UNIVERSITY OF DELHI
DEPARTMENT OF COMPUTER SCIENCE
Master of Science in Computer Science

JUNE 2023

MACHINE LEARNING

PROJECT

BIAS-VARIANCE TRADE OFF FOR MODEL SELECTION

Prepared by:
Akshita Gupta (04)
Jagriti Yadav (61)
Tanuja Bisht (52)

ACKNOWLEDGEMENT

We would like to express our special thanks to **Dr. Vasudha Bhatnagar** for giving us this opportunity to apply our theoretical knowledge to practical work. And we would also like to show our gratitude towards her for her guidance and support in completing our Project titled- Bias-Variance trade off for model selection.

INDEX

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.

INSIGHTS

Language	**Python**
Software	**Jupyter Notebook**
Libraries	**Numpy, Pandas, matplotlib, sklearn**

THEORY

BIAS

Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.

VARIANCE

Variance is the variability of model prediction for a given data point or a value which tells us spread of our data. Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before. As a result, such models perform very well on training data but has high error rates on test data.

BIAS/VARIANCE TRADEOFF

In the context of polynomial regression, as the order of the polynomial increases, small changes in the dataset cause a greater change in the fitted polynomials; thus variance increases. But a complex model on the average allows a better fit to the underlying function; thus bias decreases This is called bias/variance dilemma.

To decrease bias, the model should be flexible, at the risk of having high variance. If the variance is kept low, we may not be able to make a good fit to data and have high bias.

The ***optimal model*** is the one that has the best trade-off between the bias and the variance.

04

THEORY

UNDERFITTING AND OVERFITTING

Underfitting happens when a model unable to capture the underlying pattern of the data. Overfitting happens when our model captures the noise along with the underlying pattern in data.
If there is bias, this indicates that our model class does not contain the solution; this is underfitting. If there is variance, the model class is too general and also learns the noise; this is overfitting.

CROSS VALIDATION

In practice, the method we use to find the optimal complexity is crossvalidation. We cannot calculate the bias and variance for a model, but we can calculate the total error. Given a dataset, we divide it into two parts as training and validation sets, train candidate models of different complexities on the training set and test their error on the validation set left out during training. As the model complexity increases, training error keeps decreasing. The error on the validation set however decreases up to a certain level of complexity, then stops decreasing or does not decrease further significantly, or even increases if there is noise in the data. This corresponds to the optimal complexity level.

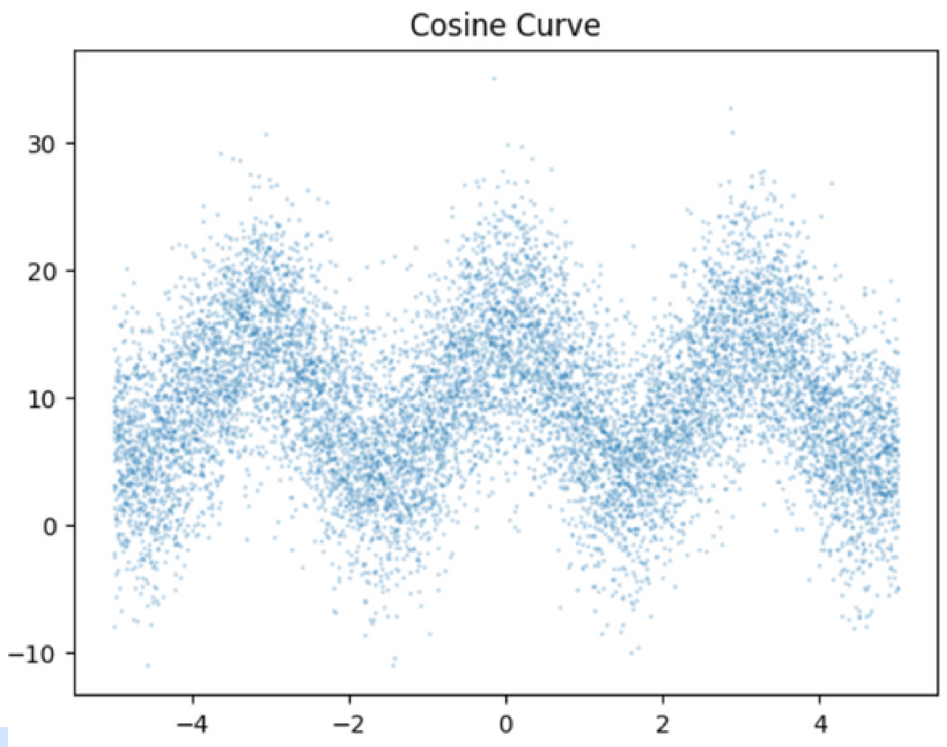
SOLUTION IN STEPS

Step 1:
Generated 10000 Random data points in a scattered plot using Numpy library following a Trigonometric function.

```
[ ]  import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import math
```

```
[ ]  n = 10000
     X = np.linspace(-5, 5, num=n)
     y = 5 * np.cos(2*X) +  10+5* np.random.normal(size=n)
```

```
[ ]  plt.scatter(X, y,s=1,alpha=0.2)
     # title for the sine curve
     plt.title('Cosine Curve')
     plt.show()
```

