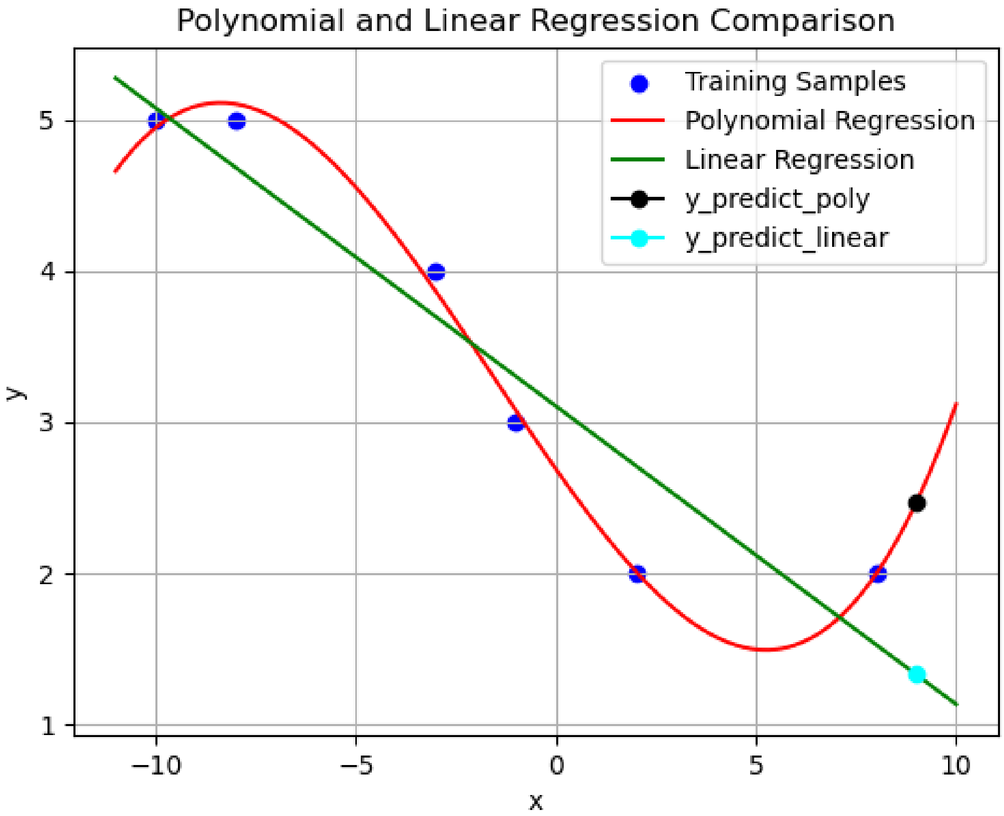
$$f(x) = w_0 + w_1 x.$$

Then, we'll compare the predicted value at $x = 9$ using the linear regression model with that from the polynomial model.

```
# (c) Linear regression for comparison
linear_model = LinearRegression()
linear_model.fit(x, y)
y_pred_linear = linear_model.predict(np.array([[9]]))

print(f"Predicted y for x=9 using the linear regression model: {y_pred_linear[0]}")
```

Q2 (c) Compare this prediction with that of a linear regression.



Q3

(Polynomial Regression, 3D data, Python)

a) Write down the expression for a 3rd order polynomial model having a 3-dimensional input.

b) Write down the $\boldsymbol{P}$ matrix for this polynomial given

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & -1 & 1 \end{bmatrix}.$$

a) Given $\boldsymbol{y} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, can a unique solution be obtained in dual form? If so, proceed to solve it.

b) Given $\boldsymbol{y} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, can the primal ridge regression be applied to obtain a unique solution? If so, proceed to solve it.

Q3 (a) Expression for a 3rd order polynomial model having a 3-dimensional input.

For a 3rd-order polynomial model with 3-dimensional input $x = [x_1, x_2, x_3]$, the expression for the polynomial model would include all terms up to the 3rd degree in x_1, x_2, and x_3. This includes:

- Constant term (degree 0)
- Linear terms (degree 1): x_1, x_2, x_3
- Quadratic terms (degree 2): $x_1x_2, x_2x_3, x_1x_3, x_1^2, x_2^2, x_3^2$
- Cubic terms (degree 3): $x_2x_1^2, x_3x_1^2, x_1x_2^2, x_3x_2^2, x_1x_3^2, x_2x_3^2, x_1x_2x_3, x_1^3, x_2^3, x_3^3$

The general expression for the 3rd-order polynomial in 3 variables can be written as:

$$f(\mathbf{x}) = w_0 + w_1x_1, +w_2x_2 + w_3\ x_3$$

$$+w_{12}x_1x_2 + w_{23}x_2x_3, +w_{13}x_1x_3 + w_{11}x_1^2, +w_{22}x_2^2 + w_{33}x_3^2$$

$$+w_{211}x_2x_1^2 + w_{311}x_3x_1^2, +w_{122}x_1x_2^2, +w_{322}x_3x_2^2 + w_{133}x_1x_3^2 + w_{233}x_2x_3^2 + w_{123}x_1x_2x_3 + w_{111}x_1^3 + w_{222}x_2^3 + w_{333}x_3^3$$

Q3(b) $\boldsymbol{P}$ matrix for this polynomial

$$f(\mathbf{x}) = w_0 + w_1x_1, +w_2x_2 + w_3\,x_3$$
$$+w_{12}x_1x_2 + w_{23}x_2x_3, +w_{13}x_1x_3 + w_{11}x_1^2, +w_{22}x_2^2 + w_{33}x_3^2$$
$$+w_{211}x_2x_1^2 + w_{311}x_3x_1^2, +w_{122}x_1x_2^2, +w_{322}x_3x_2^2 + w_{133}x_1x_3^2 + w_{233}x_2x_3^2 + w_{123}x_1x_2x_3 + w_{111}x_1^3 + w_{222}x_2^3 + w_{333}x_3^3$$

--- Equation (1)

Each row of P represents a vector formed from the corresponding row of X, and each column corresponds to one of the terms from the 3rd-order polynomial expression.

For row 1,$[\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}] = [\mathbf{1}, \mathbf{0}, \mathbf{1}]$:

$$P_1 = [1, 1, 0, 1, 1 \times 0, 0 \times 1, 1 \times 1, 1^2, 0^2, 1^2, 0 \times 1^2, 1 \times 1^2, 1 \times 0^2, 1 \times 0^2, 0 \times 1^2, 1 \times 0 \times 1, 1^3, 0^3, 1^3]$$
$$= [1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1]$$

For row 2, $[\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}] = [\mathbf{1}, -\mathbf{1}, \mathbf{1}]$:

$$P_2 = [1, 1, -1, 1, 1 \times (-1), (-1) \times 1, 1 \times 1, 1^2, (-1)^2, 1^2, (-1) \times 1^2, 1 \times 1^2, 1 \times (-1)^2, 1 \times (-1)^2, -1 \times 1^2, 1 \times (-1) \times 1, 1^3, (-1)^3, 1^3]$$
$$= [1, 1, -1, 1, -1, -1, 1, 1, 1, 1, -1, 1, 1, 1, 1, -1, -1, 1, -1, 1]$$

Therefore, the matrix $\boldsymbol{P}$ is:

$$P = \begin{bmatrix} P_1 \\ P_2 \end{bmatrix}$$

Q3(b) *P* matrix for this polynomial

```
import numpy as np
from numpy.linalg import inv
from sklearn.preprocessing import PolynomialFeatures
X = np.array([[1,0,1], [1,-1,1]])
y = np.array([0, 1])

## Generate polynomial features
order = 3
poly = PolynomialFeatures(order)
P = poly.fit_transform(X)
print(f"P={P}")
```

```
P=[[ 1.  1.  0.  1.  1.  0.  1.  0.  0.  1.  1.  0.  1.  0.  0.  1.  0.  0.
   0.  1.]
 [ 1.  1. -1.  1.  1. -1.  1.  1. -1.  1.  1. -1.  1.  1. -1.  1. -1.  1.
  -1.  1.]]
```

(Note: The arrangement of the polynomial terms in the columns of matrix **P** using PolynomialFeatures from sklearn.preprocessing might be different from that in equation(1).)

