HW1_ZheyuanHu

February 10, 2021

1 Homework 1

Name: Zheyuan Hu

NetID: zh2095

1.1 Part I General Consideration

1.1.1 Q1

$f^*(x) = a_0 + a_1x + a_2x^2$ is an obvious Bayes predictor.

Here we know that the joint distribution $P_{\mathcal{X}\times\mathcal{Y}}$ is $y = g(x) = a_0 + a_1x + a_2x^2$, so when $f^*(x) = g(x)$, the risk $R(f^*) = E[\ell(f(x), y)]$ would reach its minimum 0.

1.1.2 Q2

$f^*_{\mathcal{H}_2} = a_0 + a_1x + a_2x^2$ is a risk minimizer in $\mathcal{H}_2$.

The approximation error is computed by $R(f^*_{\mathcal{H}_2}) - R(f^*)$, and here $R(f^*_{\mathcal{H}_2}) = 0$, which is the same as $R(f^*)$, so the approximation error is 0.

1.1.3 Q3

$R(f^*_{\mathcal{H}_2}) \geq R(f^*_{\mathcal{H}_d})$.

The hypothesis space $\mathcal{H}_2$ is a subspace of $\mathcal{H}_d$ with $d > 2$, that is, the risk minimizer in $\mathcal{H}_2$ is always contained in $\mathcal{H}_d$ with $d > 2$, so the minimum risk of $\mathcal{H}_2$ cannot be less than the minimum risk of $\mathcal{H}_d$.

The risk minimizer in $\mathcal{H}_d$ is $f^*_{\mathcal{H}_d} = a_0 + a_1x + a_2x^2$, and the approximation error achieved by $f^*_{\mathcal{H}_d}$ is also 0 since $R(f^*_{\mathcal{H}_d}) = R(f^*)$.

1.1.4 Q4

$f^*_{\mathcal{H}}$ is the function in $\mathcal{H}$ that minimizes the risk $R(f_{\mathcal{H}}) = E[\ell(f_{\mathcal{H}}(x), y)] = E[\frac{1}{2}(b_1x - a_1x - a_2x^2)^2]$. That is, we need to find b_1 that minimizes $E[\frac{1}{2}(b_1x - a_1x - a_2x^2)^2]$. By calculation, $E[\frac{1}{2}(b_1x - a_1x - a_2x^2)^2] = \frac{1}{2}(\frac{1}{3}b_1^2 + \frac{1}{3}a_1^2 + \frac{1}{5}a_2^2 - \frac{2}{3}a_1b_1 - \frac{1}{2}b_1a_2 + \frac{1}{2}a_1a_2)$. The expected loss reaches its minimum when $b_1 = a_1 + \frac{3}{4}a_2$. Thus, $f^*_{\mathcal{H}}(x) = (a_1 + \frac{3}{4}a_2)x$ is the risk minimizer in $\mathcal{H}$.

Plug in $b_1 = a_1 + \frac{3}{4}a_2$ to the previous calculation, the minimum risk $R(f^*_{\mathcal{H}}) = \frac{1}{160}a_2^2$. Therefore, the approximation error achieved by $f^*_{\mathcal{H}}$ is $R(f^*_{\mathcal{H}}) - R(f^*) = \frac{1}{160}a_2^2$.

If furthermore $a_2 = 0$, then $f^*_{\mathcal{H}}(x) = a_1 x$. Therefore, $R(f^*_{\mathcal{H}})$ is 0 and the approximation error would be 0.

1.2 Part II Polynomial Regression as Linear Least Squares

1.2.1 Q5

The empirical risk is given by

$$\hat{R}_N(f) = \frac{1}{N}\sum_{i=1}^{N} \ell(f_{\mathbf{b}}(x_i), y_i)$$

The ERM $\hat{\mathbf{b}}$ minimizes the empirical risk $\hat{R}_N(f)$,

that is,

$$\hat{\mathbf{b}} = \arg\min_{\mathbf{b}} \frac{1}{N}\sum_{i=1}^{N} \ell(f_{\mathbf{b}}(x_i), y_i) = \arg\min_{\mathbf{b}} \sum_{i=1}^{N} \ell(f_{\mathbf{b}}(x_i), y_i) = \arg\min_{\mathbf{b}} \sum_{i=1}^{N} \frac{1}{2}(f_{\mathbf{b}}(x_i) - y_i)^2 = \arg\min_{\mathbf{b}} \sum_{i=1}^{N} (f_{\mathbf{b}}(x_i) - y_i)^2$$

$$\sum_{i=1}^{N} (f_{\mathbf{b}}(x_i) - y_i)^2 = \sum_{i=1}^{N} (b_0 + b_1 x_i + \cdots + b_N x_i^N - y_i)^2$$

