CSE 546 Homework #1

Lu Yu

Spring 2020

Short Answer and "True or False" Conceptual questions

A.0 The answers to these questions should be answerable without referring to external materials.

- *[2 points]* In your own words, describe what bias and variance are? What is bias-variance tradeoff?
  Bias is an error between the expected predicted value and the true value. Variance is the degree of deviation between multiple fit predictions between models. The bias–variance tradeoff is the process of simultaneously minimizing the summation of two sources of error. When the model is too simple, high bias is prone to occur; when the model is too complicated, high variance is prone to occur. The bias–variance tradeoff wants to get a balanced condition. It might be wise to use a biased estimator, so long as it reduces our variance, assuming our goal is to minimize squared error.
- *[2 points]* What happens to bias and variance when the model complexity increases/decreases?
  When the model complexity increases, bias decreases and variance increases. When the model complexity decreases, bias increases and variance decreases.
- *[1 points]* True or False: The bias of a model increases as the amount of training data available increases.
  False
- *[1 points]* True or False: The variance of a model decreases as the amount of training data available increases.
  True
- *[1 points]* True or False: A learning algorithm will generalize better if we use less features to represent our data
  False
- *[2 points]* To get better generalization, should we use the train set or the test set to tune our hyperparameters?
  The short answer is no. The test set should be used only for testing, not for hyperparameter tuning. If the train set is used for tuning hyperparameter and training, the estimate will be overoptimistic.
- *[1 points]* True or False: The training error of a function on the training set provides an overestimate of the true error of that function.
  True

Maximum Likelihood Estimation (MLE)

A.1

a. *[5 points]* the maximum-likelihood estimate of the parameter λ

$$Poi(x|\lambda) = e^{-\lambda}\frac{\lambda^x}{x!}$$

$$L(x_1, x_2, \ldots, x_5|\lambda) = \prod_{i=1}^{5} e^{-\lambda}\frac{\lambda_i^x}{x_i!} = e^{-5\lambda}\prod_{i=1}^{5}\frac{\lambda_i^x}{x_i!}$$

$$LnL = -5\lambda + \sum_{i=1}^{5}(x_i ln\lambda - lnx_i!)$$

$$\frac{dlnL}{d\lambda} = -5 + \sum_{i=1}^{5}\frac{x_i}{\lambda}$$

$$Let \frac{dlnL}{d\lambda} = 0$$

$$\lambda = \frac{1}{5}\sum_{i=1}^{5} x_i$$

b. *[5 points]* Derive the same expression for the estimate using 6 games.

$$L(x_1, x_2, \ldots, x_6|\lambda) = \prod_{i=1}^{6} e^{-\lambda}\frac{\lambda_i^x}{x_i!} = e^{-6\lambda}\prod_{i=1}^{6}\frac{\lambda_i^x}{x_i!}$$

$$lnL = -6\lambda + \sum_{i=1}^{6}(x_i ln\lambda - lnx_i!)$$

$$\frac{dlnL}{d\lambda} = -6 + \sum_{i=1}^{6}\frac{x_i}{\lambda}$$

$$Let \frac{dlnL}{d\lambda} = 0$$

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

c. *[5 points]* Given the goal counts, please give numerical estimates of λ after 5 and 6 games.
After 5 games,

$$\lambda = \frac{2+0+1+1+2}{5} = \frac{6}{5}$$

After 6 games,

$$\lambda = \frac{2+0+1+1+2+4}{6} = \frac{5}{3}$$

A.2 *[10 points]* Let $x_1, \ldots, x_n$ be independent, uniformly distributed on the continuous domain $[0, \theta]$ for some θ. What is the Maximum likelihood estimate for θ?

$$f(x|\theta) = \begin{cases} \frac{1}{\theta}, 0 \le x \le \theta \\ 0, otherwise \end{cases}$$

$$L(\theta) = \begin{cases} \frac{1}{\theta^n}, 0 \le x_i \le \theta \\ 0, otherwise \end{cases}$$

$$\hat{\theta} = max\{x_1, \ldots, x_n\}$$

Overfitting

A.3

a. *[3 points]* (bias: the test error) For all fixed f (before we've seen any data) show that:

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

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

$$\begin{aligned}
&= \mathbb{E}_{train}[\frac{1}{N_{train}} \sum_{(x,y)\in S_{train}} (f(x)-y)^2] \\
&= \frac{1}{n} \int_{(x,y)\in training\ set} \sum_{i=1}^{n} (f(x_i)-y_i)^2 P_{(x_1,y_1,\ldots,x_n,y_n)} dx_1 dy_1 \ldots dx_n dy_n \\
&= \frac{1}{n} \int_{x_1,y_1,\ldots,x_n,y_n} \sum_{i=1}^{n} (f(x_i)-y_i)^2 P_{(x_1,y_1)} P_{(x_2,y_2)} \ldots P_{x_n,y_n} dx_1 dy_1 \ldots dx_n dy_n \\
&= \frac{1}{n} \int_{x_1,y_1} n(f(x_i)-y_i)^2 P_{(x_1,y_1)} dx_1 dy_1 \qquad , (x_i,y_i)\ drawn\ i.i.d\ from\ an\ underlying\ distribution \\
&= \int_{x_i,y_i} (f(x_i)-y_i)^2 P_{(x_i,y_i)} dx_i dy_i \qquad , f\ is\ fixed\ and\ not\ related\ to\ training\ set \\
&= \int_{(x,y)\sim\mathbb{D}} (f(x)-y)^2 P_{(x,y)} dx dy \\
&= \mathbb{E}_{(x,y)\sim\mathbb{D}}[(f(x)-y)^2] \\
&= \epsilon(f).
\end{aligned}$$

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

$$\begin{aligned}
&= \mathbb{E}_{test}[\frac{1}{N_{test}} \sum_{(x,y)\in S_{test}} (f(x)-y)^2] \\
&= \frac{1}{n} \int_{(x,y)\in test\ set} \sum_{i=1}^{n} (f(x_i)-y_i)^2 P_{(x_1,y_1,\ldots,x_n,y_n)} dx_1 dy_1 \ldots dx_n dy_n \\
&= \frac{1}{n} \int_{x_1,y_1,\ldots,x_n,y_n} \sum_{i=1}^{n} (f(x_i)-y_i)^2 P_{(x_1,y_1)} P_{(x_2,y_2)} \ldots P_{x_n,y_n} dx_1 dy_1 \ldots dx_n dy_n \\
&= \frac{1}{n} \int_{x_1,y_1} n(f(x_i)-y_i)^2 P_{(x_1,y_1)} dx_1 dy_1 \qquad , (x_i,y_i)\ drawn\ i.i.d\ from\ an\ underlying\ distribution \\
&= \int_{x_i,y_i} (f(x_i)-y_i)^2 P_{(x_i,y_i)} dx_i dy_i \qquad , f\ is\ fixed\ and\ not\ related\ to\ test\ set \\
&= \int_{(x,y)\sim\mathbb{D}} (f(x)-y)^2 P_{(x,y)} dx dy \\
&= \mathbb{E}_{(x,y)\sim\mathbb{D}}[(f(x)-y)^2] \\
&= \epsilon(f).
\end{aligned}$$

Use a similar line of reasoning to show that the test error is an unbiased estimate of our true error for f. Specifically, show that:

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

f turns to be $\hat{f}$, if the i.i.d assumption still holds, the equation holds. Since the $\hat{f}$ is trained by training

set, dose nothing to do with the test set. The test set samples are still drawn i.i.d. from a distribution.

$$\begin{aligned}
&\mathbb{E}_{test}[\hat{\epsilon}_{test}(\hat{f})] \\
&\quad = \mathbb{E}_{test}[\frac{1}{N_{test}} \sum_{(x,y)\in S_{test}} (\hat{f}(x)-y)^2] \\
&\quad = \frac{1}{N_{test}} \sum_{(x,y)\in S_{test}} \int_x \int_y (\hat{f}(x)-y)^2 P_{XY(x,y)} dxdy \\
&\quad = \int_x \int_y (\hat{f}(x)-y)^2 P_{XY(x,y)} dxdy \\
&\quad = \epsilon(\hat{f}).
\end{aligned}$$