Q3(c) Given $y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, can a unique solution be obtained in dual form?

In dual form, we aim to express the model in terms of the kernel matrix $\boldsymbol{K} = \mathbf{P}\mathbf{P}^T$ and solve for the dual variables. In this question, $\boldsymbol{K}$ is invertible (non-singular), a unique solution can be obtained in the dual form. The dual solution is given by:

$$\widehat{\boldsymbol{W}} = \mathbf{P}^T(\mathbf{P}\mathbf{P}^T)^{-1} = \mathbf{P}^T \begin{bmatrix} 10 & 10 \\ 10 & 25 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\widehat{\boldsymbol{w}}^T = \quad [\,0. \;\; 0. \;\; -0.1 \;\; 0. \;\; 0. \;\; -0.1 \;\; 0. \;\; 0.1 \; - 0.1 \;\; 0. \;\; 0. \;\; -0.1 \;\; 0. \;\; 0.1 \; - 0.1 \;\; 0. \;\; -0.1 \;\; 0.1 \; - 0.1 \;\; 0.\,]$$

```
## dual solution (without ridge)
w_dual = P.T @ inv(P @ P.T) @ y
print(w_dual)
```

```
[ 0.   0.  -0.1  0.   0.  -0.1  0.   0.1 -0.1  0.   0.  -0.1  0.   0.1
 -0.1  0.  -0.1  0.1 -0.1  0. ]
```

Q3(d) Given $y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, can the primal ridge regression be applied to obtain a unique solution?

In primal ridge regression, we solve for the weights $\boldsymbol{w}$ in the primal form:

$$\widehat{\boldsymbol{W}} = \left(\mathbf{P}^{\mathrm{T}}\mathbf{P} + \lambda \boldsymbol{I}\right)^{-1}\mathbf{P}^{T}\boldsymbol{y}$$

$$\widehat{\boldsymbol{w}}^{T} = \begin{matrix} [\,9.99976692e-07 & 9.99972144e-07 & -9.99980001e-02 & 9.99971235e-07 \\ 9.99967597e-07 & -9.99980000e-02 & 9.99966687e-07 & 9.99980001e-02 \\ -9.99980001e-02 & 9.99973054e-07 & 9.99965778e-07 & -9.99980000e-02 \\ 9.99966687e-07 & 9.99980001e-02 & -9.99980001e-02 & 9.99971235e-07 \\ -9.99980001e-02 & 9.99980000e-02 & -9.99980000e-02 & 9.99970325e-07\,] \end{matrix}$$

Here, at $\lambda = 0.0001$ we observe a very close solution to that in (c) even though (d) constitutes an approximation whereas (c) is exact.

```
## primal ridge
reg_L = 0.0001*np.identity(P.shape[1])
w_primal_ridge = inv(P.T @ P + reg_L) @ P.T @ y
print(w_primal_ridge)
```

```
[ 9.99976692e-07  9.99972144e-07 -9.99980001e-02  9.99971235e-07
  9.99967597e-07 -9.99980000e-02  9.99966687e-07  9.99980001e-02
 -9.99980001e-02  9.99973054e-07  9.99965778e-07 -9.99980000e-02
  9.99966687e-07  9.99980001e-02 -9.99980001e-02  9.99971235e-07
 -9.99980001e-02  9.99980000e-02 -9.99980000e-02  9.99970325e-07]
```

```
## dual solution (without ridge)
w_dual = P.T @ inv(P @ P.T) @ y
print(w_dual)
```

```
[ 0.   0.  -0.1  0.   0.  -0.1  0.   0.1 -0.1  0.   0.  -0.1  0.   0.1
 -0.1  0.  -0.1  0.1 -0.1  0. ]
```

Q4

(Binary Classification, Python)

Given the training data:

$$\{x = -1\} \rightarrow \{\, y = class1 \,\}$$

$$\{x = \ 0 \,\} \rightarrow \{\, y = class1 \,\}$$

$$\{x = 0.5\} \rightarrow \{\, y = class2 \,\}$$

$$\{x = 0.3\} \rightarrow \{\, y = class1 \,\}$$

$$\{x = 0.8\} \rightarrow \{\, y = class2 \,\}$$

Predict the class label for $\{x = -0.1\}$ and $\{x = 0.4\}$ using linear regression with signum discrimination