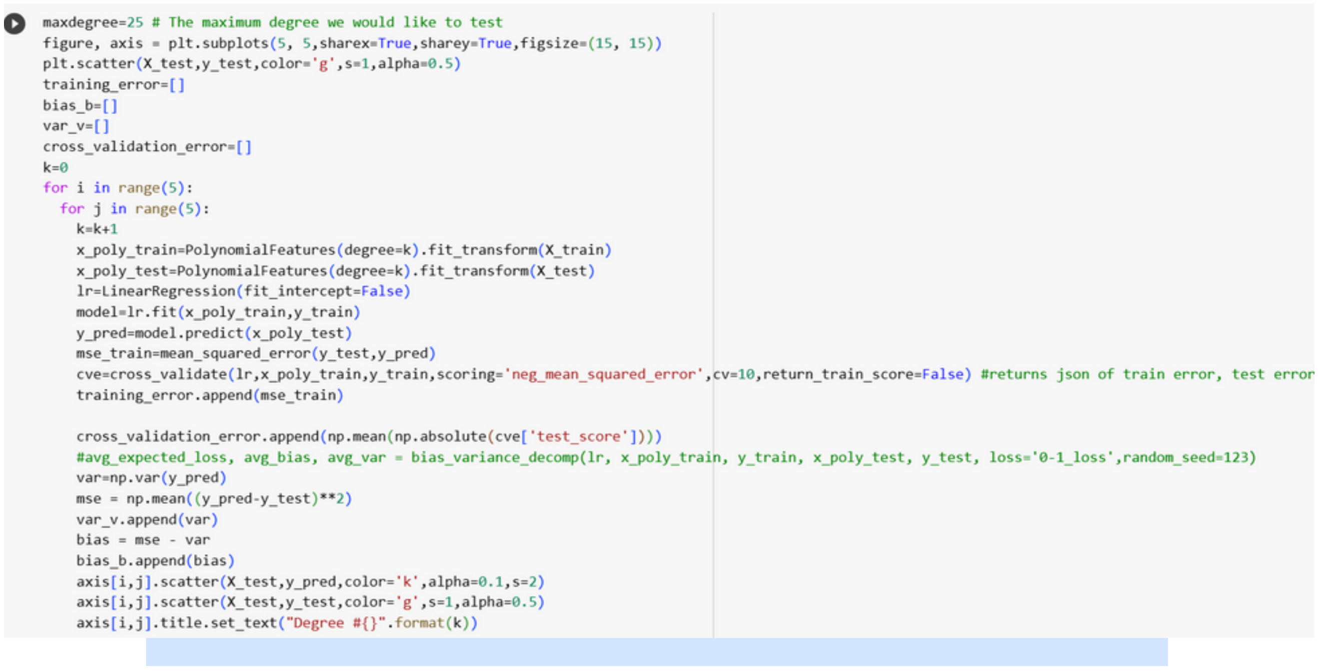


06

Step 2:
In this step we initialized 4 different arrays to store training error, bias, variance, cross validation errors in the following arrays respectively,

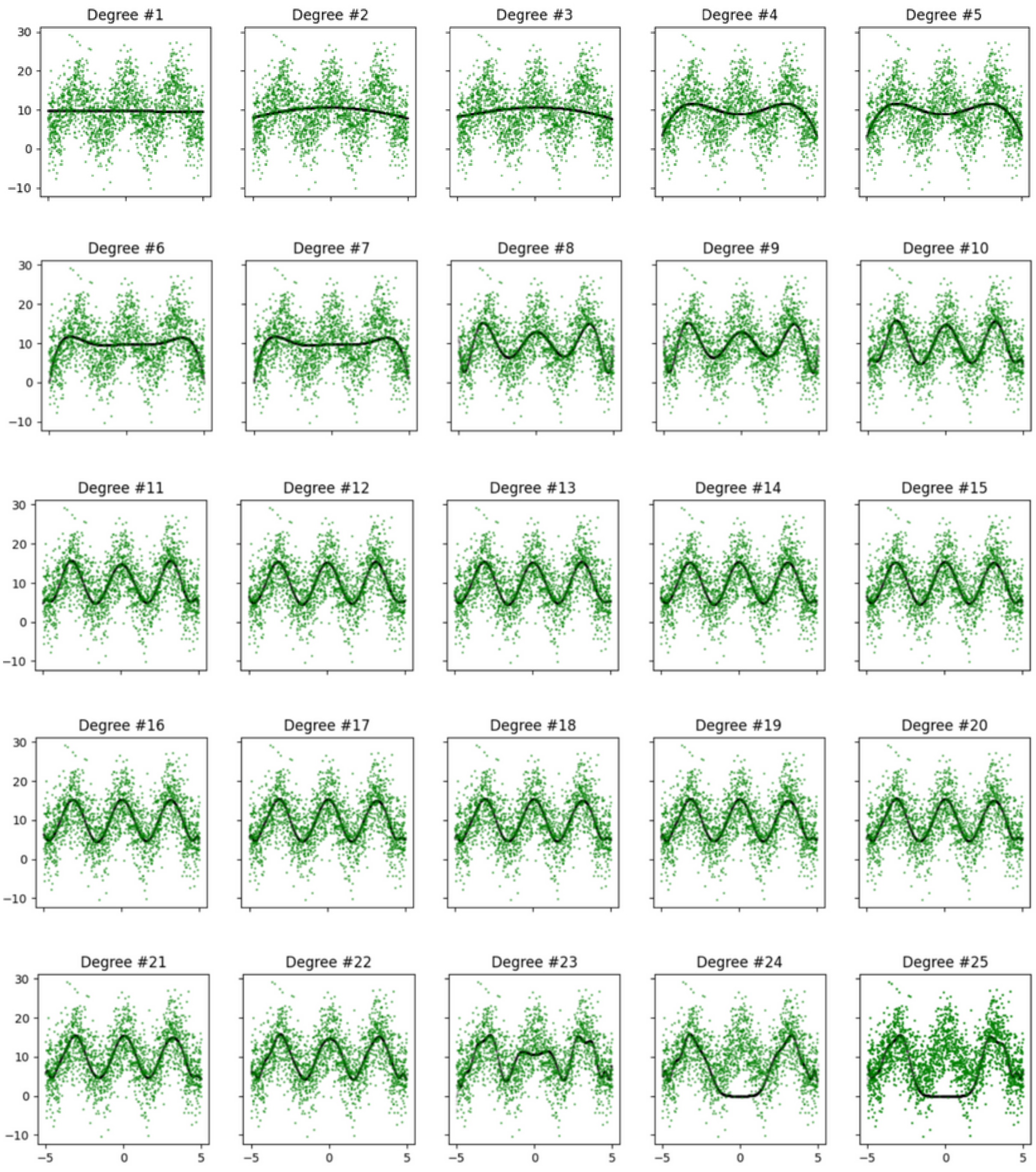
1. training_error=[]
2. bias_b=[]
3. var_v=[]
4. cross_validation_error=[]

We then automated the process of fitting different order polynomials over our dataset and cross validating each fitted model. We fitted 25 polynomial models, ranging from degree 1 to 25.
Statistic(s) used along with their formulas to compute:
Bias: E[ŷ - y] , where ŷ-> predicted target value and y -> true target value.
Variance: E[(ŷ - E[ŷ])^2] , where E[ŷ]) -> average of the predicted target values over all possible training sets.
Mean squared error: (1/n) * Σ(y - ŷ)^2 , where y -> true target values ŷ -> predicted target values.
Total error: average error or loss computed over multiple iterations of cross-validation.
We then plotted each polynomial model with their degrees in ascending order:



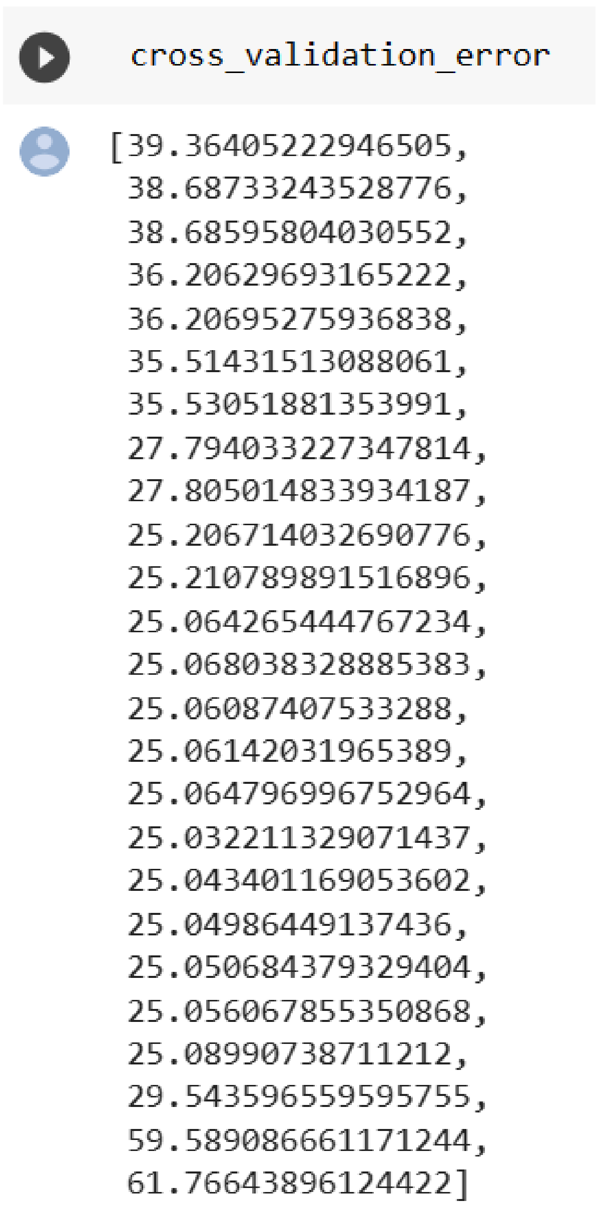
```
maxdegree=25 # The maximum degree we would like to test
figure, axis = plt.subplots(5, 5,sharex=True,sharey=True,figsize=(15, 15))
plt.scatter(X_test,y_test,color='g',s=1,alpha=0.5)
training_error=[]
bias_b=[]
var_v=[]
cross_validation_error=[]
k=0
for i in range(5):
  for j in range(5):
    k=k+1
    x_poly_train=PolynomialFeatures(degree=k).fit_transform(X_train)
    x_poly_test=PolynomialFeatures(degree=k).fit_transform(X_test)
    lr=LinearRegression(fit_intercept=False)
    model=lr.fit(x_poly_train,y_train)
    y_pred=model.predict(x_poly_test)
    mse_train=mean_squared_error(y_test,y_pred)
    cve=cross_validate(lr,x_poly_train,y_train,scoring='neg_mean_squared_error',cv=10,return_train_score=False) #returns json of train error, test error
    training_error.append(mse_train)

    cross_validation_error.append(np.mean(np.absolute(cve['test_score'])))
    #avg_expected_loss, avg_bias, avg_var = bias_variance_decomp(lr, x_poly_train, y_train, x_poly_test, y_test, loss='0-1_loss',random_seed=123)
    var=np.var(y_pred)
    mse = np.mean((y_pred-y_test)**2)
    var_v.append(var)
    bias = mse - var
    bias_b.append(bias)
    axis[i,j].scatter(X_test,y_pred,color='k',alpha=0.1,s=2)
    axis[i,j].scatter(X_test,y_test,color='g',s=1,alpha=0.5)
    axis[i,j].title.set_text("Degree #{}".format(k))
```

07



Degree #1
Degree #2
Degree #3
Degree #4
Degree #5
Degree #6
Degree #7
Degree #8
Degree #9
Degree #10
Degree #11
Degree #12
Degree #13
Degree #14
Degree #15
Degree #16
Degree #17
Degree #18
Degree #19
Degree #20
Degree #21
Degree #22
Degree #23
Degree #24
Degree #25
30
20
10
0
−10
−5
0
5

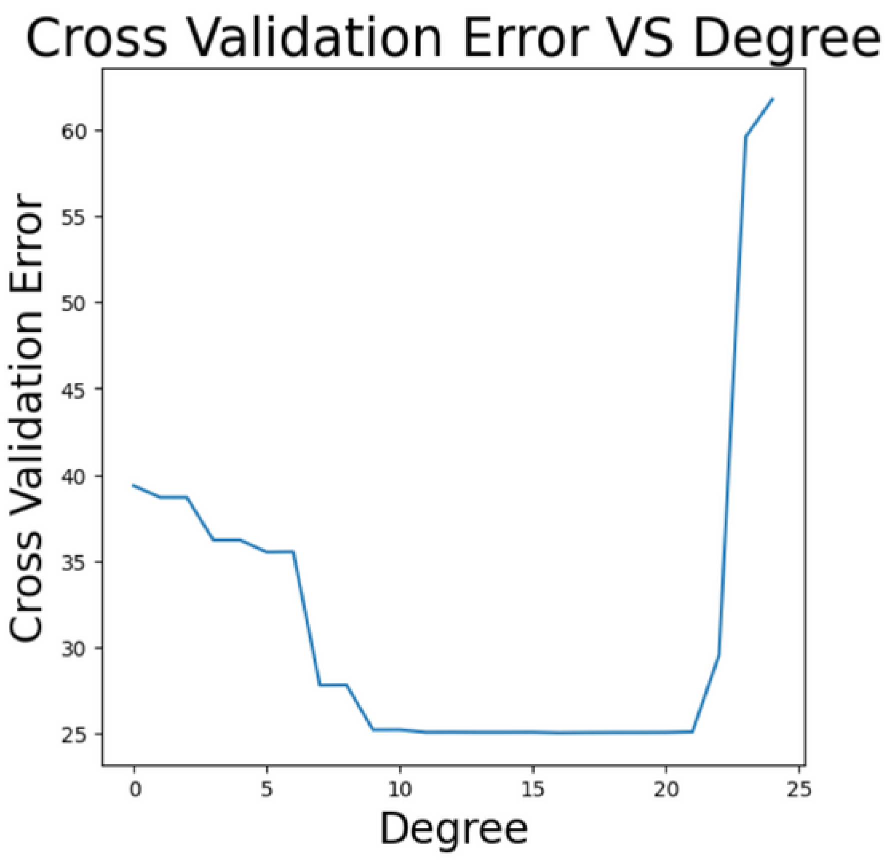
Step 3:
As we have mentioned in our theory part that "**As the model complexity increases, training error keeps decreasing. The error on the validation set however decreases up to a certain level of complexity, then stops decreasing or does not decrease further significantly, or even increases if there is noise in the data. This corresponds to the optimal complexity level.**".
We further illustrated this in our work, the cross validation error array and the corresponding graph presented below illustrates the same:

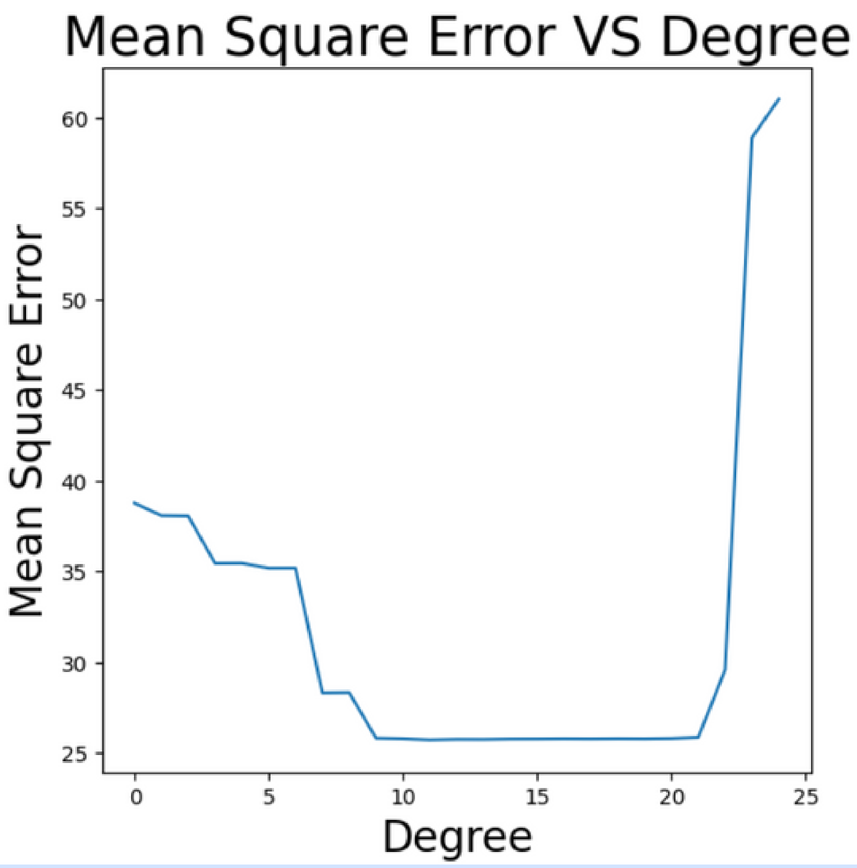


```
cross_validation_error
```

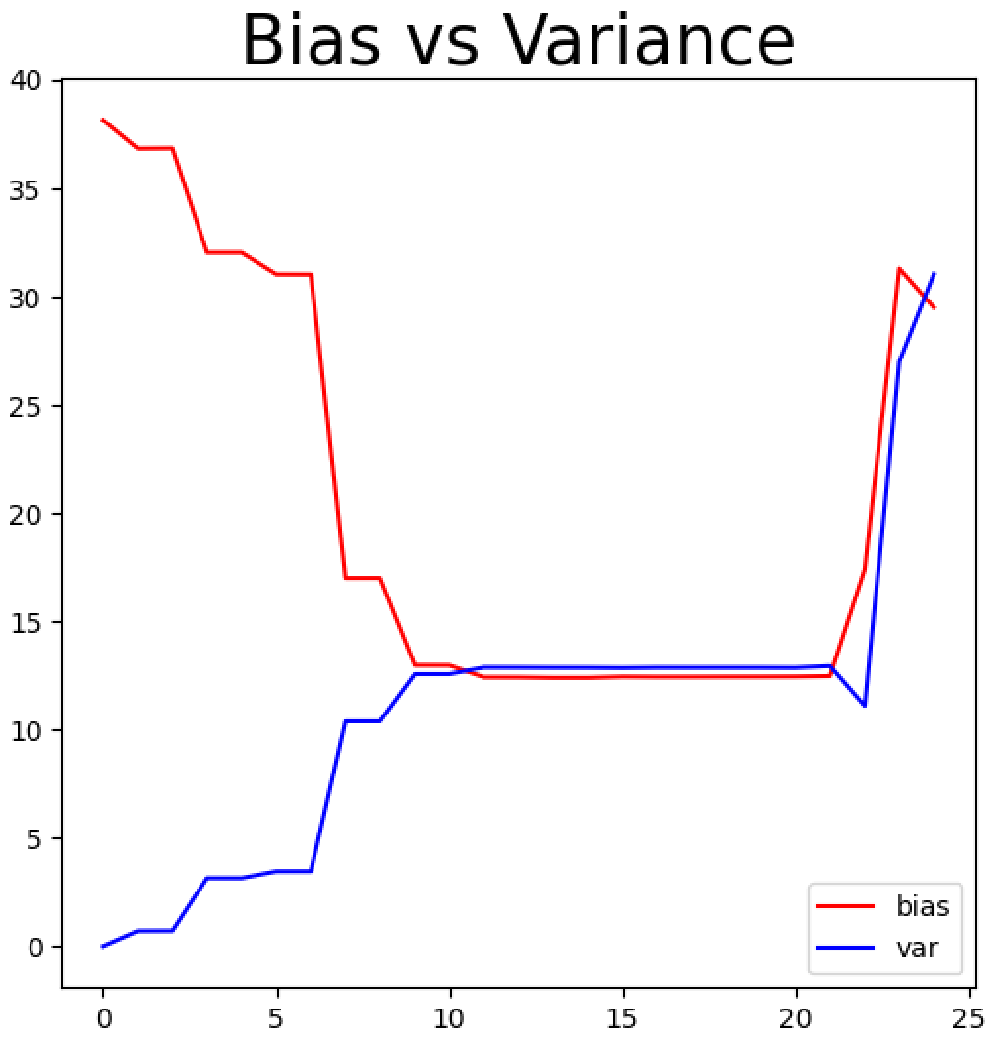
```
[39.36405222946505,
 38.68733243528776,
 38.68595804030552,
 36.20629693165222,
 36.20695275936838,
 35.51431513088061,
 35.53051881353991,
 27.794033227347814,
 27.805014833934187,
 25.206714032690776,
 25.210789891516896,
 25.064265444767234,
 25.068038328885383,
 25.06087407533288,
 25.06142031965389,
 25.064796996752964,
 25.032211329071437,
 25.043401169053602,
 25.04986449137436,
 25.050684379329404,
 25.056067855350868,
 25.08990738711212,
 29.543596559595755,
 59.589086661171244,
 61.76643896124422]
```

As we can see in the cross-validation error array, it first decreased from 39.3640 then became stagnant at ~25 for a range of degree, then spiked up to ~61.

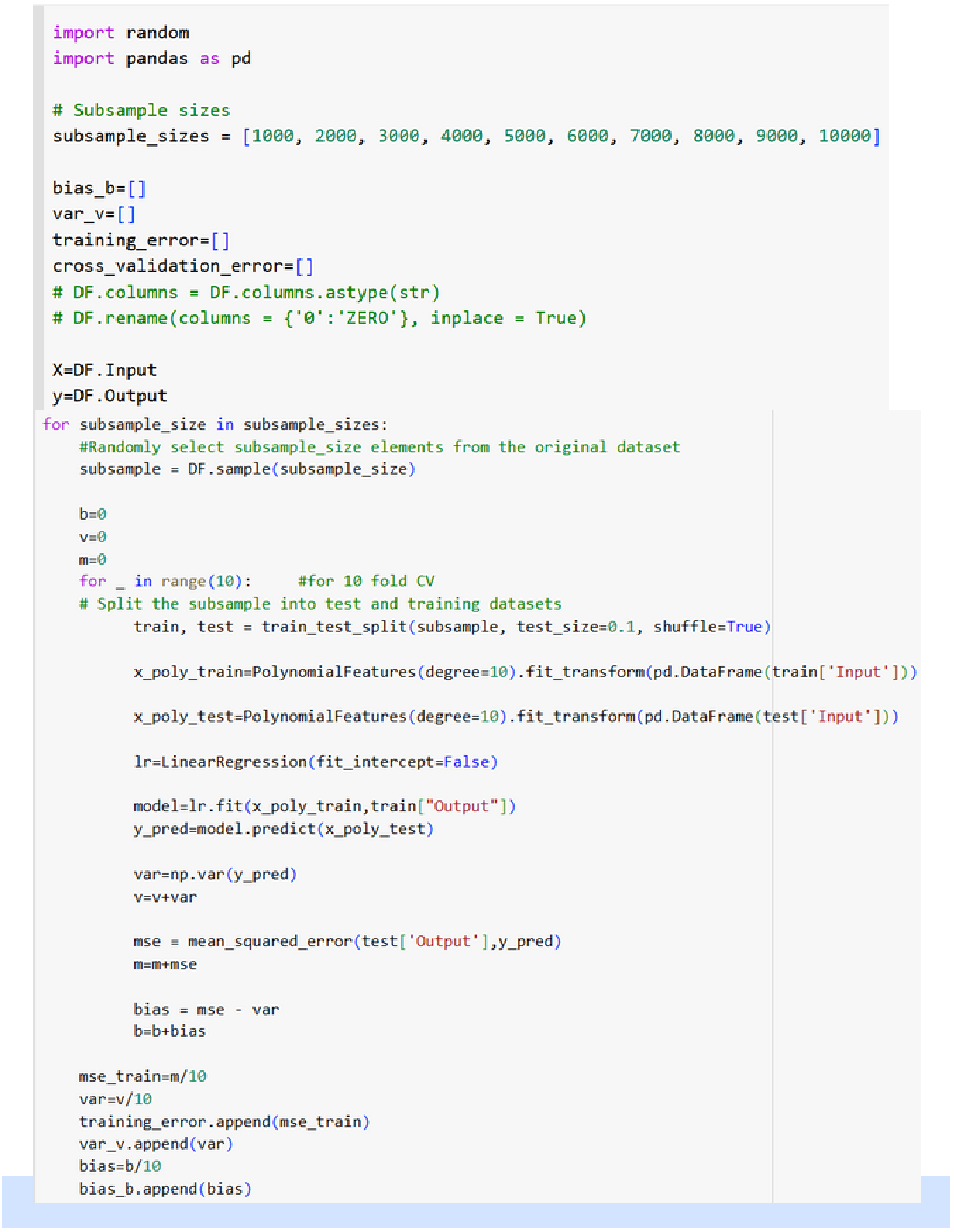

Cross Validation Error VS Degree
Cross Validation Error
Degree
60
55
50
45
40
35
30
25
0
5
10
15
20
25


Mean Square Error VS Degree
Mean Square Error
Degree
60
55
50
45
40
35
30
25
0
5
10
15
20
25

Step 4:
We further plotted the graph to represent the "BIAS-VARIANCE TRADEOFF" for all the degrees and we could from the graph mentioned below.
As we can observe that both bias and variance became stagnant after degree 10 then both spiked up.



Step 5:
We then selected the polynomial model of degree 10 which gave us a good fit and best trade-off (approximately) and with a lower complexity when compared with other models of higher complexity but with negligible improvement in error.
We then subsampled the data set into samples of sizes 1K, 2K, 3K,... 10K and applied cross-validation of each one of the samples and checked the various statistic for these sample folds too. This was automated too via a function.
Below is the graph for bias and variance for the last sample of size 10K.

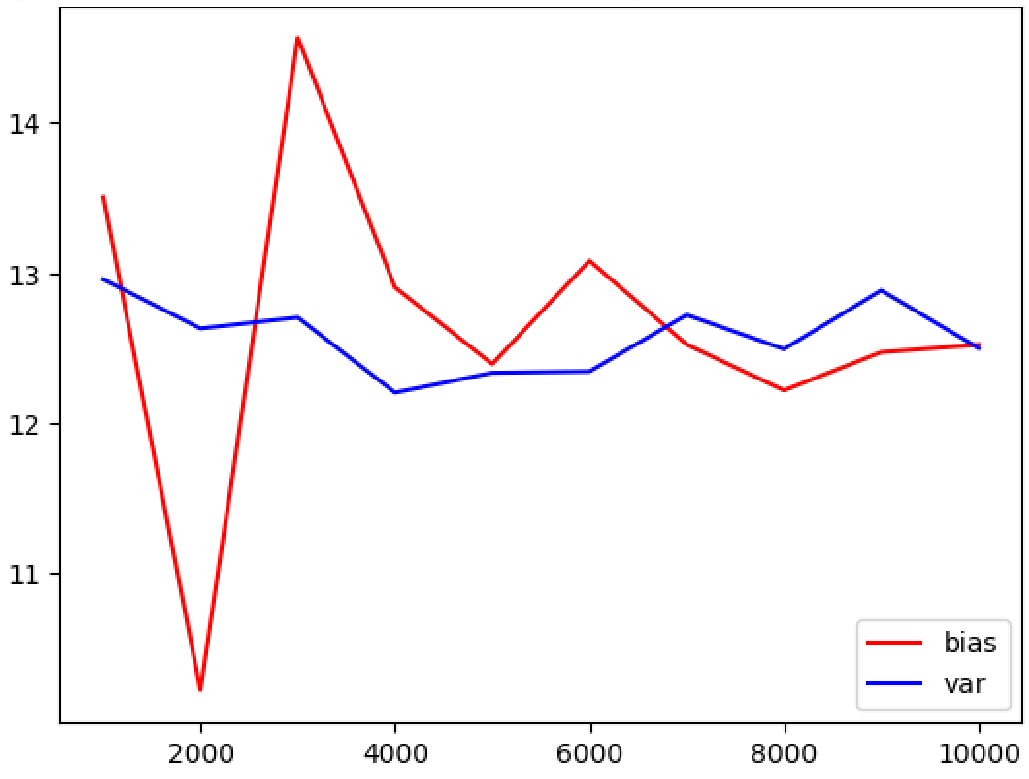


```
import random
import pandas as pd

# Subsample sizes
subsample_sizes = [1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000]

bias_b=[]
var_v=[]
training_error=[]
cross_validation_error=[]
# DF.columns = DF.columns.astype(str)
# DF.rename(columns = {'0':'ZERO'}, inplace = True)

X=DF.Input
y=DF.Output
```

```
for subsample_size in subsample_sizes:
    #Randomly select subsample_size elements from the original dataset
    subsample = DF.sample(subsample_size)

    b=0
    v=0
    m=0
    for _ in range(10):      #for 10 fold CV
    # Split the subsample into test and training datasets
        train, test = train_test_split(subsample, test_size=0.1, shuffle=True)

        x_poly_train=PolynomialFeatures(degree=10).fit_transform(pd.DataFrame(train['Input']))

        x_poly_test=PolynomialFeatures(degree=10).fit_transform(pd.DataFrame(test['Input']))

        lr=LinearRegression(fit_intercept=False)

        model=lr.fit(x_poly_train,train["Output"])
        y_pred=model.predict(x_poly_test)

        var=np.var(y_pred)
        v=v+var

        mse = mean_squared_error(test['Output'],y_pred)
        m=m+mse

        bias = mse - var
        b=b+bias

    mse_train=m/10
    var=v/10
    training_error.append(mse_train)
    var_v.append(var)
    bias=b/10
    bias_b.append(bias)
```

```
plt.plot(subsample_sizes,bias_b, color = 'r', label = "bias")
plt.plot(subsample_sizes,var_v, color = 'b', label = "var")
```

```
[<matplotlib.lines.Line2D at 0x7f8c1587cc70>]
```



CONCLUSION

We tried fitting linear and non-linear polynomial models. Our polynomials degrees ranged from order 1 to order 25 and we found out that the polynomial model of degree 10 gave us a satisfactory bias-variance trade off, hence we concluded that it is one of the optimal model for the given trigonometric function.

BIBLIOGRAPHY

1. External source
2. Introduction to Machine Learning by ETHEM ALPAYDIN