b. *[4 points]*

$$\begin{aligned}
&\mathbb{E}_{train}[\hat{\epsilon}_{train}(\hat{f})] \\
&\quad = \mathbb{E}_{train}[\frac{1}{N_{train}} \sum_{(x,y)\in S_{train}} (\hat{f}(x)-y)^2] \\
&\quad = \int_{x_1}\int_{y_1}\cdots\int_{x_n}\int_{y_n} \frac{1}{N_{train}} \sum_{(x,y)\in S_{train}} \mathbb{E}_{train}[(\hat{f}(x)-y)^2] P_{XY(x_1,y_1,\ldots,x_n,y_n)} dx_1 dy_1 \ldots dx_n dy_n,
\end{aligned}$$

since $\hat{f}$ is determined by training set, the training set samples are not drawn i.i.d. from a distribution. The i.i.d assumption doesn't hold anymore.

$$\begin{aligned}
&\neq \frac{1}{N_{train}} \sum_{(x,y)\in S_{train}} \int_x \int_y (\hat{f}(x)-y)^2 P_{XY(x,y)} dxdy \\
&= \mathbb{E}_{train}[\epsilon(\hat{f})]
\end{aligned}$$

c. *[8 points]*

$$\begin{aligned}
\mathbb{E}_{train,test}[\hat{\epsilon}_{test}(\hat{f}_{train})] &= \sum_{f\in\mathbb{F}} \mathbb{E}_{train,test}[\hat{\epsilon}_{test}\mathbf{1}\{\hat{f}_{train}=f\}] \\
&= \sum_{f\in\mathbb{F}} \mathbb{E}_{test}[\hat{\epsilon}_{test}]\mathbb{E}_{train}[\mathbf{1}\{\hat{f}_{train}=f\}] \\
&= \sum_{f\in\mathbb{F}} \mathbb{E}_{test}[\hat{\epsilon}_{test}]\mathbb{P}_{train}(\hat{f}_{train}=f)
\end{aligned}$$

$$\mathbb{E}_{train}[\hat{\epsilon}_{train}(\hat{f}_{train})] = \sum_{f\in\mathbb{F}} \mathbb{E}_{train}[\hat{\epsilon}_{train}(\hat{f}_{train}|f)]\mathbb{P}_{train}(\hat{f}_{train}=f) \quad \textit{by law of total expectation}$$

Because $\hat{\epsilon}_{train}(\hat{f}_{train}) \leq \hat{\epsilon}_{train}(f)$

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

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

From part a, we know that for a fixed f, $\mathbb{E}_{train}[\hat{\epsilon}_{train}(f)] = \mathbb{E}_{test}[\hat{\epsilon}_{test}(f)]$. So,

