Homework #1A

Spring 2020, CSE 446/546: Machine Learning
Prof. Kevin Jamieson and Prof. Jamie Morgenstern
Due: 4/25/20 11:59 PM
A: 90 points. B: 50 points

Short Answer and "True or False" Conceptual questions

A.0

a. (1)Bias is the difference between the average prediction of our model and the correct value which we are trying to predict.Variance is the variability of model prediction for a given data point or a value which tells us spread of our data. The trade-off here means to reach the balance between low bias and low variance.

b. (2)when model complexity increases, the model bias is lower and variance is higher and when model complexity decreases, the model bias is higher and variance is lower.

c. FALSE

d. TRUE

e. FALSE

f. Train set.

g. FALSE

Maximum Likelihood Estimation (MLE)

A.1

a. the maximum likelihood function is
$L = e^{-5\lambda} \frac{\lambda^{x_1+x_2+x_3+x_4+x_5}}{x_1!x_2!x_3!x_4!x_5!}$
so the log-likelihood is
$l = -5\lambda + (x_1 + x_2 + x_3 + x_4 + x_5) log\lambda + constant$
then$\frac{dl}{d\lambda} = -5 + \frac{x_1+x_2+x_3+x_4+x_5}{\lambda}$
if we set $\frac{dl}{d\lambda} = 0$, we can get $\hat{\lambda} = -\frac{x_1+x_2+x_3+x_4+x_5}{5}$

b. similarly $\hat{\lambda} = -\frac{x_1+x_2+x_3+x_4+x_5+x_6}{6}$

c. after 5 games, $\hat{\lambda} = -\frac{x_1+x_2+x_3+x_4+x_5}{5} = -1.2$
after 6 games, $\hat{\lambda} = -\frac{x_1+x_2+x_3+x_4+x_5+x_6}{6} = -\frac{5}{3}$

A.2
the maximum likelihhod function is
$L = \frac{1}{\theta^n}$ $\mathbf{1}\{0\leq$ x$_i\leq \theta\}$
It can be seen that the MLE of θ must be a value of θ for which $x_i\leq \theta$ wherei=1,2.....,n;and which maximizes L among all such values. Since L is a decreasing function of θ, the estimate will be the smallest possible value of θ, so $\theta = max\{x_1, x_2, ...x_n\}$

Overfitting

A.3

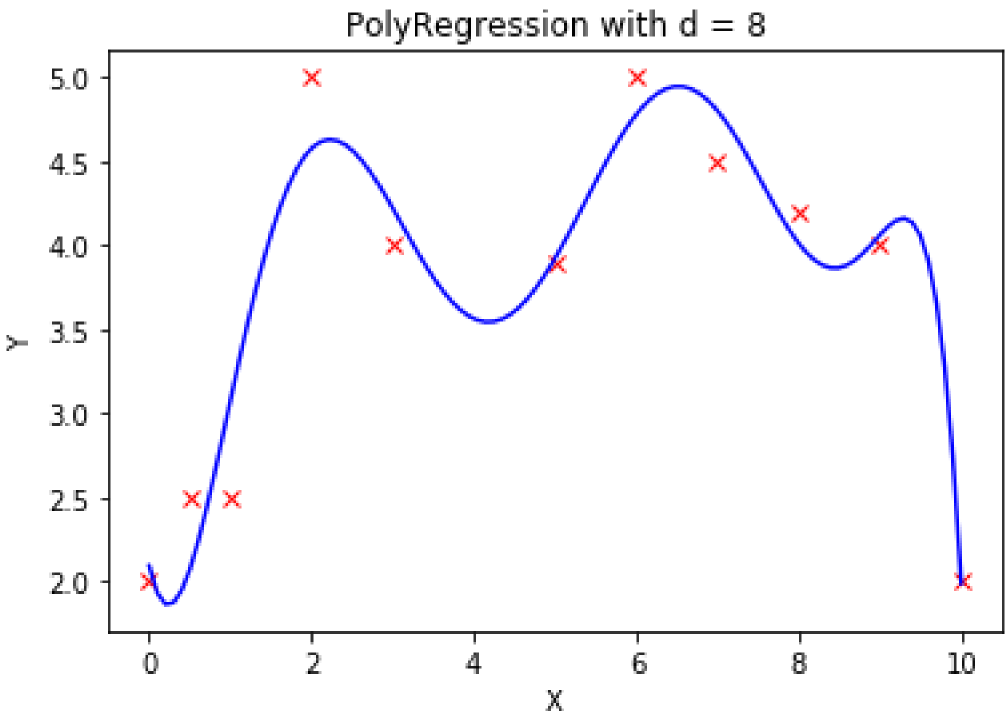
a. $E_{train}[\hat{\epsilon}_{train}(f)] = \frac{N_{train}E_D[(f(x)-y)^2]}{N_{train}} = \epsilon(f)$ and similarly
$E_{test}[\hat{\epsilon}_{test}(f)] = \frac{N_{test}E_D[(f(x)-y)^2]}{N_{test}} = \epsilon(f)$
$E_{test}[\hat{\epsilon}_{test}(\hat{f})] = \frac{N_{test}E_D[(\hat{f}(x)-y)^2]}{N_{test}} = \epsilon(\hat{f})$

