CSE446 HW1

Brian Kang*†

Conceptual Questions

A.0

- In your own words, describe what bias and variance are? What is bias-variance tradeoff?
  - Bias: How far we are from fitting out data to our ideal estimator. Or, the error introduced by approximating a real model by a much simpler model.
  - Variance: How much our estimator will change after redrawing a sample and fitting the data again to get the estimator. Or, how much $\hat{f}$ would change if we estimated a model using a different training set.
  - Bias-Variance Tradeoff: The tradeoff between bias and variance within the reducible error part of a model (test MSE) with respect to model flexibility/complexity.
- What happens to bias and variance when the model complexity increases/decreases?
  - When the model complexity increases, bias decreases (our model may be able to capture the required complexity of the true model) and variance increases (noise is introduced by fitting this complex model to sample data). When the model complexity decreases, bias increases and variance decreases. (The irreducible error stays the same regardless of complexity.)
- True or False: The bias of a model increases as the amount of training data available increases.
  - **False**. $Bias[\hat{f}(x)] = \mathbb{E}[\hat{f}(x)] - f(x)$ so training set size will not affect bias.
- True or False: The variance of a model decreases as the amount of training data available increases.
  - **True**. As training data increases, the model will be able to generalize better and variance will decrease.
- True or False: A learning algorithm will generalize better if we use less features to represent our data.
  - **Yes and No**. When facing a high variance issue, decreasing the number of features helps, but for high bias issues increasing them will help.
- To get better generalization, should we use the train set or the test set to tune our hyperparameters?
  - **Training set**. The training set will be split into training and validation sets.
- True or False: The training error of a function on the training set provides an overestimate of the true error of that function.
  - **False**. Underestimation. The function is chosen to minimize the training error. So, fitting this function on true data will return larger error because the function was not trained to minimize the true error.

---

*Collaborated with Cindy Wu

†References: An Introduction to Statistical Learning (James, Witten, Hastie, Tibshirani)

A.1

a.

$$\begin{aligned}
\hat{\theta}_{MLE} &= \underset{\theta}{\text{argmax}}\ P(x_1, x_2, x_3, x_4, x_5 \mid \lambda = \theta) \\
&= \underset{\lambda}{\text{argmax}} \prod_{i=1}^{5} \frac{e^{-\lambda}\lambda^{x_i}}{x_i!} \\
&= \underset{\lambda}{\text{argmax}}\, (e^{-\lambda})^5 \prod_{i=1}^{5} \frac{\lambda^{x_i}}{x_i!} \\
&= \underset{\lambda}{\text{argmax}}\ \log(\bullet) \\
&= \underset{\lambda}{\text{argmax}}\ 5\log(e^{-\lambda}) + \sum_{i=1}^{5}(\log(\lambda^{x_i}) - \log(x_i!)) \\
&= \underset{\lambda}{\text{argmax}}\ -5\lambda + \sum_{i=1}^{5}(x_i \log(\lambda) - \log(x_i!))\,, \text{ then}
\end{aligned}$$

$$\begin{aligned}
\frac{d}{d\lambda}[\bullet] &= -5 + \sum_{i=1}^{5}(\frac{x_i}{\lambda} - 0) \\
&= -5 + \frac{x_1 + \cdots + x_5}{\lambda} = 0\,, \text{ then} \\
5\lambda &= x_1 + \cdots + x_5\,, \\
&\boxed{\lambda = \hat{\theta}_{MLE} = \frac{x_1 + \cdots + x_5}{5}}
\end{aligned}$$

b. Based on the above work, by symmetry,

$$\boxed{\hat{\theta}_{MLE} = \frac{x_1 + \cdots + x_6}{6}}$$

c.

$$\begin{aligned}
\lambda \text{ after 5 games } &= \frac{2+0+1+1+2}{5} = \boxed{\frac{6}{5}} \\
\lambda \text{ after 6 games } &= \frac{2+0+1+1+2+4}{6} = \frac{10}{6} = \boxed{\frac{5}{3}}
\end{aligned}$$

A.2

$$\begin{aligned}
\hat{\theta}_{MLE} &= \underset{\theta}{\text{argmax}}\ P(x_1, \ldots, x_n \mid \theta) \\
&= \underset{\theta}{\text{argmax}} \prod_{i=1}^{n} \frac{1}{\theta} = \underset{\theta}{\text{argmax}}\ \theta^{-n}
\end{aligned}$$

MLE is the value of θ that maximizes $\frac{1}{\theta^n}$. Since $\frac{1}{\theta^n}$ monotonically decreases in θ, we want the smallest θ such that $\theta \geq x_i$ for $i = 1, \ldots, n$. Therefore, it has to be that $\theta = \max(x_1, \ldots, x_n)$. Then,

$$\hat{\theta}_{MLE} = \max(X_1, \ldots, X_n)$$

Do note that this MLE is not a great estimator for θ because the MLE will ALWAYS underestimate θ, since we know that $Prob(\max(X_1, \ldots, X_n) < \theta) = 1$.

A.3

a. *Proof.*

$$
\begin{aligned}
\mathbb{E}_{train}[\hat{\epsilon}_{train}(f)] &= \mathbb{E}_{train}\left[\frac{1}{N_{train}}\sum_{(x,y)\in S_{train}}(f(x)-y)^2\right] \\
&= \frac{1}{N_{train}}\sum_{(x,y)\in S_{train}}\mathbb{E}_{train}\left[(f(x)-y)^2\right] \\
&= \frac{1}{N_{train}} * N_{train} * \mathbb{E}_{train}\left[(f(x)-y)^2\right] \\
&= \mathbb{E}_{(x,y)\in S_{test}\subset D}\left[(f(x)-y)^2\right] \\
&= \epsilon(f)
\end{aligned}
$$

The last step is justified because the i.i.d. random samples are drawn from the underlying distribution D.

Likewise,

$$
\begin{aligned}
\mathbb{E}_{test}[\hat{\epsilon}_{test}(f)] &= \frac{1}{N_{test}} * N_{test} * \mathbb{E}_{test}\left[(f(x)-y)^2\right] \\
&= \mathbb{E}_{(x,y)\in S_{test}\subset D}\left[(f(x)-y)^2\right] \\
&= \epsilon(f)
\end{aligned}
$$

The last step is also justified because the i.i.d. random samples are drawn from the underlying true distribution D.

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

Using similar reasoning, if $\epsilon(\hat{f}) = \mathbb{E}_D[(\hat{f}(x)-y)^2]$, where $\hat{f}$ is the function trained using the training set, then,

$$
\begin{aligned}
\mathbb{E}_{test}[\hat{\epsilon}_{test}(\hat{f})] &= \frac{1}{N_{test}} * N_{test} * \mathbb{E}_{test}\left[(\hat{f}(x)-y)^2\right] \\
&= \mathbb{E}_{(x,y)\in S_{test}\subset D}\left[(\hat{f}(x)-y)^2\right] \\
&= \epsilon(\hat{f})
\end{aligned}
$$

The last step is justified because the i.i.d. samples are drawn from the underlying distribution D, and the training and test sets are mutually exclusive. Keeping the test set as random, the expectation over the test error (using the trained function) will be an unbiased estimator for the true error for $\hat{f}$, i.e., $\epsilon(\hat{f})$. □

b. *Proof.* Unlike above, $\mathbb{E}_{train}[\hat{\epsilon}_{train}(\hat{f})] \neq \mathbb{E}_{train}[\epsilon(\hat{f})]$. This is the fact that train error is optimistically biased because it is evaluated on the data it trained on. The $\hat{f}$ function is already trained on the test data such that $\hat{\epsilon}_{train}$ is minimized. Thus, the expectation train of this minimized error cannot be equal to the expectation train of the true error calculated from the trained function. □

c. *Proof.* First, we are given that $\exists \hat{f}_{train}$ such that, $\hat{\epsilon}_{train}(\hat{f}_{train}) \leq \hat{\epsilon}_{train}(f)\,\forall f \in F$. Then,

$$
\begin{aligned}
\mathbb{E}_{train}[\hat{\epsilon}_{train}(\hat{f}_{train})] &\leq \mathbb{E}_{train}[\hat{\epsilon}_{train}(f)]\,\forall f \in F \\
&= \epsilon(f) \text{ by (a).}
\end{aligned}
$$

Next, we are given information about $\mathbb{E}_{train,test}[\hat{\epsilon}_{test}(\hat{f}_{train})]$,

$$\begin{aligned}
\mathbb{E}_{train,test}[\hat{\epsilon}_{test}(\hat{f}_{train})] &= \sum_{f\in F} \mathbb{E}_{train,test}[\hat{\epsilon}_{test}(f) * \vec{1}\{\hat{f}_{train} = f\}] \\
&= \sum_{f\in F} \mathbb{E}_{test}[\hat{\epsilon}_{test}(f)] * \mathbb{E}_{train}[\vec{1}\{\hat{f}_{train} = f\}] \text{ by indep. between } S_{train} \text{ and } S_{test} \\
&= \sum_{f\in F} \mathbb{E}_{test}[\hat{\epsilon}_{test}(f)] * Prob_{train}(\hat{f}_{train} = f) \\
&= \sum_{f\in F} \epsilon(f) * Prob_{train}(\hat{f}_{train} = f) \text{ by (a)} \\
&= \epsilon(\hat{f}_{train}) * 1
\end{aligned}$$

We know that $\forall f \in F$, $\epsilon(f) \leq \epsilon(\hat{f}_{train})$.
Therefore, $\mathbb{E}_{train}[\hat{\epsilon}_{train}(\hat{f}_{train})] \leq \mathbb{E}_{train}[\hat{\epsilon}_{train}(f)] = \epsilon(f) \leq \epsilon(\hat{f}_{train}) = \mathbb{E}_{train,test}[\hat{\epsilon}_{test}(\hat{f}_{train})]$. □

Technical Questions

A.4

```
'''
    Template for polynomial regression
    AUTHOR Eric Eaton, Xiaoxiang Hu
'''

import numpy as np


#-----------------------------------------------------------------
#  Class PolynomialRegression
#-----------------------------------------------------------------

class PolynomialRegression:

    def __init__(self, degree=1, reg_lambda=1E-8):
        """
        Constructor
        """
        self.regLambda = reg_lambda
        self.degree = degree
        self.theta = None
        self.mu = None
        self.sd = None

    def polyfeatures(self, X, degree):
        """
        Expands the given X into an n * d array of polynomial features of
            degree d.

        Returns:
            A n-by-d numpy array, with each row comprising of
            X, X * X, X ** 3, ... up to the dth power of X.
            Note that the returned matrix will not include the zero-th power.

        Arguments:
            X is an n-by-1 column numpy array
            degree is a positive integer
        """
        n, d = X.shape   # get the row and column dimension of X
        X_ = X  # save X in different variable "X_"
        for i in range(2, degree+1):    # because last index exclusive and we want the d'th
    degree
```

```
            expX = np.power(X, i)  # make d'th power of X
            X_ = np.concatenate((X_, expX), axis=1) # add d'th power column to the right
        return X_

    def fit(self, X, y):
        """
            Trains the model
            Arguments:
                X is a n-by-1 array
                y is an n-by-1 array
            Returns:
                No return value
            Note:
                You need to apply polynomial expansion and scaling
                at first
        """
        n = len(X)
        X = self.polyfeatures(X, self.degree)  # poly expand
        xbar = np.mean(X, axis=0)   # get mean of columns
        sdhat = np.std(X, axis=0)   # get std dev of columns
        X = (X-xbar)/sdhat  # standardize
        X_ = np.c_[np.ones([n, 1]), X]  # add 1s column
        n, d = X_.shape     # get dim of new data matrix
        d = d-1  # remove 1 for the extra column of ones we added to get the original num
    features

        # construct reg matrix
        reg_matrix = self.regLambda * np.eye(d + 1)
        reg_matrix[0, 0] = 0
        # analytical solution (X'X + regMatrix)^-1 X' y
        self.theta = np.linalg.pinv(X_.T.dot(X_) + reg_matrix).dot(X_.T).dot(y)
        self.mu = xbar
        self.sd = sdhat

    def predict(self, X):
        """
        Use the trained model to predict values for each instance in X
        Arguments:
            X is a n-by-1 numpy array
        Returns:
            an n-by-1 numpy array of the predictions
        """
        n = len(X)
        X = self.polyfeatures(X, self.degree)  # poly expand
        X = (X - self.mu) / self.sd  # standardize again
        X_ = np.c_[np.ones([n, 1]), X]  # add 1s column
        return X_.dot(self.theta)   # predict the fitted values


#-----------------------------------------------------------------
#  End of Class PolynomialRegression
#-----------------------------------------------------------------



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

    Arguments:
        Xtrain -- Training X, n-by-1 matrix
        Ytrain -- Training y, n-by-1 matrix
        Xtest -- Testing X, m-by-1 matrix
        Ytest -- Testing Y, m-by-1 matrix
        regLambda -- regularization factor
        degree -- polynomial degree

    Returns:
        errorTrain -- errorTrain[i] is the training accuracy using
```

```
        model trained by Xtrain[0:(i+1)]
        errorTest -- errorTrain[i] is the testing accuracy using
        model trained by Xtrain[0:(i+1)]

    Note:
        errorTrain[0:1] and errorTest[0:1] won't actually matter, since we start displaying
    the learning curve at n = 2 (or higher)
    """

    n = len(Xtrain)
    errorTrain = np.zeros(n)
    errorTest = np.zeros(n)
    glm = PolynomialRegression(degree, reg_lambda)   # get polynom model
    for i in range(2, n+1):
        glm.fit(Xtrain[0:(i+1)], Ytrain[0:(i+1)])   # fit the training data
        errorTrain[i-1] = np.mean((glm.predict(Xtrain[0:(i+1)]) - Ytrain[0:(i+1)])**2)
        errorTest[i-1] = np.mean((glm.predict(Xtest[0:(i+1)]) - Ytest[0:(i+1)])**2)
    return errorTrain, errorTest
```

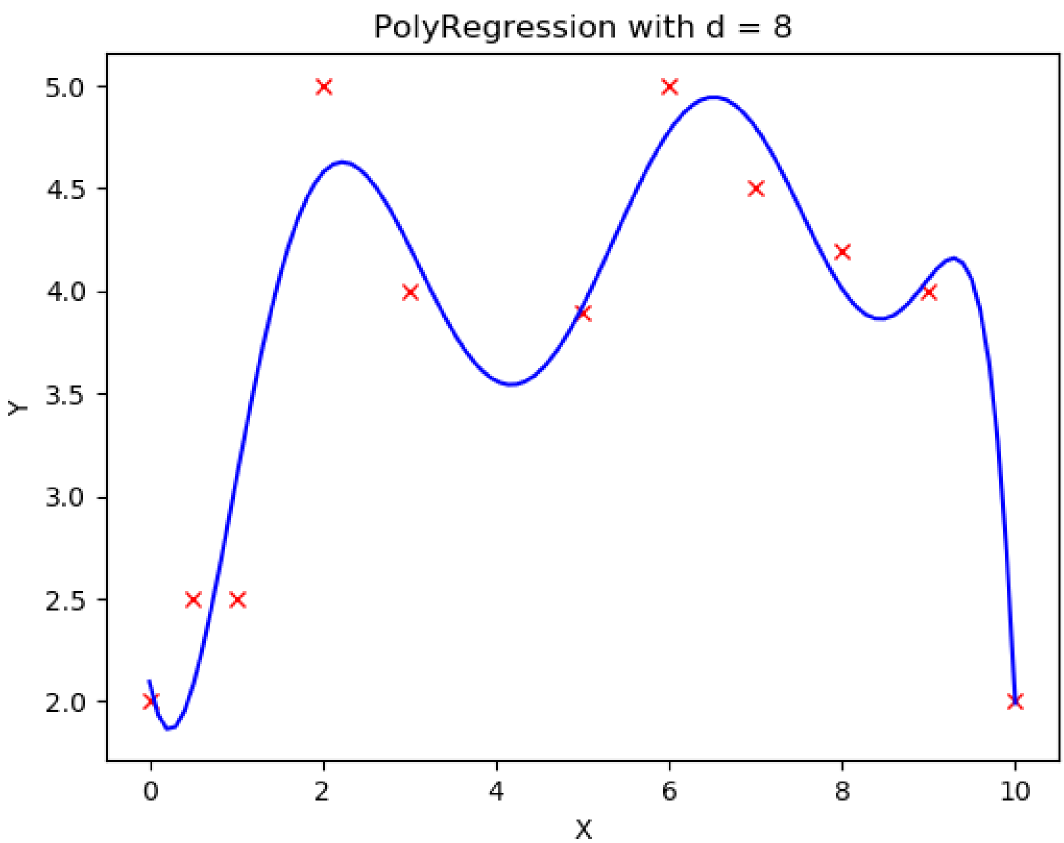

Figure 1: PolyReg Fit

A.5
Code is above with polyreg.py

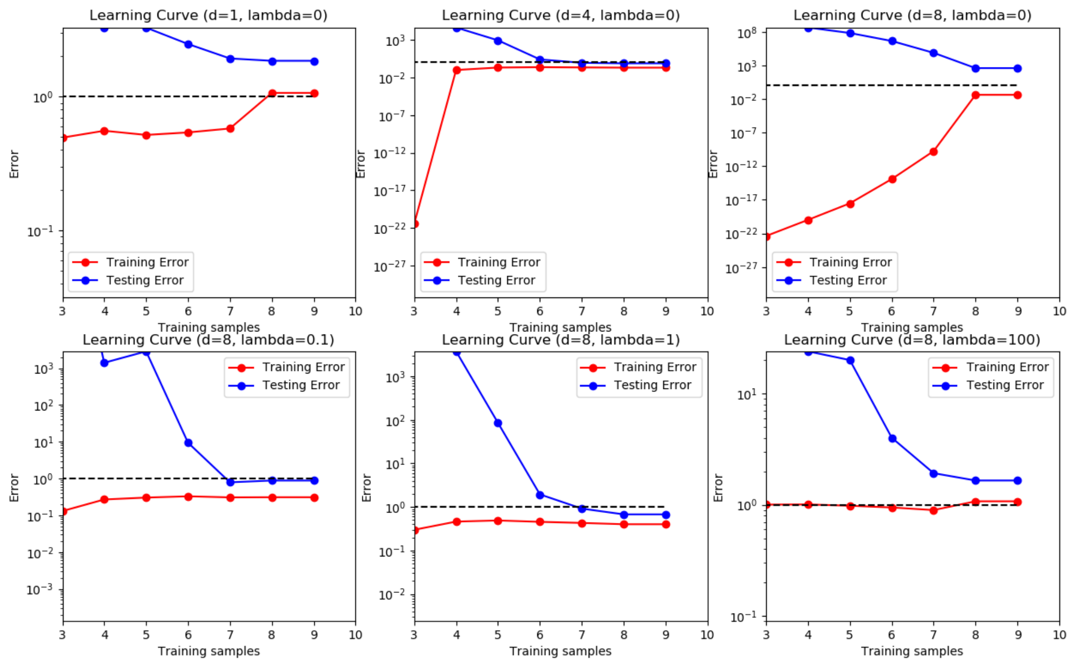

Figure 2: Learning Curves

A.6

a. *Proof.*

$$\begin{aligned}\hat{w} &= \underset{w}{\text{argmin}} \sum_{j=0}^{k} \left[||Xw_j - Ye_j||^2 + \lambda ||w_j||^2 \right] \\ &= \underset{w}{\text{argmin}} \sum_{j=0}^{k} ||Xw_j - Ye_j||^2 + \lambda \sum_{j=0}^{k} ||w_j||^2 \\ &= \underset{w}{\text{argmin}} \sum_{j=0}^{k} (Xw_j - Ye_j)^2 + \lambda \sum_{j=0}^{k} ||w_j||^2\end{aligned}$$

$$\nabla_w(\bullet) = \begin{bmatrix} 2(Xw_0 - Ye_0)^T X + 2\lambda w_0 \\ \vdots \\ 2(Xw_k - Ye_k)^T X + 2\lambda w_k \end{bmatrix}$$

$$= \begin{bmatrix} 2X^T X w_0 - 2X^T Y e_0 + 2\lambda w_0 \\ \vdots \\ 2X^T X w_k - 2X^T Y e_k + 2\lambda w_k \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

Now rearrange terms and solve for $\hat{w}$.

$$\begin{bmatrix} 2X^TX\hat{w}_0 + 2\lambda\hat{w}_0 \\ \vdots \\ 2X^TX\hat{w}_k + 2\lambda\hat{w}_k \end{bmatrix} = \begin{bmatrix} 2X^TYe_0 \\ \vdots \\ 2X^TYe_k \end{bmatrix}$$

$$\begin{bmatrix} X^TX\hat{w}_0 + \lambda\hat{w}_0 \\ \vdots \\ X^TX\hat{w}_k + \lambda\hat{w}_k \end{bmatrix} = \begin{bmatrix} X^TYe_0 \\ \vdots \\ X^TYe_k \end{bmatrix}$$

$$\begin{bmatrix} (X^TX + \lambda * 1)\hat{w}_0 \\ \vdots \\ (X^TX + \lambda * 1)\hat{w}_k \end{bmatrix} = \begin{bmatrix} X^TYe_0 \\ \vdots \\ X^TYe_k \end{bmatrix}$$

$$\begin{bmatrix} \hat{w}_0 \\ \vdots \\ \hat{w}_k \end{bmatrix} = \begin{bmatrix} (X^TX + \lambda * 1)^{-1}X^TYe_0 \\ \vdots \\ (X^TX + \lambda * 1)^{-1}X^TYe_k \end{bmatrix}$$

In matrix format, this translates to, $\hat{W} = (X^TX + \lambda I)^{-1}X^TY$. □

b. Final output:

train error: 0.14805
test error: 0.1466

```python
import numpy as np
from mnist import MNIST


def load_dataset():
    mndata = MNIST('.\data')
    mndata.gz = True
    X_train, labels_train = map(np.array, mndata.load_training())
    X_test, labels_test = map(np.array, mndata.load_testing())
    X_train = X_train / 255.0
    X_test = X_test / 255.0
    return X_train, labels_train, X_test, labels_test


x_train, label_train, x_test, label_test = load_dataset()  # load in data


# print("x_train: ")
# print(x_train, x_train.shape)
# label_train = np.asfarray(label_train)

# print("\noriginal label: ")
# print(label_train, label_train.shape)
# print(x_test, x_test.shape)
# print(label_test, label_test.shape)

def train(X, Y, reg_lambda):
    d = X.shape[1]
    # solve for \hat{w}: (XtX+lambda*I) * w_hat = XtY
    lhs = np.dot(np.transpose(X), X) + reg_lambda * np.eye(d)
    rhs = np.dot(np.transpose(X), Y)
    w_hat = np.linalg.solve(lhs, rhs)
    return w_hat

```

```python

def predict(w_hat, X):  # y values we are predicting are lables
    Y_hat = np.dot(X, w_hat)  # equal to e_j'th column of I * w_hat^T * x_i'th column from X
    # print("\nDim of y hat: ", Y_hat.shape)
    return Y_hat.argmax(axis=1)  # get argmax_j [e_j * w_hat * x_i]


num_class = 10  # this is k
onehot_label_train = np.eye(num_class)[label_train]
onehot_label_test = np.eye(num_class)[label_test]
# print("\none hot coded label: ")
# print(onehot_label_train, onehot_label_train.shape)

What = train(x_train, onehot_label_train, 1E-4)
# print("\ndim of w hat: ", What.shape)

# on training set
pred_train_label = predict(What, x_train)
# print("\npredicted label: ")
# print(pred_train_label, pred_train_label.shape)

labelDiff_train = np.equal(label_train, pred_train_label)
errorTrain = np.mean(1 - labelDiff_train)

# on test set
pred_test_label = predict(What, x_test)
labelDiff_test = np.equal(label_test, pred_test_label)
errorTest = np.mean(1 - labelDiff_test)

print("train error: ", errorTrain)
print("test error: ", errorTest)
```