Homework 1 DS GA 1003

Completed by: Joby George (jg6615)

Date: 2/2/2022

This ipython notebook includes markdown content to describe the problems given the homework assignment.

Problems 1- 4

Goal of the problem:

Using a synthetically made data set with x input data and y output where x ~ U(0,1) y is a polynomial of degree two, and there exists a vector A $\in R^3$ (a_0, a_1, a_2) such that

$$y = a_0 + a_1 * x + a_2 * x^2$$

From a random sample, we would like to learn the true underlying joint distribution between x and y, limiting our hypothesis set of functions (denoted H_d)

$$H_d = \left\{ f : x \to b_0 + b_1 x + \cdots b_d x^d ; b_k \in forallk \in \{0, \cdots d\} \right\}$$

We are trying to optimize the function we select from our hypothesis space, according to the following loss function:

$\ell(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$

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^*.

Terminology

The expected risk of our function f is defined as: E $\ell(\hat{y}, y)$ (source: slide 10, introduction to statistical learning)

We know our loss function is defined as: $\ell(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$.

Additionally, the Bayes predictor, f^* is defined as f $\in \arg\min_f R(f)$

Answer

Given we know that the true y is a function of polynomial degree two, with some constants: $a_0, a_1, a_2 \in R^3$ we would be able to faithfully recreate y given x if our function was a polynomial of degree 2, where the coefficients of our regresion $b_0, b_1, b_2 \in R^3$ eqaulled a_0, a_1, a_2, respectively.

Therefore our Bayes predictor, f^* is

$$\hat{y} = b_0 + b_1 * x + b_2 * x^2$$
$$s.t.\ b_0 = a_0\ and\ b_1 = a_1,\ and\ b_2 = a_2$$

When we calculate the R(f^*) we would get 0 as $\hat{y}$ = y.

Plugging this into our Risk function get

$$\frac{1}{2}E(\hat{y_i} - y)^2$$

This simplifies to

$$\frac{1}{2}E(0)^2\ for\ all\ i \in 1..n$$

which of course, equals 0, which is the lower bound of our Risk function

Problem 2

Using H_2 as your hypothesis class, which function $f^*_{H_2}$ is a risk minimizer in H_2? Recall the definition of the approximation error. What is the approximation error achieved by $f^*_{H_2}$?

Terminology

F_F = the Empirical risk minimizing function in our hypothesis space of functions F_* = the Bayes Predictor function

Approximation error: R(F_F) - R(F_*)

Answer

Given we know that the true y is a function of polynomial degree two, with some constants: $a_0, a_1, a_2 \in R^3$ we would be able to faithfully recreate y given x if our function was a polynomial of degree 2, where the coefficients of our regresion $b_0, b_1, b_2 \in R^3$ eqaulled a_0, a_1, a_2, respectively.

This function is in our function hypothesis set of H_2.

Therefore our Bayes predictor, F^*_{H2} is a polynomial of degree two where:

$$\hat{y} = b_0 + b_1 * x + b_2 * x^2$$
$$s.t. \, b_0 = a_0 \; and \; b_1 = a_1, \; and \; b_2 = a_2$$

When we calculate the R(f^*) we would get 0 as $\hat{y}$ = y.

Plugging this into our Risk function get

$$\frac{1}{2} E(\hat{y_i} - y)^2$$

This simplifies to

$$\frac{1}{2} E(0)^2$$

which of course, equals 0, which is the lower bound of our Risk function

Approximation Error

Since the total risk of our function is 0, the approximation error of F^*_{H2} **must be 0**

Problem 3

Considering now H_d, with $d > 2$. Justify an inequality between $R(f^*_{H_2})$ and $R(f^*_{H_d})$. Which function $f^*_{H_d}$ is a risk minimizer in H_d? What is the approximation error achieved by $f^*_{H_d}$?

Answer using information from CampusWire Forums

In this CampusWire Thread (https://campuswire.com/c/G6A12AE75/feed/23) the TA, Colin Wang clarifies how to interpret H_d. His response states that H_d represents the set of functions with degree $\leq$ d

Therefore in our hypothesis space of functions with degree $\leq$ d, where d $\geq$ 3, the Bayes Predictor (f^*) is in our hypothesis space for all d satisfying our constraints.

This is because our Bayes Predictor, as mentioned in problems 1 and 2 is:

$$\hat{y} = b_0 + b_1 * x + b_2 * x^2$$
$$s.t.\, b_0 = a_0 \; and \; b_1 = a_1, \; and \; b_2 = a_2$$

To be clear, the Bayes Predictor function would then be selected in the H_2 space of polynomial functions which is a subset of the set of functions contained in $H_{d\geq 3}$

Since approximation error is defined as: R(F_F) - R(F_*).

We observe: $F_F = F_*$.

Therefore our approximation error would be 0.

The equality we would be able to write is that R($f^*_{H_d}$) = $R(f^*_{H_2})$

Answer interpreting the question as initially stated

In the original question, the clarification is not made clear that our hypothesis set includes polynomials with a degree lower than d where d $\geq$ 3.

Given $R(f^*_{H_d})$ no longer includes our Bayes Predictor function, approximation error would exist.

The most stable fit would be to take a polynomial of d = 3 that takes on the form:

$$\hat{y} = b_0 + b_1 * x + b_2 * x^2 + b_3 * x^3$$

$$s.t.\, b_0 = a_0 \; and \; b_1 = a_1, \; and \; b_2 = a_2, \; and \; b_3 = an\; infintesmally\; small\; decimal$$

We would be able to write the inequality that R($f^*_{H_d=3}$) > $R(f^*_{H_2})$.

The risk, when examined over a large portion of data would be equal to the following formula:

$$\frac{1}{2n} * \Sigma(a_3 * x_i^3)^2 \ for \ \ i \ \in 1...n$$

Generalizing this for higher degree polynomial models the expected approximation error would be

In the below cells, we show this mathematically, using the given functions, setting a_3 to be 1e-6

In [4]:
```
#call imports
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
```

```
    return x, y

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_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.random.rand(N)
    X = get_design_mat(x, deg_true)
    y = X @ a + np.random.randn(N)
    return x, y

def empirical_risk(X, target_vector, coefficients):
    """
    Inputs:
    X: design matrix of dimmension R^(N x d+1)
    target vector: vector in dimmension R^n
    coefficients: vector in R^d+1

    Returns:
    total_risk: the empirical risk given our inputs and our loss funct
    """
    total_risk = 1/2*sum((target_vector - X@coefficients)**2)
    return(total_risk/len(target_vector))
```

```
def calc_approx_error_deg3(coef):
    """
    Inputs:
    coef: an arbitrarily small coefficient that will be assigned to fi
    degree two polynomial for the third degree

    Returns:
    total risk: float, the empirical risk of our y hat given a degree
    validation: the formula derived in the markdown cell above that es
    for a polynomial of degree 3 with arbitrarily small a_3 coefficien
    """
    a = get_a(2)
    x_train, y_train= draw_sample(2,a,1000)
    x_train_mat = get_design_mat(x_train,3)
    coefs = np.append(a,coef)
    y_pred = x_train_mat @ coefs
    total_risk = empirical_risk(x_train_mat,y_train, coefs)
    non_squared_error = (x_train_mat[:,3]*np.array(coefs[3]))
    validation = sum(non_squared_error**2)/2000
    return(total_risk, validation)
```

Problem 3 continued

The following cell actually runs the calc_approx_error_deg3() which assumes that we are taking a polynomial of degree three with an arbitrarily small coefficient for a_3 which will be multiplied against x^3.

The output shows the total risk, and the derrivation of the approximation error as hypothesized above.

Since there is no estimation risk in this case, nor optimization risk, the total risk is entirely approximation error, proving the formula above

In [2]:
```
calc_approx_error_deg3(1e-9)
```

Out[2]:
```
(7.630209622143276e-20, 7.630209335095367e-20)
```

Problem 4

Part A:

For this question we assume a_0 = 0. Considering $H = \{f : x \to b_1 x; b_1 \in R\}$, which function f_H^* is a risk minimizer in H? What is the approximation error achieved by f_H^*?

Part B:

In particular what is the approximation error achieved if furthermore a_2 =0 in the definition of true underlying relation $g(x)$ above?

Part A answer:

Solved by hand: pdf of work attached

Part B answer:

If a_2 = 0, this means our function is truly a polynomial of degree 1, which falls in our hypothesis space.

Therefore the bayes predictor function is in our hypothesis space and we are able to completely recreate any y given x, meaning total risk is 0, and therefore approximation error which is one component of risk, must be 0.