Q4

Step 1: Map Class Labels to Numeric Values

Since linear regression is a continuous method, you first need to convert the class labels (Class 1, Class 2) into numeric values. A common approach is to assign:

- Class 1 $\rightarrow y = -1$
- Class 2 $\rightarrow y = +1$

Step 2: Fit a Linear Regression Model

The linear regression model assumes a relationship of the form:

$$y = w_0 + w_1 x$$

$$\mathbf{X} = \begin{bmatrix} 1 & -1 \\ 1 & 0 \\ 1 & 0.5 \\ 1 & 0.3 \\ 1 & 0.8 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} +1 \\ +1 \\ -1 \\ +1 \\ -1 \end{bmatrix}$$

Q4

Step 3: Solve for the Weights $\mathbf{w_0}$ and $\mathbf{w_1}$

Using the normal equation:

$$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} = \begin{bmatrix} 0.3333 \\ -1.1111 \end{bmatrix}$$

Step 4: Make Predictions and Apply Signum Discrimination

$$\operatorname{sgn}(\hat{\mathbf{y}}_t) = \operatorname{sgn}(\mathbf{X}_t\hat{\mathbf{w}}) = \operatorname{sgn}\left(\begin{bmatrix} 0.4444 \\ -0.1111 \end{bmatrix}\right) = \begin{bmatrix} class +1 \\ class -1 \end{bmatrix} \begin{matrix} \rightarrow class1 \\ \rightarrow class2 \end{matrix}$$

Q4

```
import numpy as np
from numpy.linalg import inv
from sklearn.preprocessing import PolynomialFeatures
X = np.array([[1,-1], [1,0], [1,0.5], [1,0.3], [1,0.8]])
y = np.array([1, 1, -1, 1, -1])

## Linear regression for classification
w = inv(X.T @ X) @ X.T @ y
print(w)

Xt = np.array([[1,-0.1], [1,0.4]])
y_predict = Xt @ w
print(y_predict)

y_class_predict = np.sign(y_predict)
print(y_class_predict)
```

Q5

(Multi-Category Classification, Python)

Given the training data:

$$\{x = -1\} \rightarrow \{\, y = class1 \,\}$$
$$\{x = \ 0\,\} \rightarrow \{\, y = class1 \,\}$$
$$\{x = 0.5\} \rightarrow \{\, y = class2 \,\}$$
$$\{x = 0.3\} \rightarrow \{\, y = class3 \,\}$$
$$\{x = 0.8\} \rightarrow \{\, y = class2 \,\}$$

a) Predict the class label for $\{x{=}-0.1\}$ and $\{x{=}0.4\}$ based on linear regression towards a one-hot encoded target.

b) Predict the class label for $\{x{=}-0.1\}$ and $\{x{=}0.4\}$ using a polynomial model of 5th order and a one-hot encoded target.

Q5(a) Predict the class label for {x=−0.1} and {x=0.4} based on linear regression

One-hot encoding transforms class labels into a vector representation. For 3 classes, we'll have a 3-dimensional vector:

- $\text{Class1} \rightarrow [1,0,0]$
- $\text{Class2} \rightarrow [0,1,0]$
- $\text{Class3} \rightarrow [0,0,1]$

x	Class label	One-Hot Encoded y
-1	Class 1	$[1,0,0]$
0	Class 1	$[1,0,0]$
0.5	Class 2	$[0,1,0]$
0.3	Class 3	$[0,0,1]$
0.8	Class 2	$[0,1,0]$

Q5 (a) Predict the class label for {x=−0.1} and {x=0.4} based on linear regression

Step 1: Set up the training data

$$\mathbf{X} = \begin{bmatrix} 1 & -1 \\ 1 & 0 \\ 1 & 0.5 \\ 1 & 0.3 \\ 1 & 0.8 \end{bmatrix}, \quad \mathbf{Y} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{X}_\mathrm{t} = \begin{bmatrix} 1 & -0.1 \\ 1 & 0.4 \end{bmatrix}.$$

