CSE546 HW1

Haoxin Luo

December 8, 2022

Exercise A1.

a).

Bias is how much our predicted value different from the truth, variance is the amount of change each time we use a different training sample to estimate the target. Bias variance trade off means that we usually cannot always satisfy both conditions at the same time. It's hard to find the right balance without over-fitting or under-fitting the data. When we have a high bias, the model fails to learn the pattern well, but if the model is too complicated (high variance), it might fail to generalize on data that we have never seen before.

b).

When model complexity increase, we have higher variance and lower bias. When model complexity decrease, we have lower variance and higher bias.

c).

False, the variance will decrease because when the algorithm exposes to more data, it will less likely over-fit the data.

d).

We should use a validation set for hyperparameter tuning. We use training set to train the model, and the test set to predict. The basic procedure is first we need to define all the range of possible values for our hyperparameters, then sample those hyperparameter values based on certain method, define an evaluative criteria to judge the model, and finally define a cross validation method.

e).

False, underestimate.

Exercise A2-MLE

a).

$$L(\lambda; x_1, ...x_n) = \Pi_{i=1}^{n} exp(-\lambda)\frac{1}{x_i!}\lambda^x$$

$$l(\lambda; x_1, ...x_n) = -n\lambda - \sum_{i=1}^{n} ln(x_i!) + ln(\lambda)\sum_{i=1}^{n} x_i$$

Take the first derivative and set to 0

$$\frac{dl(\lambda; x_1, ...x_n)}{d\lambda} = -n + \frac{1}{\lambda}\sum_{i=1}^{n} x_i = 0$$

$$\hat{\lambda} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

b). $x = 5$, $\hat{\lambda} = \frac{1}{5}(2 + 4 + 6 + 0 + 1) = 2.6$, $P(x = 6|\lambda = 2.6) = \frac{2.6^6 e^{-2.6}}{6!} \approx 0.032$

c). $\hat{\lambda} = \frac{1}{6}(2 + 4 + 6 + 0 + 1 + 8) = 3.5$, $P(x = 6|\lambda = 3.5) = \frac{3.5^6 e^{-3.5}}{6!} \approx 0.077$

Exercise B1-Overfitting

a).

$$E_{train}[\hat{\varepsilon}_{train}(f)] = \frac{1}{N_{train}} E[\sum_{(x,y)\in S_{train}} (f(x)-y)^2]$$

$$= \int \frac{1}{N_{train}} \sum_{i}^{N} (f(x_i)-y)^2 p(x_1, y_1, x_2, y_2, ..., x_n, y_n) dx_1 dy_1 ... dx_n dy_n$$

Given x, y are independent,

$$= \int \frac{1}{N_{train}} \sum_{i}^{N} (f(x_i)-y)^2 p(x_1, y_1) P(x_2, y_2) ..., P(x_n, y_n) dx_1 dy_1 ... dx_n dy_n$$

Given x, y are identical,

$$= \int (f(x_1)-y_1)^2 p(x_1, y_1) dx_1 dy_1$$

$$= \int \frac{1}{N_{train}} \sum_{i}^{N} (f(x_i)-y)^2 p(x_1, y_1) P(x_2, y_2) ..., P(x_n, y_n) dx_1 dy_1 ... dx_n dy_n$$

Given x, y are identical,

$$= \int (f(x_1)-y_1)^2 p(x_1, y_1) dx_1 dy_1$$

$$= E_{(x,y)\approx D}[(f(x)-y)^2] = \varepsilon(f)$$

Similary, we can still have $E_{test}[\hat{\varepsilon}_{test}(\hat{f})] = \varepsilon(\hat{f})$ since our i.i.d. assumption still holds given $\hat{f}$ is not trained on the test data. Thus,

$$E_{train}[\hat{\varepsilon}_{train}(f)] = E_{test}[\hat{\varepsilon}_{test}(f)] = \varepsilon(f)$$

b).

The above equation in general is not true because we do not have the independent assumption anymore.

c).

Looking at the left hand side of the equation:

$$E_{train}[\hat{\varepsilon}_{train}(\hat{f}_{train})] = \sum_{f\in F} E_{train}[\hat{\varepsilon}_{train}(\hat{f}_{train})] P_{train}(\hat{f}_{train} = f)$$

$$\leq \sum_{f\in F} E_{train}[\hat{\varepsilon}_{train}(f)] P_{train}(\hat{f}_{train} = f)$$

Looking at the right hand side:

$$E_{train,test}[\hat{\varepsilon}_{test}(\hat{f}_{train})] = \sum_{f\in F} E_{train,test}[\hat{\varepsilon}_{test}(f) I(\hat{f}_{train} = f)]$$

$$= \sum_{f\in F} E_{train}[I(\hat{f}_{train} = f)] E_{test}[\hat{\varepsilon}_{test}(f)]$$

$$= \sum_{f \in F} E_{test}[\hat{\varepsilon}_{test}(f)] P_{train}(\hat{f}_{train} = f)$$

Thus

$$E_{train}[\hat{\varepsilon}_{train}(\hat{f}_{train})] \leq \sum_{f \in F} E_{train}[\hat{\varepsilon}_{train}(f)] P_{train}(\hat{f}_{train} = f) = E_{train,test}[\hat{\varepsilon}_{test}(\hat{f}_{train})]$$

Exercise B2 - Bias Variance Tradeoff

a).

We would expect that if we have more intervals, m will be small, resulting in lower bias and higher variance. On the other hand, when m is large, we have less intervals and bias will be higher, variance will be lower.

b).

$$\frac{1}{n}\sum_{i=1}^{n}(E[\hat{f}_m(x_i)-f(x_i))^2=\frac{1}{n}\sum_{i=1}^{n}(E[\sum_{j=1}^{n/m}c_jI\{x_i\in(\frac{(j-1)m}{n},\frac{jm}{n}]\}]-f(x_i))^2$$

$$=\frac{1}{n}\sum_{i=1}^{n}(E[\sum_{j=1}^{n/m}(\frac{1}{m}\sum_{k=m(j-1)+1}^{jm}y_k)I\{x_i\in(\frac{(j-1)m}{n},\frac{jm}{n}]\}]-f(x_i))^2$$

$$=\frac{1}{n}\sum_{i=1}^{n}(\sum_{j=1}^{n/m}(\frac{1}{m}\sum_{k=m(j-1)+1}^{jm}E(f(x_k)+\varepsilon_k)I\{x_i\in(\frac{(j-1)m}{n},\frac{jm}{n}\}]-f(x_i))^2$$

$$=\frac{1}{n}\sum_{i=1}^{n}(\sum_{j=1}^{n/m}(\frac{1}{m}\sum_{k=m(j-1)+1}^{jm}f(x_k)I\{x_i\in(\frac{(j-1)m}{n},\frac{jm}{n}\}]-f(x_i))^2$$

$$=\frac{1}{n}\sum_{i=1}^{n}(\sum_{j=1}^{n/m}\bar{f}^{(j)}I\{x_i\in(\frac{(j-1)m}{n},\frac{jm}{n}\}]-f(x_i))^2$$

Because, we have $x=i/n$ and so $i\in((j-1)m,jm]$, we get:

$$\frac{1}{n}\sum_{i=1}^{n}(E[\hat{f}_m(x_i)-f(x_i))^2=\frac{1}{n}\sum_{i=1}^{n}(\sum_{j=1}^{n/m}\bar{(}f^{(j)}-f(x_i))^2$$

c).