Problem 5

Show that the empirical risk minimizer (ERM) $\hat{b}$ is given by the following minimization

$$\hat{b} = \arg\min_b \|Xb - y\|_2^2$$

Problem 5 Answer:

Solved by hand: pdf of work attached

Problem 6

Part 1

If $N > d$ and X is full rank, show that $\hat{b} = (X^\top X)^{-1} X^\top y$.

5 Show $\hat{b}$ that is the ERM is given by the minimization

$\hat{b} = \arg\min \|Xb - y\|_2^2$

$ERM = \underset{f \in F}{\operatorname{argmin}} E[l(f(x), y)]$ $\quad l(f(x), y) = \frac{1}{2}(\hat{y} - y)^2$

$ERM = \min_{f \in F} E(\frac{1}{2}(\hat{y} - y)^2)$

$= \min_{f \in F} \frac{1}{2} E(\hat{y} - y)^2$ $\quad \hat{y} = X\hat{b}$

$\min_{f \in F} \frac{1}{2} E(X\hat{b} - y)^2$

Since f is now fully a function dependent on $\hat{b}$ we can rewrite our problem to find the $\hat{b}$ that minimizes squared distance between $X\hat{b}$ and y.

i.e $\hat{b} = \underset{b \in R^{d+1}}{\operatorname{argmin}} \frac{1}{2} E[(Xb - y)^2]$

$\hat{b} = \underset{b \in R^{d+1}}{\operatorname{argmin}} \left(\frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} (X_i b - y_i)^2 \right)$

Definition of L2 Norm: $\sqrt{\sum_{i=}^{n} (X_i b - y_i)^2} = \|Xb - y\|_2$

$\therefore \hat{b} = \underset{b \in R^{d+1}}{\operatorname{argmin}} \sum_{i=1}^{N} (X_i b - y_i)^2 = \|Xb - y\|_2^2$

Plugging in our loss function

$\hat{b} = \underset{b \in R^{d+1}}{\operatorname{argmin}} \frac{1}{2N} \|Xb - y\|_2^2$

Since $\frac{1}{2N}$ is a constant we can assert that $\frac{1}{2N}$ is not impactful of $\hat{b}$

$\therefore \hat{b} = \underset{b \in R^{d+1}}{\operatorname{argmin}} \|Xb - y\|_2^2 = ERM$

Q.E.D.

DS GA 1003 HW1 Problem 4

assume $f_{H^*} = b_1 x$ and $f^* = a_1 x + a_2 x^2$ what is the approximation error

Empirical Risk = $\frac{1}{2}[(\hat{y} - y)^2$) plugging in $\hat{y} = b_1 x$ and y

$= \frac{1}{2} E[(b_1 x - (a_1 x + a_2 x^2))^2]$ by definition this equals

$\int_{x=-\infty}^{\infty} l(\hat{y}, y)$ where $x \in [0,1]$

∴ Emp risk equals

$\frac{1}{2}\int_0^1 [(b_1 - a_1)x - a_2 x^2]^2 dx$ applying the square, we get

$\frac{1}{2}\int_0^1 (b_1 - a_1)^2 x^2 - 2(b_1 - a_1)a_2 x^3 + a_2^2 x^4$ applying the integral

$\frac{1}{2}\left|_0^1 \frac{1}{3}(b_1 - a_1)^2 x^3 - \frac{1}{2}(b_1 - a_1)a_2 x^4 + \frac{a_2^2 x^5}{5}\right.$ evaluating the indefinite integral

$\frac{1}{2}\cdot\left[\frac{1}{3}(b_1 - a_1)^2 - \frac{1}{2}(b_1 - a_1)a_2 + \frac{a_2^2}{5}\right]$ take the derivative w/ respect to b and set to 0 to find the loss minimizer

$\frac{1}{3}(b_1 - a_1) - \frac{1}{4}a_2 = 0$

$b_1 = a_1 + \frac{3}{4}a_2$ ✱ calculate approximation error using $b_1 = a_1 + \frac{3}{4}a_2$

$\frac{1}{2}\cdot\left[\frac{1}{3}(a_1 + \frac{3}{4}a_2 - a_1)^2 - \frac{a_2}{2}(a_1 + \frac{3}{4}a_2 - a_1) + \frac{a_2^2}{5}\right]$

$\frac{1}{2}\left[\frac{1}{3}\cdot\frac{9}{16}a_2^2 - \frac{3}{8}a_2^2 + \frac{a_2^2}{5}\right]$

Approx error $= \frac{9}{96}a_2^2 - \frac{3a_2^2}{16} + \frac{a_2^2}{10}$

$\frac{-9a_2^2}{96} + \frac{9.6a_2^2}{96} = \frac{.6a_2^2}{96} = \boxed{\frac{a_2^2}{160}}$

Part 2

Why do we need to use the conditions $N > d$ and X full rank ?

6.1 Answer:

The solution to linear regression using matrix notation for a full rank matrix is closed form. The problem can be expressed as finding

$$\hat{b} = \arg\min_{b} ||Xb - y||_2^2$$

expanding the L2 norm we get:

$$\hat{b} = \arg\min_{b} < Xb - y, Xb - y >$$

$$\hat{b} = \arg\min_{b} ||Xb||_2^2 + ||y|_2^2 - 2 < Xb - y >$$

expanding out our L2 norms we get:

$$\hat{b} = \arg\min_{b} b^t * X^t * X * b + y^T * y - b^t * X^T * y$$

Taking the derivative with respect to b and setting it to 0 to find the optimal b hat, we get:

$$0 = 2X^T * X * b - 2X^T * y^t$$

Moving the 2X^T x y^t term to the right hand side, divding both sides by 2, and multiplying (X^T * X) by it's inverse, we get:

$$b = (X^T * X)^- 1 * X^t * y^t$$

Q.E.D

6.2 Answer:

If X is full rank, this requires that X has at least as many columns as rows (Rank Nullity Theorem).

Since our X is $\in R^{Nxd+1}$ N must be > d, or equivalently N $\geq$ d + 1 **which is why we need the first condition**

Furthermore, we require X be a full rank matrix, because then the covariance matrix generated by the matrix multiplication of X @ X^T will also be full rank, which means that the matrix is invertible. Without X being full rank, we would not be able to take the inverse of X @ X ^T, making our solution to $\hat{b} invalid.

The linear algebra basis for this is:

$$Rank(AB) \leq min(m, n, k)$$

where A $\in R^{mxn}$ and B $\in R^{nxk}$ and rank(A^T) = rank(A)

This is why we need the second condition

Problem 7

Write a function called least_square_estimator taking as input a design matrix X $\in R^{N\times(d+1)}$ and the corresponding vector y $\in R^N$ returning $\hat{b} \in R^{(d+1)}$.

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).

Problem 7 answer:

See code cell below

In [5]:
```
def least_squares_est(X,target_vector):
    """
    Inputs:
    X: design matrix of dimmension R^(N x d+1)
    target vector: vector in dimmension R^n


    Returns:
    least squares coefficients: the set of coefficients that minimize
    distance between our prediction, y hat and the true data y
    """
    if X.shape[0]<= X.shape[1]:
        raise ValueError('number of rows in design matrix less than nu
    else:
        return(np.linalg.inv(X.T@X)@X.T@target_vector)
```

Problem 8

Recall the definition of the empical 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_b taking as input a design matrix X $\in R^{N\times(d+1)}$, a vector y $\in R^N$ and the vector b $\in R^{(d+1)}$ parametrizing the predictor.

Problem 8 answer:

See code cell below

In [6]:
```
def empirical_risk(X, target_vector, coefficients):
    """
    Inputs:
    X: design matrix of dimmension R^(N x d+1)
    target vector: vector in dimmension R^n
    coefficients: vector in R^d+1

    Returns:
    total_risk: the empirical risk given our inputs and our loss funct
    """
    total_risk = 1/2*sum((target_vector - X@coefficients)**2)
    return(total_risk/len(target_vector))
```

Problem 9

Part 1

Use your code to estimate $\hat{b}$ from x_train, y_train using $d = 5$.

Part 2

Compare $\hat{b}$ and a.

Part 3

Make a single plot (Plot 1) of the plan (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 $f_{\hat{b}}(x)$ in $[0, 1]$. Make sure to include a legend to your plot.