Step 2: Train a Linear Regression Model

- Train a linear regression model with x as the input and the one-hot encoded y as the target.

$$y = w_0 + w_1 x$$

- Solve for w_0 and w_1 by fitting the linear model to the training data.

$$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} = \begin{bmatrix} 0.4780 & 0.3333 & 0.1887 \\ -0.6499 & 0.5556 & 0.0943 \end{bmatrix}$$

Step 3: Predict for $x = -0.1$ and $x = 0.4$

Once we fit the model, we predict the output for all three classes:

$$\hat{\mathbf{Y}}_\mathrm{t} = \mathbf{X}_\mathrm{t}\hat{\mathbf{w}} = \begin{bmatrix} 1 & -0.1 \\ 1 & 0.4 \end{bmatrix} \begin{bmatrix} 0.4780 & 0.3333 & 0.1887 \\ -0.6499 & 0.5556 & 0.0943 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5430 & 0.2778 & 0.1792 \\ 0.2180 & 0.5556 & 0.2264 \end{bmatrix} \Rightarrow \begin{bmatrix} class1 \\ class2 \end{bmatrix}$$

Then, the predicted class is the one with the highest value in $\hat{Y}_t$.

Q5 (a) Predict the class label for {x=−0.1} and {x=0.4} based on linear regression

```
import numpy as np
from numpy.linalg import inv
from sklearn.preprocessing import PolynomialFeatures

X = np.array([[1,-1], [1,0], [1,0.5], [1,0.3], [1,0.8]])
Y = np.array([[1,0,0], [1,0,0], [0,1,0], [0,0,1], [0,1,0]])

## Linear regression for classification
W = inv(X.T @ X) @ X.T @ Y
print(W)

Xt = np.array([[1,-0.1], [1,0.4]])
y_predict = Xt @ W
print(y_predict)
y_class_predict = [[1 if y == max(x) else 0 for y in x] for x in y_predict ]
print(y_class_predict)
```

Q5 (b) Predict the class label for {x=−0.1} and {x=0.4} using a polynomial model

We will expand the input xx into polynomial features up to degree 5 for a 5th-order polynomial model.

$$f(\mathbf{x}) = w_0 \; + \; w_1 x_1 + w_2 x_1^2 + w_3 x_1^3 + w_4 x_1^4 + w_5 x_1^5$$

Step 1: Set up the polynomial features

For each x, we generate polynomial features up to the 5th degree:

Polynomial Features for $\boldsymbol{x} = [x^0, x^1, x^2, x^3, x^4, x^5]$

$$\mathbf{P} = \begin{bmatrix} 1.0000 & -1.0000 & 1.0000 & -1.0000 & 1.0000 & -1.0000 \\ 1.0000 & 0 & 0 & 0 & 0 & 0 \\ 1.0000 & 0.5000 & 0.2500 & 0.1250 & 0.0625 & 0.0313 \\ 1.0000 & 0.3000 & 0.0900 & 0.0270 & 0.0081 & 0.0024 \\ 1.0000 & 0.8000 & 0.6400 & 0.5120 & 0.4096 & 0.3277 \end{bmatrix}.$$

Q5 (b) Predict the class label for {x=−0.1} and {x=0.4} using a polynomial model

Step 2: Train the Polynomial Regression Model

We now fit a regression model using the polynomial features for each class. The model becomes:

$$\hat{\mathbf{w}} = \mathbf{P}^T(\mathbf{P}\mathbf{P}^T)^{-1}\mathbf{Y} = \begin{bmatrix} 1.0000 & 0 & -0.0000 \\ -5.3031 & -3.7023 & 9.0055 \\ 5.2198 & 10.8728 & -16.0926 \\ 6.6662 & 9.4698 & -16.1360 \\ -6.4765 & -12.9099 & 19.3864 \\ -2.6199 & -7.8045 & 10.4244 \end{bmatrix}.$$

Step 3: Predict for $x = -0.1$ and $x = 0.4$

Using the fitted model, predict the class labels for the given x values. The predicted class corresponds to the highest value in the output $\widehat{\boldsymbol{Y}_t}$.

