CSE 546 Homework 1

Jake Gonzales

October 19, 2023

Collaborators: compared answers with Alexey Yermakov

1 Conceptual Questions

A1.

(a) **Bias** is the difference between the expectation of the estimated model and the ground truth model. If we have a higher bias then that means our estimated model is not a good representation of the training dataset. If our bias is low or we are unbiased, then that means we do have a good estimated model.

Variance can be described as how much the model varies with respect to different portions of the dataset or new data. Essentially how much does the estimated model or predictions change when we sample data from a distribution.

The **Bias-Variance Tradeoff** is the inverse relationship between the bias and variance. If we have a higher bias then we usually have a lower variance and vice versa. Mathematically, if we are computing the mean square error (MSE) for our predicted model (estimator) then sometimes it is better to use a bias estimator as long as it reduces our variance enough. We are attempting to find the optimal tradeoff between bias and variance.

(b) Typically, when the model complexity increases, the bias decreases and variance increases. As the model complexity decreases, the bias increases and the variance decreases.

(c) **False.** We would expect the variance to decrease.

(d) You should use the validation set for tuning hyperparamters. The training set is used to train your model and optimize those parameters (w, b), and we use the validation set to tune the hyperparameters and pick the model that best performs on that validation set. The test sample is only used at the end and not for any sort of model optimization.

(e) **False.** It provides an underestimate of the true error.

2 Maximum Likelihood Estimation (MLE)

A2.

(a) Let our non-negative vectors $\{x_1, \cdots, x_n\}$ represent the scores. The probability mass function for our Poisson distribution is written as:

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

Writing out the likelihood function as the joint pmf of our scores

$$L(\lambda) = \prod_{i=1}^{n} e^{-\lambda} \frac{\lambda_i^x}{x_i!}.$$

The log of the likelihood has the same maximizer as the likelihood function itself. The log of the product is the some of the logs.

$$\log(L(\lambda)) = \sum_{i=1}^{n} \log(e^{-\lambda}\frac{\lambda_i^x}{x_i!}) = \sum_{i=1}^{n} \log(e^{-\lambda}) + \log(\lambda^{x_i}) + \log(x_i!)$$
$$= \sum_{i=1}^{n} -\lambda + x_i \log(\lambda) + \log(x_i!)$$

Lets take the derivative and set to zero

$$\frac{d}{dx}\log(L(\lambda)) = \sum_{i=1}^{n} \frac{x_i}{\lambda} - 1 = n - \sum_{i=1}^{n} \frac{x_i}{\lambda} = 0$$

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

(b) A numerical estimate for λ after the first 5 games:

$$\lambda = \sum_{i=1}^{n} \frac{x_i}{n} = \sum_{i=1}^{5} \frac{(2+4+6+0+1)}{5} = \frac{13}{5}$$

Given this λ the probability that they score 6 goals in their next game is

$$\text{Poi}(6 \mid 2.6) = \frac{e^{-2.6} \cdot 2.6^6}{6!} = 0.032.$$

If they scored 8 goals in their 6th game the updated probability for them to score exactly 6 goals in their 7th game is

$$\lambda = \sum_{i=1}^{n} \frac{x_i}{n} = \sum_{i=1}^{6} \frac{(2+4+6+0+1+8)}{6} = \frac{21}{6}$$

$$\text{Poi}(6 \mid 3.5) = \frac{e^{-3.5} \cdot 3.5^6}{6!} = 0.077.$$

---

3 Overfitting

B1.

(a) Lets start by taking the expectation with respect to all possible sets in S_{train} of the training loss $\widehat{\epsilon}_{train}(f)$ which is defined in the problem statement.

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

Because the training set S_{train} is sampled i.i.d. we can make the equivalence that picking points on by one is the same as sampling one point from the distribution N_{train} times. So we will replace the expected value over the training (and later test) set with the expected value over the distribution. With this being said, and recalling the definition of the true least squares error of f we can write

$$\implies \frac{1}{N_{train}}\mathbb{E}_{S_{train}}[\sum_{(x,y)\in S_{train}} (f(x)-y)^2] = \frac{1}{N_{train}} N_{train}\mathbb{E}_{(x,y)\sim D}[(f(x)-y)^2] = \epsilon(f)$$

Here we are also assuming f is independent of S_{train} given the problem statement.