In [7]:
```
#Problem 9.1, estimate b hat from x_train, y_train, where y is a degre
#and our estimation uses a degree 5 approximation
a = get_a(2)

x_train,y_train= draw_sample(2,a,10)
x_test, y_test = draw_sample(2,a,1000)
x_train_mat = get_design_mat(x_train,5)
x_test_mat = get_design_mat(x_test,5)
b = least_squares_est(x_train_mat, y_train)
```

In [8]:
```
# Problem 9.2, compare b and a

print(b)
print(a)

print('\nWhen comparing our coefficients in vector b \
\nand our true data generating coefficients, we can see that after deg
\nthe coefficeints are incredibly small, meaning those higher degree p
\nhave small impact on our y hat calculation')
```

```
[ 6.98712138e+00 -4.51896175e+00 -2.15464570e-01 -3.56149030e-08
  3.74002411e-08 -4.40422809e-09]
[ 6.98712139 -4.51896178 -0.21546455]

When comparing our coefficients in vector b
and our true data generating coefficients, we can see that after degr
ee 2,
the coefficeints are incredibly small, meaning those higher degree po
lynomials
have small impact on our y hat calculation
```

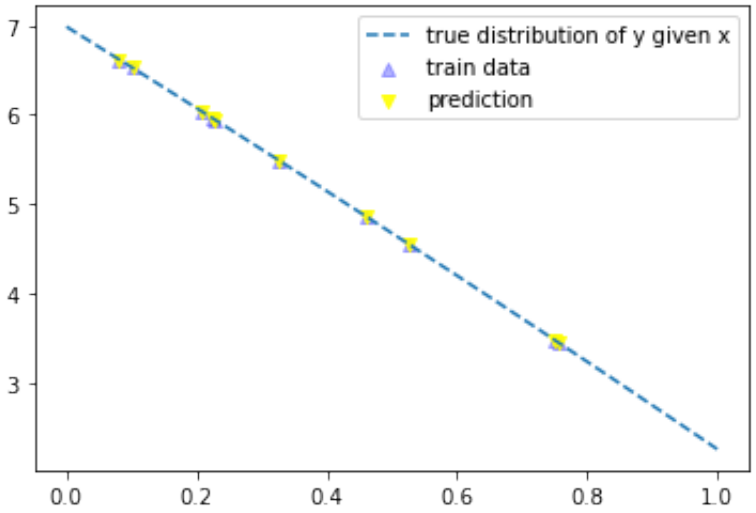
In [9]:
```
#print empirical risk on the test data set with d= 5 approximation
empirical_risk(x_test_mat,y_test,b)
```

Out[9]:
```
1.2290434477014894e-18
```

In [10]:
```
#problem 9.3
def true_func_input (x, degree):
    """
    Inputs:
    x: a vector that corresponds to the data points in our matrix
    degree: the degree to which we will transform our input vector to

    Returns:
    a matrix that contains the points that will be used to calculate t
    the data generating function
    """
    data = []
    for i in range(degree+1):
        data.append(x**i)
    return(np.array(data))

x = np.linspace(0,1, 1000)

true_func = true_func_input(x,2).T@a
y_pred = x_train_mat @ b
```

In [11]:
```
#Plot 1 for problem 9.3
plt.scatter(x_train,y_train, color = 'blue', alpha = .3, marker = '^',
plt.scatter(x_train,y_pred, color = 'yellow', alpha = .9, marker = 'v'
plt.plot(x, true_func, linestyle ='--', label = 'true distribution of
plt.legend()
#notice the overlapping between prediction and train data
```

Out[11]: <matplotlib.legend.Legend at 0x7fbd10644df0>



Problem 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?

In [12]:
```
#keeping y train and y_test the same, how well would our
#polynomial of degree i fit model a polynomial of degree 5
for i in range(9):
    x_train_mat = get_design_mat(x_train,i)
    x_test_mat = get_design_mat(x_test,i)
    b = least_squares_est(x_train_mat, y_train)
    message = 'The empirical risk of Polyonimal ' + str(i) +' is '
    emp_risk = empirical_risk(x_test_mat,y_test,b)
    print(message + str(round(emp_risk,20)))
```

```
The empirical risk of Polyonimal 0 is 1.1062846848273538
The empirical risk of Polyonimal 1 is 0.00018533395778082867
The empirical risk of Polyonimal 2 is 0.0
The empirical risk of Polyonimal 3 is 0.0
The empirical risk of Polyonimal 4 is 0.0
The empirical risk of Polyonimal 5 is 1.23e-18
The empirical risk of Polyonimal 6 is 5.54767e-15
The empirical risk of Polyonimal 7 is 2.04966397e-12
The empirical risk of Polyonimal 8 is 1.366891729748078e-05
```

Problem 10 answers

We observe a perfect fit at d = 2, which is when our sample space includes the true data generating function, which we recreate using least squares regression

With a total risk of 0, this requires approximation error to be 0 as approximation error is a non-negative component of total risk

Problem 11

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.

Part 1

Plot e_t and e_g as a function of N for $d < N < 1000$ for $d = 2$, $d = 5$ and $d = 10$. You may want to use a logarithmic scale in the plot.

Part 2

Include also plots similar to Plot 1 for 2 or 3 different values of N for each value of d.

In [28]:
```
#create lists to iterate over for d and n
deg_list = [2,5,10]
n_list = range(12,1001)

#create blank lists to store input vectors
x_train_list =[]
y_train_list = []
y_test_list = []

n_test = 1000

#create blank lists to store input matricies
x_train_mat_list = []
x_test_mat_list = []

#create blank list to store coefficients
coef_list = []

#create test sample we'll be using for the training and testing error
x_train,y_train= draw_sample_with_noise(2,a,1000)
x_test, y_test = draw_sample_with_noise(2,a,n_test)

#create blank list to store training and generalization error

training_error_dict = {}
test_error_dict = {}

for i in range(len(deg_list)):
    #determine the polynomial of the function and the coefficients of
    deg = deg_list[i]

    #create a list that will be re-initialized for each degree we char
    training_error_list = []
    test_error_list = []
```

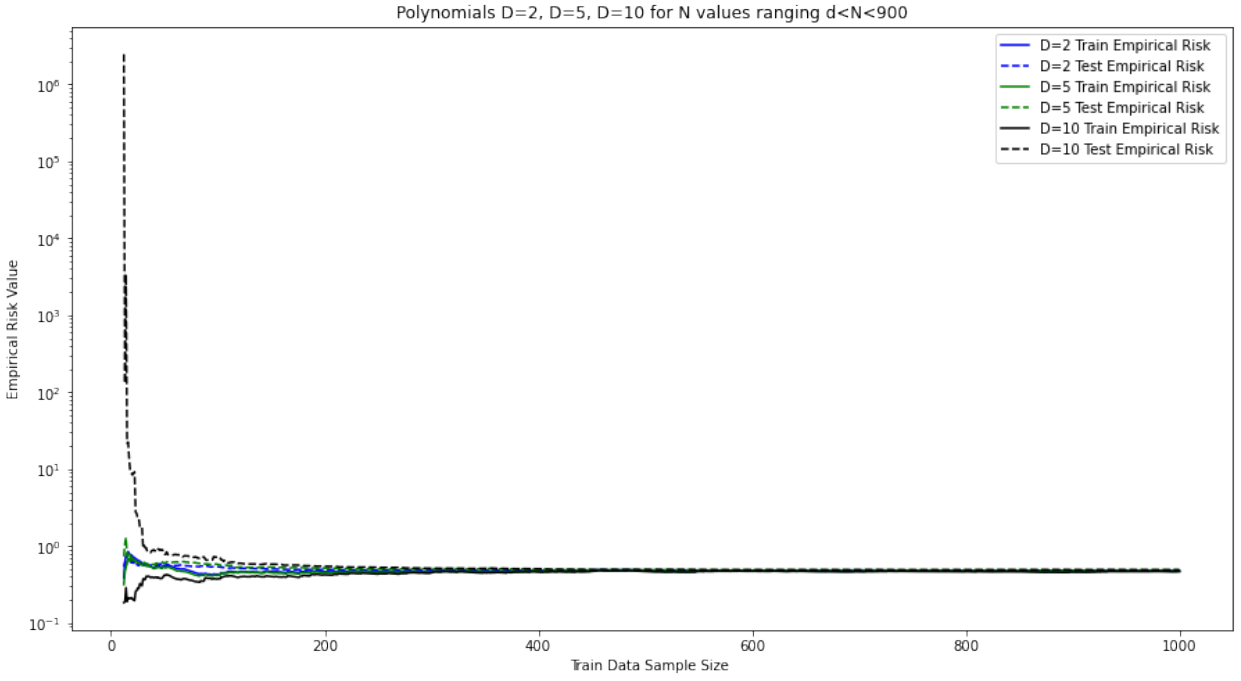
```
    for j in range(len(n_list)):
        #determine the sample size
        n = n_list[j]

        #initialize x_train, y_train, x_test, y_test



        #create input matricies

        x_train_mat = get_design_mat(x_train[0:n],deg)
        x_test_mat = get_design_mat(x_test,deg)

        #append them to matrix list
        x_train_mat_list.append(x_train_mat)
        x_test_mat_list.append(x_test_mat)
        y_train_input =  y_train[0:n]
        #run regression on x_train_mat and y
        b = least_squares_est(x_train_mat, y_train_input)



        #append coefficients to coef_list
        coef_list.append(b)

        #calculate training and testing risk, printing it
        train_emp_risk = empirical_risk(x_train_mat,y_train_input,b)

        #print('The training empirical risk of Polyonimal ' + str(deg)
        training_error_list.append(train_emp_risk)


        #print('\n')
        test_emp_risk = empirical_risk(x_test_mat,y_test,b)
        #print('The testing empirical risk of Polyonimal ' + str(deg)
        #print('\n')
        test_error_list.append(test_emp_risk)
        if j == len(n_list)- 1:
            training_error_dict[deg] = training_error_list
            test_error_dict[deg] = test_error_list
```