$$\mathbf{P}_t = \begin{bmatrix} 1.0000 & -0.1000 & 0.0100 & -0.0010 & 0.0001 & -0.0000 \\ 1.0000 & 0.4000 & 0.1600 & 0.0640 & 0.0256 & 0.0102 \end{bmatrix}.$$

$$\hat{\mathbf{Y}}_t = \mathbf{P}_t\hat{\mathbf{w}} = \begin{bmatrix} 1.5752 & 0.4683 & -1.0435 \\ -0.0521 & 0.4544 & 0.5977 \end{bmatrix} \Rightarrow \begin{bmatrix} class1 \\ class3 \end{bmatrix}.$$

Q5(b) Predict the class label for {x=−0.1} and {x=0.4} using a polynomial model

```
## Polynomial regression for ## Generate polynomial features
order = 5
poly = PolynomialFeatures(order)

## only the data column (2nd) is needed for generation of polynomial terms
reshaped = X[:,1].reshape(len(X[:,1]),1)
P = poly.fit_transform(reshaped)
reshaped = Xt[:,1].reshape(len(Xt[:,1]),1)
Pt = poly.fit_transform(reshaped)

## dual solution (without ridge)
Wp_dual = P.T @ inv(P @ P.T) @ Y
print(Wp_dual)

yp_predict = Pt @ Wp_dual
print(yp_predict)
yp_class_predict = [[1 if y == max(x) else 0 for y in x] for x in yp_predict ]
```

This line converts the predicted values into one-hot encoded class predictions by selecting the index of the maximum value for each test point.

Q6

(Multi-Category Classification, Python)

- Get the data set “from sklearn.datasets import load_iris”. Use Python to perform the following tasks.
- (a) Split the database into two sets: 74% of samples for training, and 26% of samples for testing. Hint: you might want to utilize from sklearn.model_selection import train_test_split for the splitting.
- (b) Construct the target output using one-hot encoding.
- (c) Perform a linear regression for classification (without inclusion of ridge, utilizing one-hot encoding for the learning target) and compute the number of test samples that are classified correctly.
- (d) Using the same training and test sets as in above, perform a 2nd order polynomial regression for classification (again, without inclusion of ridge, utilizing one-hot encoding for the learning target) and compute the number of test samples that are classified correctly. Hint: you might want to use from sklearn.preprocessing import PolynomialFeatures for generation of the polynomial matrix.

Q6 (a) Split the database into two sets

- Load the iris dataset from sklearn.datasets. This dataset contains 150 samples of iris flowers, with 4 features (sepal length, sepal width, petal length, and petal width) and 3 target classes (setosa, versicolor, and virginica).
- Split the dataset into two sets: 74% for training and 26% for testing. You can use train_test_split from sklearn.model_selection to do this.

```
iris_dataset = load_iris()
print(iris_dataset.frame)

## (a) split data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( iris_dataset['data'], iris_dataset['target'], test_size=0.26, random_state=0)
```

Q6(b) One-hot encode the target output

For this task, you need to one-hot encode the target labels y, which means converting the class labels (0, 1, 2) into a binary matrix format.

- Create a OneHotEncoder instance with sparse=False, which returns a dense NumPy array.
- In newer versions of scikit-learn, use sparse_output=False for the same effect

```
from sklearn.preprocessing import OneHotEncoder
onehot_encoder=OneHotEncoder(sparse=False) # Use sparse_output instead of sparse for the new version
reshaped = y_train.reshape(len(y_train), 1)

Ytr_onehot = onehot_encoder.fit_transform(reshaped)
reshaped = y_test.reshape(len(y_test), 1)

Yts_onehot = onehot_encoder.fit_transform(reshaped)
```

fit_transform transforms the labels into one-hot encoded vectors for both training and testing sets (Ytr_onehot and Yts_onehot).

Q6 (c) Perform linear regression for classification (no ridge)

In this step, you'll perform linear regression on the training set using one-hot encoded targets. Once trained, you can use this model to make predictions on the test set and then compute how many samples are classified correctly.

```
## (c) Linear Classification
bias1 = np.ones((X_train.shape[0], 1))
X_train = np.concatenate((bias1, X_train), axis = 1)

Bias2 = np.ones((X_test.shape[0], 1))
X_test = np.concatenate((Bias2, X_test), axis = 1)
w = inv(X_train.T @ X_train) @ X_train.T @ Ytr_onehot
print(w)

# calculate the output based on the estimated w and test input X and then assign to one of the classes based on one hot encoding
yt_est = X_test.dot(w);
yt_cls = [[1 if y == max(x) else 0 for y in x] for x in yt_est ]
print(yt_cls)

# compare the predicted y with the ground truth
m1 = np.matrix(Yts_onehot)
m2 = np.matrix(yt_cls)
difference = np.abs(m1 - m2)
print(difference)

# calculate the error rate/accuracy
correct = np.where(~difference.any(axis=1))[0]
accuracy = len(correct)/len(difference)
print(len(correct))
print(accuracy)
```

Converts the predictions into one-hot format by assigning 1 to the class with the maximum predicted value and 0 to others.

Q6 (d) Perform 2nd order polynomial regression for classification

```
# calculate the error rate/accuracy
correct = np.where(~difference.any(axis=1))[0]
accuracy = len(correct)/len(difference)
print(len(correct))
print(accuracy)

## (d) Polynomial Classification
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(2)
P = poly.fit_transform(X_train)
Pt = poly.fit_transform(X_test)

if P.shape[0] > P.shape[1]:
    wp = inv(P.T @ P) @ P.T @ Ytr_onehot
else:
    wp = P.T @ inv(P @ P.T) @ Ytr_onehot

print(wp)

yt_est_p = Pt.dot(wp);
yt_cls_p = [[1 if y == max(x) else 0 for y in x] for x in yt_est_p ]
print(yt_cls_p)

m1 = np.matrix(Yts_onehot)
m2 = np.matrix(yt_cls_p)
difference = np.abs(m1 - m2)
print(difference)

correct_p = np.where(~difference.any(axis=1))[0]
accuracy_p = len(correct_p)/len(difference)
print(len(correct_p))
print(accuracy_p)
```

Here, you'll perform a 2nd order polynomial regression using one-hot encoded targets. You'll generate polynomial features using PolynomialFeatures from sklearn.preprocessing, fit the model, and compute the number of correctly classified test samples.

Check the type of P matrix

Converts the predictions into one-hot format by assigning 1 to the class with the maximum predicted value and 0 to others.

```
In [3]: print("yt_est_p")
        print(yt_est_p)

yt_est_p
[[ 4.39804651e+12  2.19902326e+12  4.39804651e+12]
 [ 2.19902326e+12 -1.09951163e+12  3.29853488e+12]
 [ 4.39804651e+12  3.29853488e+12  6.59706977e+12]
 [ 3.29853488e+12  5.19943718e-01  3.29853488e+12]
```

```
In [4]: print("yt_cls_p")
        print(yt_cls_p)

yt_cls_p
[[0, 0, 1], [0, 0, 1], [0, 0, 1], [0, 0, 1], [0, 0, 1],
```

Q7

MCQ: there could be more than one answer. Given three samples of two-dimensional data points $X = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 3 & 3 \end{bmatrix}$ with corresponding target vector $\mathbf{y} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$. Suppose you want to use a full third-order polynomial model to fit these data. Which of the following is/are true?

a) The polynomials model has 10 parameters to learn

b) The polynomial learning system is an under-determined one

c) The learning of the polynomial model has infinite number of solutions

d) The input matrix X has linearly dependent samples

e) None of the above

Q8

MCQ: there could be more than one answer. Which of the following is/are true?

a) The polynomial model can be used to solve problems with nonlinear decision boundary.

b) The ridge regression cannot be applied to multi-target regression.

c) The solution for learning feature $\mathbf{X}$ with target $\mathbf{y}$ based on linear ridge regression can be written as $\hat{\boldsymbol{w}} = (\mathbf{X}^T\mathbf{X} + \lambda\,\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$ for $\lambda > 0$. As λ increases, $\hat{\boldsymbol{w}}^T\hat{\boldsymbol{w}}$ decreases.

d) If there are four data samples with two input features each, the full second-order polynomial model is an over-determined system.

THANK YOU