b. No, the training error is biased because it is evaluated on the data it trained on.

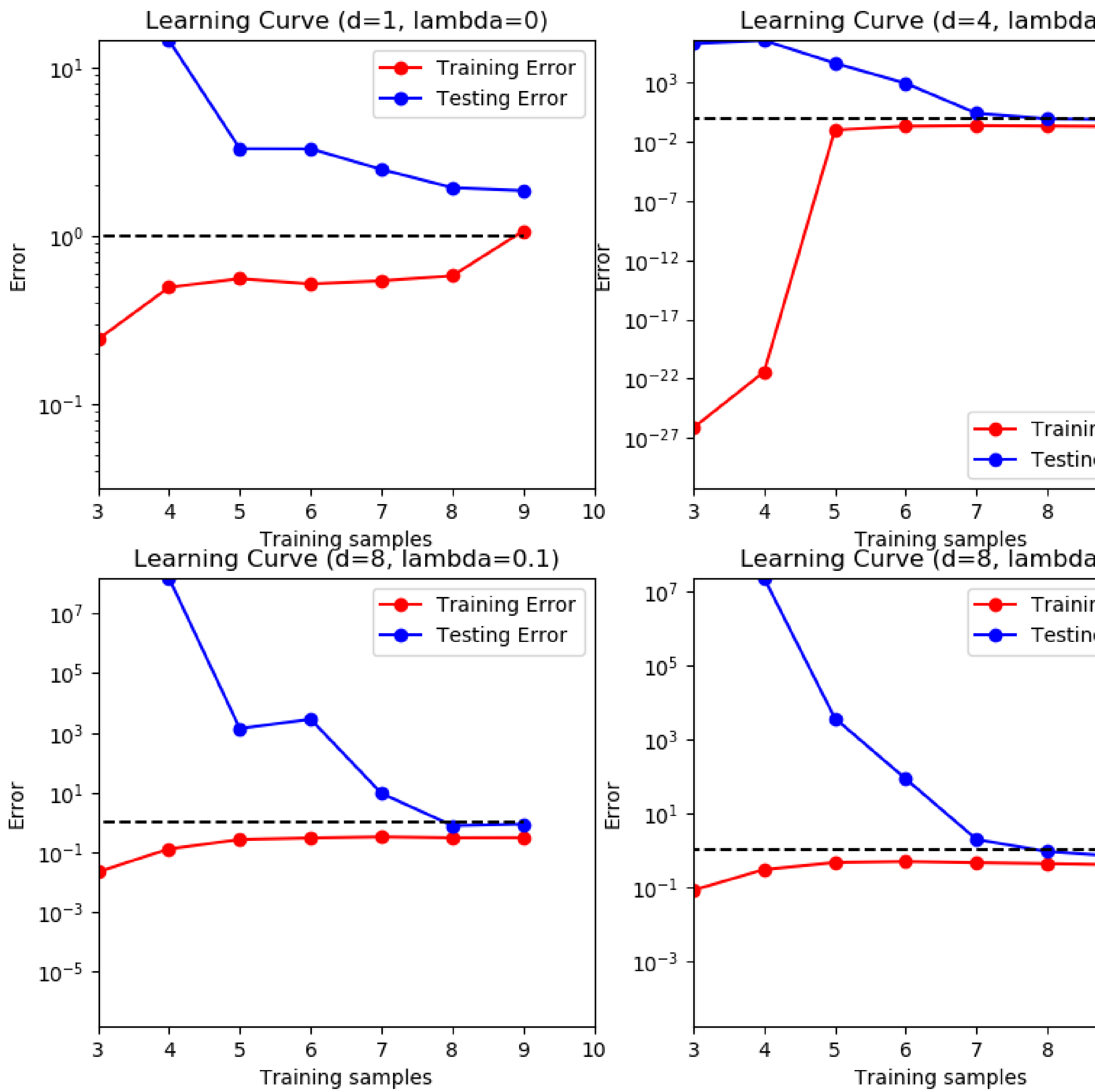
c. According to the hint,
$E_{train,test}[\hat{\epsilon}_{test}(\hat{f}_{train)}] = \Sigma E_{test}[\hat{\epsilon}_{test}(f)]P_{train}(\hat{f}_{train} = f)$
so $E_{train}[\hat{\epsilon}_{train}(\hat{f}_{train})] \leq$ $E_{train,test}[\hat{\epsilon}_{test}(\hat{f}_{train)}]$ because $\hat{f}_{train}$ minimize the error.