Problem 11 Part 1:

Plot e_t and e_g as a function of N for $d < N < 1000$for $d = 2$, $d = 5$ and $d = 10$. You may want to use a logarithmic scale in the plot.

In [27]:

```
plt.figure(figsize=(15,8))
plt.plot(n_list, training_error_dict[2], color = 'blue')
plt.xlabel("Train Data Sample Size ")
plt.ylabel("Empirical Risk Value")


plt.plot(n_list, test_error_dict[2], linestyle ='--', color ='blue')
plt.plot(n_list, training_error_dict[5], color = 'green')
plt.plot(n_list, test_error_dict[5], color = 'green', linestyle ='--')
plt.plot(n_list, training_error_dict[10], color = 'black')
plt.plot(n_list, test_error_dict[10], color = 'black',linestyle ='--')
#plt.plot(n[6:], train_error_arr_5)
#plt.plot(n[6:], test_error_arr_5)
#plt.plot(n[11:], train_error_arr_10)
#plt.plot(n[11:], test_error_arr_10)
plt.legend(labels=['D=2 Train Empirical Risk'
                   ,'D=2 Test Empirical Risk',
                   'D=5 Train Empirical Risk',
                   'D=5 Test Empirical Risk',
                   'D=10 Train Empirical Risk',
                   'D=10 Test Empirical Risk'])
plt.title("Polynomials D=2, D=5, D=10 for N values ranging d<N<900")
plt.yscale('log')
```



Observations on the plot:

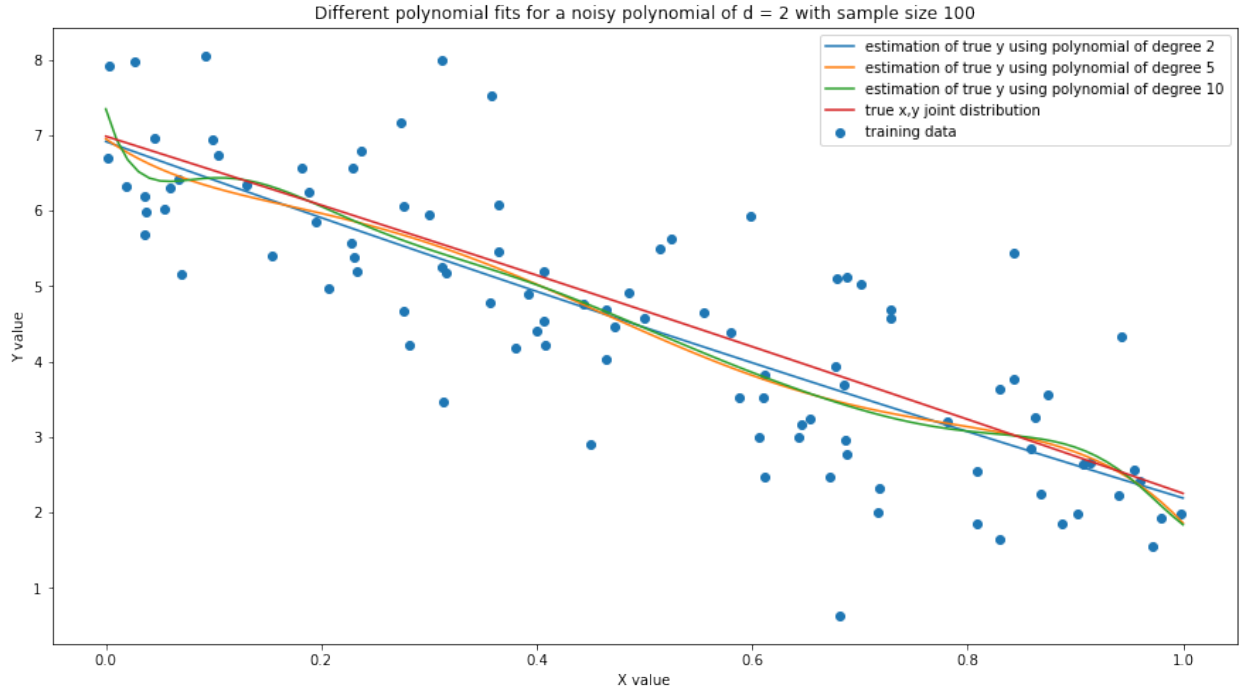
1. Test error is magnitudes higher than training error, regard
less of D sample size.
2. For a low N and a higher D, total risk goes up, driven by e
stimation error
3. Even for a high D, as N increases, Empirical Risk exponenti
anlly converges to 0,
 as the model learns more about the true g(x)

Problem 11 Part 2

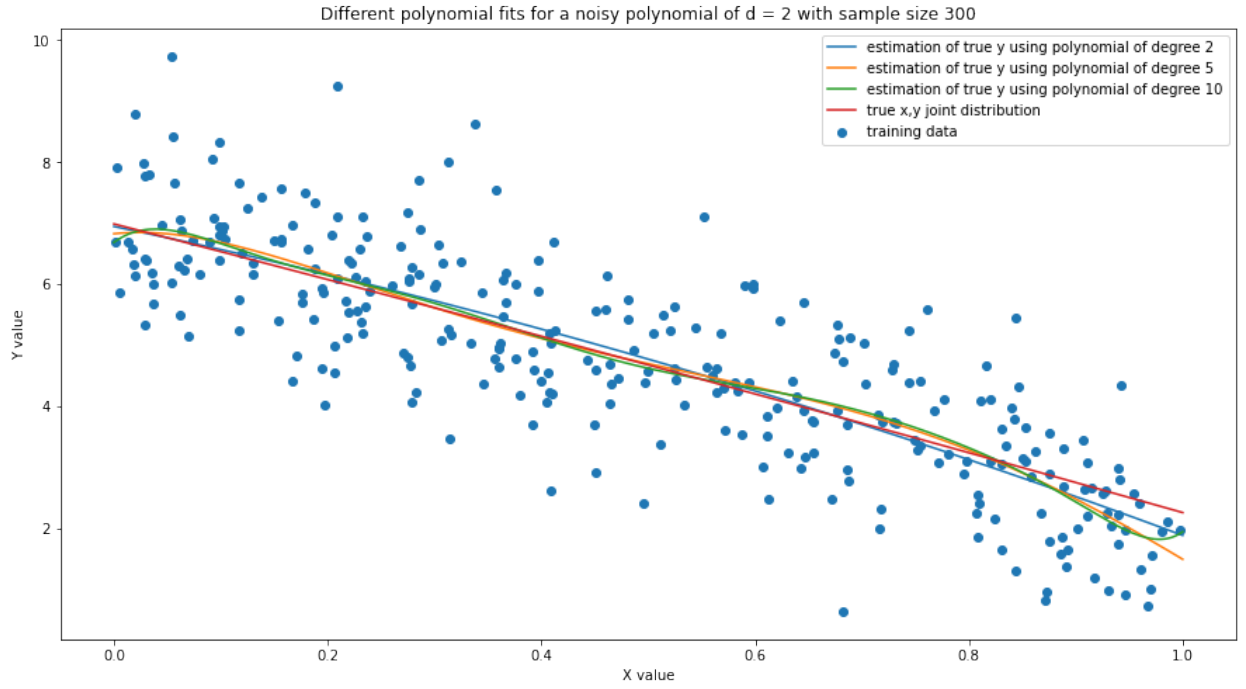
Include also plots similar to Plot 1 for 2 or 3 different values of N for each value of d.