$$E[\frac{1}{n}\sum_{i=1}^{n}(\hat{f}_m(x_i)-E[\hat{f}_m(x_i)])^2]=E[\frac{1}{n}\sum_{i=1}^{n}(\sum_{j=1}^{n/m}(c_jI\{x_i\in(\frac{(j-1)m}{n},\frac{jm}{n}]\}-E[\sum_{j=1}^{n/m}c_jI\{x_i\in(\frac{(j-1)m}{n},\frac{jm}{n}]\}])^2]$$

$$=E[\frac{1}{n}\sum_{i=1}^{n}(\sum_{j=1}^{n/m}(c_j-E[c_j])I\{x_i\in(\frac{(j-1)m}{n},\frac{jm}{n}]\}])^2$$

$$=\frac{1}{n}\sum_{j=1}^{n/m}E[\sum_{k=m(j-1)+1}^{jm}(c_j-E[c_j])^2]$$

$$=\frac{1}{n}\sum_{j=1}^{n/m}E[m(c_j-E[c_j])^2]$$

Since $E[c_j]=\frac{1}{m}\sum_{i=(k-1}m+1^{jm}E[y_k]=\bar{f}^{(j)}$, we have:

$$\frac{1}{n}\sum_{j=1}^{n/m}E[m(c_j-E[c_j])^2]=\frac{1}{n}\sum_{j=1}^{n/m}E[m(c_j\bar{f}^{(j)})^2]$$

$$= \frac{1}{n}\sum_{j=1}^{n/m} mE[\sum_{i=(j-1)m+1}^{jm} \frac{\varepsilon_i}{m})^2]$$

$$= \frac{1}{n}\sum_{j=1}^{n/m} \frac{1}{m}\sum_{i=(j-1)m+1}^{jm} E(\varepsilon_i^2)]$$

$$= \frac{1}{n}\sum_{j=1}^{n/m} \sigma^2$$

$$E[\frac{1}{n}\sum_{i=1}^{n}(\hat{f}_m(x_i) - E[\hat{f}_m(x_i)])^2] = \frac{1}{n}\sum_{j=1}^{n/m} E[m(c_j\bar{f}^{(j)})^2] = \frac{\sigma^2}{m}$$

d).

By mean value theorem, $|\bar{f}^{(j)} - f(x_i)|$ is bounded by $|max f(x_i) - min f(x_j)|$ which is $\frac{L}{n}|i-j| = \frac{Lm}{n}$

Thus, we have our bias square term to be:

$$\frac{1}{n}\sum_{j=1}^{n/m}\sum_{i=(j-1)m+1}^{jm}(f^{(j)} - f(x_i))^2 \leq \frac{1}{n}\sum_{j=1}^{n/m}\sum_{i=(j-1)m+1}^{jm}(\frac{Lm}{n})^2 = O(\frac{L^2m^2}{n^2})$$

Furthermore, as total error = bias squared + variance and we need to take the variance into account, we will have $O(\frac{L^2m^2}{n^2} + \frac{\sigma^2}{m})$ complexity. Taking the derivative with respect to m, the minimum is:

$$\frac{d}{dm}(\frac{L^2m^2}{n^2} + \frac{\sigma^2}{m}) = 0$$

$$m = (\frac{n^2\sigma^2}{2L^2})^{1/3}$$

$$O(\frac{L^2m^2}{n^2} + \frac{\sigma^2}{m}) = O((\frac{\sigma^2L}{n})^{2/3})$$

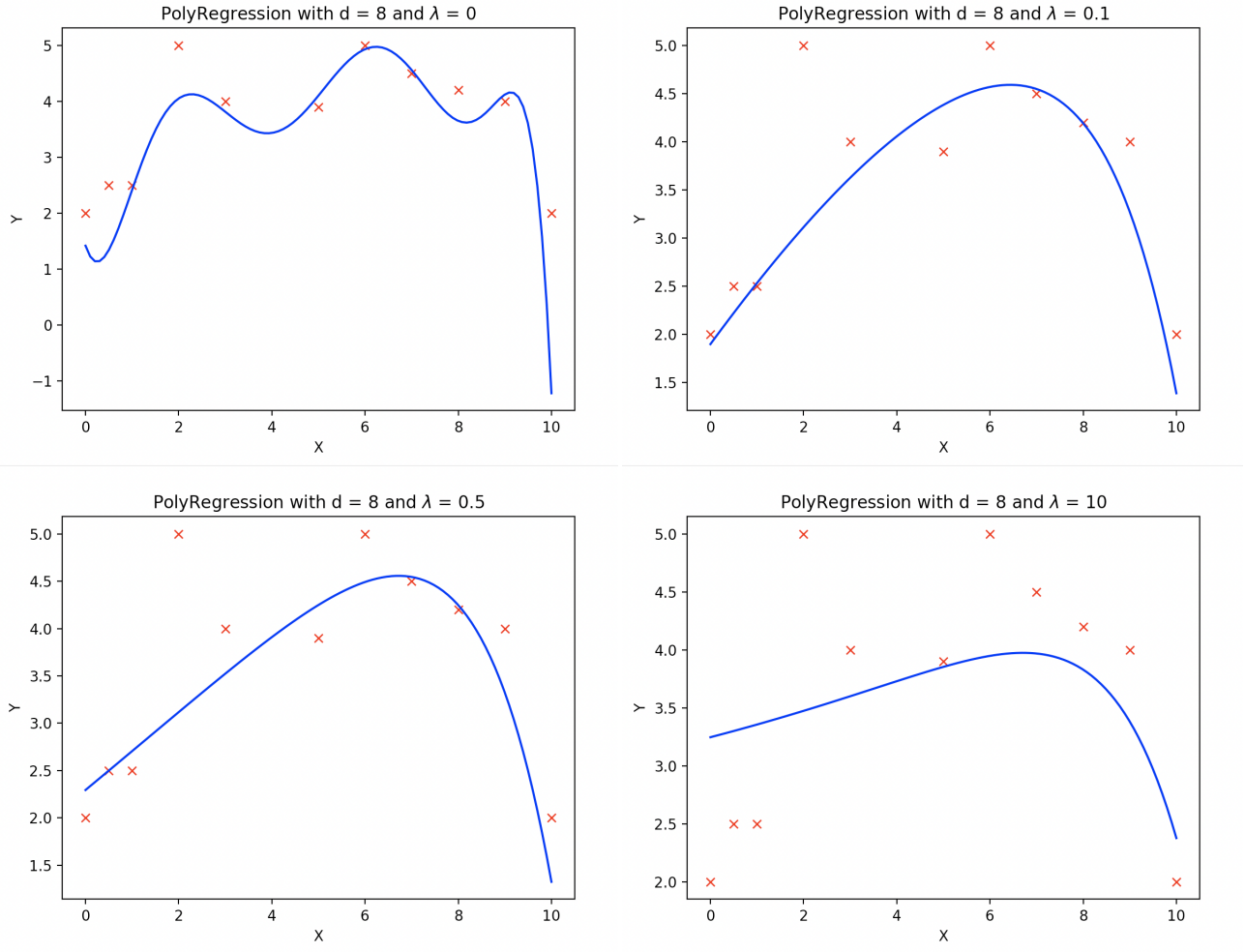
From the above, we can find out that:
when L increases, σ^2 increases and n decreases: the total error increases as the function become less smooth.
when L decreases, σ^2 decreases and n increases: the total error decreases as the function become more smooth.

Exercise A3-Polynomial Regression

a).

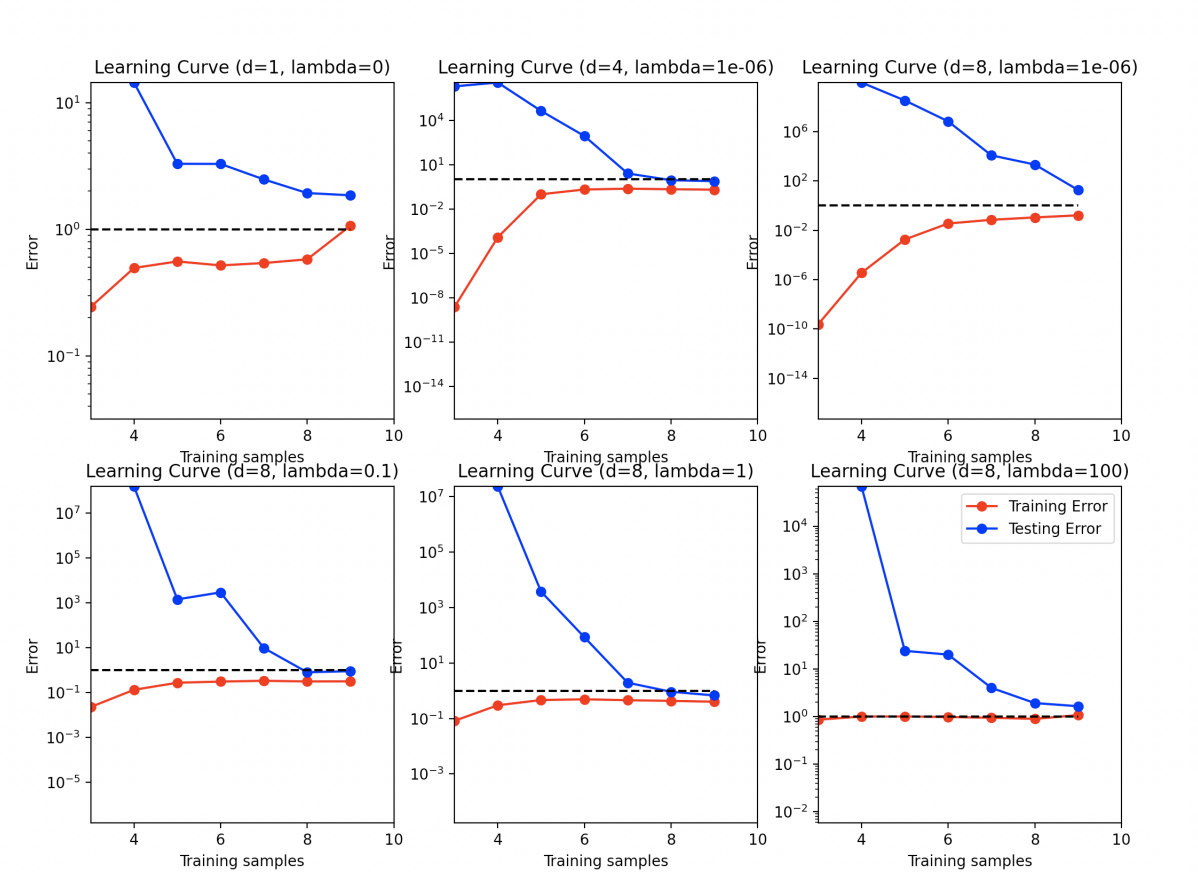
Code in gradescope

b).



From the above pictures, we can see that when we increase the size of regularization term, the fitting curve become smoother and more flat. Which also means our bias increases and our variance decreases.

Exercise A4.



Code in gradescope

Exercise Ridge Regression

a). From the prompt, we have:

$$\widehat{W} = argmin_w \sum_{i=1}^{n} (\|W^T x_i - y_i\|_2^2 + \lambda \|w_i\|_F^2)$$

$$= \sum_{j=1}^{k} [\|Xw_j - Ye_j\|^2 + \lambda \|w_j\|^2]$$

To minimize the above quantity with respect to W, we want to set the derivative to zero:

$$\frac{\partial}{\partial w_j} \sum_{j=0}^{k} \left[\|Xw_j - Ye_j\|^2 + \lambda \|w_j\|^2\right] = \sum_{j=0}^{k} \frac{\partial}{\partial w_j} \left[\|(w_j^T X^T - e_j Y^T)(X_{wj} - Y_{ej})\| + \lambda \|w_j^T w_j\|\right]$$

