Homework #1

CSE 546: Machine Learning

Ray Chen

Collaborator:

October 20, 2022

A1:

- **Parts a**
  Bias is error resulted by algorithm, or because of the estimation.
  Variance is error that because of fluctuation of the variables.
  Bias-Variance Tradeoff is get a balance between bias and variance. Because as bias increase, then variance decrease. Vice versa.

- **Parts b**
  Typically means it is under bias-variance tradeoff situation.
  When the model complexity increases, bias decrease, variance increase. Also, when the model complexity decreases, bias increase, variance decrease.

- **Parts c**
  False. Because more training data will help lower the variance instead of increasing the variance; and more Training data will help improve the algorithm.

- **Parts d** Validation and train data works. We can just adjust this parameter to get the best fit of our data.

- **Parts e**
  False. The training error should be the underestimate of the true error.

A2:

- **Part a:**

$$\begin{aligned}
\mathrm{P}(x|\lambda) &= \frac{e^{-\lambda}\lambda^{x}}{x!}\\
\mathrm{L}(x|\lambda) &= \prod_{i=1}^{n} \frac{e^{-\lambda}\lambda^{x_i}}{x_i!}\\
&= \ln(\sum_{i=1}^{n} \frac{e^{-\lambda}\lambda^{x_i}}{x_i!})\\
&= \sum_{i=1}^{n}(\ln(\lambda^{x_i}) + \ln(e^{-\lambda}) - \ln(x_i!))\\
&= \sum_{i=1}^{n}(x_i \ln(\lambda) - \lambda - \ln(x_i!))\\
&= \ln(\lambda)\sum_{i=1}^{n} x_i - n\lambda - \sum_{i=1}^{n}\ln(x_i!)\\
\frac{d}{d\lambda}\mathrm{L}(x|\lambda) &= \frac{d}{d\lambda}(\ln(\lambda)\sum_{i=1}^{n} x_i - n\lambda - \sum_{i=1}^{n}\ln(x_i!))\\
&= -n + \frac{1}{\lambda}\sum_{i=1} x_i
\end{aligned}$$

We set the previous derivative equal to zero, then we have

$$\begin{aligned}
0 &= -n + \frac{1}{\lambda}\sum_{i=1} x_i\\
\lambda &= \frac{1}{n}\sum_{i=1} x_i
\end{aligned}$$

- **Part b:**
  We plug in $x_i = [2, 4, 6, 0, 1]$ and $n = 5$

$$\begin{aligned}\lambda &= \frac{1}{5}(2+4+6+0+1) \\ &= \frac{13}{5} \\ &= 2.6\end{aligned}$$

  As $\lambda = 2.6$ and $x = 6$, we have

$$\begin{aligned}\mathbb{P}(x|\lambda) &= e^{-2.6}\frac{2.6^6}{6!} \\ &= 0.0318671\end{aligned}$$

- **Part c:**
  We plug in $x_i = [2, 4, 6, 0, 1, 8]$ and $n = 6$

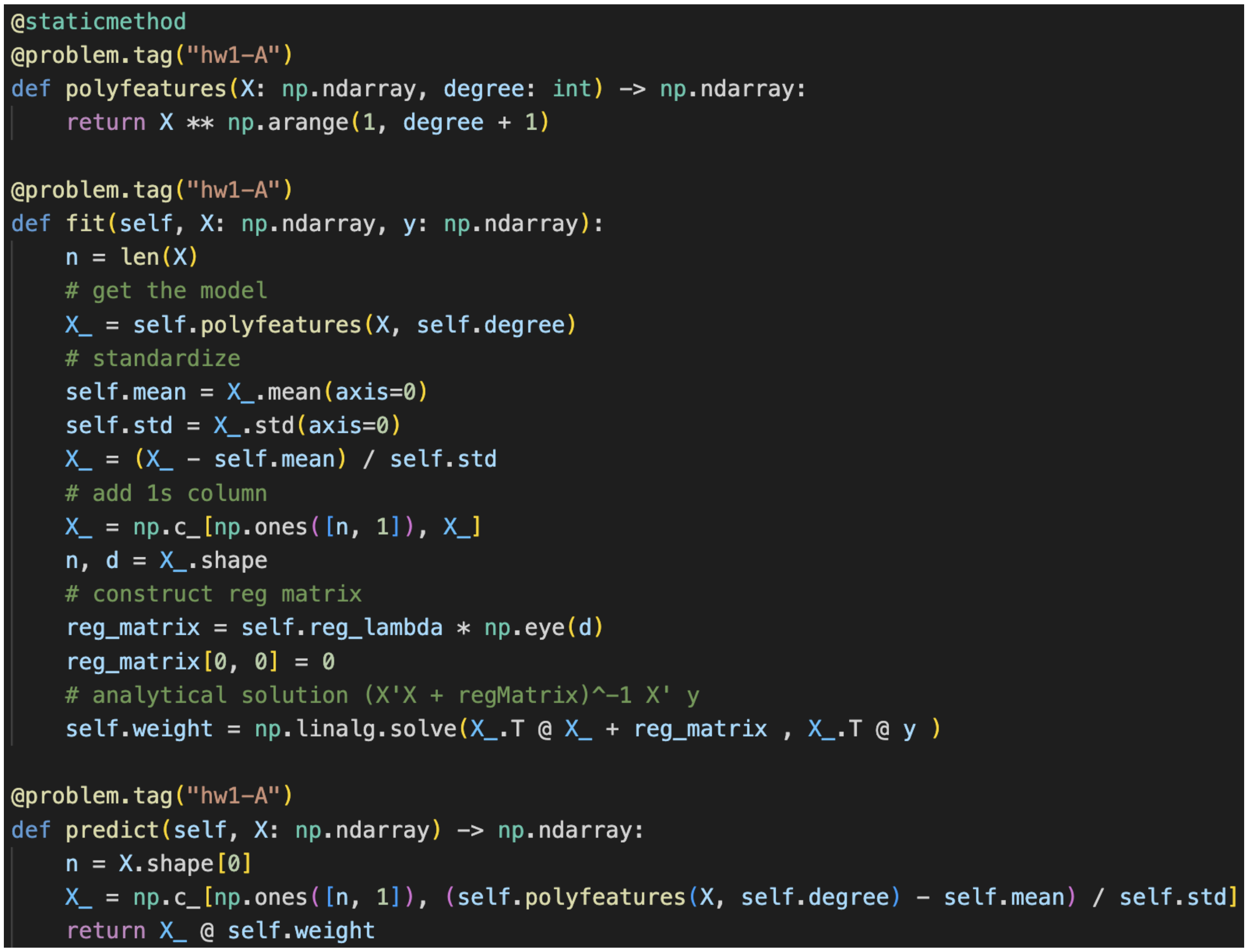
$$\begin{aligned}\lambda &= \frac{1}{6}(2+4+6+0+1+8) \\ &= \frac{21}{6} \\ &= 3.5\end{aligned}$$

  As $\lambda = 3.5$ and $x = 6$, we have

$$\begin{aligned}\mathbb{P}(x|\lambda) &= e^{-3.5}\frac{3.5^6}{6!} \\ &= 0.0770983\end{aligned}$$

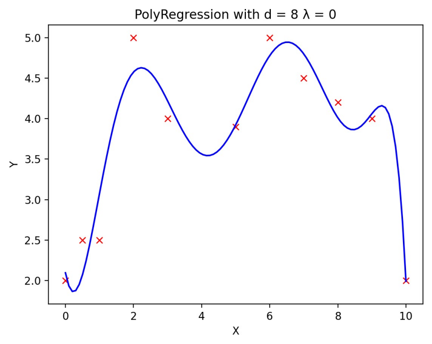
A3:

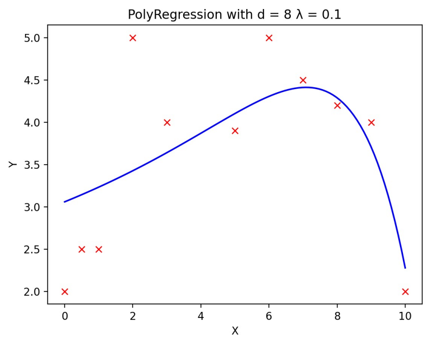
- **Part a:** Code on Gradescope through coding submission.

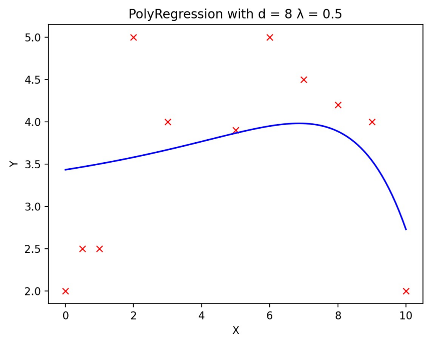


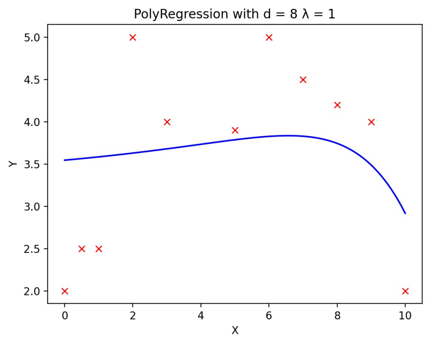
```
@staticmethod
@problem.tag("hw1-A")
def polyfeatures(X: np.ndarray, degree: int) -> np.ndarray:
    return X ** np.arange(1, degree + 1)

@problem.tag("hw1-A")
def fit(self, X: np.ndarray, y: np.ndarray):
    n = len(X)
    # get the model
    X_ = self.polyfeatures(X, self.degree)
    # standardize
    self.mean = X_.mean(axis=0)
    self.std = X_.std(axis=0)
    X_ = (X_ - self.mean) / self.std
    # 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 = X.shape[0]
    X_ = np.c_[np.ones([n, 1]), (self.polyfeatures(X, self.degree) - self.mean) / self.std]
    return X_ @ self.weight
```

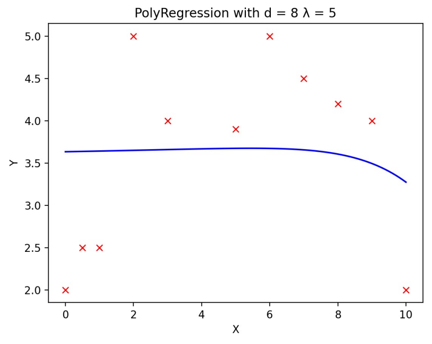
- **Part b:** 1-2 sentence description of the effect of increasing regularization. Plots before and after increase in regularization.

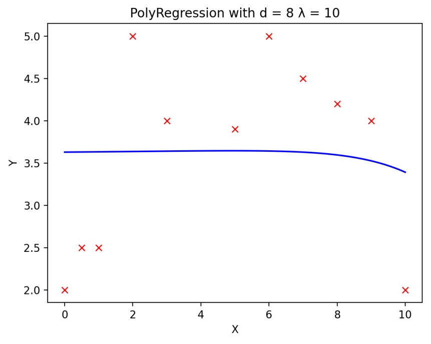


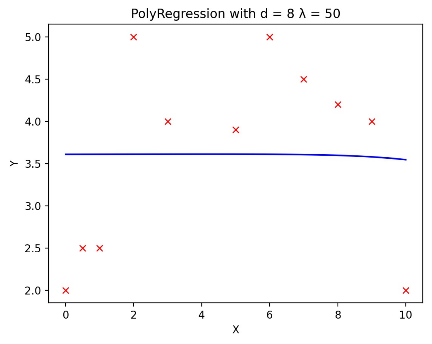


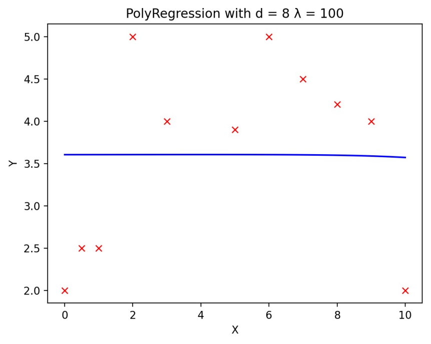








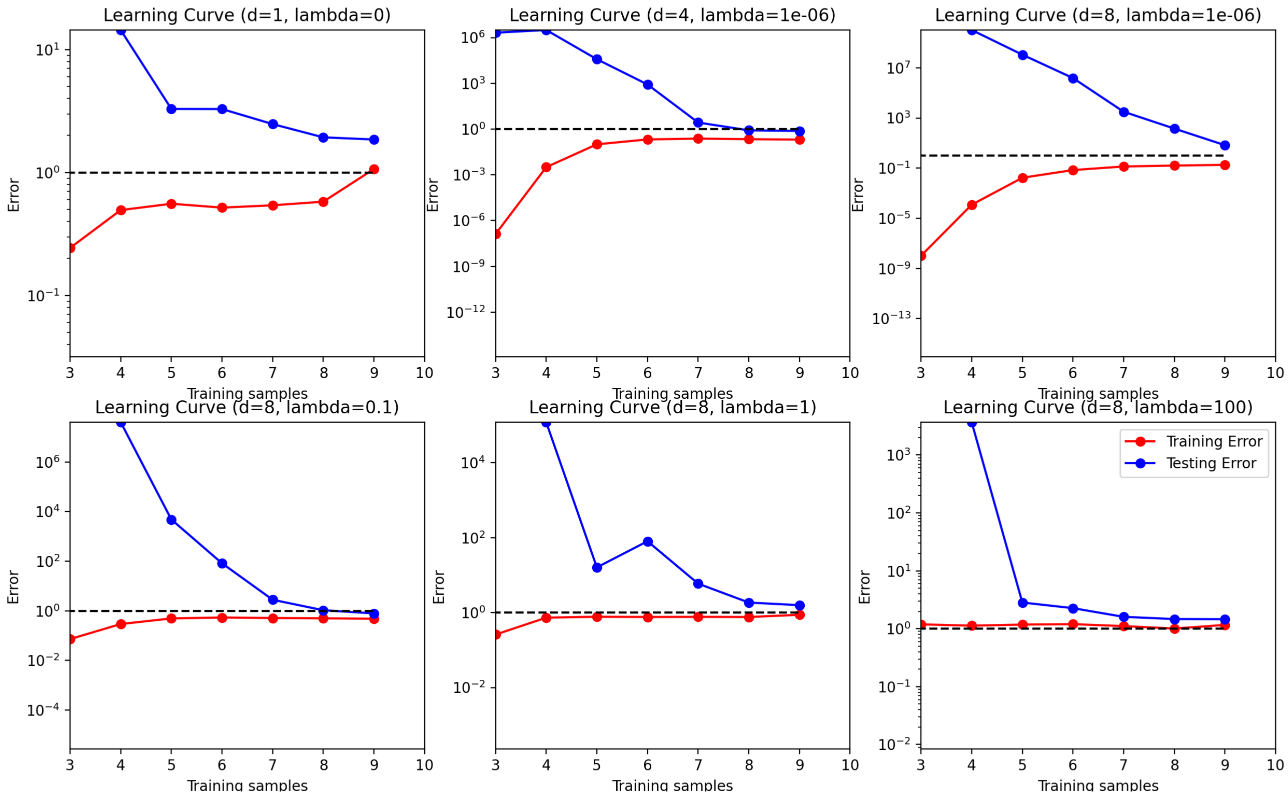




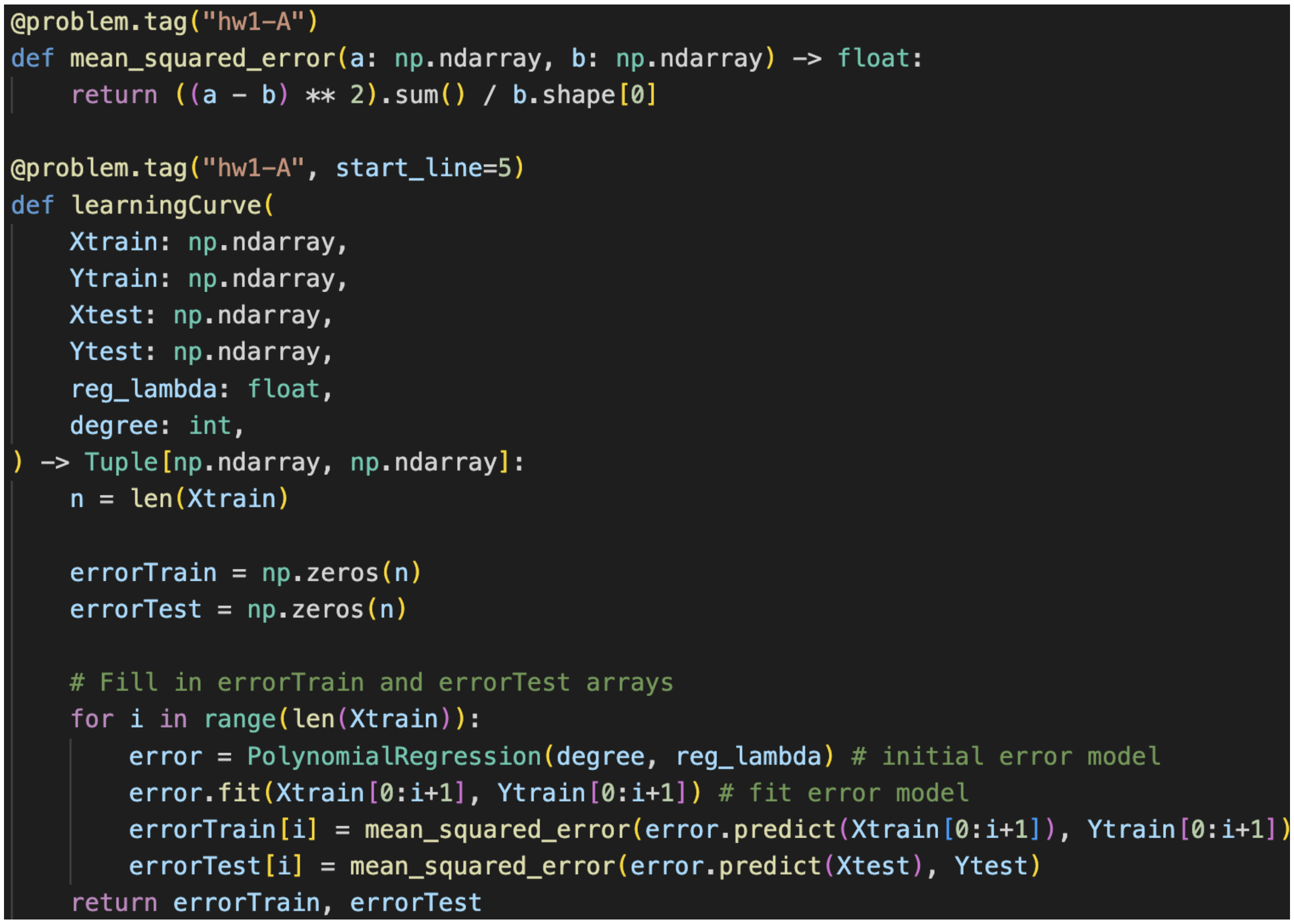
From $\lambda = 0$ to $\lambda = 100$ as the graph show above. As the λ gets bigger, the line become more flat.

A4:

- **Plots** (or single plot with many subplots) of learning curves for $(d, \lambda) \in \{(1, 0), (4, 10^{-6}), (8, 10^{-6}), (8, 0.1), (8, 1), (8, 100)\}$.



- **Code** on Gradescope through coding submission



```python
@problem.tag("hw1-A")
def mean_squared_error(a: np.ndarray, b: np.ndarray) -> float:
    return ((a - b) ** 2).sum() / b.shape[0]

@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
    for i in range(len(Xtrain)):
        error = PolynomialRegression(degree, reg_lambda) # initial error model
        error.fit(Xtrain[0:i+1], Ytrain[0:i+1]) # fit error model
        errorTrain[i] = mean_squared_error(error.predict(Xtrain[0:i+1]), Ytrain[0:i+1])
        errorTest[i] = mean_squared_error(error.predict(Xtest), Ytest)
    return errorTrain, errorTest
```

A5:

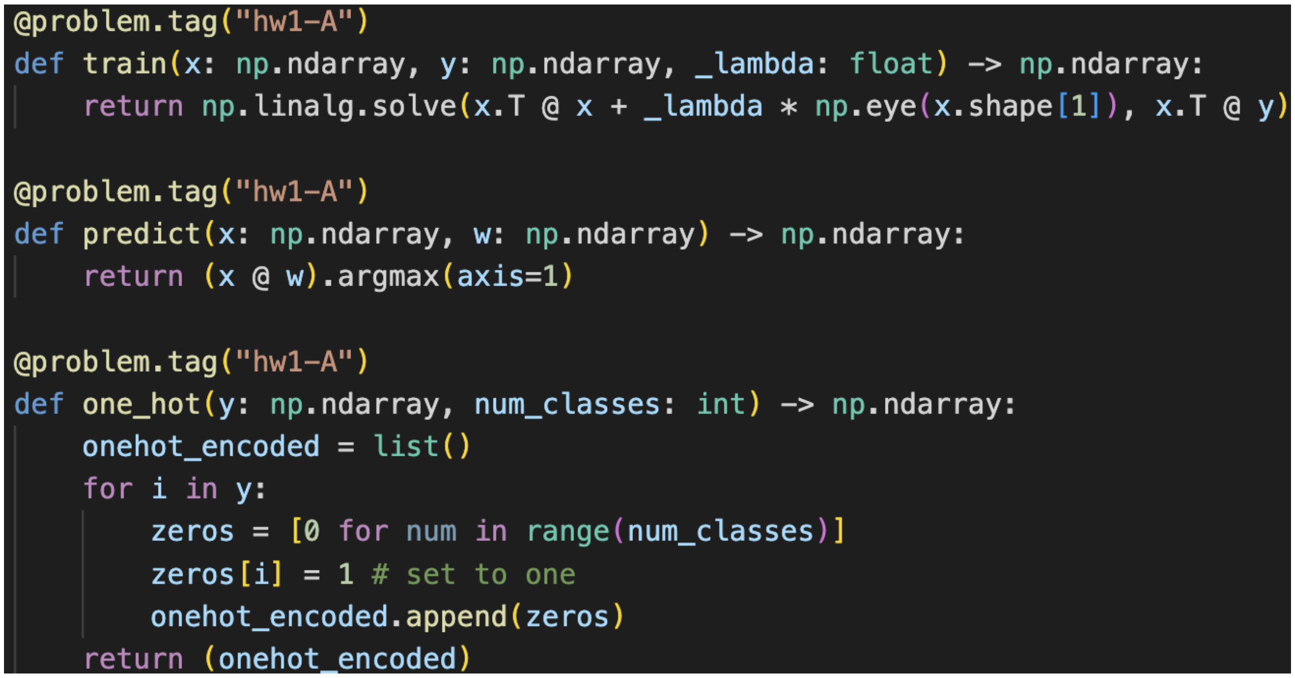
- **Part a:** Derivation of expression for $\widehat{W}$

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

We know that the derivation is of that is equls to zero, then we have:

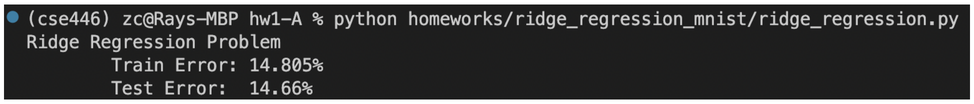
$$\begin{aligned}
0 &= \frac{d}{dw_j} \sum_{i=1}^{k} (||X w_i - y_i||_2^2 + \lambda ||w_i||_2^2) \\
0 &= \sum_{i=1}^{k} (2X^T (X w_i - Y e_i) + 2\lambda w_i) \\
\sum_{i=1}^{k} (2X^T X w_i + \lambda w_i) &= \sum_{i=1}^{k} (X^T Y e_i) \\
\sum_{i=1}^{k} w_i &= \sum_{i=1}^{k} (X^T X + \lambda I)^{-1} X^T Y e_i \qquad (Y e_i = y_i) \\
\widehat{W} &= (X^T X + \lambda I)^{-1} X^T Y
\end{aligned}$$

- **Part b: Code** on Gradescope through coding submission



```
@problem.tag("hw1-A")
def train(x: np.ndarray, y: np.ndarray, _lambda: float) -> np.ndarray:
    return np.linalg.solve(x.T @ x + _lambda * np.eye(x.shape[1]), x.T @ y)

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

@problem.tag("hw1-A")
def one_hot(y: np.ndarray, num_classes: int) -> np.ndarray:
    onehot_encoded = list()
    for i in y:
        zeros = [0 for num in range(num_classes)]
        zeros[i] = 1 # set to one
        onehot_encoded.append(zeros)
    return (onehot_encoded)
```

- **Part c:** Values of training and testing errors



```
(cse446) zc@Rays-MBP hw1-A % python homeworks/ridge_regression_mnist/ridge_regression.py
Ridge Regression Problem
        Train Error: 14.805%
        Test Error:  14.66%
```

A6:

It takes about 12 hours.