Homework #1 - A

Spring 2020, CSE 446/546: Machine Learning
Richy Yun
Due: 4/24/20 11:59 PM
Collaborators: Samantha Sun

Short Answer and "True or False" Conceptual questions

A.0

- *[2 points]* In your own words, describe what bias and variance are? What is bias-variance tradeoff?

  Bias can be thought of the error between the average output of the model and the real data (ground "truth"). Variance is the variability in the output of the model. Bias-variance tradeoff refers to how models with high bias result in low variance and vice versa, which forces designs to find the proper balance between bias and variance. In essence, we have to decide how well the model fits the train data - the better the fit the lower the variance, but generally higher bias due to overfitting. On the flip side, the more generalized the fit the higher the variance, but lower bias.

- *[2 points]* What happens to bias and variance when the model complexity increases/decreases?

  As model complexity increases, bias goes down as the result becomes more and more overfitted to the train data, but variance goes up. As complexity decreases, bias goes up due to generalization but variance goes down.

- *[1 points]* True or False: The bias of a model increases as the amount of training data available increases.

  False. The bias decreases, as more training data results in a better fit.

- *[1 points]* True or False: The variance of a model decreases as the amount of training data available increases.

  True. As there is more training data the model is better fitted, leading to less noise and lower variance.

- *[1 points]* True or False: A learning algorithm will generalize better if we use less features to represent our data.

  True. More features leads to overfitting which does not generalize well. Thus less features is a more generalized model.

- *[2 points]* To get better generalization, should we use the train set or the test set to tune our hyperparameters?

  We should always use the train set to tune our hyperparameters.

- *[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.

  False. The training error is generally smaller than the test error (an estimate of true error) as the function is overfit to the training set. The expected value of the training error is always less than or equal to the expected value of the test error.

Maximum Liklihood Estimation (MLE)

A.1

a. *[5 points]* To derive the maximum-likelihood estimate by taking the log of the likelihood to covert it from a product to a sum, then taking the derivative and setting it equal to zero:

$$\begin{aligned}
L(\lambda) &= \prod_{i=1}^{n} \text{Poi}(x_i|\lambda) \\
&= \sum_{i=1}^{n} \log(\text{Poi}(x_i|\lambda)) \\
\hat{\lambda}_{MLE} &= \arg\max_x \ (L(\lambda)) \\
&= \arg\max_x \ \sum_{i=1}^{n} \log(e^{-\lambda}\frac{\lambda_i^x}{x_i!}) \\
0 &= \sum_{i=1}^{n} \left(\frac{d}{d\lambda}\log(e^{-\lambda}\frac{\lambda_i^x}{x_i!})\right) \\
&= \sum_{i=1}^{n} \left(\frac{d}{d\lambda}(-\lambda + x_i\log(\lambda) - \log(x_i!))\right) \\
&= \sum_{i=1}^{n} \left(-1 + \frac{x_i}{\lambda}\right) \\
n\lambda &= \sum_{i=1}^{n} x_i \\
\hat{\lambda}_{MLE} &= \frac{1}{n}\sum_{i=1}^{n} x_i \\
&= \frac{1}{5}\sum_{i=1}^{5} x_i
\end{aligned}$$

Thus, the maximum likelihood estimate is the average of the scores.

b. *[5 points]* Similar to above, the maximum likelihood estimate is the average:

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

c. *[5 points]* For 5 and 6 games, we simply find the average of the goal counts to get the estimate of λ. For 5 games, $\lambda = \frac{6}{5}$. For 6 games, $\lambda = \frac{10}{6} = \frac{5}{3}$

A.2 *[10 points]* We first approach this problem similar to part a:

$$
\begin{aligned}
&= \begin{cases} L(\theta) = \prod_{i=1}^{n} \text{Uniform}(x_i|\theta) & 0 \leq x_i \leq \theta \\ 0 & \text{otherwise} \end{cases} \\
&= \sum_{i=1}^{n} \log(\text{Uniform}(x_i|\theta)) \\
\hat{\theta}_{MLE} &= \arg\max_x \left(L(\theta)\right) \\
&= \arg\max_x \sum_{i=1}^{n} \log\left(\frac{1}{\theta}\right) \\
0 &= \sum_{i=1}^{n} \left(\frac{d}{d\theta}(\log(1) - \log(\theta))\right) \\
&= \sum_{i=1}^{n} -\frac{1}{\theta} \\
&= -\frac{n}{\theta}
\end{aligned}
$$