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

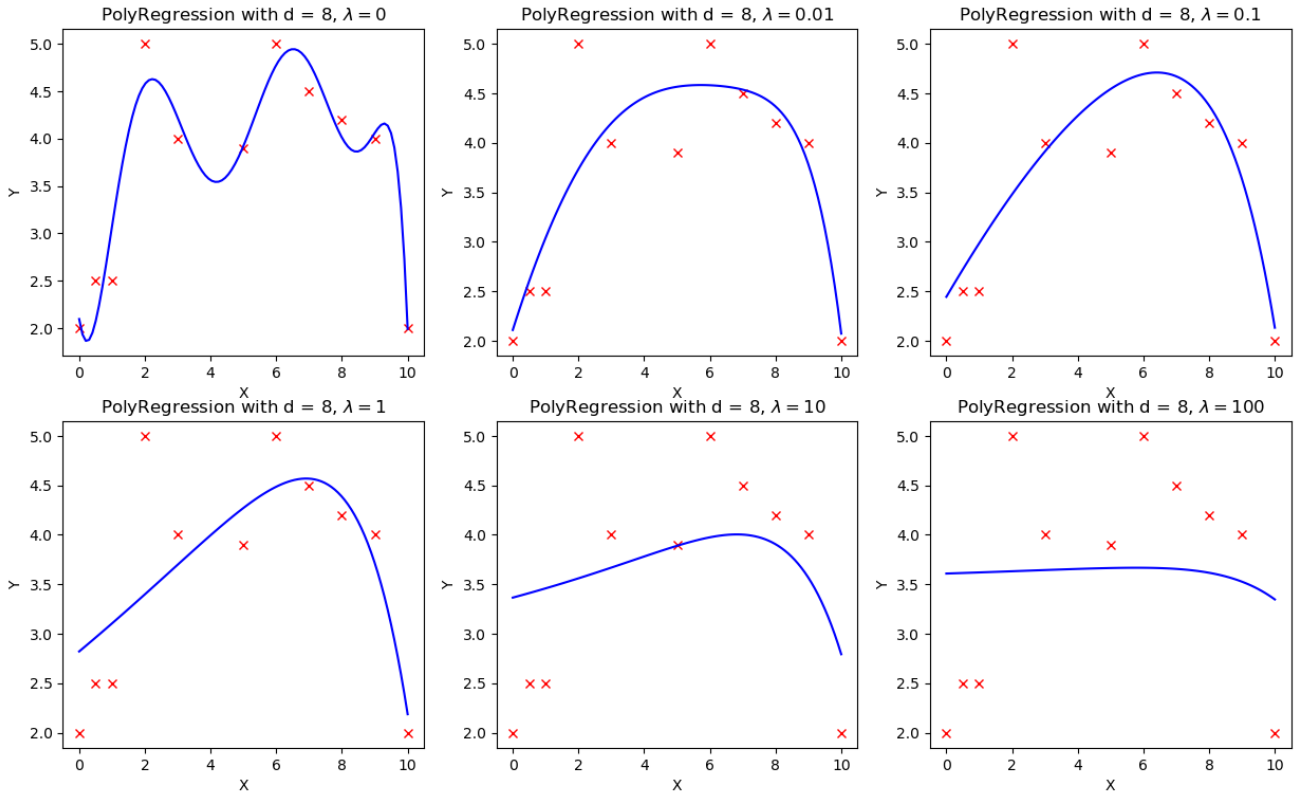


Figure 1: A4.1

Polynomial Regression

A.4 *[10 points]*
As increasing the amount of regularization, the resulting effect on the function becomes more and more flat.

A.5 *[10 points]*

Ridge Regression on MNIST

A.6. *[20 points]*

a. *[10 points]* Show that$\hat{W} = (X^T X + \lambda I)^{-1} X^T Y$

$$\bigtriangledown_W = 2X^T(XW - YI + 2\lambda W$$

$$Let \ \bigtriangledown_W = 0$$

$$2X^T X\hat{W} - 2X^T Y + 2\lambda\hat{W} = 0$$

$$X^T X\hat{W} + \lambda\hat{W} = X^T Y$$

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

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

b. *[10 points]* What is the training and testing error?

$$training \ error = 14.805\%$$

$$testing \ error = 14.66\%$$

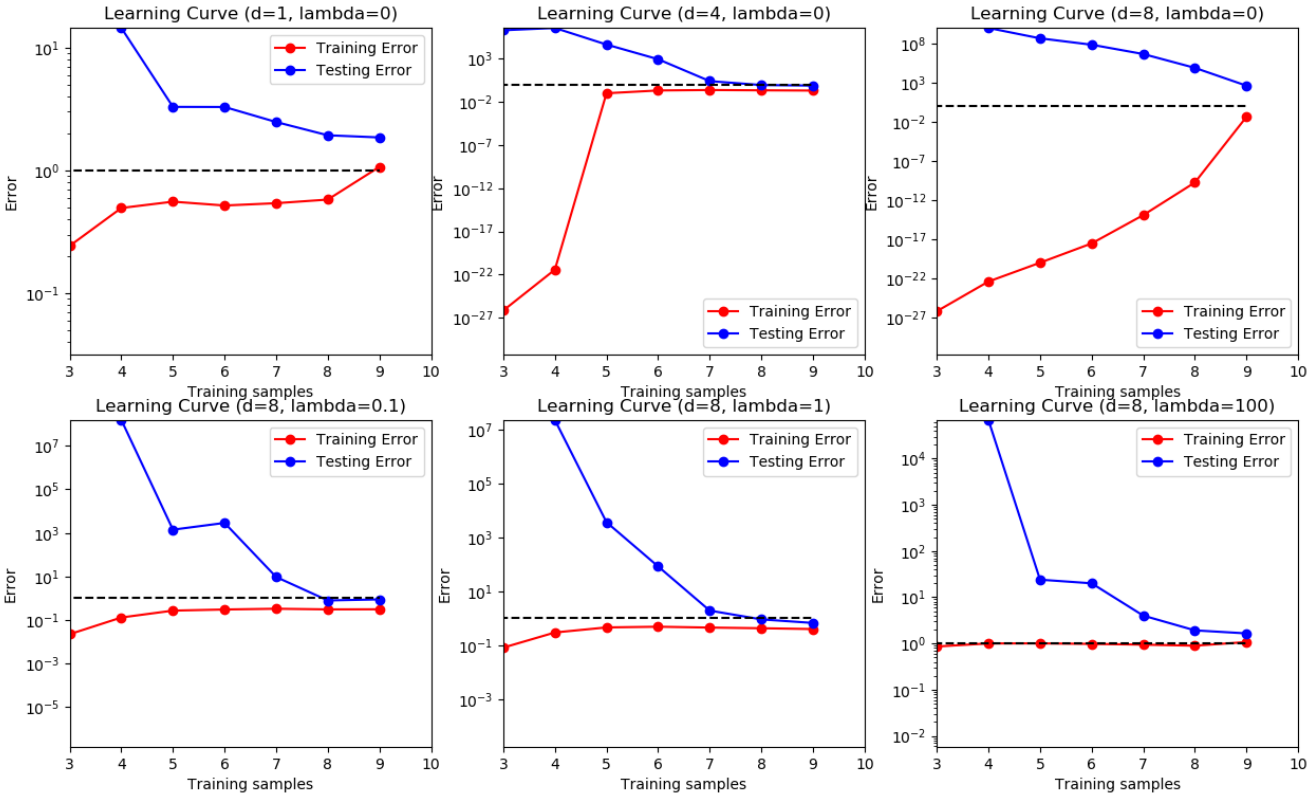

Learning Curve (d=1, lambda=0)
Learning Curve (d=4, lambda=0)
Learning Curve (d=8, lambda=0)
Learning Curve (d=8, lambda=0.1)
Learning Curve (d=8, lambda=1)
Learning Curve (d=8, lambda=100)
Training Error
Testing Error
Error
Training samples

Figure 2: A.5

[1]:
```
'''
    A.4 & A.5
    Template for polynomial regression
    AUTHOR Eric Eaton, Xiaoxiang Hu
'''

import numpy as np
import matplotlib.pyplot as plt

#----------------------------------------------------------------------
#  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.ScaleMean=None
        self.ScaleSigma=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 = len(X)
        XX = np.zeros((n,degree))
        for i in range(0,n):
            for j in range(1,self.degree+1):
                XX[i,j-1] = X[i]**j

        return XX #n-by-d
```

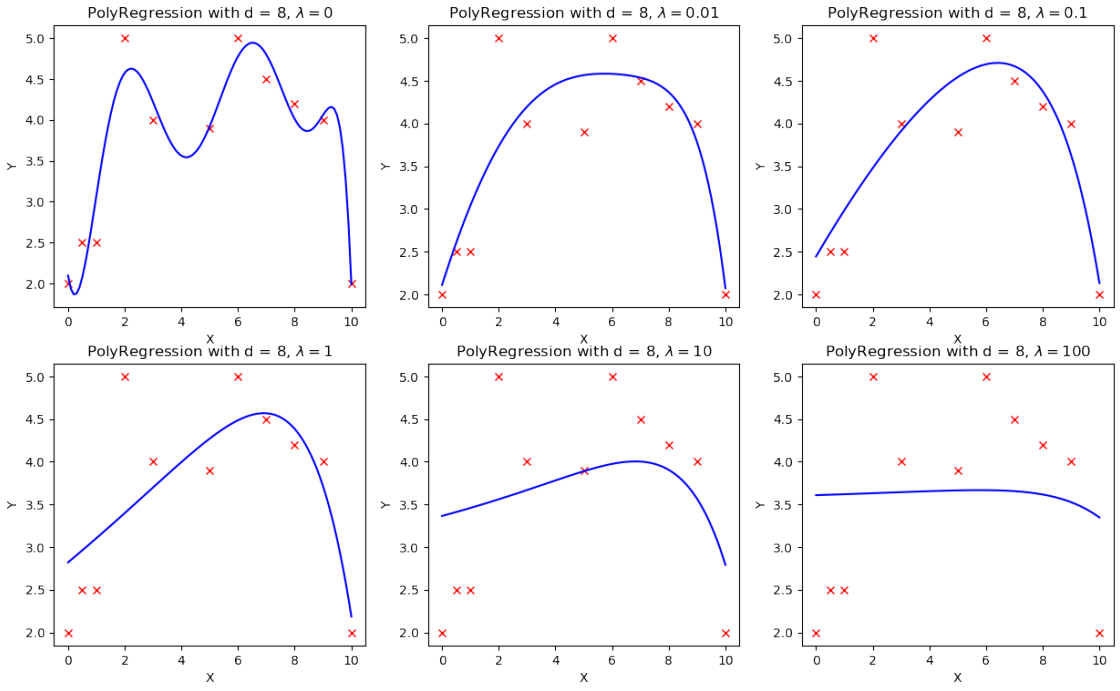
```
    def fit(self, X, y):
        """
            Trains the model
            Arguments:
                X is a n-by-1 array
                y is an n-by-1 array
            Returns:
                No return value
            Note:
                You need to apply polynomial expansion and scaling
                at first
        """
        n = len(X)
        XX = self.polyfeatures(X,self.degree)
        ##standalize
        self.ScaleMean = np.mean(XX, axis=0)
        self.ScaleSigma = np.std(XX, axis=0)
        XX = (XX - self.ScaleMean)/self.ScaleSigma
        #print(XX)


        # add 1s column
        XX_ = np.c_[np.ones([n, 1]),XX]

        n, d = XX_.shape
        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(XX_.T.dot(XX_) + reg_matrix).dot(XX_.T).
↪dot(y)


    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)
```

```
        XX = self.polyfeatures(X, self.degree)
        XX = (XX - self.ScaleMean)/self.ScaleSigma

        # add 1s column
        XX_ = np.c_[np.ones([n, 1]),XX]

        # predict
        return XX_.dot(self.theta)

#---------------------------------------------------------------------
#  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)
    #TODO -- complete rest of method; errorTrain and errorTest are already the␣
 ↪correct shape
    from polyreg import PolynomialRegression
    model = PolynomialRegression(degree=degree, reg_lambda=reg_lambda)

    for i in range(1,n):
```

```
        #fit model
        model.fit(Xtrain[0:(i+1)], Ytrain[0:(i+1)])
        errorTest[i] = ((model.predict(Xtest[0:(i+1)])-Ytest[0:(i+1)])**2).mean()
        errorTrain[i] = ((model.predict(Xtrain[0:(i+1)])-Ytrain[0:(i+1)])**2).
 ↪mean()

    return errorTrain, errorTest
```

[2]:
```
"""
    TEST SCRIPT FOR POLYNOMIAL REGRESSION 1
    AUTHOR Eric Eaton, Xiaoxiang Hu
"""

import numpy as np
import matplotlib.pyplot as plt
from polyreg import PolynomialRegression

def plotLearningCurve(X, y, reg_lambda, d):
    """
        plot computed learning curve
    """
        # regression with degree = d
    model = PolynomialRegression(degree=d, reg_lambda=reg_lambda)
    model.fit(X, y)

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

    plt.plot(X, y, 'rx')
    plt.title('PolyRegression with d = '+str(d) + ", $\lambda = $" +␣
 ↪str(reg_lambda))
    plt.plot(xpoints, ypoints, 'b-')
    plt.xlabel('X')
    plt.ylabel('Y')

if __name__ == "__main__":
    '''
        Main function to test polynomial regression
    '''

    # load the data
    filePath = "data/polydata.dat"
    file = open(filePath,'r')
    allData = np.loadtxt(file, delimiter=',')

    X = allData[:, [0]]
```

```
    y = allData[:, [1]]

    plt.figure(figsize=(15, 9), dpi=100)
    plt.subplot(2, 3, 1)
    plotLearningCurve(X, y, 0, 8)
    plt.subplot(2, 3, 2)
    plotLearningCurve(X, y, 0.01, 8)
    plt.subplot(2, 3, 3)
    plotLearningCurve(X, y, .1, 8)
    plt.subplot(2, 3, 4)
    plotLearningCurve(X, y, 1, 8)
    plt.subplot(2, 3, 5)
    plotLearningCurve(X, y,10, 8)
    plt.subplot(2, 3, 6)
    plotLearningCurve(X, y, 100, 8)
    plt.show()
```

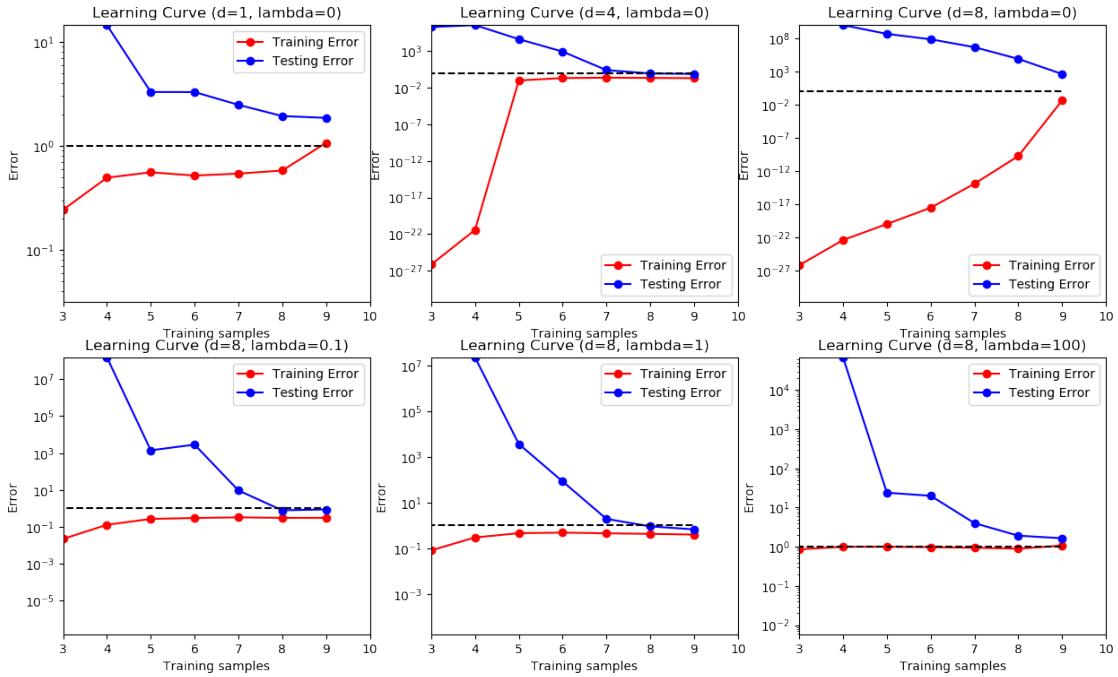


As increasing the amount of regularization, the resulting effect on the function becomes more and more flat.

[6]:
```
"""
    TEST SCRIPT FOR POLYNOMIAL REGRESSION 1
    AUTHOR Eric Eaton, Xiaoxiang Hu
"""

import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn import model_selection
from polyreg import learningCurve

#---------------------------------------------------
# Plotting tools

def plotLearningCurve(errorTrain, errorTest, regLambda, degree):
    """
        plot computed learning curve
    """
    minX = 3
    maxY = max(errorTest[minX+1:])

    xs = np.arange(len(errorTrain))
    plt.plot(xs, errorTrain, 'r-o')
    plt.plot(xs, errorTest, 'b-o')
    plt.plot(xs, np.ones(len(xs)), 'k--')
    plt.legend(['Training Error', 'Testing Error'], loc='best')
    plt.title('Learning Curve (d='+str(degree)+', lambda='+str(regLambda)+')')
    plt.xlabel('Training samples')
    plt.ylabel('Error')
    plt.yscale('log')
    plt.ylim(top=maxY)
    plt.xlim((minX, 10))


def generateLearningCurve(X, y, degree, regLambda):
    """
        computing learning curve via leave one out CV
    """

    n = len(X)

    errorTrains = np.zeros((n, n-1))
    errorTests = np.zeros((n, n-1))

    loo = model_selection.LeaveOneOut()
    itrial = 0
    for train_index, test_index in loo.split(X):
        #print("TRAIN indices:", train_index, "TEST indices:", test_index)
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        #regLambda = 0
        #degree =8
        (errTrain, errTest) = learningCurve(X_train, y_train, X_test, y_test,␣
↪regLambda, degree)
```

```
        errorTrains[itrial, :] = errTrain
        errorTests[itrial, :] = errTest
        itrial = itrial + 1

    errorTrain = errorTrains.mean(axis=0)
    errorTest = errorTests.mean(axis=0)

    plotLearningCurve(errorTrain, errorTest, regLambda, degree)



#-----------------------------------------------

if __name__ == "__main__":
    '''
        Main function to test polynomial regression
    '''

    # load the data
    filePath = "data/polydata.dat"
    file = open(filePath,'r')
    allData = np.loadtxt(file, delimiter=',')

    X = allData[:, [0]]
    y = allData[:, [1]]

    # generate Learning curves for different params
    plt.figure(figsize=(15, 9), dpi=100)
    plt.subplot(2, 3, 1)
    generateLearningCurve(X, y, 1, 0)
    plt.subplot(2, 3, 2)
    generateLearningCurve(X, y, 4, 0)
    plt.subplot(2, 3, 3)
    generateLearningCurve(X, y, 8, 0)
    plt.subplot(2, 3, 4)
    generateLearningCurve(X, y, 8, .1)
    plt.subplot(2, 3, 5)
    generateLearningCurve(X, y, 8, 1)
    plt.subplot(2, 3, 6)
    generateLearningCurve(X, y, 8, 100)
    plt.show()
```



A.6. *[20 points]*

a. *[10 points]* Show that$\hat{W} = (X^T X + \lambda I)^{-1} X^T Y$

$$\bigtriangledown_W = 2X^T(XW - YI) + 2\lambda W$$

$$Let \bigtriangledown_W = 0$$

$$2X^T X\hat{W} - 2X^T Y + 2\lambda\hat{W} = 0$$

$$X^T X\hat{W} + \lambda\hat{W} = X^T Y$$

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

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

b. *[10 points]* What is the training and testing error?

$$training\ error = 14.805\%$$

$$testing\ error = 14.66\%$$

[7]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 16 13:10:52 2020
CSE 546 HW 1 A.6
@author: Leah
"""
```

```
import numpy as np
from mnist import MNIST

def load_dataset():
    mndata = MNIST('./data/')
    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

    #show the image
    #import matplotlib.pyplot as plt
    #%matplotlib inline # Only use this if using iPython
    #img_h = img_w = 28
    #image_index = 1 # You may select anything up to 60,000
    #print(labels_train[image_index]) # The label is 8
    #plt.imshow(X_train[image_index].reshape((img_h, img_w)), cmap='Greys')

    return X_train, labels_train, X_test, labels_test

def train(X,Y,lamda):
    img_h = img_w = 28              # MNIST images are 28x28
    img_size_flat = img_h * img_w  # 28x28=784, the total number of pixels
    W_hat = np.linalg.pinv(X.T.dot(X)+lamda*np.eye(img_size_flat)).dot(X.T).
 ↪dot(Y)
    return W_hat

def predict(W,XX):
    X_ = W.T.dot(XX.T)
    pre_ = X_.argmax(axis=0)
    return pre_

#------------------------------------------------------------------
#  Main
#------------------------------------------------------------------

n_classes = 10                  # Number of classes, one class per digit

#LOAD data
X_train, labels_train, X_test, labels_test = load_dataset()

## transform labels_train(n-by-1) into n-by-k
Y_train = np.zeros((len(labels_train),n_classes))
for i in range(0,len(Y_train)):
    Y_train[i,labels_train[i]] = 1

#get w_hat
```

```
W_hat = train(X_train,Y_train,lamda = 0.0001)

#predict
Train_pre = predict(W_hat,X_train)
Test_pre = predict(W_hat,X_test)

##training, test error
train_error = sum(Train_pre != labels_train)/len(labels_train)
test_error = sum(Test_pre != labels_test)/len(labels_test)

print("training error :" + str(train_error))
print("test error :" + str(test_error))
```

```
training error :0.14805
test error :0.1466
```