DS-GA 1003 - Homework 1

Eric Niblock

February 3rd, 2021

For the first part of this assignment we will consider a synthetic prediction problem to develop our intuition about the error decomposition. Consider the random variables $x \in \mathcal{X} = [0, 1]$ distributed uniformly ($x \sim \text{Unif}([0, 1])$) and $y \in \mathcal{Y} = \mathbb{R}$ defined as a polynomial of degree 2 of x: there exists $(a_0, a_1, a_2) \in \mathbb{R}^3$ such that the values of x and y are linked as $y = g(x) = a_0 + a_1 x + a_2 x^2$. Note that this relation fixes the joint distribution $P_{\mathcal{X}\times\mathcal{Y}}$.

From the knowledge of a sample $\{x_i, y_i\}_{i=1}^N$, we would like to predict the relation between x and y, that is find a function f to make predictions $\hat{y} = f(x)$. We note $\mathcal{H}_d$, the set of polynomial functions on $\mathbb{R}$ of degree d:

$$\mathcal{H}_d = \left\{ f : x \to b_0 + b_1 x + \cdots + b_d x^d ; b_k \in \mathbb{R} \forall k \in \{0, \cdots d\} \right\}$$

We will consider the hypothesis classes $\mathcal{H}_d$ varying d. We will minimize the squared loss $\ell(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$ to solve the regression problem.

1. **Recall the definition of the expected risk $R(f)$ of a predictor f. While this cannot be computed in general note that here we defined $P_{\mathcal{X}\times\mathcal{Y}}$. Which function f^* is an obvious Bayes predictor? Make sure to explain why the risk $R(f^*)$ is minimum at f^*.**

   Here the obvious Bayes predictor is simply $f^* = g(x) = a_0 + a_1 x + a_2 x^2$. This is because $R(f^*)$ is minimized, as shown,

$$R(f^*) = E[\ell(f^*(x), g(x))] = E[\ell(g(x), g(x))] = E\left[\frac{1}{2}(g(x) - g(x))\right] = 0 \tag{1}$$

2. **Using $\mathcal{H}_2$ as your hypothesis class, which function $f^*_{\mathcal{H}_2}$ is a risk minimizer in $\mathcal{H}_2$? Recall the definition of the approximation error. What is the approximation error achieved by $f^*_{\mathcal{H}_2}$?**

   It is clear that $f^*_{\mathcal{H}_2}(x) = a_0 + a_1 x + a_2 x^2$ is a risk minimizer in $\mathcal{H}_2$. The approximation error is then given by,

$$
\begin{aligned}
R(f^*_{\mathcal{H}_2}) - R(f^*) = R(f^*_{\mathcal{H}_2}) &= E[\ell(f^*_{\mathcal{H}_2}(x), g(x))] = E\left[\frac{1}{2} f^*_{\mathcal{H}_2}(x) - g(x))\right] \\
&= E\left[\frac{1}{2}\left((b_0 - a_0) + (b_1 - a_1)x + (b_2 - a_2)x^2\right)\right] \\
&= \frac{1}{2}(b_0 - a_0) + (b_1 - a_1)E[x] + (b_2 - a_2)E[x^2] \\
&= \frac{1}{2}(b_0 - a_0) + \frac{1}{2}(b_1 - a_1) + \frac{1}{4}(b_2 - a_2)
\end{aligned} \tag{2}
$$

We are free to choose $b_i = a_i$. Doing so minimizes the risk, and yields,

$$
R(f^*_{\mathcal{H}_2}) - R(f^*) = R(f^*_{\mathcal{H}_2}) = 0 \tag{3}
$$

3. Considering now $\mathcal{H}_d$, with $d > 2$. Justify an inequality between $R(f^*_{\mathcal{H}_2})$ and $R(f^*_{\mathcal{H}_d})$. Which function $f^*_{\mathcal{H}_d}$ is a risk minimizer in $\mathcal{H}_d$? What is the approximation error achieved by $f^*_{\mathcal{H}_d}$?

From the above result, we have the value for $R(f^*_{\mathcal{H}_2}) = 0$. For $R(f^*_{\mathcal{H}_d})$ for all d, we can simply make $f^*_{\mathcal{H}_d} = f^*_{\mathcal{H}_2}$ by setting all $b_i = 0$ for $i \in \{3, ..., d\}$. This function is therefore $f^*_{\mathcal{H}_d}$, the risk minimizer of $\mathcal{H}_d$. Then it is obvious that,

$$
R(f^*_{\mathcal{H}_2}) = R(f^*_{\mathcal{H}_d}) \tag{4}
$$

Our approximation error of $f^*_{\mathcal{H}_d}$ is therefore given by,

$$
R(f^*_{\mathcal{H}_d}) - R(f^*) = 0 \tag{5}
$$

4. For this question we assume $a_0 = 0$. Considering $\mathcal{H} = \{f : x \to b_1 x; b_1 \in \mathbb{R}\}$, which function $f^*_{\mathcal{H}}$ is a risk minimizer in $\mathcal{H}$? What is the approximation error achieved by $f^*_{\mathcal{H}}$? In particular what is the approximation error achieved if furthermore $a_2 = 0$ in the definition of true underlying relation $g(x)$ above?

