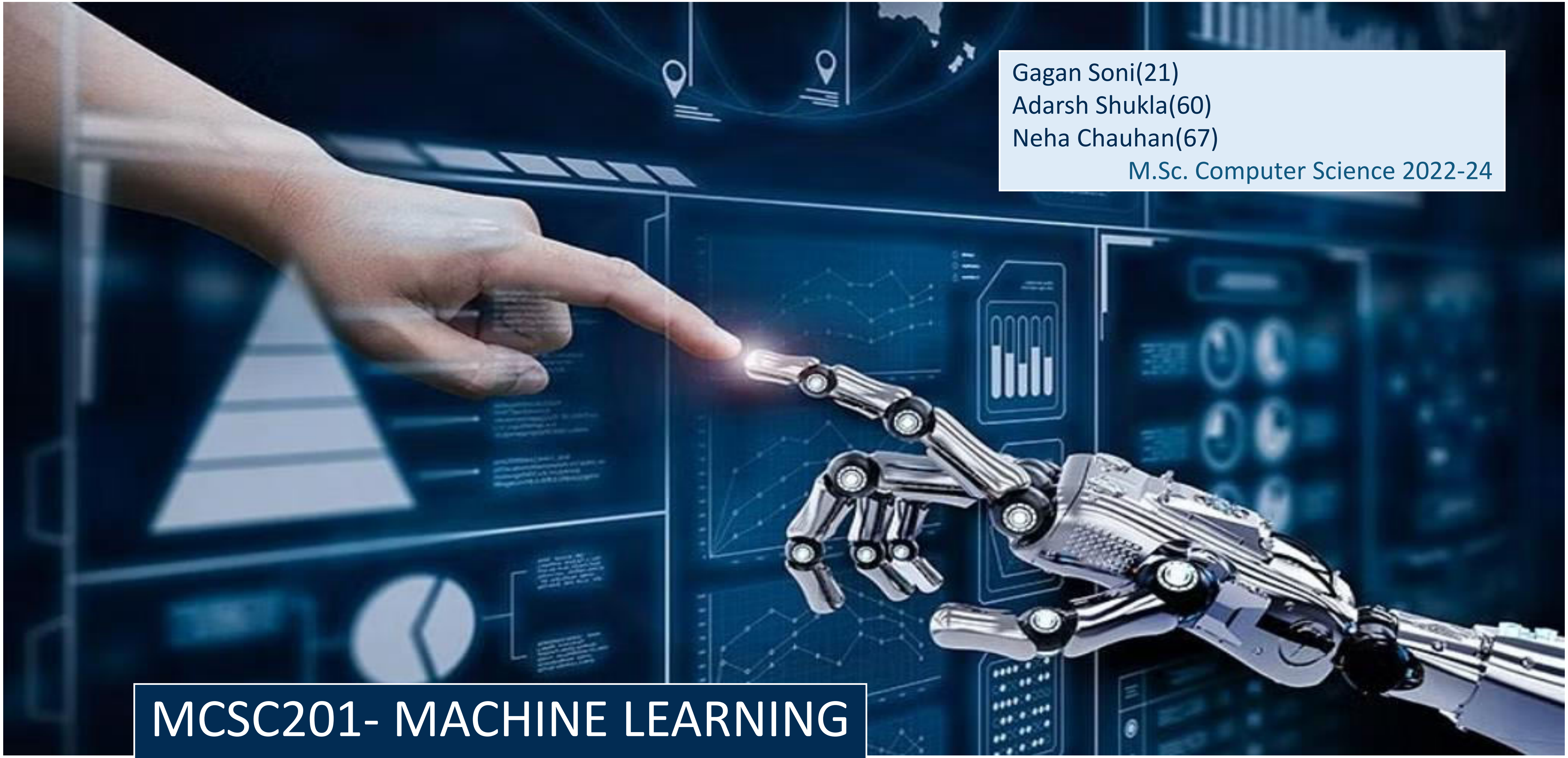


Gagan Soni(21)
Adarsh Shukla(60)
Neha Chauhan(67)
M.Sc. Computer Science 2022-24

MCSC201- MACHINE LEARNING ASSIGNMENT 1

Bias-Variance trade-off for Model Selection

Table Of Content

Problem Statement

Take a trigonometric function and a choose an error function (N(0, sigma-sq)). Generate data set of 10,000 instances. Fit polynomials of order 1 - 10 and estimate and plot total error, Bias, Variance,

training and validation error for each using 10-fold cross validation. Create the folds using Python function and compute all errors using the equations given in textbook. Select the optimal model.

For the selected model, estimate and plot total error, Bias, Variance, training and validation error for training set sizes 1K, 2K, ... 10K. Use 10-fold cross validation for each training set, and the functions coded earlier for estimating the error.

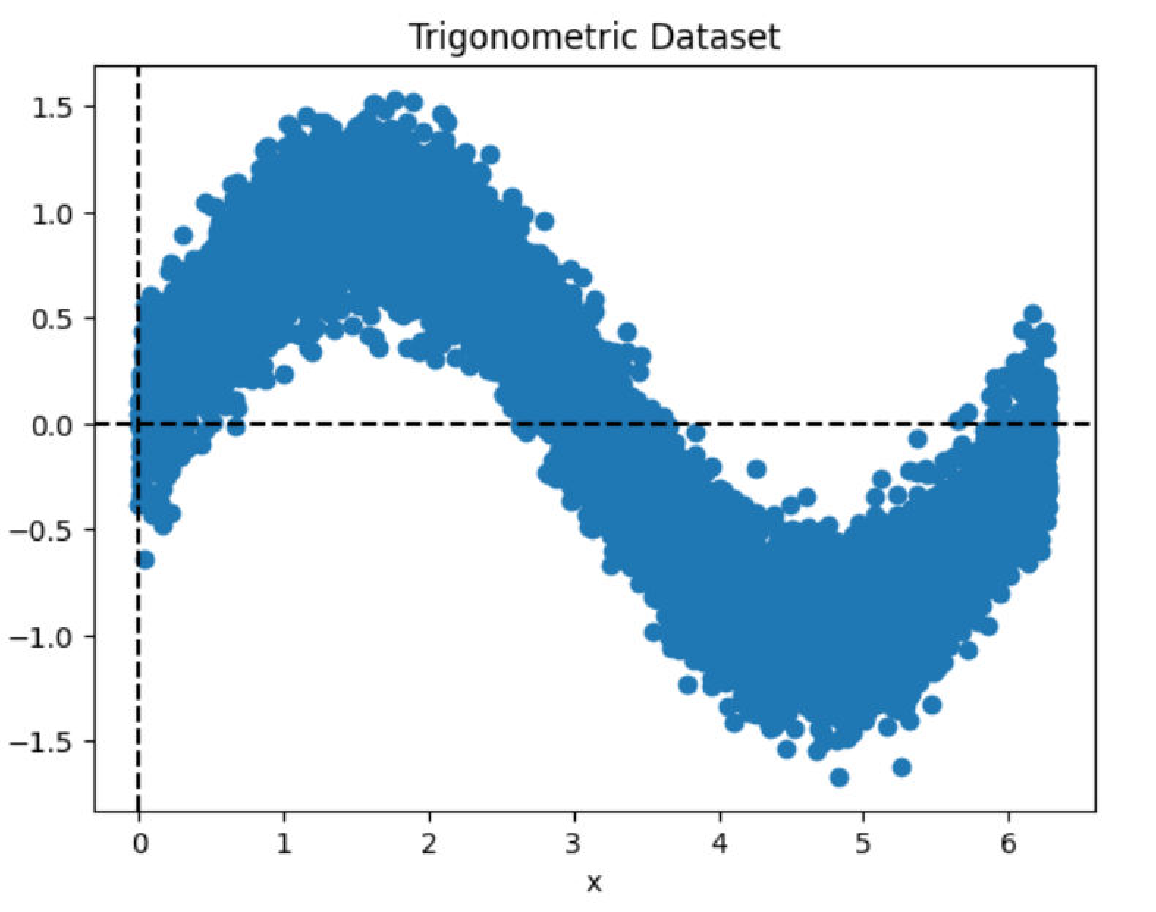
Dataset Generation

- Trignometric function is taken as user input using format np.trig_func e.g. np.sin for sine.
- Angles are generated using random function and noise using normal distribution(randomly).
- Dataframe is generated from the dataset generated.

```
def generate_dataset(num_instances, sigma, trig_func):
    dataset = []
    for _ in range(num_instances):
        angle = np.random.uniform(0, 2 * np.pi)  # Random angle between 0 and 2*pi
        error = np.random.normal(0, sigma)  # Random error from a normal distribution
        value = trig_func(angle) + error  # Compute the value with the user-defined trigonometric function and error
        dataset.append([angle, value])
    return dataset

num_instances = 10000
sigma = 0.2  # Adjust the value of sigma to control the spread of the errors

# Get the user's input for the trigonometric function
trig_func_input = input("Enter the trigonometric function (e.g., 'np.sin' for sine): ")
trig_func = eval(trig_func_input)  # Evaluate the user's input to obtain the function

dataset = generate_dataset(num_instances, sigma, trig_func)

# Create a DataFrame from the dataset
df = pd.DataFrame(dataset, columns=['x', 'y'])
df.head()
```

Dataset Representation

- Input trignometric function : np.sin(i.e. sine function).
- x-axis represents the angle between 0 and 2π.
- y-axis represents value of trignometric function for the given angle.



Value Calculation

- Training and testing error is calculated using mean squared error between actual and predicted values of training and testing dataset.
- **Mean Squared Error :** $\sum(\boldsymbol{y}-\boldsymbol{y}_{\mathrm{i}})^2/\mathrm{n}$
- Bias and variance is calculated testing data.
- Total error= bias +variance

```python
def calculate_values(dataset, train_indices, test_indices, degree):
    x_train = dataset[train_indices, 0]
    y_train = dataset[train_indices, 1]
    x_test = dataset[test_indices, 0]
    y_test = dataset[test_indices, 1]

    coefficients = np.polyfit(x_train, y_train, degree)
    y_train_pred = np.polyval(coefficients, x_train)
    y_test_pred = np.polyval(coefficients, x_test)

    train_error = mean_squared_error(y_train, y_train_pred)
    test_error = mean_squared_error(y_test, y_test_pred)

    bias = np.mean((y_test - np.mean(y_test_pred))**2)
    variance = np.var(y_test_pred)

    total_error = (bias+variance)

    return train_error, test_error, bias, variance, total_error
```

Polynomial Fitting

- We need to calculate bias, variance, training error, testing error and total error using 10-fold cross validation.

- **K-fold cross validation** : K-Fold is validation technique in which we split the data into k-subsets and the holdout method is repeated k-times where each of the k subsets are used as test set and other k-1 subsets are used for the training purpose. Then the average error from all these k trials is computed.

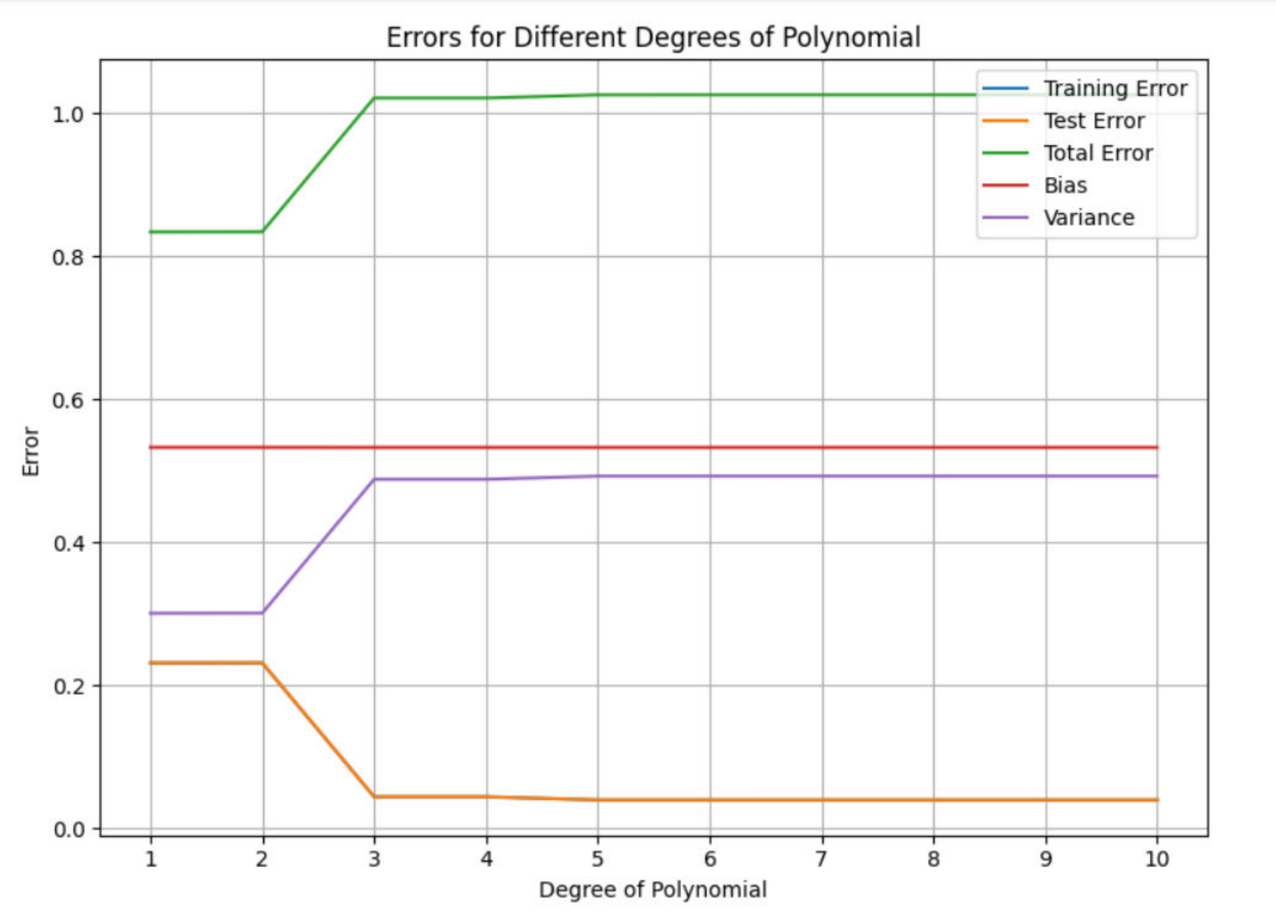
```
def fit_polynomial(dataset, degrees, cv_folds=10):
    kf = KFold(n_splits=cv_folds)
    total_errors = []
    biases = []
    variances = []
    train_errors = []
    test_errors = []

    for d in degrees:
        total_errors.append([])
        biases.append([])
        variances.append([])
        train_errors.append([])
        test_errors.append([])

    for train_indices, test_indices in kf.split(dataset):
        for idx, degree in enumerate(degrees):

            train_error, test_error, bias, variance, total_error = calculate_values(dataset, train_indices, test_indices, degree)
            total_errors[idx].append(total_error)
            biases[idx].append(bias)
            variances[idx].append(variance)
            train_errors[idx].append(train_error)
            test_errors[idx].append(test_error)

    for idx in range(0, len(degrees)):
        total_errors[idx] = np.mean(total_errors[idx])
        biases[idx] = np.mean(biases[idx])
        variances[idx] = np.mean(variances[idx])
        train_errors[idx] = np.mean(train_errors[idx])
        test_errors[idx] = np.mean(test_errors[idx])
    return total_errors, biases, variances, train_errors, test_errors
```

Bias, Variance, Training Error, Testing Error and Total Error

Training Set Size	Total Error	Bias	Variance	Training Error	Testing Error
1000	1.06137	0.551319	0.510056	0.0404971	0.0410599
2000	1.05077	0.54495	0.505817	0.0382795	0.0385956
3000	1.03895	0.539134	0.499813	0.0393754	0.0395735
4000	1.04276	0.541245	0.501511	0.0397937	0.0399523
5000	1.0469	0.543395	0.503509	0.0399773	0.0400813
6000	1.04431	0.542235	0.502071	0.0398689	0.0399881
7000	1.03702	0.538437	0.498581	0.0397866	0.0398784
8000	1.03994	0.539881	0.500062	0.0398606	0.0399548
9000	1.03823	0.539036	0.49919	0.0398423	0.0399195
10000	1.03281	0.53637	0.496444	0.0400131	0.0400757

Error Plotting

- Bias, Variance, Training error, Testing error and Total error generated earlier is plotted for different degree of polynomial.
- x-axis represents the degree of ploynomial.
- y-axis represents the error.



Optimal degree Calculation

- **Optimal degree** is the degree of polynomial for which the model is best fit.
- For the taken function sin(in our case) the optimal degree is 7.

```
optimal_degree_idx = np.argmin(test_errors)
optimal_degree = degrees[optimal_degree_idx]
print(optimal_degree)
```

```
7
```

Fitting Optimal Polynomial

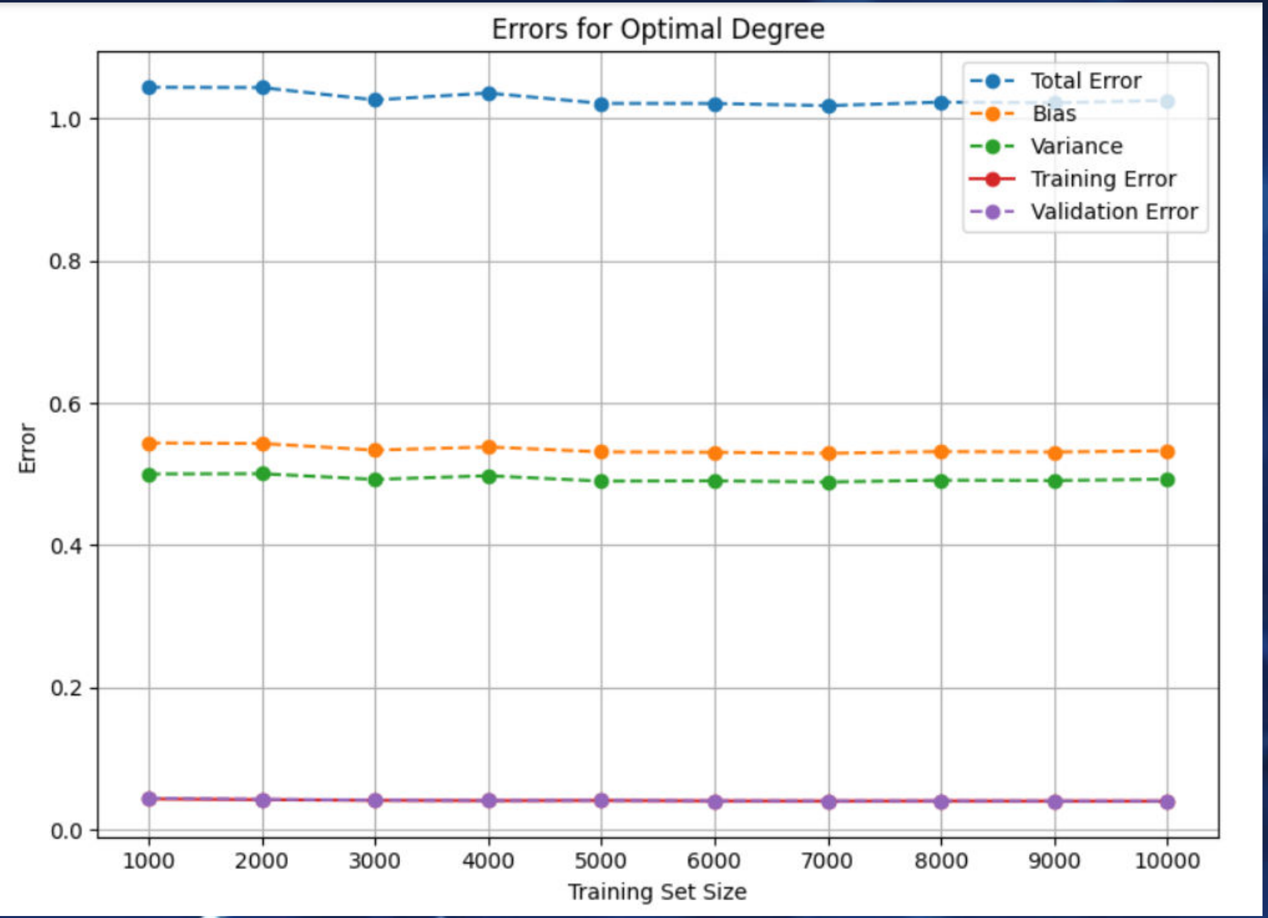
- Till now optimal degree is calculated i.e. 7, now the optimal polynomial of degree 7 is needed to be fitted.

- Now the bias, variance, test error, training error and total error is calculated each for 1k,2k,3k,.....,10k size dataset using 10-fold cross validation.

```
def fit_optimal_polynomial(dataset, degree, sizes):
    total_errors = []
    biases = []
    variances = []
    train_errors = []
    test_errors = []

    for idx, size in enumerate(sizes):
        partial_dataset = np.array(dataset[:size])  # Convert the dataset to a numpy array
        total_error, bias, variance, train_error, test_error = fit_polynomial(partial_dataset, [degree])
        total_errors.append(total_error[0])
        biases.append(bias[0])
        variances.append(variance[0])
        train_errors.append(train_error[0])
        test_errors.append(test_error[0])
    return total_errors, biases, variances, train_errors, test_errors
```

Optimal Degree Error Plotting

- Finally the error plotting for the optimal degree i.e. 7(in our case) is done.
- x-axis shows Training set size i.e. 1k, 2k, 3k,......,10k.
- y-axis shows error.



References

- Introduction to Machine Learning by Ethem Alpaydin.

Thank You...