Homework #1B

Spring 2020, CSE 446/546: Machine Learning
Prof. Kevin Jamieson and Prof. Jamie Morgenstern
Due: 4/24/20 11:59 PM
A: 90 points. B: 50 points

Bias-Variance tradeoff

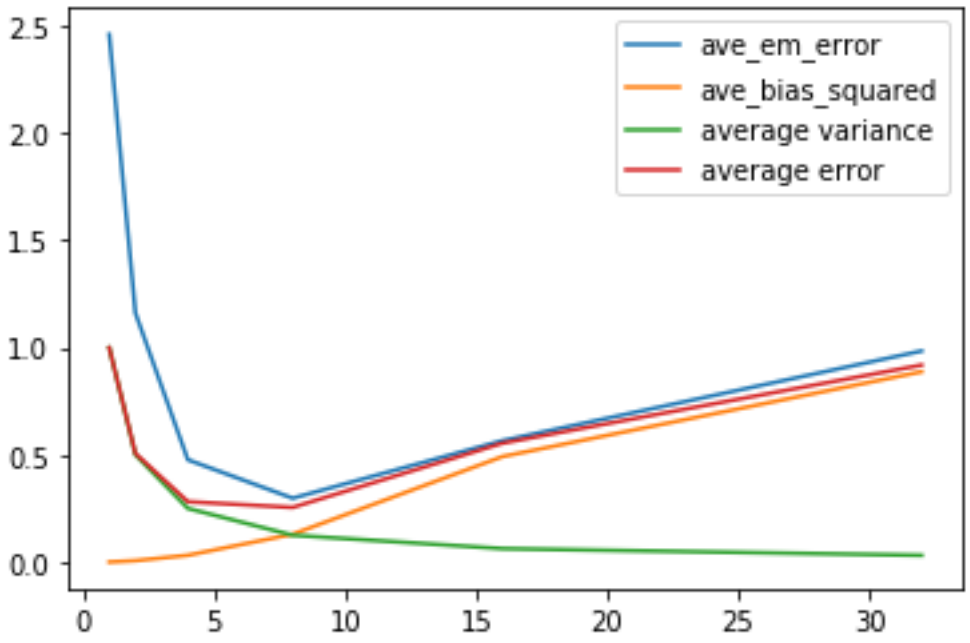
B.1

a. For small values of m, the bias will decrease and the variance will increase. And for large values of m, the bias will increase and the variance will decrease.

b. $\frac{1}{n}\sum_1^n(E[\hat{f}_m(x_i)] - f(x_i))^2 = \frac{1}{n}\sum_{n/m}^{j=1}\sum_{i=(j-1)m+1}^{jm}(E[\hat{f}_m(x_i)] - f(x_i))^2$
and since $E[\hat{f}_m(x_i)] = \bar{f}^{(j)}$, the result certainly holds here.

c. For the first equality,
$E[\frac{1}{n}\sum_1^n(\hat{f}_m - E[\hat{f}_m])^2] = \frac{1}{n}\sum_{j=1}^{n/m} mE[(\hat{f}_m - E[\hat{f}_m])^2] = \frac{1}{n}\sum_{j=1}^{n/m} mE[(c_j - \bar{f}^{(j)})^2]$
For the second equality
$E[(c_j - \bar{f}^{(j)})^2] = \sigma^2$so we can get
$\frac{1}{n}\sum_{j=1}^{n/m} mE[(c_j - \bar{f}^{(j)})^2] = \frac{\sigma^2}{m}$

d. the plot is as follows:



Listing 1: $\text{hw1}_b1_4 code$

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

```
n=256;
sigma_2 = 1;

def f(x):
    return 4*np.sin(x*np.pi)*np.cos(6*np.pi*x**2)

def hat_fm(x,y,m,n):
    h=np.array([sum(y[int((i-1)*m):int(i*m)])/m for i in np.arange(1, n/(1+m))])
    l=np.array([h[int(j-1)]*int((j-1)*m/n<x<=j*m/n) for j in np.arange(1,n/(1+m))]).sum
    return l

def c_square(x):
    return (x-x.mean())**2

#initial
x = np.arange(1, 1+n)/n
y = f(x) + np.random.randn(n)
m = np.array([1,2,4,8,16,32])
ave_em_err = np.zeros(len(m))
ave_bias_sq = np.zeros(len(m))

for i in np.arange(len(m)):
    ave_em_err[i] = np.array([(hat_fm(j, y, m[i], n) - f(j)) ** 2 for j in x]).mean()

for i in np.arange(len(m)):
    ave_bias_sq[i]=np.array([c_square(np.array([f(j) for j in x[int((p-1)*m[i]) : int(p


plt.plot(m, ave_em_err, label='ave_em_error')
plt.plot(m, ave_bias_sq, label = 'ave_bias_squared')
plt.plot(m, 1/m, label = 'average variance')
plt.plot(m, ave_bias_sq + 1/m, label = 'average error')

plt.legend()
plt.show()
```

e. $\frac{1}{n}\sum_{n/m}^{j=1}\sum_{i=(j-1)m+1}^{jm}(\bar{f}^{(j)} - f(x_i))^2$
$\leq \frac{1}{n}\sum_{n/m}^{j=1}\sum_{i=(j-1)m+1}^{jm}(max - min)$
$\leq \frac{1}{n}\sum_{n/m}^{j=1}\sum_{i=(j-1)m+1}^{jm}\frac{L^2}{n^2}m^2$ and thus it's $O(\frac{L^2}{n^2}m^2)$
To minimize m in $f = \frac{L^2}{n^2}m^2 + \frac{\sigma^2}{m}$ we need to figure out $\frac{df}{dm}$ and set this to 0.
Finally we get $\frac{df}{dm} = \frac{2L^2}{n^2}m - \frac{\sigma^2}{m^2}$ and $\hat{m} = (\frac{n^2\sigma^2}{2L^2})^(1/3)$. m increases as n and σ increase and L decrease.

Ridge Regression on MNIST

B.2

a. from the plot, we take p = 6000

b. the test error when p=6000 is 0.8967
and the confidence interval is [0.8831189848425939, 0.9102810151574062]

Listing 2: hw1$_b$2*code*

```
import numpy as np
```

```
from scipy.linalg import solve as sol
from mnist import MNIST
import matplotlib.pyplot as plt

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()


thistrain = np.random.choice(range(X_train.shape[0]), int(0.8*X_train.shape[0]), replace = F
X_thistrain = X_train[thistrain]
labels_thistrain = labels_train[thistrain]
X_valid = np.delete(X_train, thistrain, 0)
labels_valid = np.delete(labels_train, thistrain, 0)
one_hot_thistrain = np.zeros((labels_thistrain.size, labels_thistrain.max() + 1))
one_hot_thistrain[np.arange(labels_thistrain.size), labels_thistrain] = 1

dim = X_thistrain.shape[1]
trainerror = np.zeros(12)
validerror = np.zeros(12)
for p in range(500, 6001, 500):
    G = np.random.normal(0, np.sqrt(0.1), p*dim).reshape(p, dim)
    b = np.random.uniform(0, 2*np.pi, p).reshape(p, 1)
    What = train(np.cos(G.dot(X_thistrain.T)+b).T, one_hot_thistrain, 10e-4)
    labels_thistrain_pred = predict(What, np.cos(G.dot(X_thistrain.T)+b).T)
    labels_valid_pred = predict(What, np.cos(G.dot(X_valid.T)+b).T)
    trainerror[int(p/500)-1] = np.array([labels_thistrain[j] != labels_thistrain_pred[j] for
    validerror[int(p/500)-1] = np.array([labels_valid[j] != labels_valid_pred[j] for j in ra

plt.plot(np.arange(500, 6001, 500), trainerror, label = 'trainerror')
plt.plot(np.arange(500, 6001, 500), validerror, label = 'validation_error')
plt.xlabel('p')
plt.ylabel('error')
plt.show()

#Following is B2(b) and we take p=6000 here

G = np.random.normal(0, np.sqrt(0.1), 6000*dim).reshape(6000, dim)
```

```
b = np.random.uniform(0, 2*np.pi, 6000).reshape(6000, 1)
What = train(np.cos(G.dot(X_thistrain.T)+b).T, one_hot_thistrain, 10e-4)
labels_test_pred = predict(What, np.cos(G.dot(X_thistrain.T)+b).T)
testerror = np.array([labels_test[j] != labels_test_pred[j] for j in range(labels_test.size)
CI = [testerror-np.sqrt(np.log(2/0.05)/2/len(labels_test)), testerror+np.sqrt(np.log(2/0.05)

print('Test error is ', testerror)
print('Confidence Interval is ', CI)
```