$$= \sum_{j=0}^{k} \frac{\partial}{\partial w_j} [w_j^T X^T X w_j - w_j^T X^T Y_{ej} - e_j^T Y^T Y e_j + e_j^T Y^T Y_{ej} + \lambda w_j^T w_j] = 0$$

$$\sum_{j=0}^{k} (X^T X + \lambda I) w_j = \sum_{j=0}^{k} X^T Y_{ej}$$

$$\sum_{j=0}^{k} \hat{W}_j = \sum_{j=0}^{k} (X^T X + \lambda I)^{-1} X^T Y_{ej}$$

So we get

$$\boxed{\widehat{W} = (X^T X + \lambda I)^{-1} X^T Y}$$

b).

Code:

```
import numpy as np
from mnist.loader import MNIST
import scipy.linalg as LA
import matplotlib.pyplot as plt
from utils import load_dataset, problem


@problem.tag("hw1-A")
def train(x: np.ndarray, y: np.ndarray, regLambda: float) -> np.ndarray:
    """Train function for the Ridge Regression problem.
    Should use observations (`x`), targets (`y`) and regularization parameter (`_lambda`)
    to train a weight matrix $$\\hat{W}$$.


    Args:
        x (np.ndarray): observations represented as `(n, d)` matrix.
            n is number of observations, d is number of features.
        y (np.ndarray): targets represented as `(n, k)` matrix.
            n is number of observations, k is number of classes.
```

```
        _lambda (float): parameter for ridge regularization.

    Raises:
        NotImplementedError: When problem is not attempted.

    Returns:
        np.ndarray: weight matrix of shape `(d, k)`
            which minimizes Regularized Squared Error on `x` and `y` with hyperparameter `_la
    """
    #raise NotImplementedError("Your Code Goes Here")
    W_hat = np.linalg.solve(x.T.dot(x) + regLambda*np.eye(x.shape[1]), x.T.dot(y))
    return W_hat



@problem.tag("hw1-A")
def predict(x: np.ndarray, w: np.ndarray) -> np.ndarray:
    """Train function for the Ridge Regression problem.
    Should use observations (`x`), and weight matrix (`w`) to generate predicated class for e

    Args:
        x (np.ndarray): observations represented as `(n, d)` matrix.
            n is number of observations, d is number of features.
        w (np.ndarray): weights represented as `(d, k)` matrix.
            d is number of features, k is number of classes.

    Raises:
        NotImplementedError: When problem is not attempted.

    Returns:
        np.ndarray: predictions matrix of shape `(n,)` or `(n, 1)`.
    """
    #raise NotImplementedError("Your Code Goes Here")
    pred = np.argmax(w.T.dot(x.T), axis = 0)
    return pred


@problem.tag("hw1-A")
def one_hot(y: np.ndarray, num_classes: int) -> np.ndarray:
    """One hot encode a vector `y`.
    One hot encoding takes an array of integers and coverts them into binary format.
    Each number i is converted into a vector of zeros (of size num_classes), with exception o

    Args:
        y (np.ndarray): An array of integers [0, num_classes), of shape (n,)
        num_classes (int): Number of classes in y.

    Returns:
```

```
        np.ndarray: Array of shape (n, num_classes).
        One-hot representation of y (see below for example).

    Example:
        ```python
        > one_hot([2, 3, 1, 0], 4)
        [
            [0, 0, 1, 0],
            [0, 0, 0, 1],
            [0, 1, 0, 0],
            [1, 0, 0, 0],
        ]
        ```
    """
    #raise NotImplementedError("Your Code Goes Here")
    return np.squeeze(np.eye(num_classes)[y.reshape(-1)])

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()
```

c).

The training error is: 14.805%

The test error is: 14.66%

Exercise A6

about 15 hours