Similarly and following the same reasoning, for the test error we can show:

$$\mathbb{E}_{S_{test}}[\widehat{\epsilon}_{test}(f)] = \mathbb{E}_{S_{test}}[\frac{1}{N_{test}} \sum_{(x,y)\in S_{test}} (f(x)-y)^2] = \frac{1}{N_{test}}\mathbb{E}_{S_{test}}[\sum_{(x,y)\in S_{test}} (f(x)-y)^2]$$
$$= \frac{1}{N_{test}} N_{test}\mathbb{E}_{(x,y)\sim D}[(f(x)-y)^2] = \epsilon(f).$$

Now we want to show that the test error is an unbiased estimate of our true error for $\widehat{f}$. Specifically we show that:

$$\mathbb{E}_{S_{test}}[\widehat{\epsilon}_{test}(\widehat{f})] = \mathbb{E}_{S_{test}}[\frac{1}{N_{test}} \sum_{(x,y)\in S_{test}} (\widehat{f}(x)-y)^2] = \frac{1}{N_{test}}\mathbb{E}_{S_{test}}[\sum_{(x,y)\in S_{test}} (\widehat{f}(x)-y)^2]$$
$$= \frac{1}{N_{test}} N_{test}\mathbb{E}_{(x,y)\sim D}[(f(x)-y)^2] = \epsilon(\widehat{f}).$$

(b) In general this is <u>not true</u>. In the previous statement we have shown this equivalency, but usually $\widehat{f}$ is not independent of S_{train} so the above does not hold.

(c) Using the hint we see that:

$$\mathbb{E}_{S_{\text{train}},S_{\text{test}}}[\hat{\epsilon}_{\text{train}}(\hat{f}_{\text{train}})] = \sum_{f\in\mathcal{F}} \mathbb{E}_{test}[\hat{\epsilon}_{\text{test}}(f)]\mathbb{P}_{\text{train}}(\hat{f}_{\text{train}} = f)$$

And as shown in the previous part:

$$\mathbb{E}_{\text{train}}[\hat{\epsilon}_{\text{test}}(f)] = \mathbb{E}_{\text{train}}[\hat{\epsilon}_{\text{train}}(f)]$$

combining we get

$$\sum_{f\in\mathcal{F}} \mathbb{E}_{\text{test}}[\hat{\epsilon}_{\text{test}}(f)]\mathbb{P}_{\text{train}}(\hat{f}_{\text{train}} = f) = \sum_{f\in\mathcal{F}} \mathbb{E}_{\text{train}}[\hat{\epsilon}_{\text{train}}(f)]\mathbb{P}_{\text{train}}(\hat{f}_{\text{train}} = f)$$

For any $f \in \mathcal{F}$ we know that $\hat{\epsilon}_{\text{train}}(\hat{f}_{\text{train}}) \leq \hat{\epsilon}_{\text{train}}(f)$ so we assume that holds true for the expectation case:

$$\mathbb{E}_{\text{train}}[\hat{\epsilon}_{\text{train}}(\hat{f}_{\text{train}})] \leq \mathbb{E}_{\text{train}}[\hat{\epsilon}_{\text{train}}(f)]$$

and thus we have

$$\geq \sum_{f\in\mathcal{F}} \mathbb{E}_{\text{train}}[\hat{\epsilon}_{\text{train}}(f)]\mathbb{P}_{\text{train}}(\hat{f}_{\text{train}} = f) = \mathbb{E}_{\text{train}}[\hat{\epsilon}_{\text{train}}(f)] \sum_{f\in\mathcal{F}} \mathbb{P}_{\text{train}}(\hat{f}_{\text{train}} = f).$$

The summation shown is the sum over the probabilities of all possible values of f which must sum to 1 given its a pmf so:

$$= \mathbb{E}[\hat{\epsilon}_{\text{train}}(f)]$$

hence we have proved

$$\mathbb{E}_{S_{\text{train}}}[\widehat{\epsilon}_{\text{train}}(\widehat{f}_{\text{train}})] \leq \mathbb{E}_{S_{\text{train}},S_{\text{test}}}[\widehat{\epsilon}_{\text{test}}(\widehat{f}_{\text{train}})].$$

4 Bias-Variance Tradeoff

B2.

(a) From the problem statement, we have an estimator $\widehat{f}_m(x)$ that partitions values of n into intervals and predicts the average of the observations within each interval. Figure 1 in the problem statement has a good representation of of this, we are using data points and our estimator as a step function estimator. For a **small** m this means the estimator would have less observations per interval for approximation and would predict the average of a much small interval. Therefore, I would expect there to be a **higher variance** and **lower bias**. Conversely, I would expect a **larger** m to have a **lower variance** and **higher bias**. The variation a single data point would have on the estimator is lower.

(b) The indicator function in the definition of $\widehat{f}_m(x)$ will be one when the x_i is in the $j-th$ interval and will otherwise be zero so let's write

$$\mathbb{E}[\widehat{f}_m(x_i)] = \mathbb{E}[c_j] = \mathbb{E}[\frac{1}{m}\sum_{i=(j-1)m+1}^{jm} y_i] = \frac{1}{m}\sum_{i=(j-1)m+1}^{jm}\mathbb{E}[y_i] = \frac{1}{m}\sum_{i=(j-1)m+1}^{jm} f(x_i) = \overline{f}^{(j)}.$$

Now let's look at the average-bias squared and decompose this into two summations over the intervals and then each of the data points in each interval. Also noting the previous equivalence we just obtained in the previous step. Then we have:

$$\frac{1}{n}\sum_{i=1}^{n}(\mathbb{E}[\widehat{f}_m(x_i)] - f(x_i))^2 = \frac{1}{n}\sum_{j=1}^{n/m}\sum_{i=(j-1)m+1}^{jm}(\mathbb{E}[\widehat{f}_m(x_i)] - f(x_i))^2 = \sum_{j=1}^{n/m}\sum_{i=(j-1)m+1}^{jm}(\overline{f}^{(j)} - f(x_i))^2.$$

(c) Looking at the defined average variance as two summations similar to the previous problem we have

$$\mathbb{E}[\frac{1}{n}\sum_{i=1}^{n}(\widehat{f}_m(x_i) - \mathbb{E}[\widehat{f}_m(x_i)])^2] = \frac{1}{n}\sum_{j=1}^{n/m}\sum_{i=(j-1)m+1}^{jm}\mathbb{E}[(\widehat{f}_m(x_i) - \mathbb{E}[\widehat{f}_m(x_i)])^2]$$

Now similar to the previous part we can show that for a fixed partition j and for all x_i in this partition we have $\widehat{f}_m(x_i) = c_j$. Also we showed $\mathbb{E}[\widehat{f}_m(x_i)] = \overline{f}^{(j)}$, so putting this together we write

$$= \frac{1}{n}\sum_{j=1}^{n/m} m\mathbb{E}[(\widehat{f}_m(x_i) - \mathbb{E}[\widehat{f}_m(x_i)])^2] = \frac{1}{n}\sum_{j=1}^{n/m} m\mathbb{E}[(c_j - \overline{f}^{(j)})^2]$$

thus we have shown the first equality. Furthermore, writing out c_j we get

$$= \frac{1}{n}\sum_{j=1}^{n/m} m\mathbb{E}[(\frac{1}{m^2}\sum_{i=(j-1)m+1}^{jm}(y_i - f(x_i))^2] = \frac{1}{mn}\sum_{j=1}^{n/m}\sum_{i=(j-1)m+1}^{jm}\mathbb{E}[(y_i - f(x_i))^2] = \frac{n\sigma^2}{mn} = \frac{\sigma^2}{m}.$$

(d) By the mean value theorem we know that $\widehat{f}^{(j)}$ is in an interval between the maximum and minimum values within its partition $f(x_i)$. As stated previously the average bias squared is

$$\sum_{j=1}^{n/m}\sum_{i=(j-1)m+1}^{jm}(\overline{f}^{(j)} - f(x_i))^2 = \sum_{j=1}^{n/m}\sum_{i=(j-1)m+1}^{jm}|\overline{f}^{(j)} - f(x_i)|^2$$

We know that both $f(x_i)$ and $\widehat{f}^{(j)}$ are bounded by the maximum and minimum value on the interval for $i \in \{(j-1)m+1, \cdots, jm\}$ for all j

$$\leq \frac{1}{n}\sum_{j=1}^{n/m}\sum_{i=(j-1)m+1}^{jm}|f(x_i)_{\min} - f(x_i)_{\max}|^2$$

And suppose f is L - Lipschitz as define in the problem statement we can write that

$$|f(x_i)_{\min} - f(x_i)_{\max}| \leq \frac{L}{n}|jm - ((j-1)m+1)| = \frac{L}{n}|m-1|$$

then plugging back in and simplifying

$$\begin{aligned} &\leq \frac{1}{n}\sum_{j=1}^{n/m}\sum_{i=(j-1)m+1}^{jm}(\frac{L}{n}|m-1|)^2 \\ &= \frac{L^2(m-1)^2}{n^3}\sum_{j=1}^{n/m}\sum_{i=(j-1)m+1}^{jm}1 \\ &= \frac{L^2(m-1)^2}{n^2} \leq \frac{L^2m^2}{n^2} \end{aligned}$$

Therefore the average bias-squared is $O(\frac{L^2m^2}{n^2})$.

Now considering the expression for average variance as above the total error behaves like $O(\frac{L^2m^2}{n^2} + \frac{\sigma^2}{m})$. To minimize lets take the derivative:

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

after algebra:

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

Parameters n and m are proportional, meaning increasing the amount of data decreases the error and increases m. Increasing σ which is the variance or noise, increases error and increases m. And decreasing L decreases the error and thus we'd increase m this is intuitive because the smoother a function is the easier it is for the estimator to fit the function.

---

5 Polynomial Regression

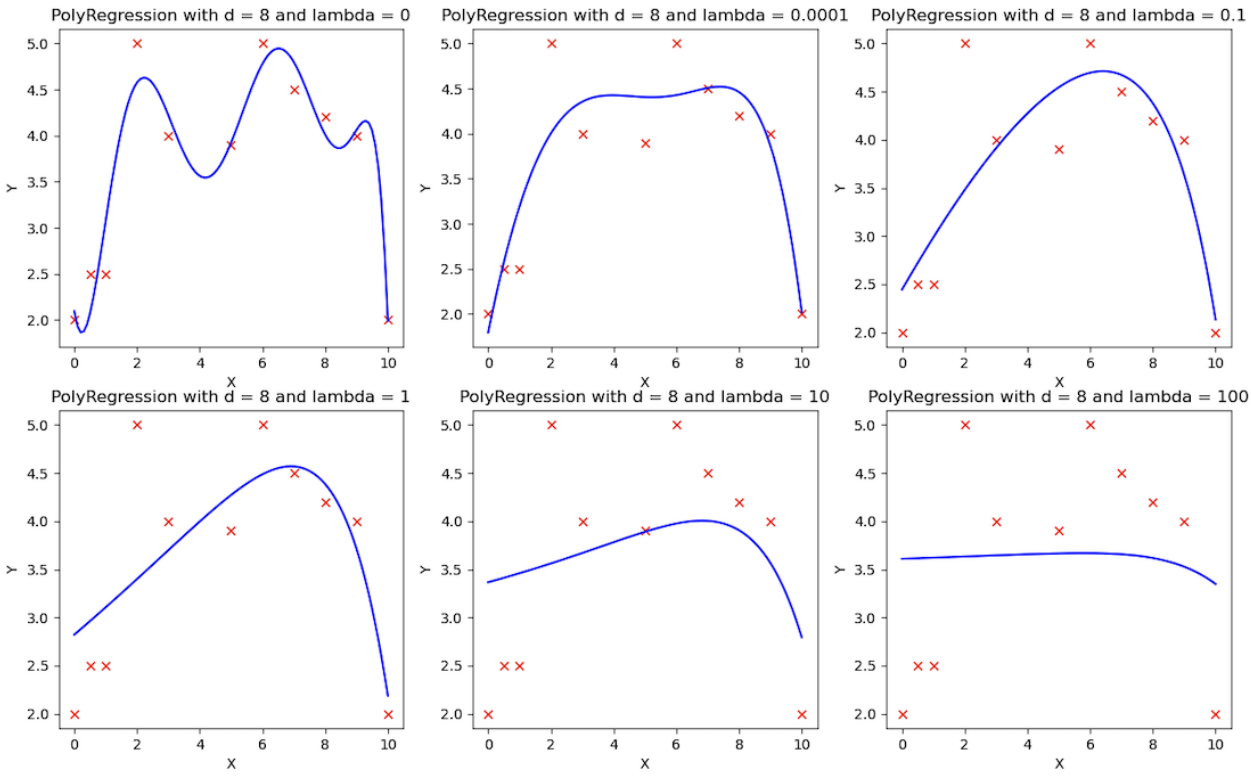
A3.

(a) Code:

```
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
        # Fill in with matrix with the correct shape
        self.weight: np.ndarray = None  # type: ignore
        # You can add additional fields
        self.mean = 0
        self.std = 0


    @staticmethod
```

```python
@problem.tag("hw1-A")
def polyfeatures(X: np.ndarray, degree: int) -> np.ndarray:

    poly_X = X

    for k in range (1, degree):
        poly_X = np.c_[poly_X, X**(k+1)]


    return poly_X


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

    poly_X = self.polyfeatures(X, self.degree)
    self.mean = np.mean(poly_X, axis=0)
    self.std = np.std(poly_X, axis=0)
    poly_X = (poly_X - self.mean) / self.std
    #column stack
    poly_X = np.c_[np.ones([len(poly_X), 1]), poly_X]
    n, d = poly_X.shape
    # closed form regression
    regMatrix = self.reg_lambda * np.eye(d)
    regMatrix[0,0] = 0
    # solution
    X = poly_X
    self.weight = np.linalg.pinv(X.T.dot(X) + regMatrix).dot(X.T).dot(y)


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

    poly_X = self.polyfeatures(X, self.degree)
    poly_X = (poly_X - self.mean) / self.std
    poly_X = np.c_[np.ones([len(poly_X), 1]), poly_X]

    sol = poly_X.dot(self.weight)

    return sol
```

(b) A large value for λ results increasing regularization, and as we can see in the plots the bias increases and the variance decreases. Meaning, if the regularization is too large then our model becomes too simple and we instead under-fit our data.

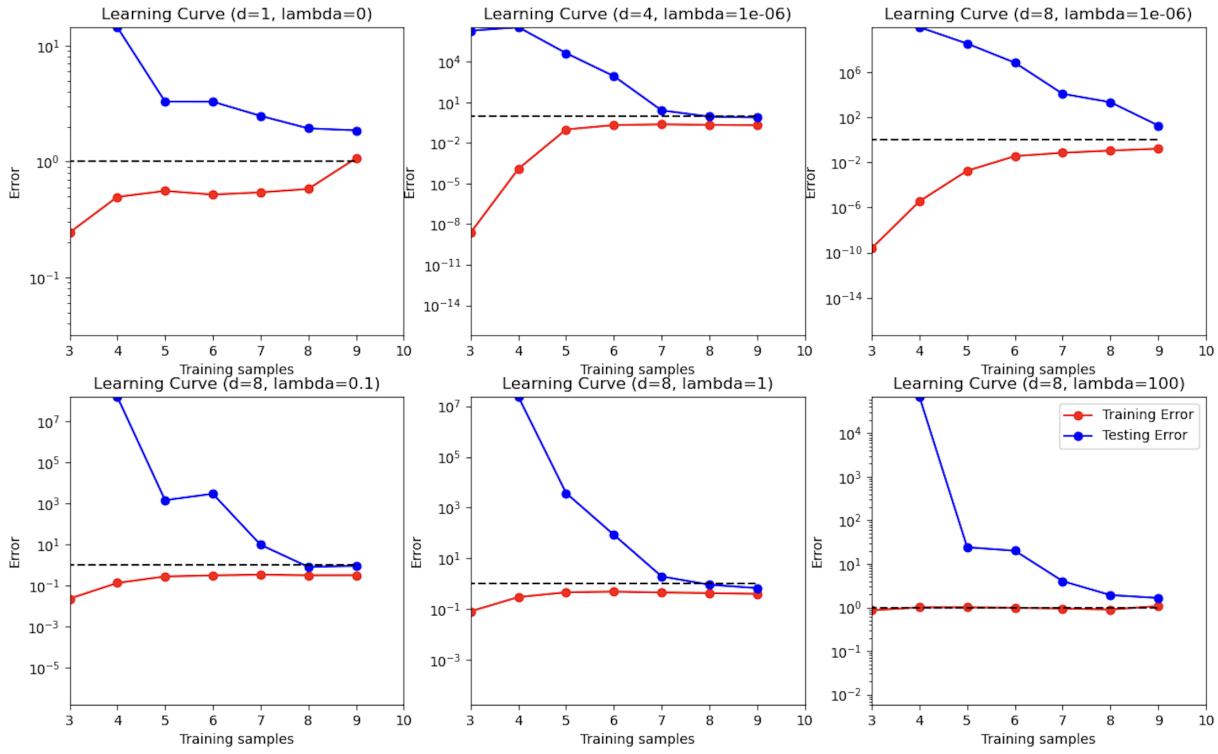


6 Learning Curve

A4.
Below are the plots for the learning curve and the corresponding code.

```python
@problem.tag("hw1-A")
def mean_squared_error(a: np.ndarray, b: np.ndarray) -> float:

    result = np.mean((a-b)**2)
    return result


@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)
    # Fill in errorTrain and errorTest arrays

    model = PolynomialRegression(degree=degree, reg_lambda=reg_lambda)
```

```
for i in range(1, n):
    train_observations = Xtrain[0:(i+1)]
    train_targets = Ytrain[0:(i+1)]

    model.fit(train_observations, train_targets)

    train_predict = model.predict(train_observations)
    test_predict = model.predict(Xtest)

    err_train = mean_squared_error(train_predict, train_targets)
    err_test = mean_squared_error(test_predict, Ytest)

    errorTrain[i] = err_train
    errorTest[i] = err_test

return errorTrain, errorTest
```



7 Ridge Regression

A5.

(a) From the problem statement we have

$$\widehat{W} = \text{argmin}_{W \in^{d \times k}} \sum_{i=1}^{n} \|W^T x_i - y_i\|_2^2 + \lambda \|W\|_F^2$$

Note that $\|W\|_F$ corresponds to the Frobenius norm of W, i.e. $\|W\|_F^2 = \sum_{i=1}^{d} \sum_{j=1}^{k} W_{i,j}^2$. To classify a point x_i we will use the rule $\arg\max_{j=0,\ldots,9} e_{j+1}^T \widehat{W}^T x_i$. Note that if $W = \begin{bmatrix} w_1 & \ldots & w_k \end{bmatrix}$

then

$$\begin{aligned}\sum_{i=1}^{n} \|W^T x_i - y_i\|_2^2 + \lambda \|W\|_F^2 &= \sum_{j=1}^{k} \left[\sum_{i=1}^{n} (e_j^T W^T x_i - e_j^T y_i)^2 + \lambda \|W e_j\|^2 \right] \\ &= \sum_{j=1}^{k} \left[\sum_{i=1}^{n} (w_j^T x_i - e_j^T y_i)^2 + \lambda \|w_j\|^2 \right] \\ &= \sum_{j=1}^{k} \left[\|X w_j - Y e_j\|^2 + \lambda \|w_j\|^2 \right]\end{aligned}$$

where $X = \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix}^\top \in^{n \times d}$ and $Y = \begin{bmatrix} y_1 & \dots & y_n \end{bmatrix}^\top \in^{n \times k}$. We want to show that