In [15]:
```
f create_plot_train(n):
  plt.figure(figsize=(15,8))
  index = n-12
  #plot training data
  plt.scatter(x_train[0:n], y_train[0:n], label = 'training data')
  #plot true_function
  #plot different polynomial functions
  y_pred_list =[]
  x = np.linspace(0,1,n)
  x_for_y_pred = true_func_input(x,2).T
  true_y = x_for_y_pred@a

  for i in range(len(x_train_mat_list)//988):
      x_for_y_pred = true_func_input(x,deg_list[i]).T
      y_pred = x_for_y_pred @ coef_list[index + i + 988*i]
      y_pred_list.append(y_pred)
      plt.plot(x,y_pred_list[i], label = 'estimation of true y using p

  message = "Different polynomial fits for a noisy polynomial of d = 2
  plt.title(message)
  plt.xlabel("X value")
  plt.ylabel("Y value")


  plt.plot(x, true_y[0:n], label= 'true x,y joint distribution')
  plt.legend()
  plt.show()
```

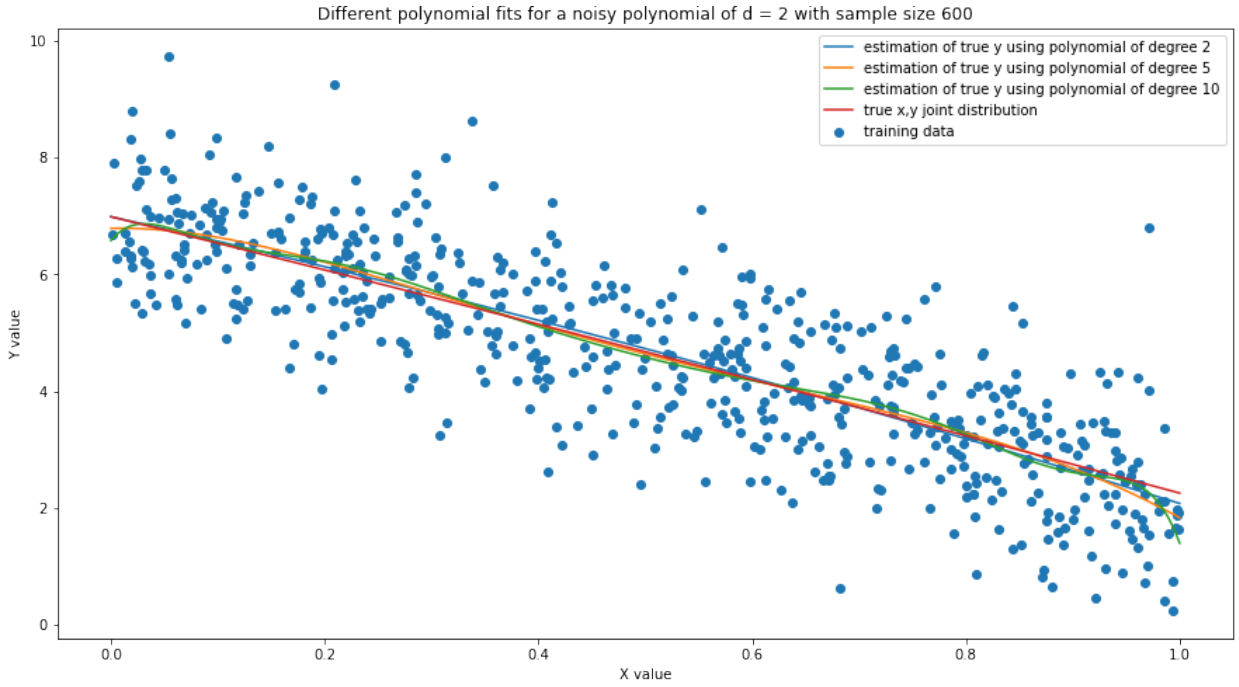
In [16]:
```
#recreating "plot 1" with a noisy dataset using n = 100, d = 2,5,10
create_plot_train(100)
```



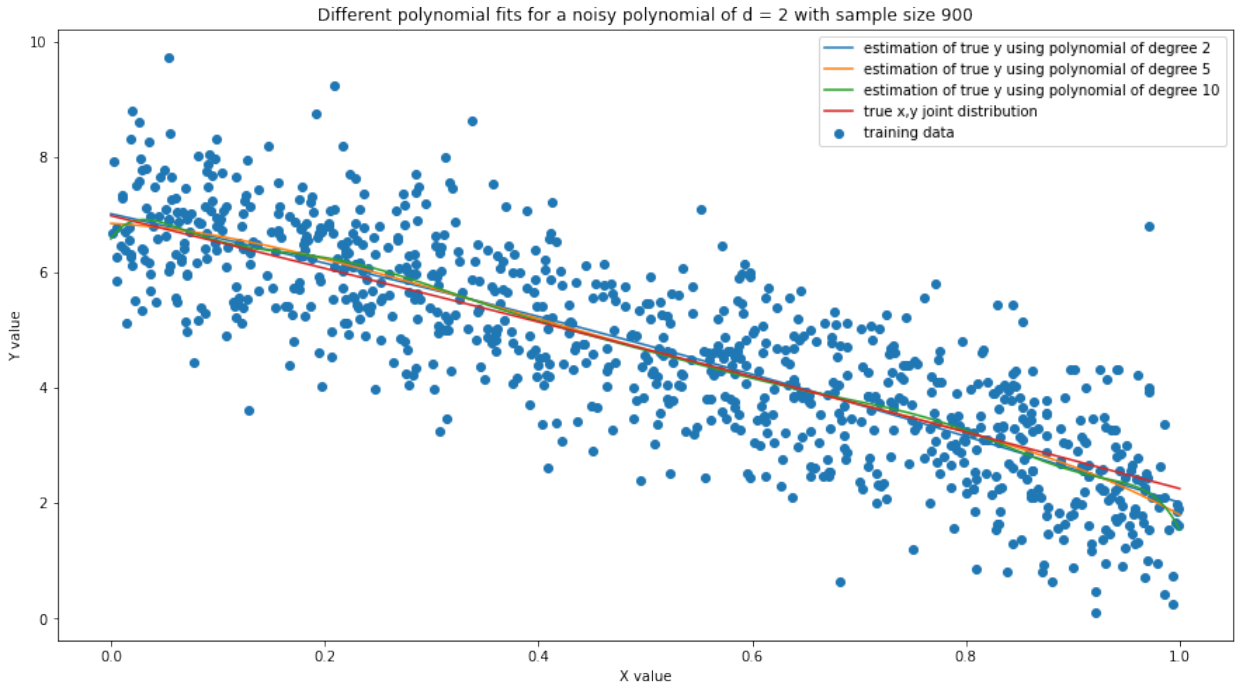
In [17]:
```
#recreating "plot 1" with a noisy dataset using n = 300, d = 2,5,10

create_plot_train(300)
```



In [18]:
```
#recreating "plot 1" with a noisy dataset using n = 600, d = 2,5,10
create_plot_train(600)
```



In [19]:
```
#recreating "plot 1" with a noisy dataset using n = 900, d = 2,5,10

create_plot_train(900)
```



Problem 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)

Terminology

Estimation error = $\hat{R}(f_n)$- $\hat{R}$(F$_F$)

Problem 12 Answer

As we have observed in problem 11, the total error on both the train and test data sets approaches to .5 for all values of D as N approahces 200.

The below code cell highlights this

In [20]:
```
print('The training error using degree = 2 and n = 200 is ' + str(roun
print('The test error using degree = %s and n = 200 is ' + str(round(t

print('\n')
print('The training error using degree = 5 and n = 200 is ' + str(roun
print('The test error using degree = 5 and n = 200 is ' + str(round(te

print('\n')
print('The training error using degree = 10 and n = 200 is ' + str(rou
print('The test error using degree = 10 and n = 200 is ' + str(round(t
```

```
The training error using degree = 2 and n = 200 is 0.549
The test error using degree = %s and n = 200 is 0.5


The training error using degree = 5 and n = 200 is 0.541
The test error using degree = 5 and n = 200 is 0.517


The training error using degree = 10 and n = 200 is 0.538
The test error using degree = 10 and n = 200 is 0.519
```

Problem 12 Answer Continued

However, we see that total error is much higher on the test error for a high degree polynomial fit with a low n.

The below cell highlights this.