Polynomial Regression

A.4



A.5



Listing 1: A.4A.5 code

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

import numpy as np
```

```python
#--------------------------------------------------------------------------------
#   Class PolynomialRegression
#--------------------------------------------------------------------------------

class PolynomialRegression:

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

    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
        """
        return np.array([[i[0, ] ** d for d in np.arange(1,self.degree +1)] for i in 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
        """
        X_exp=self.polyfeatures(X, self.degree)
        n = len(X_exp)
        if n>1:
            self.trainmean= np.mean(X_exp, axis=0)
            self.trainsd=np.std(X_exp, axis=0)
        else:
            self.trainmean=X_exp
            self.trainsd=np.ones((1, n))
```

```python
        X_scaled = (X_exp-self.trainmean)/self.trainsd

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

        n, d = X_.shape
        d = d-1  # remove 1 for the extra column of ones we added to get the original num fe

        # 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)

    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
        """
        X_exp = self.polyfeatures(X, self.degree)
        n = len(X_exp)
        X_scaled = (X_exp-self.trainmean)/self.trainsd
        # add 1s column
        X_ = np.c_[np.ones([n, 1]), X_scaled]

        # predict
        return X_.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
"""

n = len(Xtrain)

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

for i in np.arange(1,n+1):
    M = PolynomialRegression(degree = degree, reg_lambda= reg_lambda);
    M.fit(Xtrain[0:i], Ytrain[0:i])
    Ytrain_pred = M.predict(Xtrain[0:i])
    Ytest_pred = M.predict(Xtest)
    errorTrain[i-1]=np.mean((Ytrain_pred-Ytrain[0:i])**2)
    errorTest[i-1]=np.mean((Ytest_pred-Ytest)**2)

return errorTrain, errorTest
```

Ridge Regression on MNIST

A.6

a. To get $\hat{W}$, we need to take the derivative of $\sum_{j=0}^{k}[\|Xw_j - Ye_j\|^2 + \lambda\|w_j\|^2]$, which is $\sum_{j=0}^{k}[2X^T\|Xw_j - Ye_j\| + 2\lambda\|w_j\|$ and if we set it to 0, we can get $\hat{W} = (X^TX)^{-1}X^TY$

b. The training error is 0.14815 and the test error is 0.1465.

```
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 24 21:39:49 2020

@author: ASUS
"""
import numpy as np
from scipy.linalg import solve as sol
from mnist import MNIST

def train(X, Y, lamda):
    dim = X.shape[1]
    return sol(X.T.dot(X)-lamda * np.eye(dim), np.eye(dim)).dot(X.T).dot(Y)

def predict(W, X_prime):
    vector = np.zeros(X_prime.shape[0])
    for i in range(X_prime.shape[0]):
        vector[i]=W.T.dot(X_prime[i]).argmax()
    return (vector)


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,labels_train, X_test, labels_test = load_dataset()



one_hot_train = np.zeros((labels_train.size, labels_train.max() + 1))
one_hot_train[np.arange(labels_train.size), labels_train] = 1
What = train(X_train, one_hot_train, 10e-4)
labels_train_pred=predict(What, X_train)
labels_test_pred= predict(What, X_test)

trainerror = np.array([labels_train[j] != labels_train_pred[j] for j in range(labels_tra
testerror = np.array([labels_test[j] != labels_test_pred[j] for j in range(labels_test.


print(trainerror)
print(testerror)
```