We note that $f^*_{\mathcal{H}}$ is such that,

$$f^*_{\mathcal{H}} \in \arg\min_{f \in \mathcal{H}} E[\ell(f(x), y)] \tag{6}$$

So, we begin by calculating the expectation,

$$\begin{aligned} E[\ell(f(x), y)] &= E[\frac{1}{2}(a_1 x + a_2 x^2 - b_1 x)^2] \\ &= \frac{1}{2}(a_1 - b_1)^2 E[x^2] + a_2(a_1 - b_1)E[x^3] + \frac{1}{2}a_2^2 E[x^4] \\ &= \frac{1}{8}(a_1 - b_1)^2 + \frac{1}{8}a_2(a_1 - b_1) + \frac{1}{32}a_2^2 \end{aligned} \tag{7}$$

Now we can minimize this function with respect to b_1 by taking a derivative,

$$\frac{\partial E[\ell(f(x), y)]}{\partial b_1} = b_1 - a_1 - \frac{a_2}{2} = 0 \tag{8}$$

$$b_1 = a_1 + \frac{a_2}{2} \tag{9}$$

So, we have the risk minimizer as being,

$$f^*_{\mathcal{H}}(x) = (a_1 + \frac{a_2}{2})x \tag{10}$$

In practice, $P_{\mathcal{X}\times\mathcal{Y}}$ is usually unknown and we use the empirical risk minimizer (ERM). We will reformulate the problem as a d-dimensional linear regression problem. First note that functions in $\mathcal{H}_d$ are parametrized by a vector $\boldsymbol{b} = [b_0, b_1, \cdots b_d]^\top$, we will use the notation $f_{\boldsymbol{b}}$. Similarly we will note $\boldsymbol{a} \in \mathbb{R}^3$ the vector parametrizing $g(x) = f_{\boldsymbol{a}}(x)$. We will also gather data points from the training sample in the following matrix and vector:

$$X = \begin{bmatrix} 1 & x_1 & \cdots & x_1^d \\ 1 & x_2 & \cdots & x_2^d \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & \cdots & x_N \end{bmatrix}, \quad \boldsymbol{y} = [y_0, y_1, \cdots, y_N]^\top. \tag{11}$$

These notations allow us to take advantage of the very effective linear algebra formalism. X is called the design matrix.

5. Show that the empirical risk minimizer (ERM) $\hat{b}$ is given by the following minimization $\hat{\boldsymbol{b}} = \arg\min_b \|Xb - \boldsymbol{y}\|_2^2$.

From the definition of an empirical risk minimizer, $\hat{f}_n$, we have that,

$$\hat{f}_n \in \arg\min_{f\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i) \implies \hat{f}_n \in \arg\min_{f\in\mathcal{F}} \sum_{i=1}^{n} (f(x_i) - y_i)^2 \tag{12}$$

Now, it is clear that we can write this sum of losses in a more compact notation,

$$\begin{bmatrix} f(x_1) - y_1 \\ \vdots \\ f(x_n) - y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 & \cdots & x_1^d \\ 1 & x_2 & \cdots & x_2^d \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & \cdots & x_N \end{bmatrix} \begin{bmatrix} b_0 \\ \vdots \\ b_d \end{bmatrix} - \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = Xb - y \tag{13}$$

Where the vector b contains all the information regarding function f. Then we see that,

$$\begin{aligned} \sum_{i=1}^{n} (f(x_i) - y_i)^2 &= (f(x_1) - y_1)^2 + ... + (f(x_n) - y_n)^2 \\ &= \left\| \begin{bmatrix} f(x_1) - y_1 \\ \vdots \\ f(x_n) - y_n \end{bmatrix} \right\| = \|Xb - \boldsymbol{y}\|_2^2 \end{aligned} \tag{14}$$

So the empirical risk minimizer (ERM) $\hat{\boldsymbol{b}}$ is given by the following minimization $\hat{\boldsymbol{b}} = \arg\min_b \|Xb - \boldsymbol{y}\|_2^2$.

6. If $N > d$ and X is full rank, show that $\hat{\boldsymbol{b}} = (X^\top X)^{-1} X^\top \boldsymbol{y}$. (Hint: you should take the gradients of the loss above with respect to $\boldsymbol{b}$). Why do we need to use the conditions $N > d$ and X full rank?

First, we take the gradient of the loss function with respect to b, yielding,

$$\frac{\partial}{\partial b}\left(\|Xb - \boldsymbol{y}\|_2^2\right) = \frac{\partial}{\partial b}\left((Xb - \boldsymbol{y})^T (Xb - \boldsymbol{y})\right) = 2X^T(Xb - y) \tag{15}$$

Setting this equal to zero provides us with $\hat{b}$,

$$2X^T(X\hat{b} - y) = 0 \tag{16}$$

$$X^T X\hat{b} = X^T y \tag{17}$$

$$\hat{b} = (X^T X)^{-1} X^T y \tag{18}$$

Which is the desired result. Obviously, X must be full rank because otherwise $X^T X$ would not be full rank, and therefore, non-invertible. This would prevent us from solving for $\hat{b}$. Furthermore, since $d + 1$ represents the number of columns and N represents the number of rows, if $N \leq d$, the matrix is wide, and there exists linear dependency. This would imply X is not full rank. As before, we would be unable to solve for $\hat{b}$.

Open the source code file *hw1_code_source.py* from the *.zip* folder. Using the function `get_a` get a value for a, and draw a sample `x_train, y_train` of size $N = 10$ and a sample `x_test, y_test` of size $N_{\text{test}} = 1000$ using the function `draw_sample`.

We used the following code to produce the requested a value and accompanying samples,

```
a = get_a(2)
x_train,y_train = draw_sample(2, a, 10)
x_test,y_test = draw_sample(2,a, 1000)
```

7. **Write a function called `least_square_estimator` taking as input a design matrix $X \in \mathbb{R}^{N\times(d+1)}$ and the corresponding vector $\boldsymbol{y} \in \mathbb{R}^N$ returning $\hat{\boldsymbol{b}} \in \mathbb{R}^{(d+1)}$. Your function should handle any value of N and d, and in particular return an error if $N \leq d$. (Drawing x at random from the uniform distribution makes it almost certain that any design matrix X with $d \geq 1$ we generate is full rank).**

Here, we have the function `least_square_estimator`:

```
def least_square_estimator(X,y):
    if X.shape[0] > X.shape[1]-1:
        b = np.linalg.inv(X.T @ X) @ X.T @ y
        return(b)
    else:
        print('Error: N <= d')
```

8. Recall the definition of the empirical risk $\hat{R}(\hat{f})$ on a sample $\{x_i, y_i\}_{i=1}^N$ for a prediction function $\hat{f}$. Write a function `empirical_risk` to compute the empirical risk of $f_{\boldsymbol{b}}$ taking as input a design matrix $X \in \mathbb{R}^{N\times(d+1)}$, a vector $\boldsymbol{y} \in \mathbb{R}^N$ and the vector $\boldsymbol{b} \in \mathbb{R}^{(d+1)}$ parametrizing the predictor.

Here, we have the function `empirical_risk`:

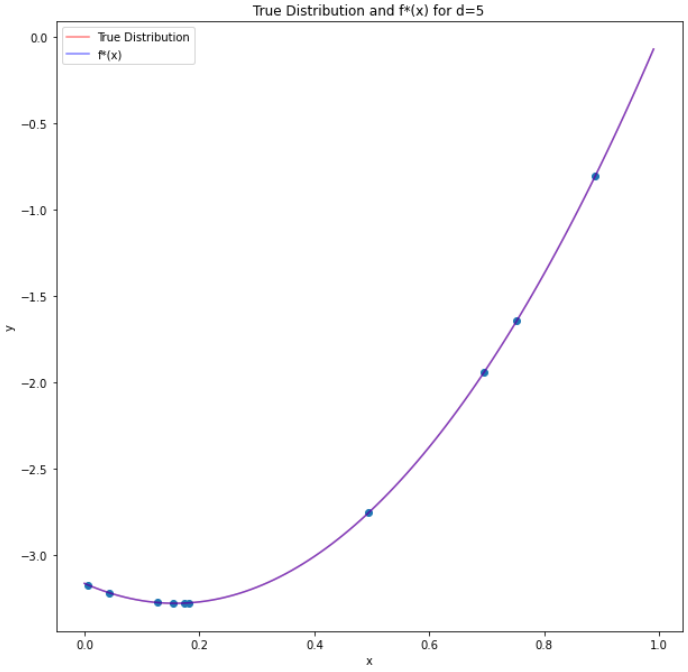
```
def empirical_risk(X,y,b):
    risk = (1/X.shape[0])*(np.linalg.norm(X@b-y))**2
    return(risk)
```

9. Use your code to estimate $\hat{\boldsymbol{b}}$ from `x_train`, `y_train` using $d = 5$. Compare $\hat{\boldsymbol{b}}$ and $\boldsymbol{a}$. Make a single plot (Plot 1) of the plane (x, y) displaying the points in the training set, values of the true underlying function $g(x)$ in $[0, 1]$ and values of the estimated function $f_{\hat{\boldsymbol{b}}}(x)$ in $[0, 1]$. Make sure to include a legend to your plot.

Using $d = 5$, we find the following values of a and $\hat{b}$,

```
d = 5
X_train = get_design_mat(x_train, d)
b = least_square_estimator(X_train,y_train)

a = [-3.1633736 , -1.46077299,  4.62770649]
b = [-3.16337360e+00, -1.46077299e+00,  4.62770649e+00,  3.73347575e-10,
   -7.12134351e-10,  3.27872840e-10]
```

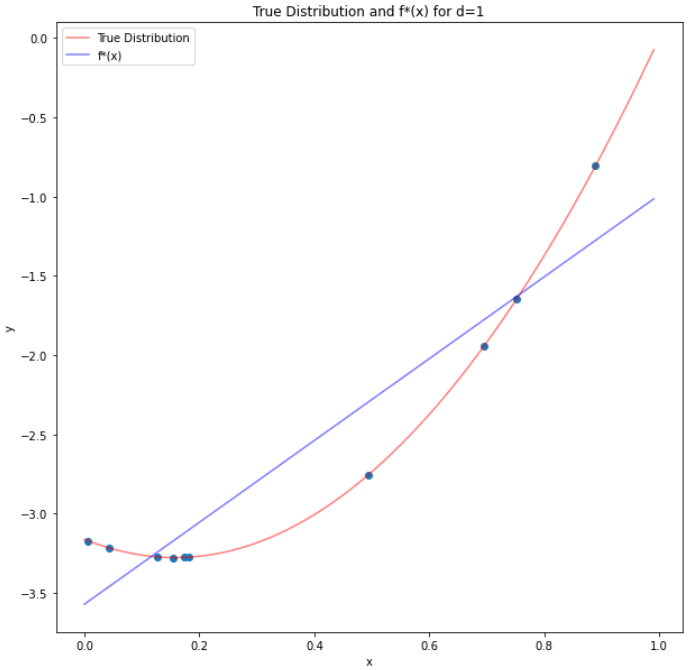
Notice that a and $\hat{b}$ are virtually identical, since the trailing values of $\hat{b}$ all equal zero. We then plot the resulting values of the data generating function, and our empirical risk minimizer, $f^*(x)$,



The resulting quadratic is purple, because $f^*(x)$ and the data generating function are indeed the same function.

10. Now you can adjust d. What is the minimum value for which we get a "perfect fit"? How does this result relates with your conclusions on the approximation error above?

We note that $d = 2$ is the minimum value for which we get a "perfect fit". Observe the plot for $d = 1$,



Now we will modify the true underlying $P_{\mathcal{X}\times\mathcal{Y}}$, adding some noise in $y = g(x) + \epsilon$, with $\epsilon \sim \mathcal{N}(0,1)$ a standard normal random variable independent from x. We will call training error e_t the empirical risk on the train set and generalization error e_g the empirical risk on the test set.

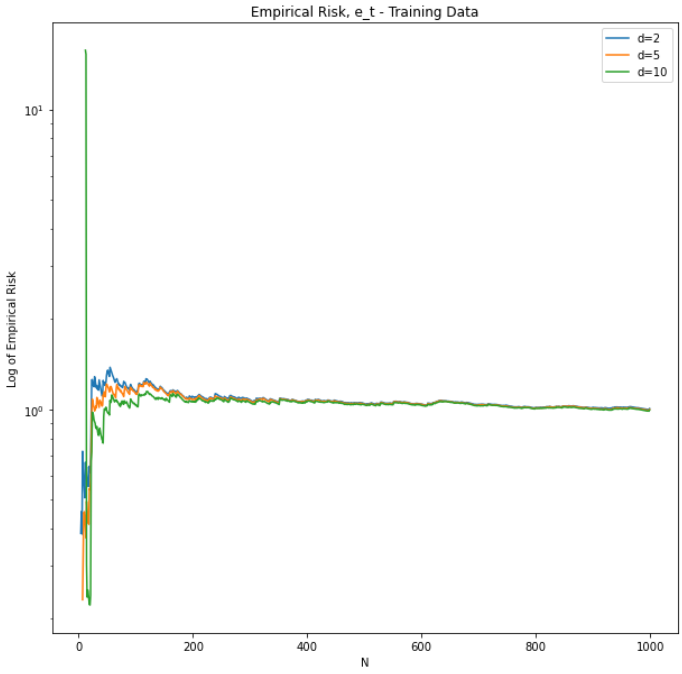
The following code was used for questions 11 and 12,

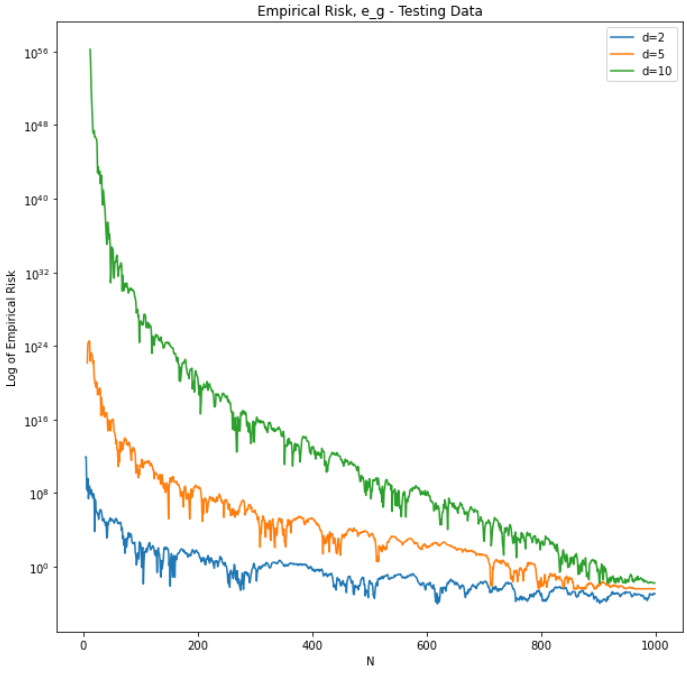
```
risks_train = []
risks_test_a = []
risks_test_b = []
estimation_err = []

x_test,y_test = draw_sample_with_noise(2, a, 1000)
x_train,y_train = draw_sample_with_noise(2, a, 1000)

for d in [2,5,10]:
    X_train = get_design_mat(x_train, d)
    X_test = get_design_mat(x_test, d)
    X_test_a = get_design_mat(x_test, 2)
    risk_test_a = []
    risk_test_b = []
    risk_train = []
    esti_err = []
    for ind in range(d+1,1000):
        b = least_square_estimator(X_train[:ind],y_train[:ind])
```

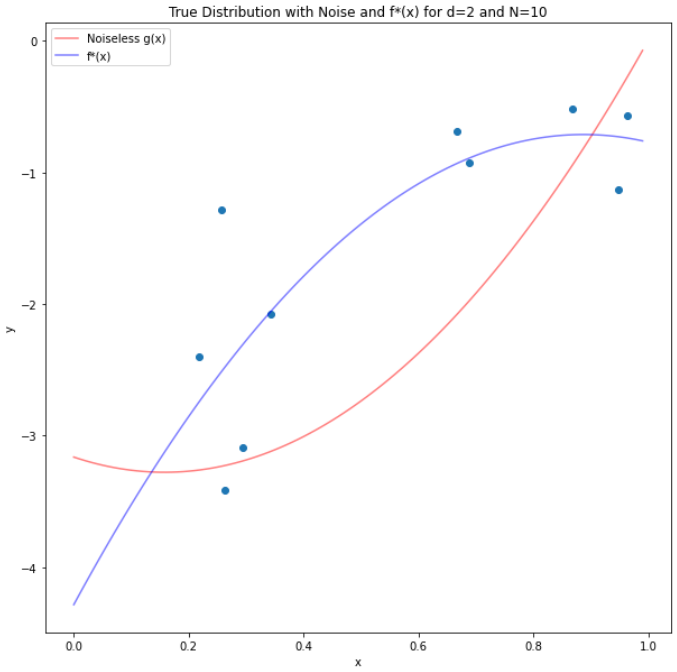
```
        r_train = empirical_risk(X_train[:ind],y_train[:ind],b)
        r_test_b = empirical_risk(X_test,y_test,b)
        r_test_a = empirical_risk(X_test_a,y_test,a)
        esti = r_test_b - r_test_a
        risk_test_a.append(r_test_a)
        risk_test_b.append(r_test_b)
        risk_train.append(r_train)
        esti_err.append(esti)
    estimation_err.append(esti_err)
    risks_train.append(risk_train)
    risks_test_a.append(risk_test_a)
    risks_test_b.append(risk_test_b)
```

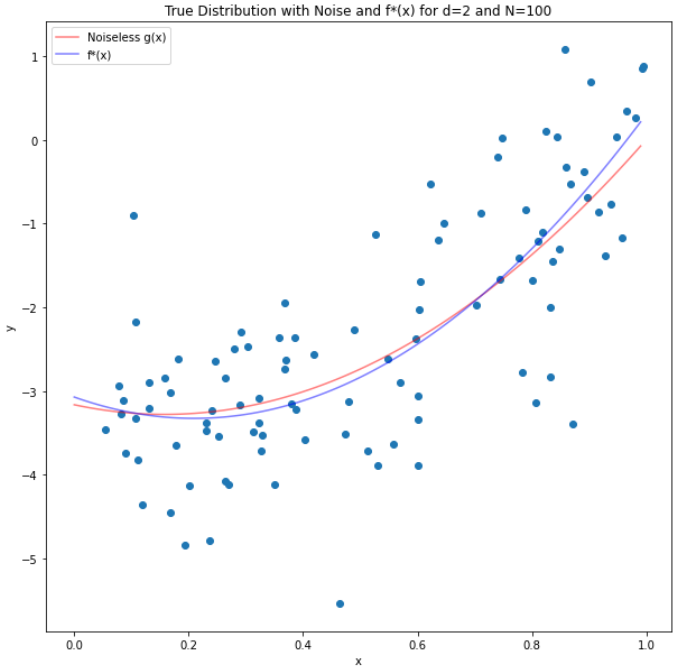
11. Plot e_t and e_g as a function of N for $d < N < 1000$ for $d = 2$, $d = 5$ and $d = 10$ (Plot 2). You may want to use a logarithmic scale in the plot. Include also plots similar to Plot 1 for 2 or 3 different values of N for each value of d.

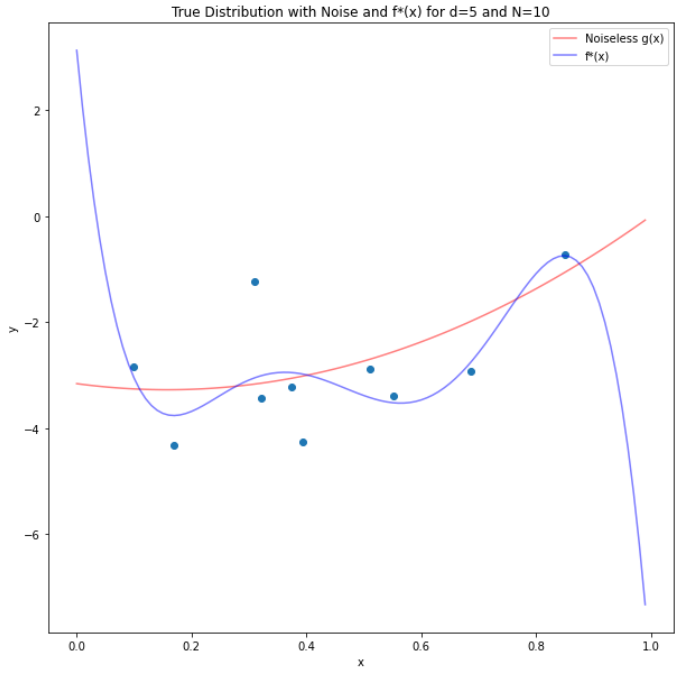
First, we plot e_t and e_g as functions of N and d. As N increases, we see that each function stabilizes to the same training error. A similar result is true of the testing data, though $d = 2$ still out performs the other degrees. Furthermore, we also plot similar plots to Plot 1.

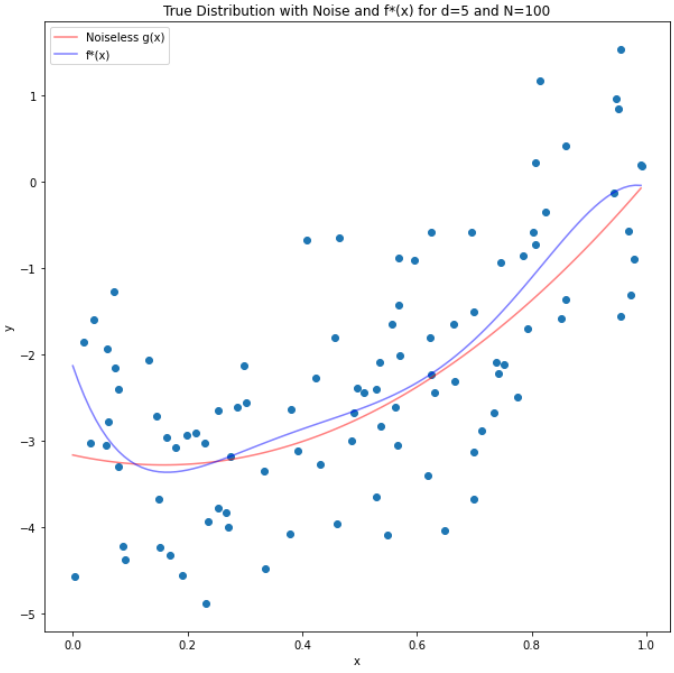


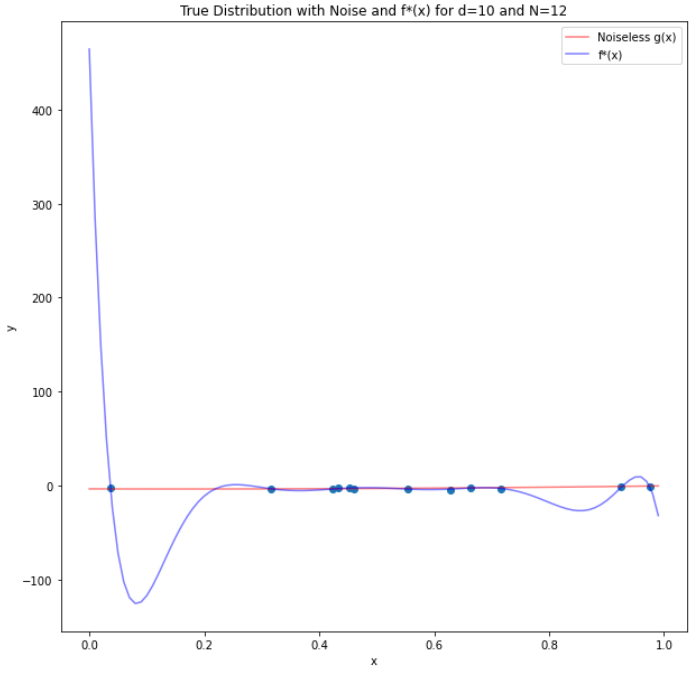


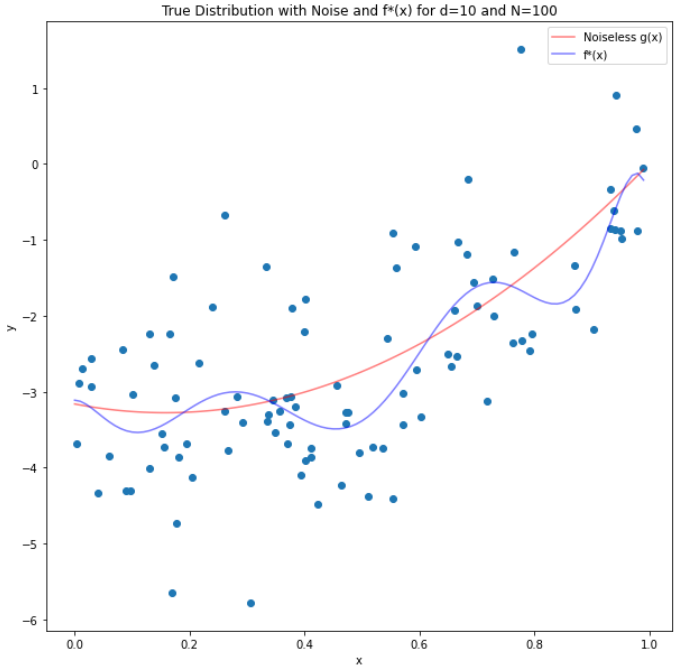

True Distribution with Noise and f*(x) for d=2 and N=10
Noiseless g(x)
f*(x)
x
y


True Distribution with Noise and f*(x) for d=2 and N=100
Noiseless g(x)
f*(x)
x
y


True Distribution with Noise and f*(x) for d=5 and N=10
Noiseless g(x)
f*(x)
x
y


True Distribution with Noise and f*(x) for d=5 and N=100
Noiseless g(x)
f*(x)
x
y





12. Recall the definition of the estimation error. Using the test set, (which we intentionally chose large so as to take advantage of the law of large numbers) give an empirical estimator of the estimation error. For the same values of N and d above plot the estimation error as a function of N (Plot 3).

We can find the estimation error by use of the following formula,

$$Estimation\,Error = R(\hat{f}_n) - R(f_{\mathcal{F}}) \tag{19}$$

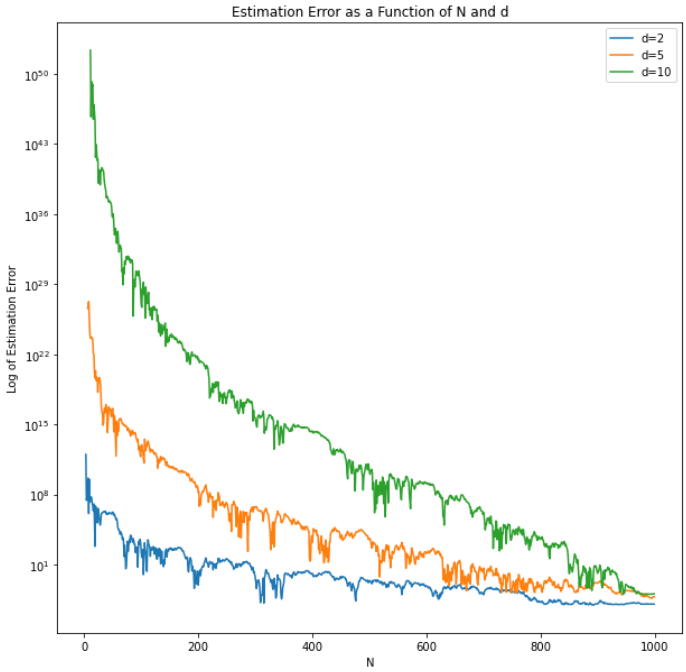
$$Estimation\,Error = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2 - E[(g(x) - (g(x) + \mathcal{N}(0,1)))^2] \tag{20}$$

$$Estimation\,Error = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2 - E[(g(x) - (g(x) + \mathcal{N}(0,1)))^2] \tag{21}$$

$$Estimation\,Error = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2 \tag{22}$$

Where $\hat{f}_n$ is the least-squares fitted function by use of n data points of the training set and $f_{\mathcal{F}}$ is $g(x)$. However, since we have a limited sample size, $R(f_{\mathcal{F}})$ is generally non-zero

and constant. We then have the following result,



We note that this result is similar in form to the previous empirical risk on the test set, e_g. This is because $f_{\mathcal{F}}$ is $g(x)$, which produces a constant $R(f_{\mathcal{F}})$. Therefore, the estimation error is simply the empirical risk on the test set, but shifted by $R(f_{\mathcal{F}})$.

13. The generalization error gives, in practice, information related to the estimation error. Comment on the results of Plot 2 and 3. What is the effect of increasing N? What is the effect of increasing d?

As mentioned, the estimation error is simply the empirical risk on the test set, but shifted by $R(f_{\mathcal{F}})$, which is constant when observed as a function of N. Increasing N, the amount of training data used, results in a lower generalization error and estimation error. Increasing d, when the data generating function is of degree two, produces greater generalization error and estimation error.

14. Besides from the approximation and estimation there is a last source of error we have not discussed here. Can you comment on the optimization error of the algorithm we are implementing?

There is no optimization error given the algorithm that we are using, because we are using the closed form solution. If we used something like gradient descent, we would then have

optimization error.

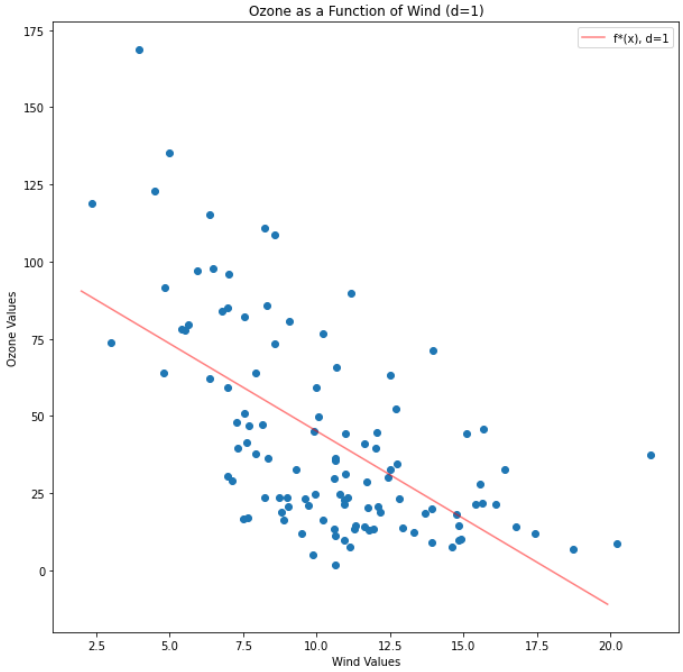
(Optional) You can now use the code we developed on the synthetic example on a real world dataset. Using the command `np.loadtxt('ozone_wind.data')` load the data in the *.zip*. The first column corresponds to ozone measurements and the second to wind measurements. You can try polynomial fits of the ozone values as a function of the wind values.

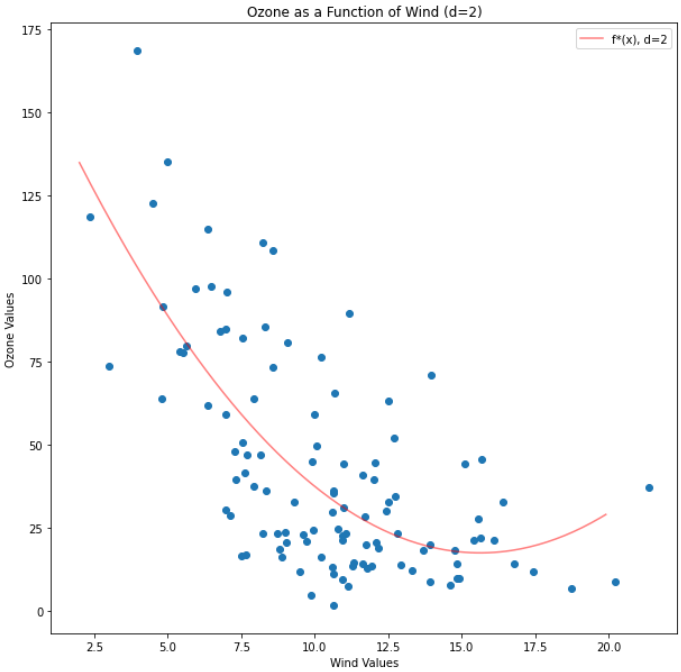
15. Reporting plots, discuss the again in this context the results when varying N (subsampling the training data) and d.

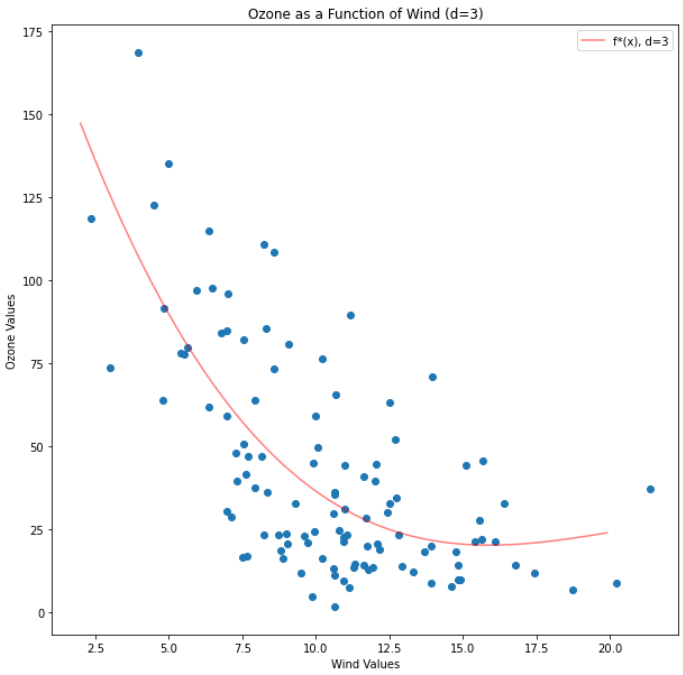
The following code was used to create fits for the data,

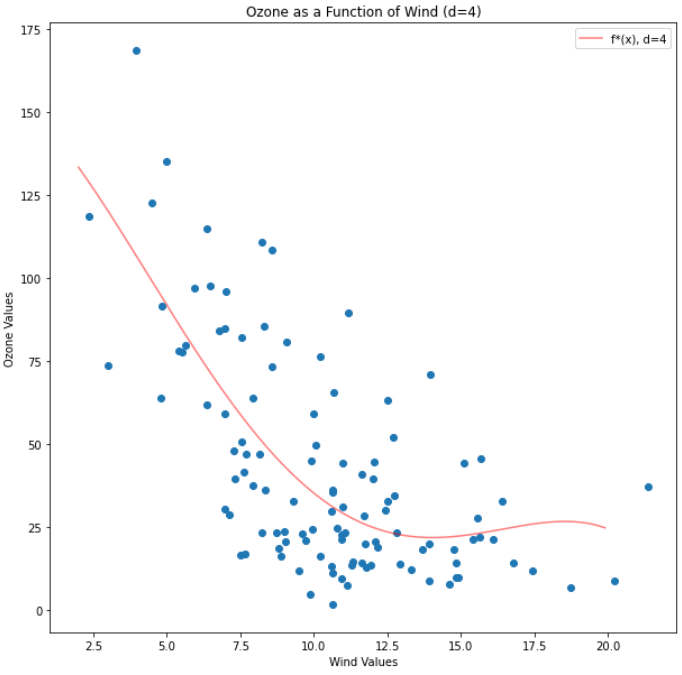
```
data = np.loadtxt('ozone_wind.data')

wind = data[:,1]
ozone = data[:,0]

degree = 4
windX = get_design_mat(wind, degree)
x_vals = np.arange(2,20,0.1)
X_vals = get_design_mat(x_vals, degree)
b = least_square_estimator(windX,ozone)
```

We see that when using the training data, $d = 1$ provides the best fit for a number of reasons. All other degree-fits seem to suggest a positive relationship between wind and ozone near the higher end wind values. Nothing provides strong evidence of such a relationship.


Ozone as a Function of Wind (d=1)
f*(x), d=1
Ozone Values
Wind Values


Ozone as a Function of Wind (d=2)
f*(x), d=2
Ozone Values
Wind Values


Ozone as a Function of Wind (d=3)
f*(x), d=3
Ozone Values
Wind Values


Ozone as a Function of Wind (d=4)
f*(x), d=4
Ozone Values
Wind Values