In [21]:
```
print('The test error using degree = %s and n = 12 is ' + str(round(te

print('\n')
print('The test error using degree = 5 and n = 12 is ' + str(round(tes

print('\n')
print('The test error using degree = 10 and n = 12 is ' + str(round(te
```

```
The test error using degree = %s and n = 12 is 1.343


The test error using degree = 5 and n = 12 is 1.249


The test error using degree = 10 and n = 12 is 944771.672
```

Problem 12 Answer Continued

Using the following formula:

$$totalerror = approximationerror + estimationerror + optimizationerror + bayes$$

Note bayesian error is the minimal risk associated with the bayesian precictor.

For a given N, D and test set, we can solve for estimation error if we know approximation error, bayesian error and optimization error.

In our situation:

```
    1. Approximation error is 0 for all polynomials of degree
greater than 2, because the
    true data generating function is within our function hypot
hesis set
    2. Optimization error is 0 because we are employing the cl
osed form solution to linear regression,
    which finds the optimal solution to the problem according
to our error function
    3. Bayesian error is .5, this is becasue of the random noi
se term.
```

Further elaboration on Bayesian Error

The Bayesian Error is .5 as the risk of the ideal function can be calcluated as:

$$x@a = \hat{y}$$

Where x ∈ R^3 taking the form x ∈ {x_0, x_1, x_1^2} and a ∈ R^3 where a are the coefficients used to calcluate y = x @ a + ϵ where epsilon N(0,1)

Plugging this into our Empirical Risk function we get:

$$\frac{1}{2}E(\hat{y} - y)^2$$

This equals:

$$\frac{1}{2}E(x * a - x * a + \epsilon)^2$$

which simplfies to:

$$\frac{1}{2}E(\epsilon)^2$$

which further reduces to:

$$\frac{1}{2}E(\epsilon^2)$$

Lastly, since this is a mean 0 normally distributed variable, we know that

$$E(\epsilon^2) = V(\epsilon) = 1$$

Therefore:

$$bayes\ error = 1/2 * 1 = 1/2$$

Empircal estimation of estimation error

Thus we know our formula:

$$totalerror = approximationerror + estimationerror + optimizationerror + bayes$$

simplifies to

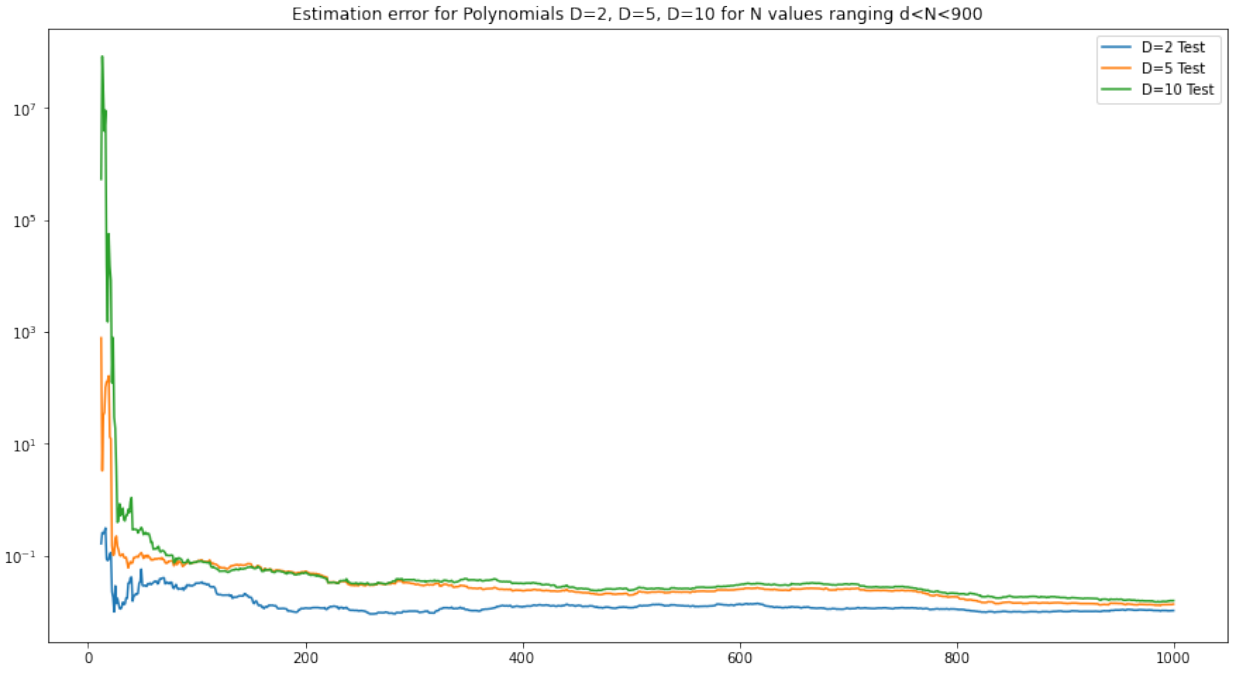
$$totalerror = 0 + estimationerror + 0 + .5$$

Solving for estimation error, we get:

$$estimationerror = totalerror - .5$$

The plots below demonstrate this

In [29]:
```
# Plot 3
estimation_error_2 = np.array(test_error_dict[2])-.5
estimation_error_5 = np.array(test_error_dict[5])-.5
estimation_error_10 = np.array(test_error_dict[10])-.5




plt.figure(figsize=(15,8))
plt.plot(n_list, np.array(estimation_error_2))
plt.plot(n_list, np.array(estimation_error_5))
plt.plot(n_list, np.array(estimation_error_10))


plt.legend(labels=['D=2 Test', 'D=5 Test', 'D=10 Test'])
plt.title("Estimation error for Polynomials D=2, D=5, D=10 for N value
plt.yscale('log')
```



Problem 13

The generalization error gives in practice an information related to the estimation error.

Part 1

Comment on the results of (Plot 2 and 3). What is the effect of increasing N?

Part 2

What is the effect of increasing D?

Problem 13 Part 1 answer.

We observe for any d, that increasing the N exponentially decreases estimation error until it converges to 0 as N increases.

Additional comments on Plot 2 were stated above, but repeated here:

```
1. Test error is magnitudes higher than training error, regard
less of D sample size.
2. For a low N and a higher D, total risk goes up, driven by e
stimation error
3. Even for a high D, as N increases, Empirical Risk exponenti
anlly converges to 0,
 as the model learns more about the true g(x)
```

Problem 13 Part 2 answer:

Increasing D, for a small N results in large generalization error. However, as mentioned above even for a large D, as N increases, estimation error still converges to 0 by N = 200 for d in 5 and 10.

Problem 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?

Problem 14 answer:

The optimization error of all models we have made is 0, as the closed form linear solution to linear regression can be obtained. This was guaranteed in our function by taking random uniform variables as our x vector, and raising an exception if there were more columns than rows.

Had we taken an approach like Gradient Descent to find the optimal coefficients for our linear regression problem, there would be optimizaiton error, as it is not guaranteed to converge to the global minima of the error function using this optimization technique

Problem 15

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

In [30]:
```
ozone_dat = np.loadtxt("ozone_wind.data")
#ozone data is first wind data is second column

#create test train split of 80/20
oz_train = ozone_dat[:,0][:80]
oz_test = ozone_dat[:,0][80:]

wind_train = ozone_dat[:,1][:80]

wind_test = ozone_dat[:,1][80:]
```

In [32]:
```
# goal, fit ozone as our response variable using various polynomial fi
for i in range(1,15):
    wind_train_mat = get_design_mat(wind_train, i)
    test_deg = get_design_mat(wind_test,i)
    coef = least_squares_est(wind_train_mat, oz_train)
    emp_risk = empirical_risk(test_deg,oz_test, coef)

    print("empirical risk is ",str(round(emp_risk,3)) + " at d = ",str
```

```
empirical risk is  258.392 at d =  1
empirical risk is  218.997 at d =  2
empirical risk is  227.226 at d =  3
empirical risk is  226.595 at d =  4
empirical risk is  219.326 at d =  5
empirical risk is  234.803 at d =  6
empirical risk is  256.298 at d =  7
empirical risk is  274.947 at d =  8
empirical risk is  326.599 at d =  9
empirical risk is  419.309 at d =  10
empirical risk is  336.149 at d =  11
empirical risk is  103941.631 at d =  12
empirical risk is  309.682 at d =  13
empirical risk is  674.436 at d =  14
```

Problem 15 commentary:

We observere minimum generalization error at d = 2 using an 80 twenty test split. However, let's observe how generalization error changes as train N and d change.

In [33]:
```
ozone_n_list = [20,30,50,70]
ozone_coef_list = []
for j in range(len(ozone_n_list)):
    for i in range(1,15):

        oz_train = ozone_dat[:,0][:ozone_n_list[j]]
        oz_test = ozone_dat[:,0][ozone_n_list[j]:]

        wind_train = ozone_dat[:,1][0:ozone_n_list[j]]
        wind_test = ozone_dat[:,1][ozone_n_list[j]:]



        wind_train_mat = get_design_mat(wind_train, i)
        test_deg = get_design_mat(wind_test,i)
        coef = least_squares_est(wind_train_mat, oz_train)
        ozone_coef_list.append(coef)
        emp_risk = empirical_risk(test_deg,oz_test, coef)

        print("generealization error is " + str(round(emp_risk,3)) + "
              + " and train n = " + str(ozone_n_list[j])
             )
```

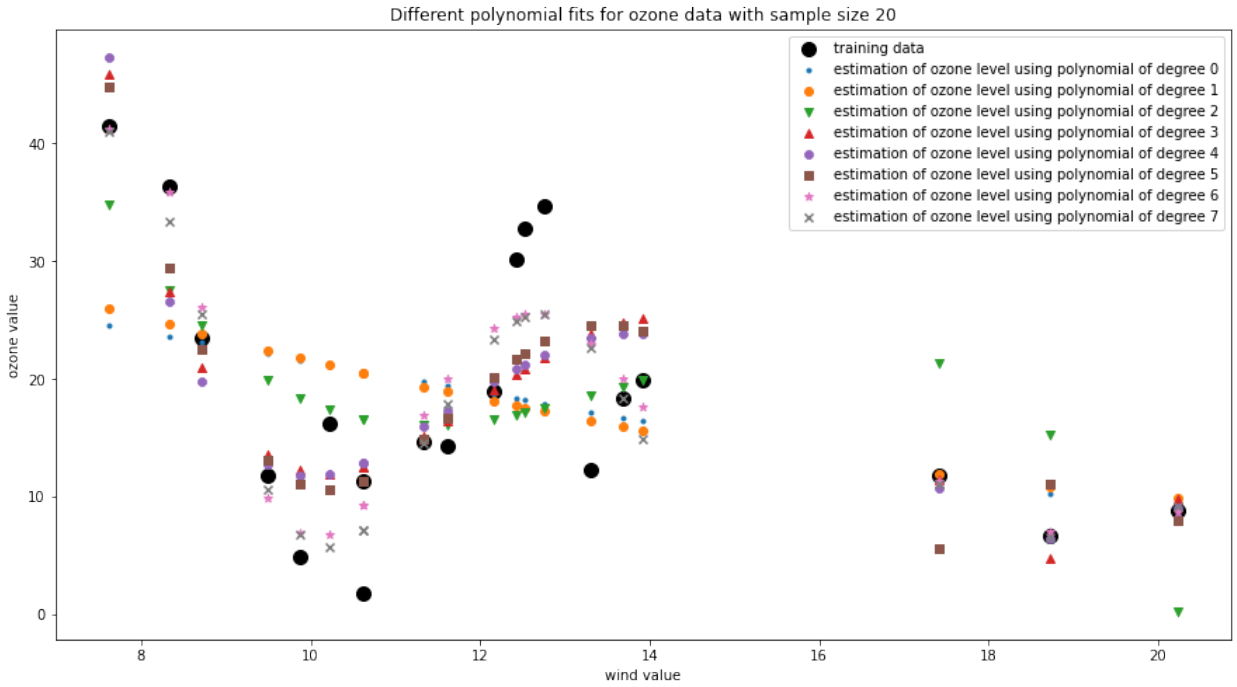
```
generealization error is 844.193 at d = 1 and train n = 20
generealization error is 792.641 at d = 2 and train n = 20
generealization error is 483.601 at d = 3 and train n = 20
generealization error is 4392.876 at d = 4 and train n = 20
generealization error is 9344.144 at d = 5 and train n = 20
generealization error is 15100.893 at d = 6 and train n = 20
generealization error is 2389481.796 at d = 7 and train n = 20
generealization error is 288600.76 at d = 8 and train n = 20
generealization error is 234038.814 at d = 9 and train n = 20
generealization error is 199898.431 at d = 10 and train n = 20
generealization error is 943778.571 at d = 11 and train n = 20
generealization error is 2318495.036 at d = 12 and train n = 20
generealization error is 93168.365 at d = 13 and train n = 20
generealization error is 542622.437 at d = 14 and train n = 20
generealization error is 635.082 at d = 1 and train n = 30
generealization error is 384.276 at d = 2 and train n = 30
generealization error is 682.502 at d = 3 and train n = 30
generealization error is 6884.389 at d = 4 and train n = 30
generealization error is 5213.945 at d = 5 and train n = 30
generealization error is 6147.763 at d = 6 and train n = 30
generealization error is 18765.308 at d = 7 and train n = 30
generealization error is 73422.664 at d = 8 and train n = 30
generealization error is 905544.144 at d = 9 and train n = 30
generealization error is 27651.612 at d = 10 and train n = 30
generealization error is 190986.286 at d = 11 and train n = 30
generealization error is 25702740393.877 at d = 12 and train n = 30
generealization error is 15655.832 at d = 13 and train n = 30
generealization error is 7914995.427 at d = 14 and train n = 30
generealization error is 392.838 at d = 1 and train n = 50
generealization error is 330.215 at d = 2 and train n = 50
generealization error is 409.68 at d = 3 and train n = 50
generealization error is 532.668 at d = 4 and train n = 50
generealization error is 546.196 at d = 5 and train n = 50
generealization error is 1840.36 at d = 6 and train n = 50
generealization error is 3351.74 at d = 7 and train n = 50
generealization error is 728.791 at d = 8 and train n = 50
generealization error is 482.998 at d = 9 and train n = 50
generealization error is 1402.507 at d = 10 and train n = 50
generealization error is 205802.849 at d = 11 and train n = 50
generealization error is 14612.97 at d = 12 and train n = 50
generealization error is 73579.867 at d = 13 and train n = 50
generealization error is 139052.423 at d = 14 and train n = 50
generealization error is 327.741 at d = 1 and train n = 70
generealization error is 239.924 at d = 2 and train n = 70
generealization error is 279.196 at d = 3 and train n = 70
generealization error is 420.193 at d = 4 and train n = 70
generealization error is 222.487 at d = 5 and train n = 70
generealization error is 222.839 at d = 6 and train n = 70
generealization error is 217.728 at d = 7 and train n = 70
generealization error is 831.042 at d = 8 and train n = 70
generealization error is 1308.174 at d = 9 and train n = 70
generealization error is 370.704 at d = 10 and train n = 70
generealization error is 15579.962 at d = 11 and train n = 70
```

```
generealization error is 10105.689 at d = 12 and train n = 70
generealization error is 2210533.541 at d = 13 and train n = 70
generealization error is 102180.44 at d = 14 and train n = 70
```

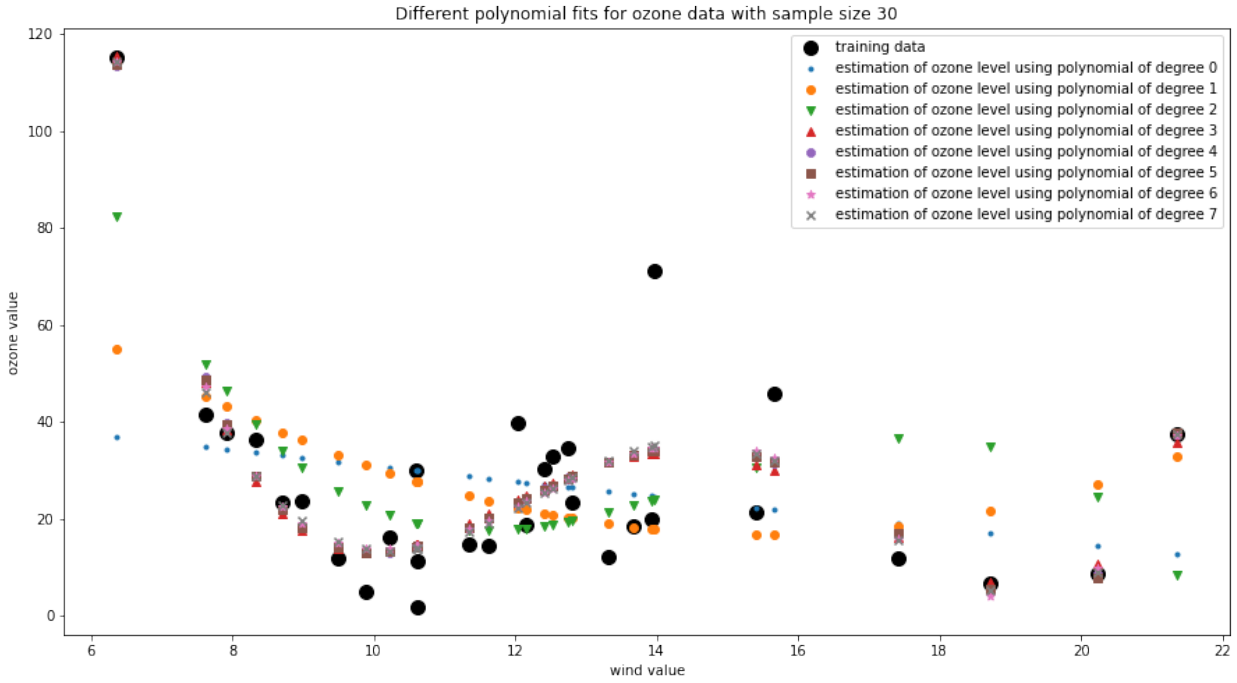
Interestingly when n = 70 and d = 7 we see the lowest generalizaiton error

In [39]:
```
def oz_create_plot_train(n):
    plt.figure(figsize=(15,8))
    index = n-12
    n_dict_list = {20:0,30:14,50:28,70:42}
    #plot training data
    plt.scatter(ozone_dat[:,1][:n], ozone_dat[:,0][:n], \
                s= 100, label = 'training data', color ='black')
    #plot true_function
    #plot different polynomial functions
    y_pred_list =[]
    shape_list = [".", "o","v", "^", "8", "s", "*", "x"]
    for j in range(0,8):
        if j == 14:
            pass
        else:
            wind_train = ozone_dat[:,1][0:n]
            wind_train_mat = get_design_mat(wind_train, j+1)

            y_pred = wind_train_mat @ ozone_coef_list[n_dict_list[n]+j
            y_pred_list.append(y_pred)
            plt.scatter(wind_train,y_pred_list[j], \
                     label = 'estimation of ozone level using polynomi
                    marker = shape_list[j])

    message = "Different polynomial fits for ozone data with sample si
    plt.title(message)
    plt.xlabel("wind value")
    plt.ylabel("ozone value")


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

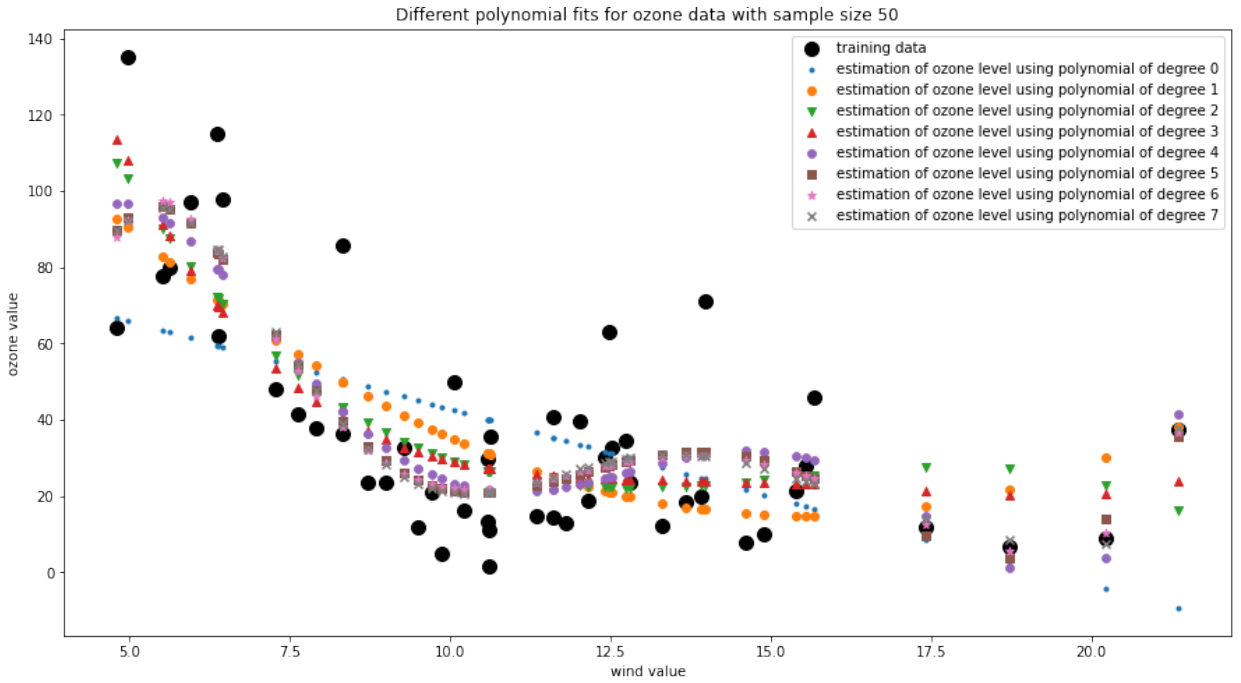
In [40]:
```
oz_create_plot_train(20)
```



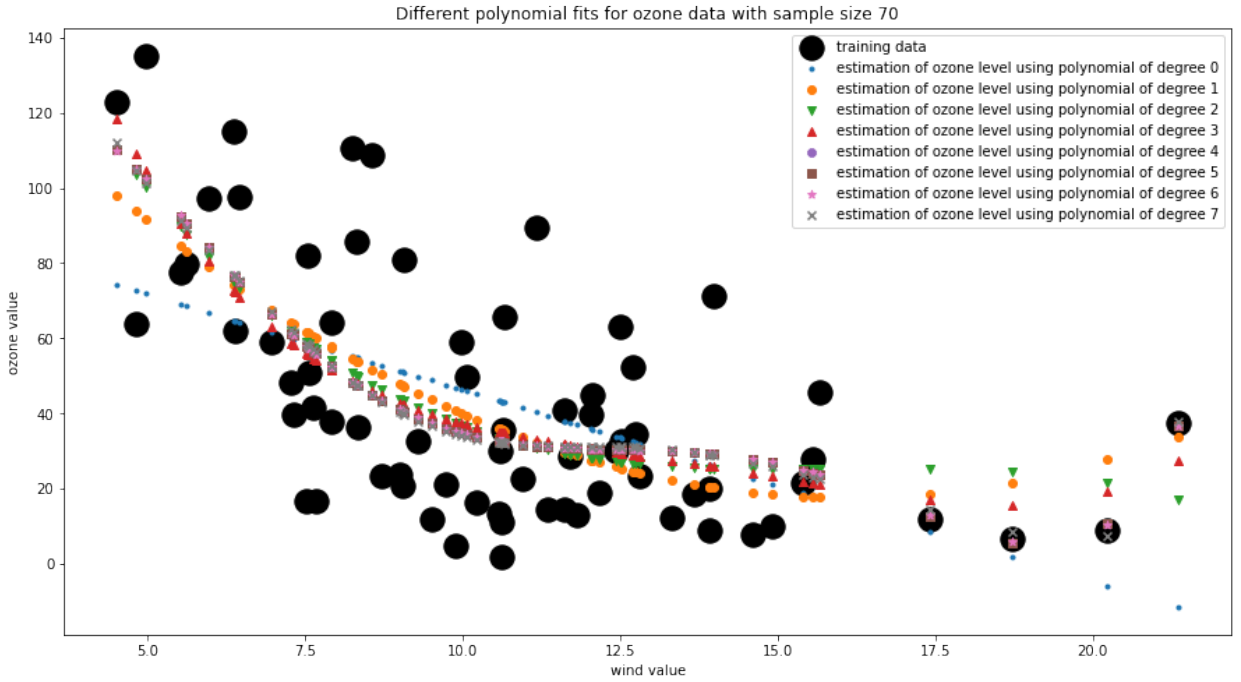
In [41]:
```
oz_create_plot_train(30)
```



In [42]:
```
oz_create_plot_train(50)
```



In [38]:
```
oz_create_plot_train(70)
```



Plot Commentary

When looking at our fits of the data for varying d and n, it seems that low degree polynomials do a good job at lower values of wind, however, as wind increases a higher degree polynomial fit works better in predicting ozone values.

In the later graphs we see a d between 4-7 seems to reasonably fit our data, especially when given higher N

In [ ]: