Homework #1

Hyungseok Choi
Student No. : 2250166

Collaborated with Yann Hyunh

Short Answer and "True or False" Conceptual questions

A1.

a. Bias is measurement that measure how far the estimated value is from the actual value. Variance represents the difference between the estimated value. Bias-variance tradeoff means reducing both bias and variance is impossible.

b. As the complexity of the model increases, the variance tends to increase, whereas when the complexity decreases, the bias increases.

c. **False.** the variance is reduced because it is less likely to overfit the model.

d. Validation data sets.
First, train the models with only using the training data sets. Then, evaluate the performance and tune the hyperparameters for the models using the validation data sets.

e. **False.** The training error underestimate the true error.

Maximum Likelihood Estimation (MLE)

A2.

$$\text{Poi}(x|\lambda) = e^{-\lambda}\frac{\lambda^x}{x!}.$$

a. Let λ_{mle} is the MLE of λ. Since log of the likelihood has the same maximizer,

$$log\prod_{i=1}^{n}\text{Poi}(x_i|\lambda) = -n\lambda + (x_1 + x_2 + \cdots + x_n)log(\lambda) - (log(x_1!) + log(x_2!) + \cdots + log(x_n!))$$

Differentiate this by λ we get,

$$\frac{\partial}{\partial\lambda}\left(log\prod_{i=1}^{n}\text{Poi}(x_i|\lambda)\right) = -n + \frac{(x_1 + x_2 + \cdots + x_n)}{\lambda}$$

$$\lambda_{\text{mle}} = \frac{1}{n}\left(\sum_{i=1}^{n}x_i\right)$$

b. $\lambda = \frac{2+4+6+0+1}{5} = 2.6$
$\text{Poi}(6|\lambda) = e^{-2.6}\cdot\frac{2.6^6}{6!} = 0.03187$

c. $\lambda = \frac{2+4+6+0+1+8}{6} = 3.5$
$\text{Poi}(7|\lambda) = e^{-3.5}\cdot\frac{3.5^6}{6!} = 0.07709$

Polynomial Regression

A3.

By increasing lambda, the polynomial regression model is less overfitting the test data. This means that the model reflects the overall trend of the data rather than fully fitting the results of the test data.

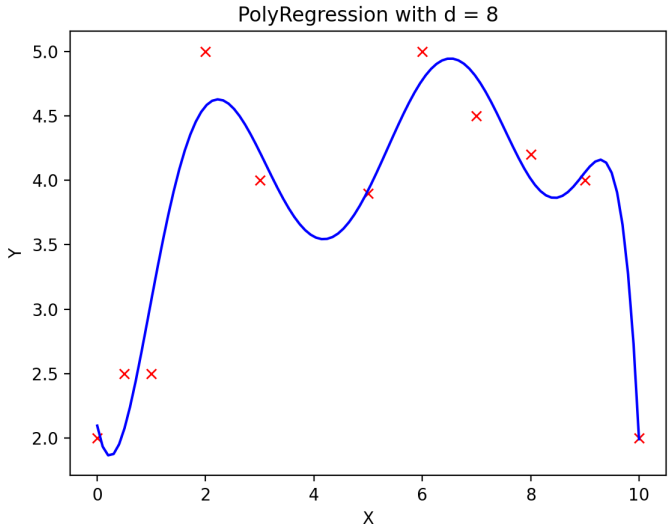

Figure 1: Fit of polynomial regression with $\lambda = 0$ and $d = 8$

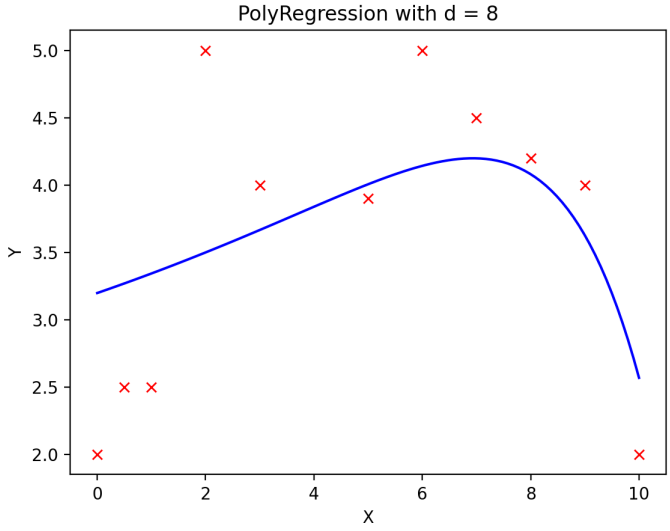

Figure 2: Fit of polynomial regression with $\lambda = 5$ and $d = 8$

A4.

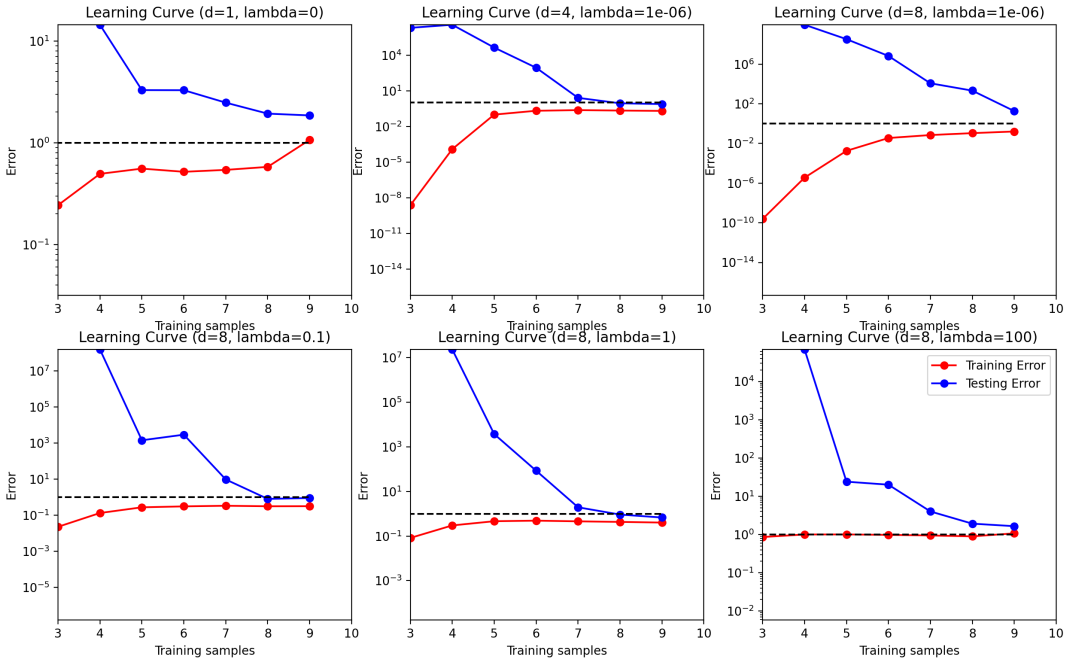

Figure 3: Learning curves with various degree and λ

```
// polyreg.py

from typing import Tuple
import numpy as np
from utils import problem


class PolynomialRegression:
    @problem.tag("hw1-A", start_line=5)
    def __init__(self, degree: int = 1, reg_lambda: float = 1e-8):
        self.degree: int = degree
        self.reg_lambda: float = reg_lambda
        self.weight: np.ndarray = None  # type: ignore
        self.xmeans: np.ndarray = None
        self.xsds: np.ndarray = None

    @staticmethod
    @problem.tag("hw1-A")
    def polyfeatures(X: np.ndarray, degree: int) -> np.ndarray:
        n = len(X)
        X_ = np.c_[np.ones([n, 1]), X]
        for i in range(degree - 1):
            X_ = np.c_[X_, X_[:, -1] * X_[:, 1]]

        return X_[:, 1:]

    @problem.tag("hw1-A")
    def fit(self, X: np.ndarray, y: np.ndarray):
        n = len(X)

        # get polynomial form and standardize
        X_ = self.polyfeatures(X, self.degree)
        self.xmeans = np.mean(X_, axis=0)
        self.xsds = np.std(X_, axis=0)

        X_ = (X_ - self.xmeans) / self.xsds

        # add 1s column
        X_ = np.c_[np.ones([n, 1]), X_]
        n, d = X_.shape

        # construct reg matrix
        reg_matrix = self.reg_lambda * np.eye(d)
        reg_matrix[0, 0] = 0

        # analytical solution (X'X + regMatrix)^-1 X' y
        self.weight = np.linalg.solve(X_.T @ X_ + reg_matrix, X_.T @ y)

    @problem.tag("hw1-A")
    def predict(self, X: np.ndarray) -> np.ndarray:
        n = len(X)

        # get polynomial form and standardize
        X_ = self.polyfeatures(X, self.degree)
        X_ = (X_ - self.xmeans) / self.xsds
```

```
        # add 1s column
        X_ = np.c_[np.ones([n, 1]), X_]

        # predict
        return X_.dot(self.weight)


@problem.tag("hw1-A")
def mean_squared_error(a: np.ndarray, b: np.ndarray) -> float:
    return np.square(np.subtract(a , b)).mean()


@problem.tag("hw1-A", start_line=5)
def learningCurve(
    Xtrain: np.ndarray,
    Ytrain: np.ndarray,
    Xtest: np.ndarray,
    Ytest: np.ndarray,
    reg_lambda: float,
    degree: int,
) -> Tuple[np.ndarray, np.ndarray]:
    n = len(Xtrain)
    errorTrain = np.zeros(n)
    errorTest = np.zeros(n)

    model = PolynomialRegression(degree, reg_lambda)
    for i in range(1, n):
        model.fit(Xtrain[:i + 1], Ytrain[:i + 1])
        errorTrain[i] = mean_squared_error(model.predict(Xtrain[:i + 1]), Ytrain[:i + 1
        errorTest[i] = mean_squared_error(model.predict(Xtest), Ytest)

    return (errorTrain, errorTest)
```

Ridge Regression on MNIST

A5.

- **Part A:** Differentiate $\sum_{j=1}^{k}\left[\|Xw_j - Ye_j\|^2 + \lambda\|w_j\|^2\right]$ by w_i we get,

$$\begin{aligned}\frac{\partial}{\partial w_i}\sum_{j=1}^{k}\left[||Xw_j - Ye_j||^2 + \lambda||w_j||^2\right] &= \tfrac{\partial}{\partial w_i}\left(||Xw_i - Ye_i||^2 + \lambda||w_i||^2\right) + \tfrac{\partial}{\partial w_i}\textstyle\sum_{j\neq i}\left[||Xw_j - Ye_j||^2 + \lambda||w_j||^2\right] \\ &= \tfrac{\partial}{\partial w_i}(Xw_i - Ye_i)^T(Xw_i - Ye_i) + \tfrac{\partial}{\partial w_i}\lambda w_i^T w_i \\ &= 2(X)^T(Xw_i - Ye_i) + 2\lambda w_i\end{aligned}$$

The value of w_i that makes this expression zero is,

$$\begin{aligned}0 &= 2(X)^T(Xw_i - Ye_i) + 2\lambda w_i \\ &= (X)^T(Xw_i - Ye_i) + \lambda w_i\end{aligned}$$

Therefore,

$$\begin{aligned}X^TXw_i + \lambda w_i &= X^TYe_i \\ (X^TX + \lambda I)w_i &= X^TYe_i \\ \therefore w_i &= (X^TX + \lambda I)^{-1}X^TYe_i\end{aligned}$$

Since w_i is ith element of gradient of $\widehat{W}$,

$$\widehat{W} = (X^TX + \lambda I)^{-1}X^TY$$

- **Part B:**
Train Error: 14.805% Test Error: 14.66%

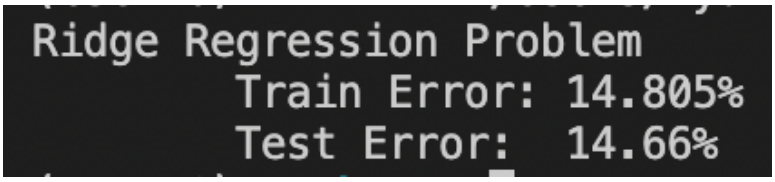

Figure 4: Ridge regression train, test error

```
// ridge_regression.py

import numpy as np

from utils import load_dataset, problem


@problem.tag("hw1-A")
def train(x: np.ndarray, y: np.ndarray, _lambda: float) -> np.ndarray:
    n, d = x.shape
    reg_matrix = _lambda * np.eye(d)
    return np.linalg.solve(x.T @ x + reg_matrix, x.T @ y)


@problem.tag("hw1-A")
def predict(x: np.ndarray, w: np.ndarray) -> np.ndarray:
    return np.argmax(x.dot(w), axis=1)


@problem.tag("hw1-A")
def one_hot(y: np.ndarray, num_classes: int) -> np.ndarray:
    one_hot = np.zeros((len(y), num_classes))
    one_hot[np.arange(len(y)), y] = 1
    return one_hot


def main():

    (x_train, y_train), (x_test, y_test) = load_dataset("mnist")
    # Convert to one-hot
    y_train_one_hot = one_hot(y_train.reshape(-1), 10)

    _lambda = 1e-4

    w_hat = train(x_train, y_train_one_hot, _lambda)

    y_train_pred = predict(x_train, w_hat)
    y_test_pred = predict(x_test, w_hat)

    print("Ridge␣Regression␣Problem")
    print(
        f"\tTrain␣Error:␣{np.average(1␣-␣np.equal(y_train_pred,␣y_train))␣*␣100:.6g}%"
    )
    print(f"\tTest␣Error:␣␣{np.average(1␣-␣np.equal(y_test_pred,␣y_test))␣*␣100:.6g}%")

if __name__ == "__main__":
    main()
```

[language=python]

Administrative

A6.

15 hours