CSE546 - Homework # 1 - Solutions

Cheng-Yen Yang

April 23, 2021

Problem A.0

a.

Bias is the expected difference between the predictions or estimates of our model and the ground-truth values from the data, or in my own words, how **far** it is off the target. **Variance** is the expected value of the squared difference between the estimates of our model and the expected value of the estimate, or in my own words, how **much** the model will changes when we retrain the model using a new dataset with the same distribution.**Bias-Variance Trade-off** is the effect that when a model have a lower bias it is often found with a higher variance or vice versa. This is because that the error can be derived into bias squared-and variance error through bias-variance decomposition which implies that there is a trade-off effect between bias and variance in the models.

b. Complexity $\uparrow$ Bias $\downarrow$ Variance $\uparrow$ / Complexity $\downarrow$ Bias $\uparrow$ Variance $\downarrow$

Typically, as the model complexity increases, the bias tends to decrease and the variance tends to increase; as the model complexity decreases, the bias tends to increase and the variance tends to decrease.

c. False.

The bias is not likely to be affect by the number of data we use in training, the number of data will direct influence towards variance.

d. True.

The variance of a model decreases as the amount of training data available increases.

e. True.

A learning algorithm will always generalize better if we use fewer features to represent our data.

f. Train set.

Ones should not use the test set for tuning at any circumstances during the training stage. Making a proper validation set from the original train set is also one of the option if we want to obtain better performance on unseen data.

g. False.

The training error of a function on the training set usually provides an underestimate of the true error of that function.

Problem A.1

a.

Since the log of the likelihood function $logL(\lambda)$ has the same maximum as the likelihood function $L(\lambda)$ itself, we have:

$$L(\lambda) = \prod_{i=1}^{n} \text{Poisson}(x_i|\lambda) = \prod_{i=1}^{n} e^{-\lambda} \frac{\lambda^{x_i}}{x_i!} \tag{1}$$

and,

$$log\left(L(\lambda)\right) = log\left(\prod_{i=1}^{n} e^{-\lambda} \frac{\lambda^{x_i}}{x_i!}\right) \tag{2}$$

$$= \sum_{i=1}^{n} log\left(e^{-\lambda} \frac{\lambda^{x_i}}{x_i!}\right) \tag{3}$$

$$= \sum_{i=1}^{n} \left(-\lambda + x_i log(\lambda) - log(x_i!)\right). \tag{4}$$

By making the derivative of the log of likelihood function 0:

$$\frac{d}{d\lambda} log\left(L(\lambda)\right) = \sum_{i=1}^{n} \left(-1 + \frac{x_i}{\lambda}\right) = 0, \tag{5}$$

we have:

$$\lambda = \sum_{i=1}^{n} \frac{x_i}{n}. \tag{6}$$

As for the case in A.1.a, the maximum-likelihood estimate of λ is:

$$\lambda = \sum_{i=1}^{5} \frac{x_i}{5} = \frac{1}{5}(x_1 + x_2 + x_3 + x_4 + x_5) \tag{7}$$

b.

Following similar procedure in A.1.a, as for the case in A.1.b, the maximum-likelihood estimate of λ is:

$$\lambda = \sum_{i=1}^{6} \frac{x_i}{6} = \frac{1}{6}(x_1 + x_2 + x_3 + x_4 + x_5 + x_6) \tag{8}$$

c.

By using the x_i given in the problem statement, we have $\lambda_{5+} = 13/5$ and $\lambda_{6+} = 16/6 = 8/3$.

Problem A.2

The maximum likelihood estimate for θ can be express as:

$$\theta_{MLE} = \text{argmax} \prod_{i=1}^{n} \frac{1}{\theta} \mathbf{1}\{x \in [0, \theta]\} \tag{9}$$

$$= \frac{1}{\theta^n} \tag{10}$$

so we need to pick the smallest possible θ which satisfy $x_i \in [0, \theta]$ for all i:

$$\theta_{MLE} = max(x_i). \tag{11}$$

Problem A.3

a.

To prove that $\mathbb{E}_{\text{train}}[\widehat{\epsilon}_{\text{train}}(f)] = \mathbb{E}_{\text{test}}[\widehat{\epsilon}_{\text{test}}(f)] = \epsilon(f)$, we begin by showing that the train error:

$$\mathbb{E}_{\text{train}}[\widehat{\epsilon}_{\text{train}}(\widehat{f})] = \mathbb{E}\left[\frac{1}{N_{\text{train}}} \sum_{(x,y)\in S_{\text{train}}} \left((f(x)-y)^2\right)\right] \tag{12}$$

$$= \frac{1}{N_{\text{train}}} \sum_{(x,y)\in S_{\text{train}}} \mathbb{E}\left[(f(x)-y)^2\right], \tag{13}$$

$$\tag{14}$$

supposed that the training set S_{train} and the test set S_{test} are drawn i.i.d. from the underlying distribution $\mathcal{D}$. We can actually see the training set as sampling N_{train} times from the distribution $\mathcal{D}$ which gives us:

$$\mathbb{E}_{\text{train}}[\widehat{\epsilon}_{\text{train}}(f)] = \frac{1}{N_{\text{train}}} \sum_{(x,y)\in S_{\text{train}}} \mathbb{E}\left[(f(x)-y)^2\right] \tag{15}$$

$$= \frac{1}{N_{\text{train}}} N_{\text{train}} \sum_{(x,y)\sim D} \mathbb{E}\left[(f(x)-y)^2\right] \tag{16}$$

$$= \epsilon(f). \tag{17}$$

And the same logic is used for the test error:

$$\mathbb{E}_{\text{test}}[\widehat{\epsilon}_{\text{test}}(f)] = \mathbb{E}\left[\frac{1}{N_{\text{test}}} \sum_{(x,y)\in S_{\text{test}}} \left((f(x)-y)^2\right)\right] \tag{18}$$

$$= \frac{1}{N_{\text{test}}} \sum_{(x,y)\in S_{\text{test}}} \mathbb{E}\left[(f(x)-y)^2\right] \tag{19}$$

$$= \frac{1}{N_{\text{test}}} N_{\text{test}} \sum_{(x,y)\sim D} \mathbb{E}\left[(f(x)-y)^2\right] \tag{20}$$

$$= \epsilon(f). \tag{21}$$

Finally from the above derivation, we let $f = \widehat{f}$, we have:

$$\mathbb{E}_{\text{test}}[\widehat{\epsilon}_{\text{test}}(\widehat{f})] = \epsilon(\widehat{f}) \tag{22}$$

which the test error is an unbiased estimate of our true error for $\widehat{f}$.

b.

No, the above equation will not be true with regards to the training set, which means $\mathbb{E}_{\text{train}}[\widehat{\epsilon}_{\text{train}}(\widehat{f})] \neq \epsilon(\widehat{f})$. To be more specific, Eq. 16 will not hold as $(f(x_i) - y_i)^2$, or (x_i, y_i), are not guaranteed as the are no longer independent to $\widehat{f}$ when $(x_i, y_i) \in S_{\text{train}}$.

c.

From Problem A.2.a we have:

$$\mathbb{E}_{\text{train}}[\widehat{\epsilon}_{\text{train}}(f)] = \mathbb{E}_{\text{test}}[\widehat{\epsilon}_{\text{test}}(f)],\ \forall\ f \in \mathcal{F}, \tag{23}$$

and from Problem A.2.c's hint we have:

$$\mathbb{E}_{\text{train,test}}[\widehat{\epsilon}_{\text{test}}(\widehat{f}_{\text{train}})] = \sum_{f \in \mathcal{F}} \mathbb{E}_{\text{test}}[\widehat{\epsilon}_{\text{test}}(f)]\mathbb{P}_{\text{train}}(\widehat{f}_{\text{train}} = f). \tag{24}$$

Together we can derive:

$$\begin{aligned}
\mathbb{E}_{\text{train,test}}[\widehat{\epsilon}_{\text{test}}(\widehat{f}_{\text{train}})] &= \sum_{f \in \mathcal{F}} \mathbb{E}_{\text{train}}[\widehat{\epsilon}_{\text{train}}(f)]\mathbb{P}_{\text{train}}(\widehat{f}_{\text{train}} = f) && (25)\\
&= \mathbb{E}_{\text{train}}[\widehat{\epsilon}_{\text{train}}(f)], && (26)
\end{aligned}$$

and in general we know that $\mathbb{E}_{\text{train}}[\widehat{\epsilon}_{\text{train}}(f)] \geq \mathbb{E}_{\text{train}}[\widehat{\epsilon}_{\text{train}}(\widehat{f})]$ as $\widehat{f}_{\text{train}}$ minimize the training error such that $\widehat{\epsilon}_{\text{train}}(\widehat{f}_{\text{train}}) \leq \widehat{\epsilon}_{\text{train}}(f)$ for all $f \in \mathcal{F}$ which complete the last portion of the inequality equation.

Problem B.1

a.

For some $m \le n$ such that n/m is an integer define the step function estimator:

$$\widehat{f}_m(x) = \sum_{j=1}^{n/m} c_j \mathbf{1}\{x \in \left(\frac{(j-1)m}{n}, \frac{jm}{n}\right]\} \quad \text{where} \quad c_j = \frac{1}{m} \sum_{i=(j-1)m+1}^{jm} y_i. \tag{27}$$

By the bias-variance decomposition at some x_i we have

$$\mathbb{E}\left[(\widehat{f}_m(x_i) - f(x_i))^2\right] = \underbrace{(\mathbb{E}[\widehat{f}_m(x_i)] - f(x_i))^2}_{\text{Bias}^2(x_i)} + \underbrace{\mathbb{E}\left[(\widehat{f}_m(x_i) - \mathbb{E}[\widehat{f}_m(x_i)])^2\right]}_{\text{Variance}(x_i)} \tag{28}$$

So intuitively, for a very large m, it is expected to have high bias and low variance (e.g. we choose $m = n$, the model will predict the exact average of all samples). For a very small m, it is expected to have low bias and high variance (e.g. we choose $m = 1$, the model will predict the exact value if falls into the interval of the single sample).

b.

We first expand the summation of average bias-squared into:

$$\frac{1}{n}\sum_{i=1}^{n}(\mathbb{E}[\widehat{f}_m(x_i)] - f(x_i))^2 = \frac{1}{n}\sum_{j=1}^{n/m}\sum_{i=jm-m+1}^{jm}(\mathbb{E}[\widehat{f}_m(x_i)] - f(x_i))^2, \tag{29}$$

then from Eq. (27) we know that the estimator will predicts the average of the observations within each interval partitioned by m and j is the index of the corresponding partition which gives:

$$\mathbb{E}[\widehat{f}_m(x_i)] = \mathbb{E}[c_j] = \mathbb{E}\left[\frac{1}{m}\sum_{i=jm-m+1}^{jm} y_i\right] = \frac{1}{m}\sum_{i=jm-m+1}^{jm}\mathbb{E}[y_i] = \frac{1}{m}\sum_{i=jm-m+1}^{jm} f(x_i) = \bar{f}^{(j)}. \tag{30}$$

Combining Eq. (26) and (27), we have:

$$\frac{1}{n}\sum_{i=1}^{n}(\mathbb{E}[\widehat{f}_m(x_i)] - f(x_i))^2 = \frac{1}{n}\sum_{j=1}^{n/m}\sum_{i=(j-1)m+1}^{jm}(\bar{f}^{(j)} - f(x_i))^2 \tag{31}$$

c.

We first expand the summation of average variance into:

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(\widehat{f}_m(x_i)-\mathbb{E}[\widehat{f}_m(x_i)])^2\right]=\mathbb{E}\left[\frac{1}{n}\sum_{j=1}^{n/m}\sum_{i=jm-m+1}^{jm}(\widehat{f}_m(x_i)-\mathbb{E}[\widehat{f}_m(x_i)])^2\right] \tag{32}$$

then from Eq. (30) we have $\mathbb{E}[\widehat{f}_m(x_i)]=\mathbb{E}[c_j]$ and the fact that $\widehat{f}_m(x_i)$ will be identical across all if the x_i terms in this given j-th partition:

$$\begin{aligned}\mathbb{E}\left[\frac{1}{n}\sum_{j=1}^{n/m}\sum_{i=jm-m+1}^{jm}(\widehat{f}_m(x_i)-\mathbb{E}[\widehat{f}_m(x_i)])^2\right]&=\frac{1}{n}\sum_{j=1}^{n/m}\sum_{i=jm-m+1}^{jm}\mathbb{E}\left[(\widehat{f}_m(x_i)-\mathbb{E}[\widehat{f}_m(x_i)])^2\right] &(33)\\ &=\frac{1}{n}\sum_{j=1}^{n/m}\sum_{i=jm-m+1}^{jm}\mathbb{E}\left[(c_j-\mathbb{E}[c_j])^2\right] &(34)\\ &=\frac{1}{n}\sum_{j=1}^{n/m}m\mathbb{E}\left[(c_j-\bar{f}^{(j)})^2\right] &(35)\\ &=\frac{\sigma^2}{m}. &(36)\end{aligned}$$

Then

$$\mathbb{E}\left[\frac{1}{n}\sum_{j=1}^{n/m}\sum_{i=jm-m+1}^{jm}(\widehat{f}_m(x_i)-\mathbb{E}[\widehat{f}_m(x_i)])^2\right]=\frac{1}{n}\sum_{j=1}^{n/m}\sum_{i=jm-m+1}^{jm}\mathbb{E}\left[(\widehat{f}_m(x_i)-\mathbb{E}[\widehat{f}_m(x_i)])^2\right] \tag{37}$$

$$\tag{38}$$

d.

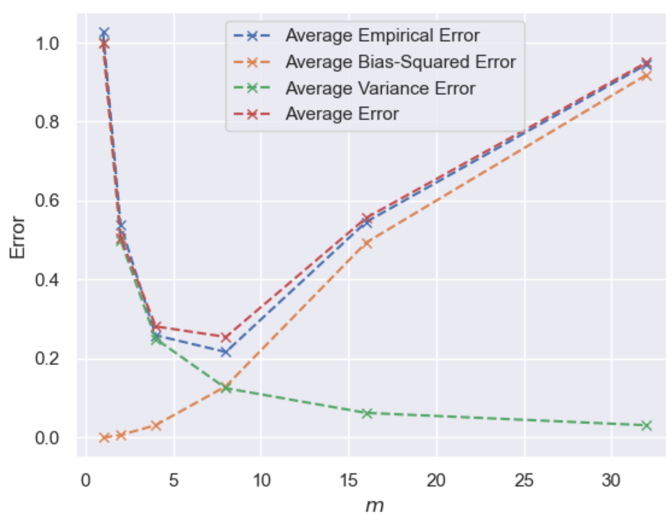


Figure 1: Plot of different error under different m value.

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

sns.set()

n = 256
sigma = 1
M = [1, 2, 4, 8, 16, 32]
eps = np.random.normal(0, sigma, n)

X = np.arange(1, n+1) / n
Y = 4 * np.sin(np.pi * X) * np.cos(6 * np.pi * X * X) + eps
Y_true = 4 * np.sin(np.pi * X) * np.cos(6 * np.pi * X * X)

# average empirical error
average_empirical_error = []
for m in M:
    f_m = np.array([np.mean(Y[j*m-m:j*m]) for j in np.arange(1, n//m + 1)])
    f_hat = np.array([f_m[int((i-1)//m)] for i in np.arange(1, n+1)])
    err = np.mean((f_hat - Y_true)**2)
    average_empirical_error.append(err)

# average bias-squared error
average_bias_squared_error = []
for m in M:
    f_j = np.array([np.mean(Y_true[j * m - m:j * m]) for j in np.arange(1, n // m + 1)])
    f_hat = np.array([f_j[int((i - 1) // m)] for i in np.arange(1, n + 1)])
    err = np.mean((f_hat - Y_true) ** 2)
    average_bias_squared_error.append(err)

# average variance error
average_variance_error = []
for m in M:
    average_variance_error.append(sigma**2/m)

plt.plot(M, average_empirical_error, 'x--', label='Average Empirical Error')
plt.plot(M, average_bias_squared_error, 'x--', label='Average Bias-Squared Error')
plt.plot(M, average_variance_error, 'x--', label='Average Variance Error')
plt.plot(M, np.array(average_bias_squared_error) + np.array(average_variance_error), 'x--', label='Average Error')
plt.legend()
plt.xlabel('$m$')
plt.ylabel('Error')
plt.savefig('B-1-4.png')
```

Figure 2: Code used to generate Fig.8.

e.

From B.1.b, we have the average bias-squared and we can derived:

$$\frac{1}{n}\sum_{i=1}^{n}(\widehat{f}^{(j)} - f(x_i))^2 \leq \frac{1}{n}\sum_{i=1}^{n}(\min f(x_i) - f(x_i))^2 \tag{39}$$

$$\leq \left(\frac{L}{n}\right)^2 (\operatorname{argmin} f(x_i) - i)^2 \tag{40}$$

$$\leq \frac{L^2}{n^2}m^2 \sim O(\frac{L^2m^2}{n^2}) \tag{41}$$

Problem A.4

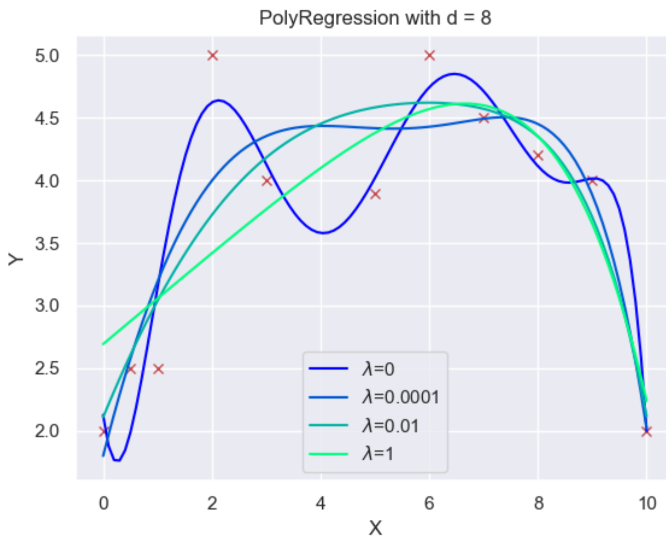


Figure 3: Plot of PolyRegression with $d = 8$ under different λ.

```python
class PolynomialRegression:

    def __init__(self, degree=1, reg_lambda=1E-8):
        """..."""
        self.degree = degree
        self.reg_lambda = reg_lambda
        self.theta = None
        self.mean = None
        self.std = None

    def polyfeatures(self, X, degree):
        """..."""
        poly_feat_X = X
        for i in range(1, self.degree):
            poly_feat_X = np.c_[poly_feat_X, X**i]
        assert poly_feat_X.shape == (X.shape[0], self.degree)
        return poly_feat_X

    def fit(self, X, y):
        """..."""
        n = len(X)

        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
        X_ = np.c_[np.ones([n, 1]), X_]

        n, d = X_.shape

        reg_matrix = self.reg_lambda * np.eye(d)
        reg_matrix[0, 0] = 0

        self.theta = np.linalg.pinv(X_.T.dot(X_) + reg_matrix).dot(X_.T).dot(y)

    def predict(self, X):
        """..."""
        n, _ = X.shape
        X_ = self.polyfeatures(X, self.degree)
        X_ = (X_ - self.mean) / self.std
        X_ = np.c_[np.ones([n, 1]), X_]

        return X_.dot(self.theta)
```

Figure 4: Screenshot of the polyreg.py.

```python
if __name__ == "__main__":
    '''...'''

    # load the data
    filePath = "data/polydata.dat"
    file = open(filePath,'r')
    allData = np.loadtxt(file, delimiter=',')
    X = allData[:, [0]]
    y = allData[:, [1]]

    # regression with degree = d
    d = 8
    model = PolynomialRegression(degree=d, reg_lambda=0)
    model.fit(X, y)

    # output predictions
    xpoints = np.linspace(np.max(X), np.min(X), 100).reshape(-1, 1)
    ypoints = model.predict(xpoints)

    # plot curve
    plt.figure()
    plt.plot(X, y, 'rx')
    plt.plot(xpoints, ypoints, color=cmap(0), label='$lambda$=0')

    model = PolynomialRegression(degree=d, reg_lambda=1e-4)
    model.fit(X, y)
    ypoints = model.predict(xpoints)
    plt.plot(xpoints, ypoints, color=cmap(0.35), label='$lambda$=0.0001')

    model = PolynomialRegression(degree=d, reg_lambda=1e-2)
    model.fit(X, y)
    ypoints = model.predict(xpoints)
    plt.plot(xpoints, ypoints, color=cmap(0.7), label='$lambda$=0.01')

    model = PolynomialRegression(degree=d, reg_lambda=1)
    model.fit(X, y)
    ypoints = model.predict(xpoints)
    plt.plot(xpoints, ypoints, color=cmap(1.2), label='$lambda$=1')

    plt.title('PolyRegression with d = ' + str(d))
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.legend()
    plt.savefig('A_4.png')
```

Figure 5: Screenshot of the test_polyreg_univariate.py.

Problem A.5

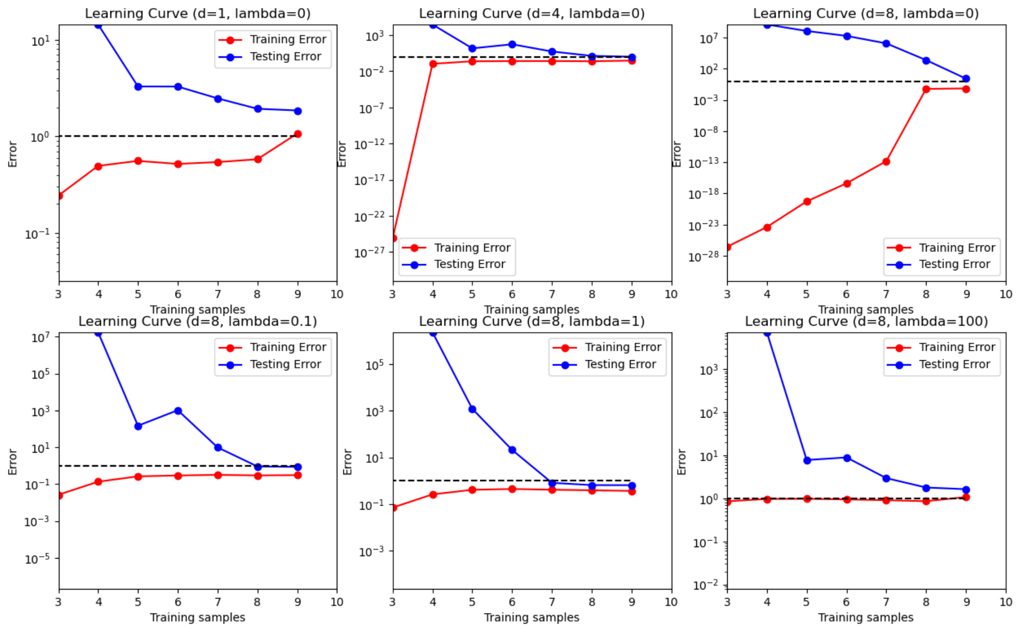


Figure 6: Plot of PolyRegression with $d = 8$ under different λ.

```python
def learningCurve(Xtrain, Ytrain, Xtest, Ytest, reg_lambda, degree):
    """Compute learning curve..."""

    n = len(Xtrain)

    errorTrain = np.zeros(n)
    errorTest = np.zeros(n)

    #TODO -- complete rest of method; errorTrain and errorTest are already the correct shape
    assert Xtrain.shape == Ytrain.shape
    assert Xtest.shape == Ytest.shape

    model = PolynomialRegression(degree=degree, reg_lambda=reg_lambda)
    for i in range(1, n):
        X = Xtrain[0:i+1]
        y = Ytrain[0:i+1]

        model.fit(X,y)

        ytrain = model.predict(X)
        ytest = model.predict(Xtest)

        errorTrain[i] = np.mean((ytrain-y)**2)
        errorTest[i] = np.mean((ytest-Ytest)**2)

    return errorTrain, errorTest
```

Figure 7: Screenshot of the polyreg.py.

Problem A.6

a.

In this problem we will choose a linear classifier to minimize the regularized least squares objective:

$$\widehat{W} = \text{argmin}_{W \in \mathbb{R}^{d\times k}} \sum_{i=1}^{n} \|W^T x_i - y_i\|_2^2 + \lambda \|W\|_F^2, \tag{42}$$

and from the problem we can deduce the term we ought to minimize:

$$\begin{aligned}
\sum_{i=1}^{n} \|W^T x_i - y_i\|_2^2 + \lambda \|W\|_F^2 &= \sum_{j=1}^{k} \left[\sum_{i=1}^{n} (e_j^T W^T x_i - e_j^T y_i)^2 + \lambda \|W e_j\|^2 \right] &(43)\\
&= \sum_{j=1}^{k} \left[\sum_{i=1}^{n} (w_j^T x_i - e_j^T y_i)^2 + \lambda \|w_j\|^2 \right] &(44)\\
&= \sum_{j=1}^{k} \left[\|X w_j - Y e_j\|^2 + \lambda \|w_j\|^2 \right], &(45)
\end{aligned}$$

where $X = \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix}^\top \in \mathbb{R}^{n\times d}$ and $Y = \begin{bmatrix} y_1 & \dots & y_n \end{bmatrix}^\top \in \mathbb{R}^{n\times k}$. So we take the derivative of the term and set the value to zero to find the minimizer of the regularized least squares objective:

$$\frac{d}{dw_j} \sum_{j=1}^{k} \left[\|X w_j - Y e_j\|^2 + \lambda \|w_j\|^2 \right] = 0 \tag{46}$$

$$\sum_{j=1}^{k} \left[2X^T (X w_j - Y e_j) + 2\lambda w_j \right] = 0 \tag{47}$$

$$\sum_{j=1}^{k} \left[X^T X w_j - X^T Y e_j + \lambda w_j \right] = 0 \tag{48}$$

$$\sum_{j=1}^{k} X^T X w_j + \lambda w_j = \sum_{j=1}^{k} X^T Y e_j \tag{49}$$

$$\sum_{j=1} w_j = \sum_{j=1}^{k} (X^T X + \lambda I)^{-1} X^T Y e_j \tag{50}$$

Finally we know that $\widehat{W}$ can be view as a matrix formed by w_j and the fact that $Y e_j = y_j$, so in the last equation, the summation sum over k making that the minimizer for this problem:

$$\widehat{W} = \text{argmin}_{W \in \mathbb{R}^{d\times k}} \sum_{i=1}^{n} \|W^T x_i - y_i\|_2^2 + \lambda \|W\|_F^2 = (X^T X + \lambda I)^{-1} X^T Y \tag{51}$$

b.

After we train $\widehat{W}$ on the MNIST training data with $\lambda = 0.0001$, the training error is 14.81% and testing error is 14.66%.

```python
import ...


def load_dataset():
    mndata = MNIST('./data/')
    X_train, Y_train = map(np.array, mndata.load_training())
    X_test, Y_test = map(np.array, mndata.load_testing())
    X_train = X_train / 255.0
    X_test = X_test / 255.0

    return X_train, Y_train, X_test, Y_test


def train(X, Y, reg_lambda=1e-16):
    I = np.eye(X.shape[1])
    # https://numpy.org/doc/stable/reference/generated/numpy.linalg.solve.html
    W_hat = np.linalg.solve(X.T.dot(X) + reg_lambda * I, X.T.dot(Y))
    assert W_hat.shape == (X.shape[1], Y.shape[1])
    return W_hat


def predict(W, X):
    preds = np.argmax(W.T.dot(X.T), axis=0)
    assert preds.shape == (X.shape[0],)
    return preds


def main():
    print("Loading MNIST dataset from source")
    X_train, Y_train, X_test, Y_test = load_dataset()
    print("Finished loading MNIST dataset from source")

    # # one-hot encoding the 10 classes in MNIST
    # Y_train = np.eye(10)[Y_train]
    # Y_test = np.eye(10)[Y_test]

    W_hat = train(X_train, np.eye(10)[Y_train], 1e-4)

    Y_train_preds = predict(W_hat, X_train)
    Y_test_preds = predict(W_hat, X_test)

    train_acc = np.count_nonzero(Y_train == Y_train_preds) / Y_train.shape[0]
    print("Training Error:", 1-train_acc)
    test_acc = np.count_nonzero(Y_test == Y_test_preds) / Y_test.shape[0]
    print("Testing Error:", 1 - test_acc)
```

Figure 8: Screenshot of the train$_{M}NIST.py$.

Problem B.2

a.

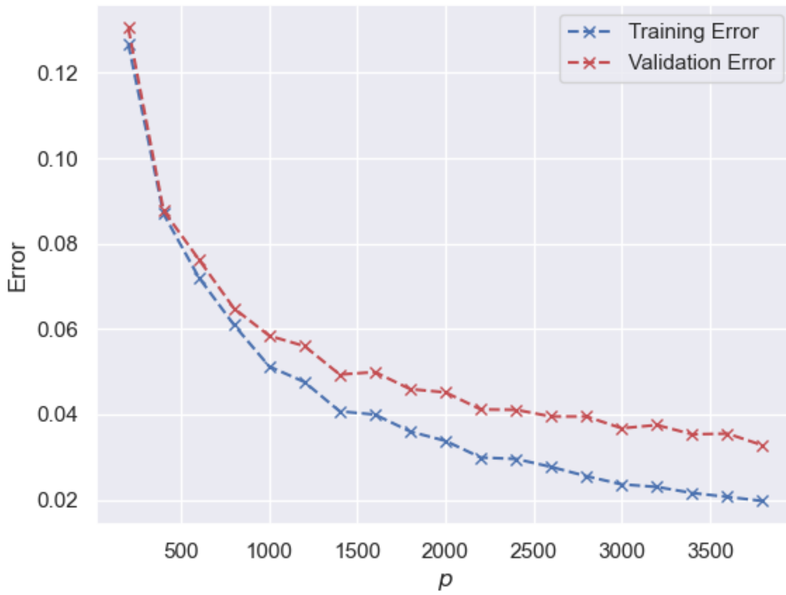
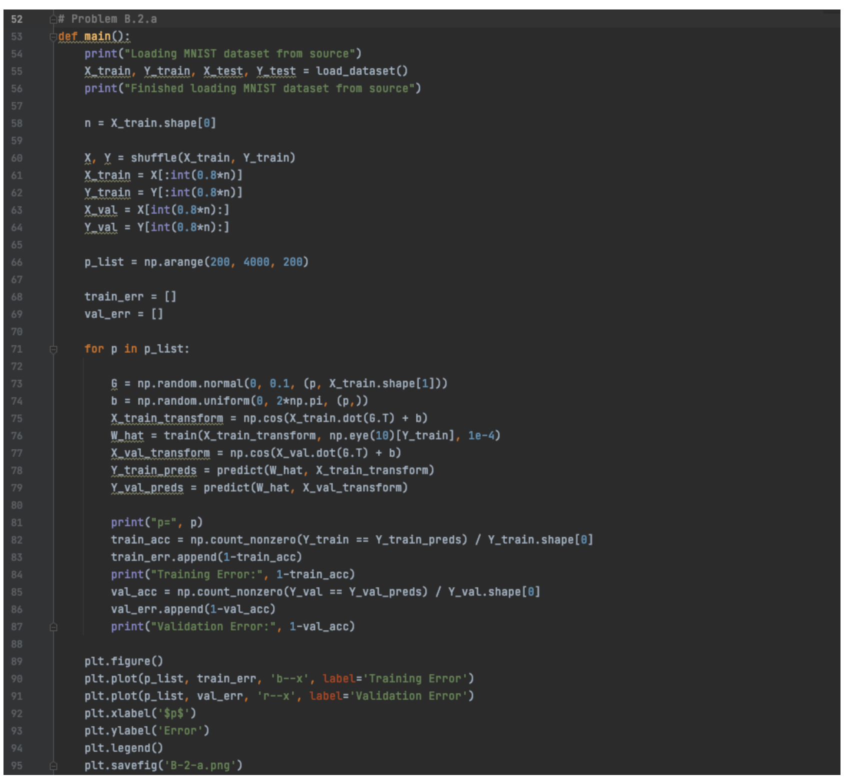


Figure 9: Plot of training and validation error under different p value.



```python
# Problem B.2.a
def main():
    print("Loading MNIST dataset from source")
    X_train, Y_train, X_test, Y_test = load_dataset()
    print("Finished loading MNIST dataset from source")

    n = X_train.shape[0]

    X, Y = shuffle(X_train, Y_train)
    X_train = X[:int(0.8*n)]
    Y_train = Y[:int(0.8*n)]
    X_val = X[int(0.8*n):]
    Y_val = Y[int(0.8*n):]

    p_list = np.arange(200, 4000, 200)

    train_err = []
    val_err = []

    for p in p_list:

        G = np.random.normal(0, 0.1, (p, X_train.shape[1]))
        b = np.random.uniform(0, 2*np.pi, (p,))
        X_train_transform = np.cos(X_train.dot(G.T) + b)
        W_hat = train(X_train_transform, np.eye(10)[Y_train], 1e-4)
        X_val_transform = np.cos(X_val.dot(G.T) + b)
        Y_train_preds = predict(W_hat, X_train_transform)
        Y_val_preds = predict(W_hat, X_val_transform)

        print("p=", p)
        train_acc = np.count_nonzero(Y_train == Y_train_preds) / Y_train.shape[0]
        train_err.append(1-train_acc)
        print("Training Error:", 1-train_acc)
        val_acc = np.count_nonzero(Y_val == Y_val_preds) / Y_val.shape[0]
        val_err.append(1-val_acc)
        print("Validation Error:", 1-val_acc)

    plt.figure()
    plt.plot(p_list, train_err, 'b--x', label='Training Error')
    plt.plot(p_list, val_err, 'r--x', label='Validation Error')
    plt.xlabel('$p$')
    plt.ylabel('Error')
    plt.legend()
    plt.savefig('B-2-a.png')
```

Figure 10: Code used to generate Fig.11.

b.

We choose $p = 4000$ with the training error being 0.017833, validation error being 0.0340, and testing error $\widehat{\epsilon}_{\text{test}}(\widehat{f})$ being 0.03020. From the Lemma, we have:

$$\mathbb{P}\left(\left|\left(\frac{1}{m}\sum_{i=1}^{m} X_i\right) - \mu\right| \geq \sqrt{\frac{(b-a)^2 \log(2/\delta)}{2m}}\right) \leq \delta,$$

where $a = 0$, $b = 1$, $\sigma = 0.05$ and $m = 10000$ which let $\sqrt{\frac{(b-a)^2 \log(2/\delta)}{2m}} \sim 0.0136$ so that the 95% confidence interval of the testing error will be 0.0302 ± 0.0136.

```python
        G = np.random.normal(0, 0.1, (p, X_train.shape[1]))
        b = np.random.uniform(0, 2*np.pi, (p,))

        X_train_transform = np.cos(X_train.dot(G.T) + b)
        X_val_transform = np.cos(X_val.dot(G.T) + b)
        X_test_transform = np.cos(X_test.dot(G.T) + b)

        W_hat = train(X_train_transform, np.eye(10)[Y_train], 1e-4)

        Y_train_preds = predict(W_hat, X_train_transform)
        Y_val_preds = predict(W_hat, X_val_transform)
        Y_test_preds = predict(W_hat, X_test_transform)

        print("p=", p)
        train_acc = np.count_nonzero(Y_train == Y_train_preds) / Y_train.shape[0]
        train_err.append(1-train_acc)
        print("Training Error:", 1-train_acc)
        val_acc = np.count_nonzero(Y_val == Y_val_preds) / Y_val.shape[0]
        val_err.append(1-val_acc)
        print("Validation Error:", 1-val_acc)
        test_acc = np.count_nonzero(Y_test == Y_test_preds) / Y_test.shape[0]
        test_err.append(1 - test_acc)
        print("Test Error:", 1 - test_acc)
```

Figure 11: Code used to get the test error.