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

We'll start by taking the derivative of the last expression in the hint with respect to w and setting equal to zero, then solving further:

$$\begin{aligned}\frac{\partial}{\partial w_j} [\sum_{j=1}^{k} \|X w_j - Y e_j\|^2 + \lambda \|w_j\|^2] &= \frac{\partial}{\partial w_j} [(X w_j - y_j)^T (X w_j - y_j) + \lambda w^T w] \\ &= (2X^T X w_j - 2X^T y_j + 2\lambda w_j) \\ &= 2(X^T X w_j - X^T y_j + \lambda I w_j) \\ &= (X^T X w_j - X^T y_j + \lambda w_j) = 0 \\ &\implies (X^T X w_j + \lambda I w_j = (X^T y_j) \\ &= w_j = \frac{X^T y_j}{X^T X + \lambda I}\end{aligned}$$

Then we have completed the derivation:

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

(b) Code:

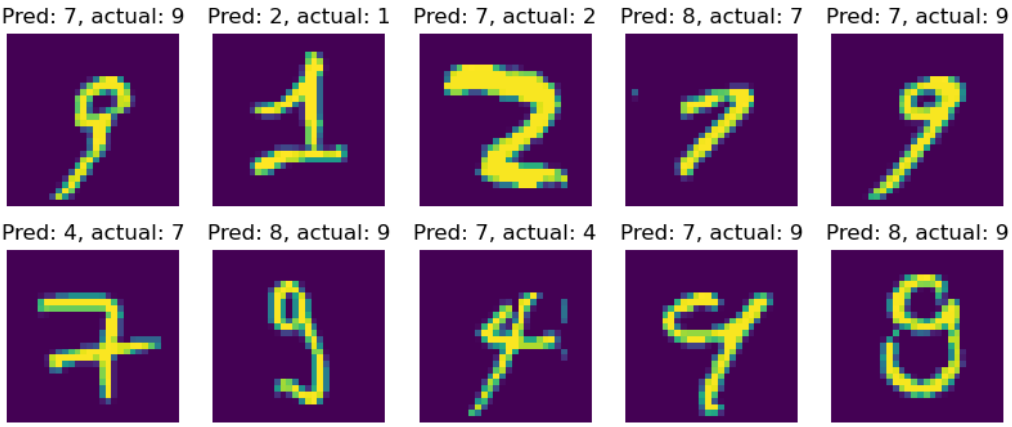
```
import numpy as np

from utils import load_dataset, problem


@problem.tag("hw1-A")
def train(x: np.ndarray, y: np.ndarray, _lambda: float) -> np.ndarray:

    rows, columns = x.shape
    a = x.T @ x + _lambda * np.eye(columns)
    b = x.T @ y
    w = np.linalg.solve(a,b)

    return w


@problem.tag("hw1-A")
def predict(x: np.ndarray, w: np.ndarray) -> np.ndarray:
```

```
    prediction = np.argmax(np.dot(x, w), axis=1)

    return prediction


@problem.tag("hw1-A")
def one_hot(y: np.ndarray, num_classes: int) -> np.ndarray:

    convert_array = np.zeros([y.shape[0], num_classes])

    for i in range(y.shape[0]):
        convert_array[i][y[i]] = 1

    return convert_array


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 and test errors are in the screenshot below:

(d) The picture below shows 10 images that were inaccurately predicted according to the label. After inspection we can see that the reason they are incorrectly predicted is because they are either poorly written or have attributions of other numbers. For one example, the 7 is written with

the line across the body, and this is a normal way humans write numbers however the machine mistakes this for a 4 which makes sense because a 4 is similarly written. The training data probably has "confuses" these similarities so our predictions are wrong.



8 Administrative

A6. The homework probably took around 15 hours to do.