However, there is no global maxima. Thus we need to determine θ by evaluating the likelihood conceptually:

$$
\begin{aligned}
L(\theta) &= \prod_{i=1}^{n} \text{Uniform}(x_i|\theta) \\
&= \frac{1}{\theta^n}
\end{aligned}
$$

To maximize $L(\theta)$ we clearly need to minimize θ. However, $\theta \geq x_i$ for all x_i as we cannot have sampled outside the limits of the distribution. Thus, the value that satisfies both conditions is the maximum of all samples, or:

$$\hat{\theta}_{MLE} = \max(x_i)$$

Overfitting

A.3

a. *[3 points]*

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

Since each sample (x, y) in S is i.i.d. we can remove the sum and the $1/N_{\text{train}}$. Since f is fixed and not dependent on the training set (we haven't see any data yet), the expected value does not have to be over the train either. We thus have:

$$\mathbb{E}_{\text{train}}[\hat{\epsilon}_{\text{train}}(f)] = \mathbb{E}[(f(x)-y)^2]$$

Using the exact same logic on $\mathbb{E}_{\text{test}}[\hat{\epsilon}_{\text{test}}(f)]$ we get:

$$\mathbb{E}_{\text{train}}[\hat{\epsilon}_{\text{train}}(f)] = \mathbb{E}_{\text{test}}[\hat{\epsilon}_{\text{test}}(f)] = \epsilon(f) = \mathbb{E}[(f(x)-y)^2]$$

And using the same reasoning, since $\hat{f}$ is not dependent on the testing set, we can also show:

$$\mathbb{E}_{\text{test}}[\hat{\epsilon}_{\text{test}}(\hat{f})] = \epsilon(\hat{f}) = \mathbb{E}[(\hat{f}(x)-y)^2]$$

b. *[4 points]* No, because $\hat{f}$ is dependent on the training set, so (x,y) are not random variables anymore. Thus $\mathbb{E}_{\text{train}}[(\hat{f}(x)-y)^2] \neq \mathbb{E}[(\hat{f}(x)-y)^2]$

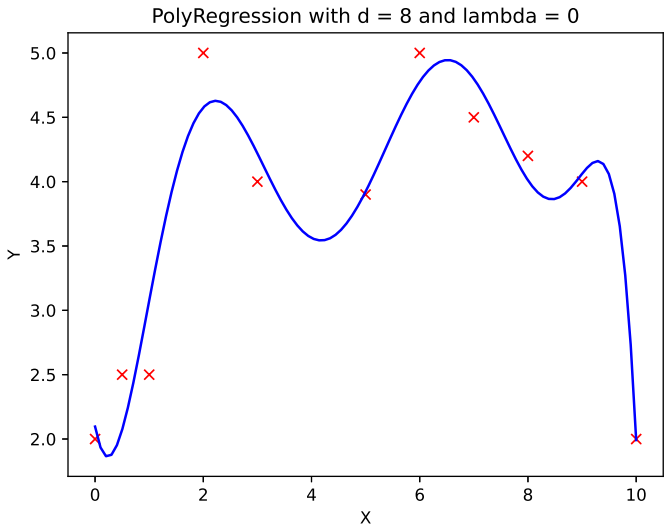
c. *[8 points]* We can use the hint to further simplify the right hand side of the equation:

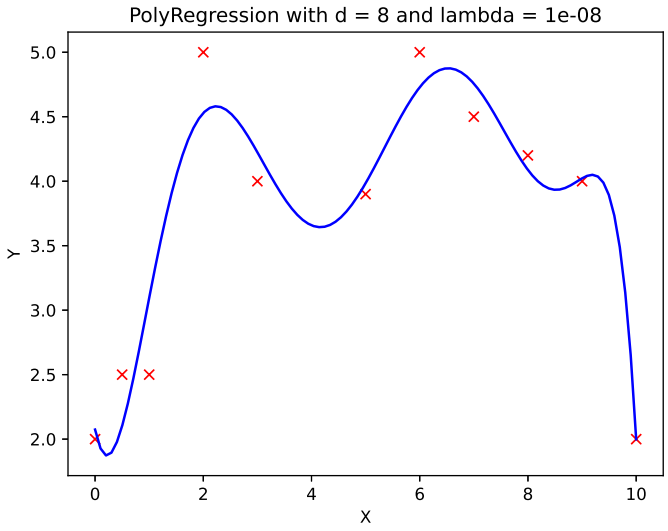
$$\begin{aligned}\mathbb{E}_{\text{train}}[\hat{\epsilon}_{\text{train}}(\hat{f}_{\text{train}})] &\leq \mathbb{E}_{\text{train,test}}[\hat{\epsilon}_{\text{test}}(\hat{f}_{\text{train}})] \\ &\leq \sum_{f\in\mathcal{F}}\mathbb{E}_{\text{test}}[\hat{\epsilon}_{\text{test}}(f)]\mathbb{P}_{\text{train}}(\hat{f}_{\text{train}}=f)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

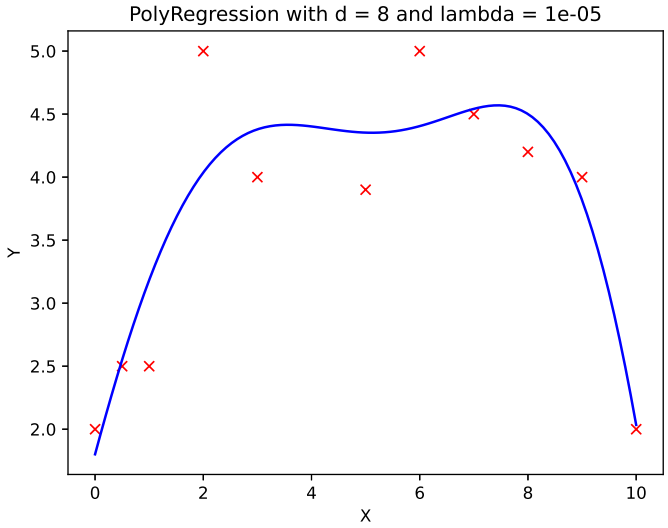
with the last step due to part (a). Since $\hat{\epsilon}_{\text{train}}(\hat{f}_{\text{train}}) = \min(\hat{\epsilon}_{\text{train}}(f))$, we know $\mathbb{E}_{\text{train}}[\min(\hat{\epsilon}_{\text{train}}(f))] \leq \mathbb{E}_{\text{train}}[\hat{\epsilon}_{\text{train}}(f)]$ must be true and so the original inequality stands.

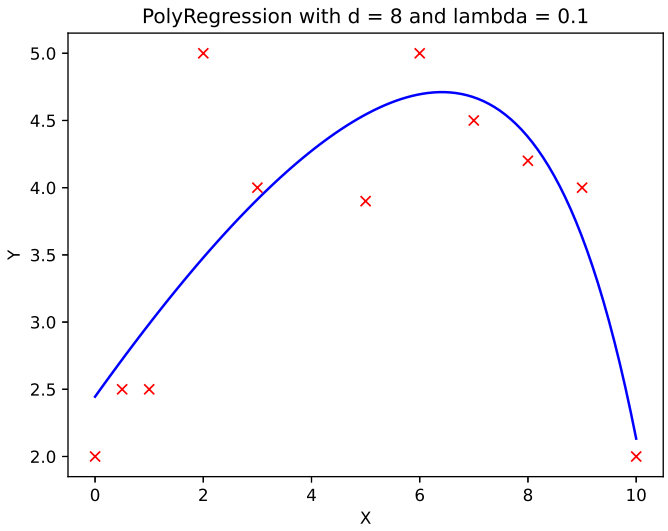
Polynomial Regression

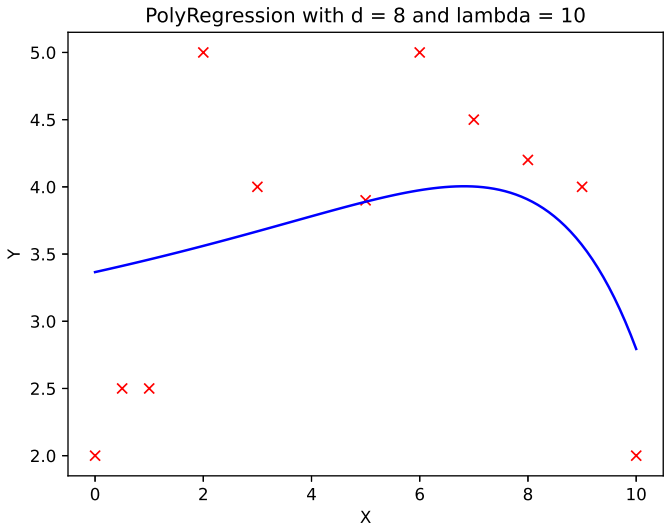
A.4 *[10 points]* As we increase regularization, we end up with less complex, more generalized solutions. We see subtle changes with λ as low as $1e-8$, with many features being lost at $1e-5$. At $\lambda = 0.1$ we essentially have a quadratic solution, with even those features becoming dimished at $\lambda = 10$.



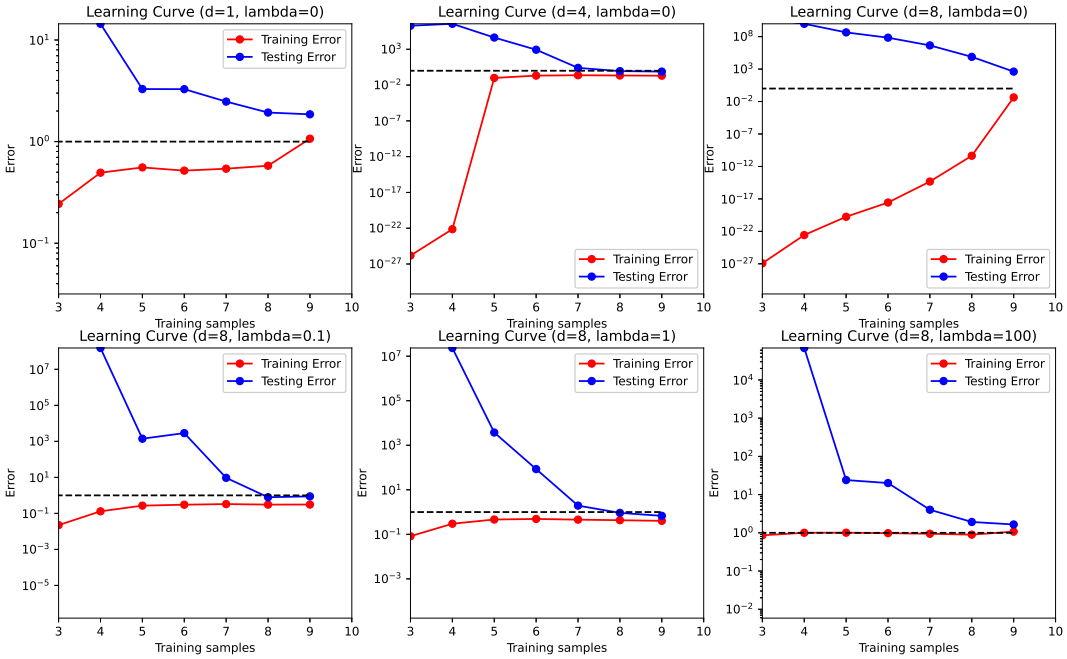



PolyRegression with d = 8 and lambda = 1e-05
5.0
4.5
4.0
3.5
3.0
2.5
2.0
Y
0
2
4
6
8
10
X


PolyRegression with d = 8 and lambda = 0.1
5.0
4.5
4.0
3.5
3.0
2.5
2.0
Y
0
2
4
6
8
10
X


PolyRegression with d = 8 and lambda = 10
5.0
4.5
4.0
3.5
3.0
2.5
2.0
Y
0
2
4
6
8
10
X

A.5 *[10 points]* As expected, we see greater training error with more samples as the model cannot perfectly fit all the data, which is compensated by the increased number of degrees (top row). With higher regularization we also see higher training error due to the increased generalization (bottom row). For testing error, we see an increase with the increase of degrees likely due to overfitting to the training set (top row), which is mitigated by inccreasing regularization for generalization of the model (bottom row).



A.4 and A.5 (polyreg.py)

```
'''
    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.degree = degree
        self.reg_lambda = reg_lambda
        self.theta = None
        self.avg = None      # for normalization
        self.std = None      # for normalization
```

```
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
    """

    # Polynomial expansion
    n = len(X)
    xpoly = np.empty([n, degree])
    for x in range(0, n):
        for d in range(0, degree):
            xpoly[x, d] = np.power(X[x], d+1)

    return xpoly

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
    """
    # Polynomial expansion
    x = self.polyfeatures(X, self.degree)

    # Normalization
    self.avg = np.mean(x, axis=0)
    self.std = np.std(x, axis=0)
    x = x-self.avg[None, :]
    x = x/self.std[None, :]

    # From linreg_closedform.py
    ################################
    n = len(X)

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

    n, d = X_.shape
    d = d - 1  # remove 1 for the extra column of ones we added
```

```
        # construct reg matrix
        reg_matrix = self.reg_lambda * 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
        """

        # Polynomial expansion
        x = self.polyfeatures(X, self.degree)

        # Normalization
        x = x - self.avg[None, :]
        x = x / self.std[None, :]

        # From linreg_closedform.py
        ################################
        n = len(X)

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

        # 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 -- errorTest[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)

reg = PolynomialRegression(degree, reg_lambda)

for i in range(1, n):
    reg.fit(Xtrain[0:i+1], Ytrain[0:i+1])
    trainpredict = reg.predict(Xtrain[0:i+1])
    errorTrain[i] = np.sum((trainpredict - Ytrain[0:i+1])**2)/len(Ytrain[0:i+1])
    testpredict = reg.predict(Xtest)
    errorTest[i] = np.sum((testpredict - Ytest)**2)/len(Ytest)

return errorTrain, errorTest
```

A.6

a. *[10 points]* To find $\widehat{W}$ we take the gradient with respect to W and set it equal to zero to find the minimum as shown during lecture:

$$\nabla_W \left[\sum_{j=0}^{k} ||Xw_j - Ye_j||^2 + \lambda ||w_j||^2 \right] = 2X^T(XW - Y) + 2\lambda W = 0$$

$$\begin{aligned} X^T(XW - Y) + \lambda W &= 0 \\ X^T XW - X^T Y + \lambda W &= 0 \\ X^T XW + \lambda W &= X^T Y \\ (X^T X + \lambda I)W &= X^T Y \\ \widehat{W} &= (X^T X + \lambda I)^{-1} X^T Y \end{aligned}$$

e is omitted from the calculations as it is a simple one-hot encoding (identity matrix). The identity matrix, I, is included with λ when isolating W to form a matrix and allow for the addition as λ is a scalar.

b. *[10 points]* The training error is 14.805% and the testing error is 14.66%.

A.6b

```
import numpy as np
import matplotlib.pyplot as plt
from mnist import MNIST
from scipy import linalg

def train(X, Y, lam):
    d = int(X.size/len(X))
    reg_matrix = lam * np.eye(d)
    a = np.matmul(X.transpose(), X) + reg_matrix
    b = np.matmul(X.transpose(), Y)
    W = linalg.solve(a, b)
    return W

def predict(W, X):
    d = len(X)
    Y = np.zeros(d)
    temp = np.matmul(X, W)
    for i in range(0, d):
        ind = np.where(temp[i, :] == np.amax(temp[i, :]))
        Y[i] = ind[0]
    return Y

# Load data
mndata = MNIST('./data/')
X_train, labels_train = map(np.array, mndata.load_training())
X_test, labels_test = map(np.array, mndata.load_testing())

# Rearrange data
X_train = X_train/255.0
X_test = X_test/255.0
Y_train = np.zeros((len(X_train), 10))
Y_train[range(0, len(X_train)), labels_train] = 1
Y_test = np.zeros((len(X_test), 10))
```

```
Y_test[range(0, len(X_test)), labels_test] = 1

# Train model
lam = 0.0001
W = train(X_train, Y_train, lam)

# Get predictions
P_train = predict(W, X_train)
P_test = predict(W, X_test)

# Error calculation
train_error = (len(X_train) - np.sum(P_train == labels_train))/len(X_train)
test_error = (len(X_test) - np.sum(P_test == labels_test))/len(X_test)

# Print output
print('Training error: ' + str(train_error*100) + '%')
print('Testing error: ' + str(test_error*100) + '%')
```