We have

$$X = \begin{bmatrix} 1 & x_1 & \cdots & x_1^d \\ 1 & x_2 & \cdots & x_2^d \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & \cdots & x_N \end{bmatrix}, \quad \mathbf{y} = [y_0, y_1, \cdots y_N]^\top. \tag{1}$$

notice that

$$\sum_{i=1}^{N} (b_0 + b_1 x_i + \cdots + b_N x_i^N - y_i)^2 = \|X\mathbf{b} - \mathbf{y}\|_2^2$$

Therefore,

$$\hat{\mathbf{b}} = \arg\min_{\mathbf{b}} \frac{1}{N}\sum_{i=1}^{N} \ell(f_{\mathbf{b}}(x_i), y_i) = \arg\min_{\mathbf{b}} \|X\mathbf{b} - \mathbf{y}\|_2^2$$

1.2.2 Q6

Define

$$J(\mathbf{b}) = \|X\mathbf{b} - \mathbf{y}\|_2^2$$

then

$$\hat{\mathbf{b}} = \arg\min_{\mathbf{b}} \|X\mathbf{b} - \mathbf{y}\|_2^2 = \arg\min_{\mathbf{b}} J(\mathbf{b})$$

Want to show that

$$\hat{\mathbf{b}} = (X^\top X)^{-1} X^\top \mathbf{y} \text{ minimizes } J(\mathbf{b})$$

calculate the gradient of $J(\mathbf{b})$ and its 2nd order:

$$\nabla J(\mathbf{b}) = X^\top X\mathbf{b} - X^\top \mathbf{y}$$

$$\nabla^2 J(\mathbf{b}) = X^\top X$$

then $\hat{\mathbf{b}}$ must satisfy two conditions to be the global minimum:

$$\hat{\mathbf{b}} \text{ is a solution to } \nabla J(\mathbf{b}) = 0 \text{ , and } \nabla^2 J(\mathbf{b}) > 0$$

$N > d$ means that the # of equations > # of unknown variablesso $ $\nabla J(\mathbf{b}) = 0$$existsasolution, and$

$$\hat{\mathbf{b}} = (X^\top X)^{-1} X^\top \mathbf{y} \text{ is a solution to } \nabla J(\mathbf{b}) = 0$$

X is full rank implies that $X^\top X$ is invertible and positive definite, that is,

$$\nabla^2 J(\mathbf{b}) = X^\top X > 0$$

Therefore, $\hat{\mathbf{b}} = (X^\top X)^{-1} X^\top \mathbf{y}$ is the ERM.

1.3 Part III Hands on

[33]:
```
import numpy as np
import matplotlib.pyplot as plt

def get_a(deg_true):
    """
    Inputs:
    deg_true: (int) degree of the polynomial g

    Returns:
    a: (np array of size (deg_true + 1)) coefficients of polynomial g
    """
    return 5 * np.random.randn(deg_true + 1)

def get_design_mat(x, deg):
    """
    Inputs:
    x: (np.array of size N)
    deg: (int) max degree used to generate the design matrix

    Returns:
    X: (np.array of size N x (deg_true + 1)) design matrix
    """
    X = np.array([x ** i for i in range(deg + 1)]).T
    return X

def draw_sample(deg_true, a, N):
```

```
    """
    Inputs:
    deg_true: (int) degree of the polynomial g
    a: (np.array of size deg_true) parameter of g
    N: (int) size of sample to draw

    Returns:
    x: (np.array of size N)
    y: (np.array of size N)
    """
    x = np.sort(np.random.rand(N))
    X = get_design_mat(x, deg_true)
    y = X @ a
    return x, y

def draw_sample_with_noise(deg_true, a, N):
    """
    Inputs:
    deg_true: (int) degree of the polynomial g
    a: (np.array of size deg_true) parameter of g
    N: (int) size of sample to draw

    Returns:
    x: (np.array of size N)
    y: (np.array of size N)
    """
    x = np.sort(np.random.rand(N))
    X = get_design_mat(x, deg_true)
    y = X @ a + np.random.randn(N)
    return x, y
```

1.3.1 Q7

[34]:
```
def least_square_estimator(X, y):
    N = X.shape[0]
    d = X.shape[1]-1

    if N <= d:
        print('there is no solution')
    else:
        b = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)

    return b
```

1.3.2 Q8

The empirical risk is given by

$$\hat{R}_N(f) = \frac{1}{N}\sum_{i=1}^{N}\ell(f_{\mathbf{b}}(x_i), y_i) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}(f_{\mathbf{b}}(x_i) - y_i)^2 = \frac{1}{2N}\sum_{i=1}^{N}(b_0 + b_1 x_i + \cdots + b_N x_i^N - y_i)^2 = \frac{1}{2N}\|X\mathbf{b} - \mathbf{y}\|_2^2$$

[166]:
```
def empirical_risk(X, y, b):
    N = X.shape[0]
    norm = np.linalg.norm(X.dot(b) - y)
    R = norm**2/(2*N)
    return R
```

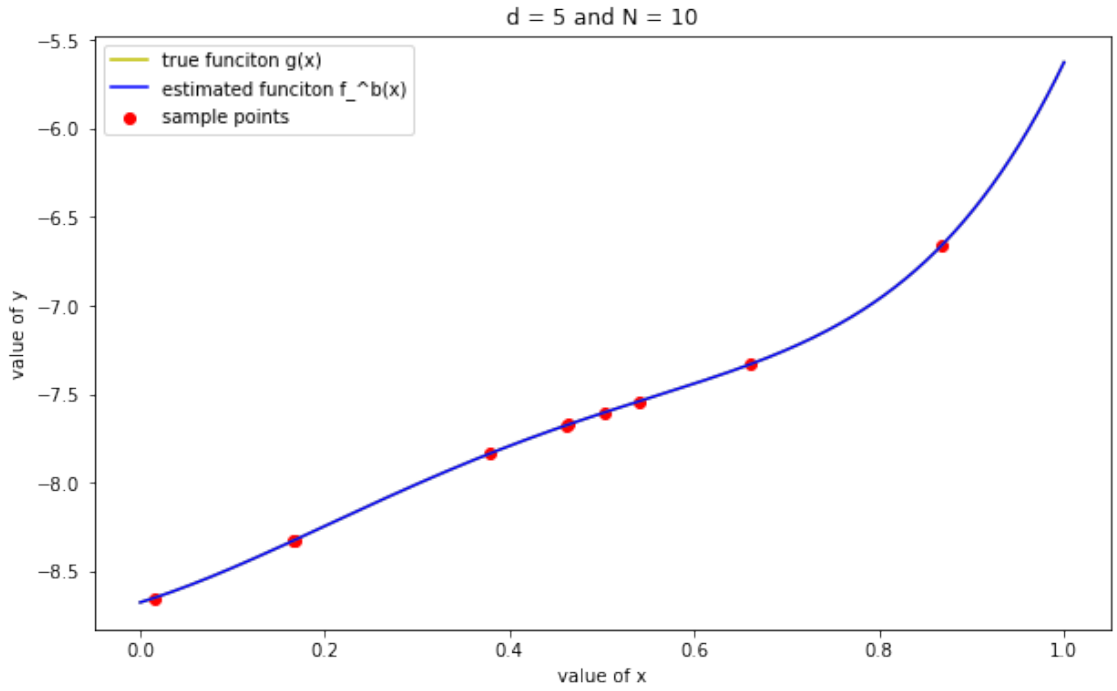
1.3.3 Q9

[234]:
```
# draw training sample with d=5
d = 5
N = 10
a = get_a(d)
x_train, y_train = draw_sample(d, a, N)
```

[235]:
```
# compare a with b
X = get_design_mat(x_train, d)
b = least_square_estimator(X, y_train)

print('The actual coefficients vector a is ', a)
print('The predicted coefficients vector b is ', b)
```

```
The actual coefficients vector a is  [-8.67824082  1.60874666  4.27978095
-8.08384795  1.3345093   3.91413356]
The predicted coefficients vector b is  [-8.67824082  1.60874667  4.27978094
-8.08384795  1.33450931  3.91413355]
```

[236]:
```
x, y = draw_sample(d, a, 10000)
X_axis = get_design_mat(x, d)

# plot training set points, the function g(x) and the estimated function f_b_hat)
plt.figure(figsize=(10,6))
plt.xlim([-0.05,1.05])
plt.xlabel('value of x')
plt.ylabel('value of y')
plt.title('d = %d and N = %d'%(d,N))

plt.scatter(x_train, y_train, c='r', label = 'sample points')
plt.plot(x, y, 'y', label = 'true funciton g(x)')
plt.plot(x, X_axis.dot(b), 'b', label = 'estimated funciton f_^b(x)')
plt.legend()
plt.show()
```



Seems that the true function and the estimated function are almost the same.

1.3.4 Q10

We can test d from small to large to find the minimum satisfied value.

[238]:
```
for d in range(7):
    X_train = get_design_mat(x_train, d)
    b = least_square_estimator(X_train, y_train)
    R = empirical_risk(X_train, y_train, b)
    print('when d = ' + str(d) + ', the minimum empirical risk is ' + str(R))
```

```
when d = 0, the minimum empirical risk is 0.14587116277126952
when d = 1, the minimum empirical risk is 0.0012216991501511878
when d = 2, the minimum empirical risk is 0.000831097960346169
when d = 3, the minimum empirical risk is 0.00041456256389581816
when d = 4, the minimum empirical risk is 7.831478775377083e-07
when d = 5, the minimum empirical risk is 5.093540171727528e-19
when d = 6, the minimum empirical risk is 1.5296192767148228e-14
```

Say that we consider the risk less than 1e-5 as a 'perfect fit', then the minimum value for which we get a perfect fit is d=4.

It is related to the conclusion in Q3 that $R(f^*_{\mathcal{H}_2}) \geq R(f^*)$, that is, the approximation error is always greater than or equal to 0.

1.4 Part IV In presence of noise

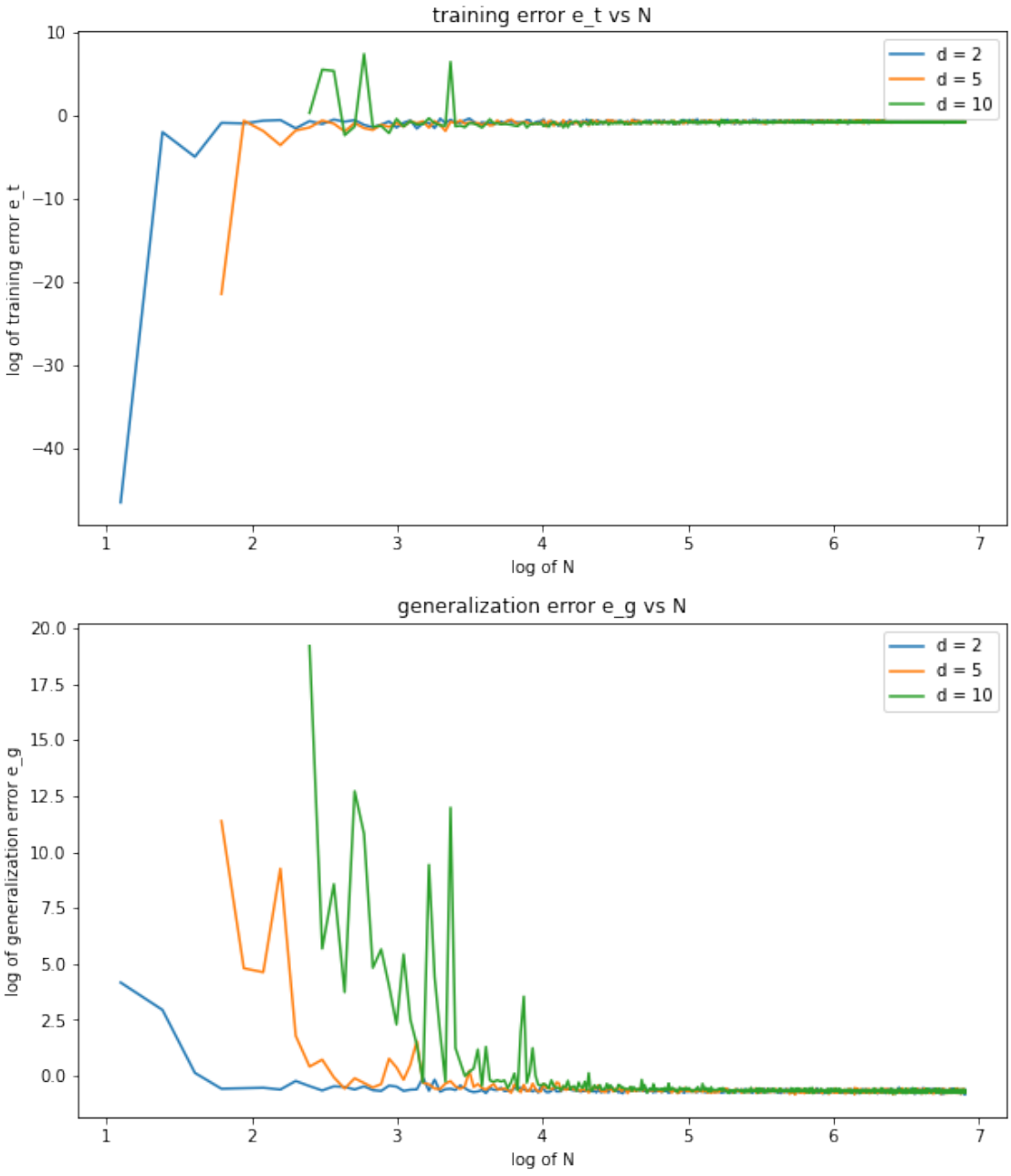
1.4.1 Q11

Plot 2

[182]:
```
N_test = 1000
d_set = [2, 5, 10]

fig, (ax1,ax2) = plt.subplots(2,figsize=(10,12))
fig.suptitle('Plot 2')
for d in d_set:
    R_train_set = []
    R_test_set = []
    N_set = np.arange(d+1,1000)
    logN_set = np.log(N_set)

    for N in N_set:
        a = get_a(d)
        x_train,y_train = draw_sample_with_noise(d, a, N)
        X_train = get_design_mat(x_train, d)
        b = least_square_estimator(X_train, y_train)
        # compute the training error e_t
        R_train = empirical_risk(X_train, y_train, b)
        R_train_set.append(R_train)
        logR_train_set = np.log(R_train_set)

        # compute the generalization error e_g
        x_test,y_test = draw_sample_with_noise(d, a, N_test)
        X_test = get_design_mat(x_test, d)
        R_test = empirical_risk(X_test, y_test, b)
        R_test_set.append(R_test)
        logR_test_set = np.log(R_test_set)

    # plot e_t as a function of N
    ax1.set_xlabel('log of N')
    ax1.set_ylabel('log of training error e_t')
    ax1.set_title('training error e_t vs N')
    ax1.plot(logN_set, logR_train_set, label = 'd = %d'%d)
    ax1.legend()

    # plot e_g as a function of N
    ax2.set_xlabel('log of N')
    ax2.set_ylabel('log of generalization error e_g')
    ax2.set_title('generalization error e_g vs N')

    ax2.plot(logN_set, logR_test_set, label = 'd = %d'%d)
    ax2.legend()
```

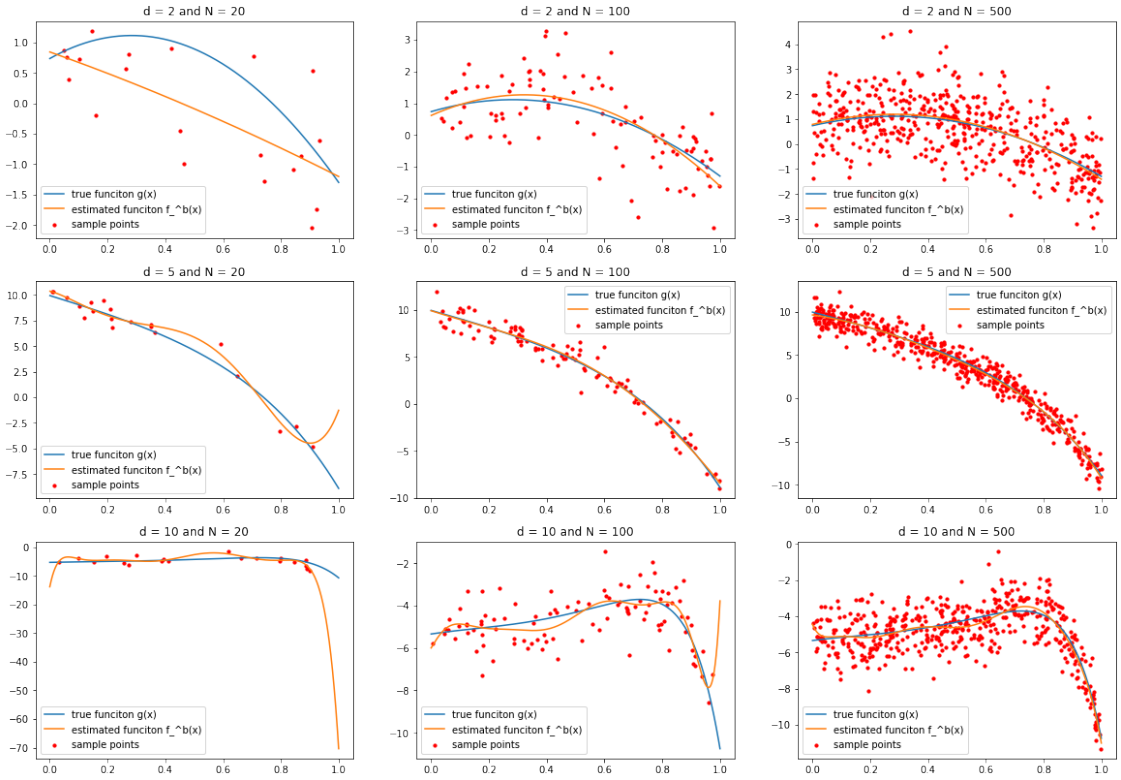
```
plt.show()
```

Plot 2



Plots similar to Plot 1

[133]:
```
# here we test with N = 20, 100, 500
N_test = 1000
d_set = [2, 5, 10]
N_set = [20, 100, 500]
fig, axs = plt.subplots(3, 3, figsize=(20,14))
i = 0
for d in d_set:
    a = get_a(d)
    j = 0
    for N in N_set:
        x_train, y_train = draw_sample_with_noise(d, a, N)
        X_train = get_design_mat(x_train, d)
        b = least_square_estimator(X_train, y_train)
        # use 10000 pts to plot for a continuous line
        x, y_null = draw_sample_with_noise(d, a, 10000)
        X = get_design_mat(x, d)
        y = X @ a

        axs[i,j].set_xlim([-0.05,1.05])
        axs[i,j].set_title('d = %d and N = %d'%(d,N))
        axs[i,j].scatter(x_train, y_train, s = 10, c= 'r', label = 'sample␣
 ↪points')
        axs[i,j].plot(x, y, label = 'true funciton g(x)')
        axs[i,j].plot(x, X.dot(b), label = 'estimated funciton f_^b(x)')
        axs[i,j].legend()

        j += 1
    i += 1
plt.show()
```



1.4.2 Q12

The estimation error is given by

$$R(\hat{f}_N) - R(f_{\mathcal{F}})$$

$R(\hat{f}_N)$ is the empirical risk computed on the test set using predictor (b) learnt from training set

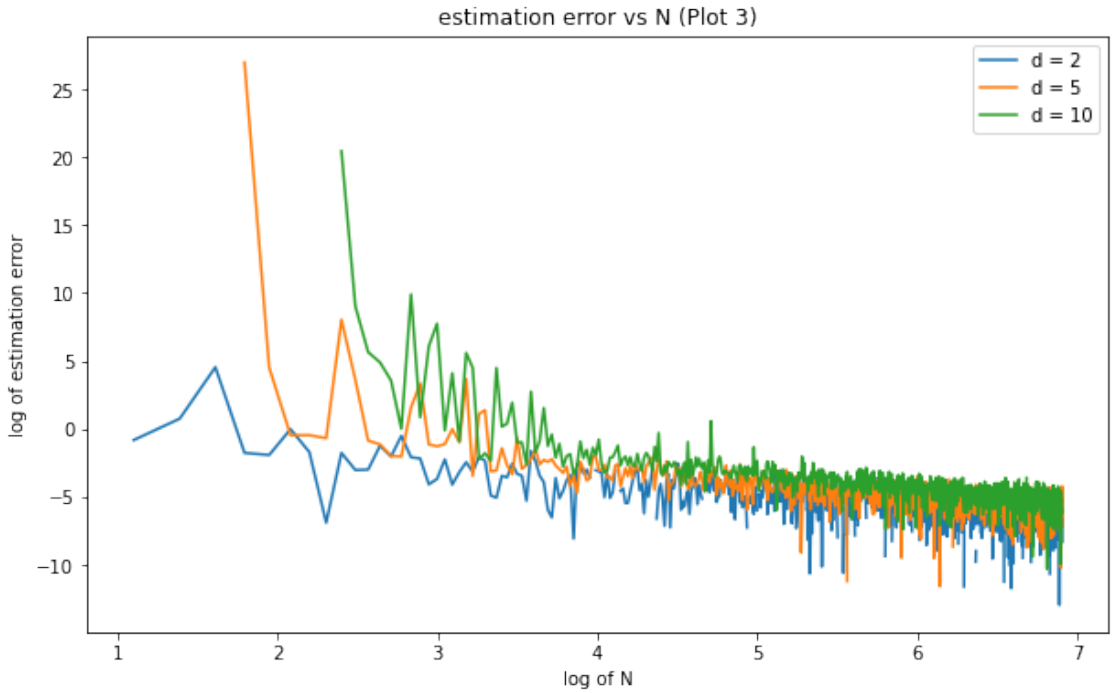
$R(f_{\mathcal{F}})$ is the risk of $y = g(x)$ since $g(x)$ is the best prediction function we can get from the hypothesis space, and here we can consider the empirical risk computed on the test set using the true coefficients (a) to be $R(f_{\mathcal{F}})$ since the size of the testset is large enough

[ ]:

[181]:
```
N_test = 1000
d_set = [2, 5, 10]

fig, ax = plt.subplots(figsize=(10,6))
for d in d_set:
    estierr_set = []
    N_set = np.arange(d+1,1000)
    logN_set = np.log(N_set)

    for N in N_set:
```

```
        a = get_a(d)
        x_train,y_train = draw_sample_with_noise(d, a, N)
        X_train = get_design_mat(x_train, d)
        b = least_square_estimator(X_train, y_train)

        x_test,y_test = draw_sample_with_noise(d, a, N_test)
        X_test = get_design_mat(x_test, d)
        R_fhat = empirical_risk(X_test, y_test, b)
        R_f = empirical_risk(X_test, y_test, a)
        estierr = R_fhat - R_f
        estierr_set.append(estierr)
        log_estierr_set = np.log(estierr_set)
    ax.set_xlabel('log of N')
    ax.set_ylabel('log of estimation error')
    ax.set_title('estimation error vs N (Plot 3)')
    ax.plot(logN_set, log_estierr_set, label = 'd = %d'%d)
    ax.legend()

plt.show()
```

```
<ipython-input-181-955a306df513>:22: RuntimeWarning: invalid value encountered
in log
  log_estierr_set = np.log(estierr_set)
<ipython-input-181-955a306df513>:22: RuntimeWarning: invalid value encountered
in log
  log_estierr_set = np.log(estierr_set)
<ipython-input-181-955a306df513>:22: RuntimeWarning: invalid value encountered
in log
  log_estierr_set = np.log(estierr_set)
```



1.4.3 Q13

The increasing of d will increase both of the generalization error and the estimation error, while the increasing of N will decrease the generalization error and the estimation error.

1.4.4 Q14

There is no optimization error in the algorithm we are implementing, because our algorithm computes the ERM directly by matrix multiplication ($\hat{\mathbf{b}} = (X^\top X)^{-1} X^\top \mathbf{y}$) without any optimization process.

[ ]: