Assignment 2

Report

By Aryan Gupta,

Task 1

Understanding what function the method LinearRegression().fit() performs.

LinearRegression().fit()

Fit linear model.

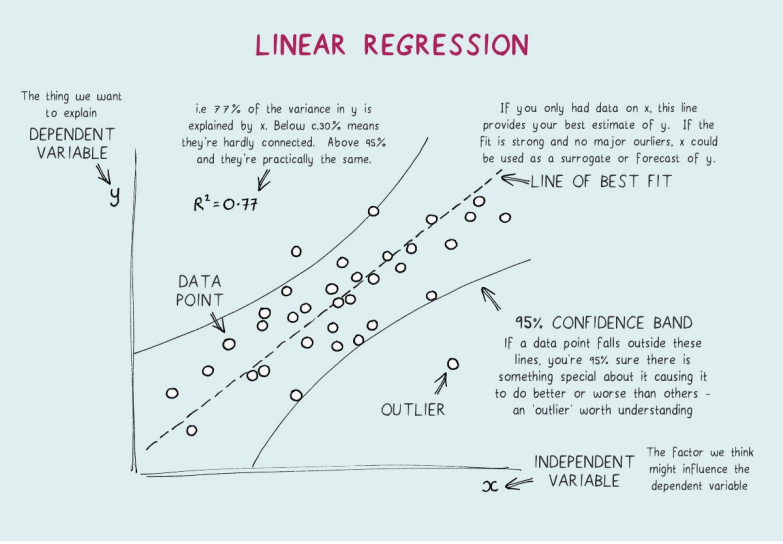
Parameters:	**X :** ***{array-like, sparse matrix} of shape (n_samples, n_features)*** Training data. **y :** ***array-like of shape (n_samples,) or (n_samples, n_targets)*** Target values. Will be cast to X's dtype if necessary. **sample_weight :** ***array-like of shape (n_samples,), default=None*** Individual weights for each sample. *New in version 0.17:* parameter *sample_weight* support to LinearRegression.
Returns:	**self :** ***object*** Fitted Estimator.

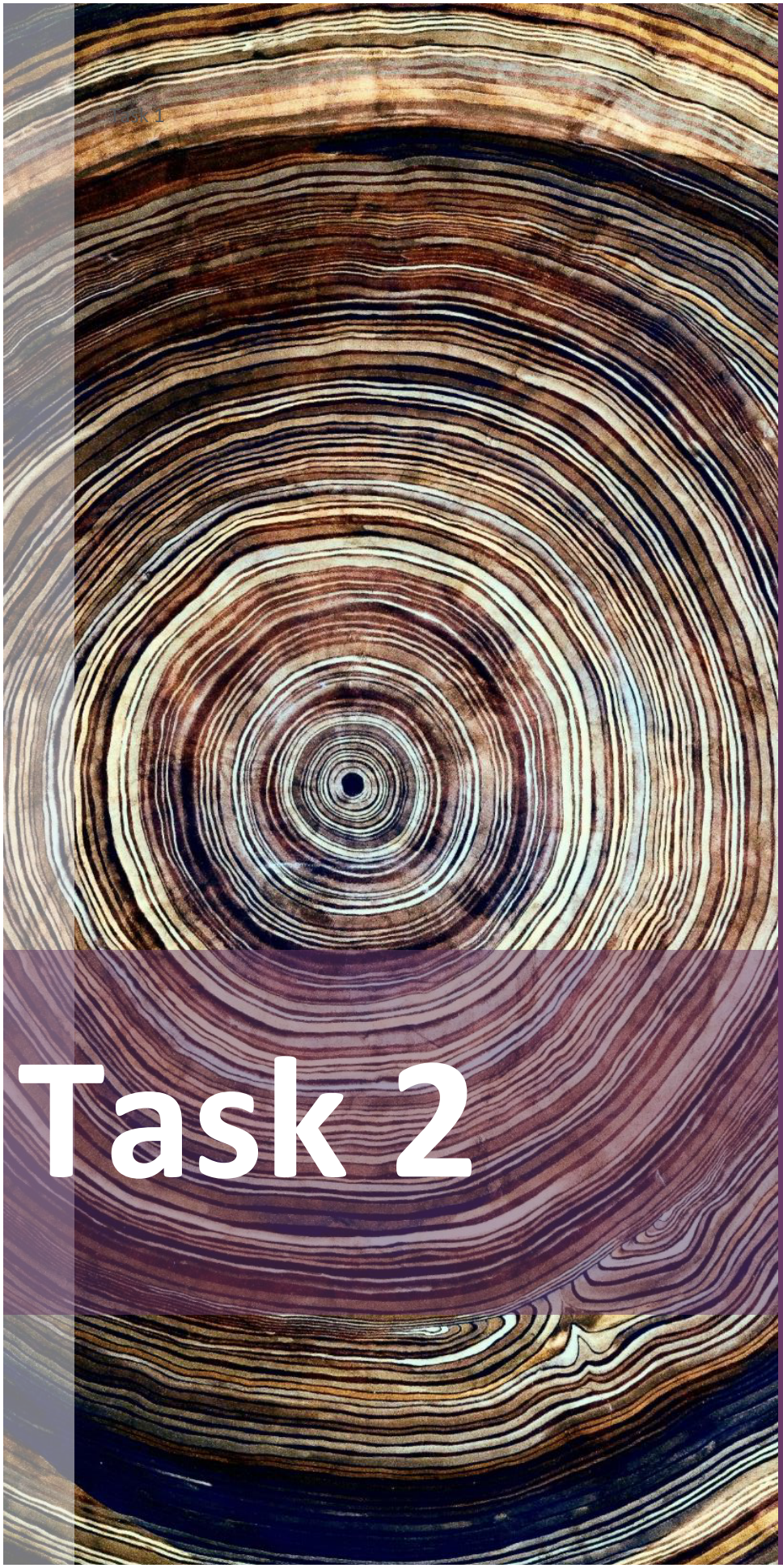
The linear regression model is trained on a given dataset using the fit() method. The target variable, which is represented as a 1D array-like object, and the input features or predictors, which are both represented as 2D array-like objects, are the two mandatory arguments this method requires when it is called. Once the coefficients defining the linear relationship between the input features and the target variable have been estimated, the method fits the linear regression model to the data.

LinearRegression() is a part of a class in the scikit-learn library, which is used for implementing linear regression

Fit() will teach the LinearRegression object the coefficients that minimise the sum of the squared residuals between the predicted values and the actual target values(MSE). By calling the predict() method of the LinearRegression object, these coefficients can then be used to make predictions on new data. It returns a straight line,

y = reg.coef_ * X + reg.intercept_





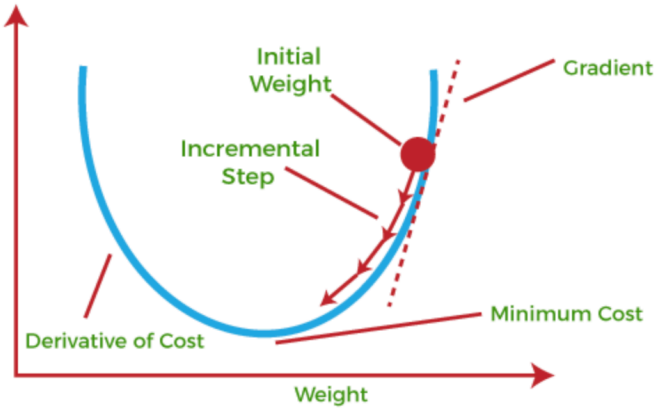
Task 2

Understanding Gradient Descent

The basic idea of gradient descent is to iteratively adjust the model parameters (i.e., coefficients) in the direction of steepest descent of the cost function. The cost function is a measure of the error between the predicted and actual values of the dependent variable, and the goal is to minimize it.

Let's understand with an example!

Gradient descent is an optimization algorithm commonly used in machine learning to find the optimal parameters of a model.



To begin with the process let's assume

$F(x, y) = ax + bx^2$;

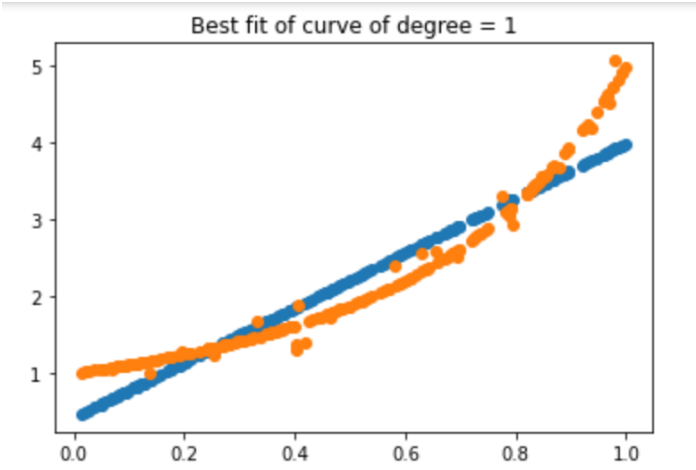
Now we must find the ideal value of a & b. To do so we first Start with some initial values for the coefficients (e.g., 0 or a random value). Then we measure the error between the predicted and actual values of the dependent variable for the current set of coefficients. Take the partial derivates wrt each term and then walk towards the minimization of the error. We repeat this process till we gain satisfactory coefficients.

Gradient descent is an iterative algorithm that adjusts the coefficients of a model to minimize the cost function by taking small steps in the direction of steepest descent. This is done by calculating the gradients of the cost function with respect to the coefficients and adjusting them by subtracting a fraction of the gradient in each iteration.

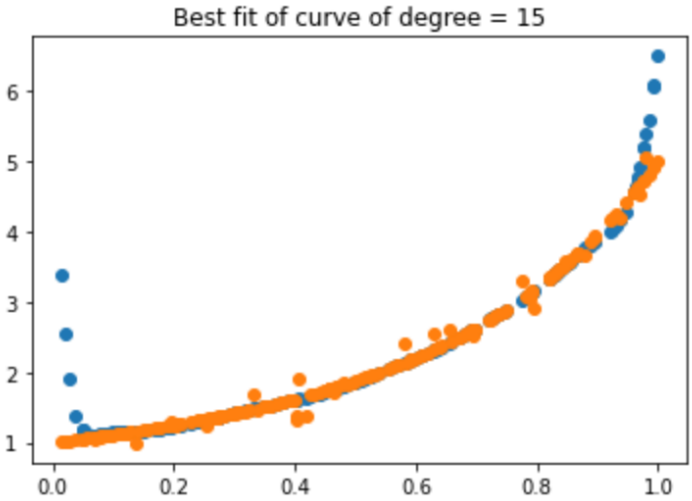
Degree	MSE	Mean of bias square	Mean of bias	Mean of variance
1	0.123073	0.114392	0.269398	0.00868095
2	0.0133656	0.0121412	0.0862565	0.00122436
3	0.00504545	0.00470811	0.0332718	0.000337339
4	0.0046061	0.0042391	0.0242826	0.000366999
5	0.00465952	0.00419759	0.0238793	0.000461938
6	0.00477993	0.00419841	0.0239554	0.00058152
7	0.00510317	0.00418638	0.02483	0.000916796
8	0.00602264	0.00426172	0.0248874	0.00176092
9	0.0131353	0.00485845	0.0304184	0.00827683
10	0.0109153	0.00441441	0.0286639	0.00650085
11	0.0390896	0.00639682	0.0365903	0.0326928
12	0.941428	0.0569339	0.0709172	0.884494
13	0.0618877	0.00757067	0.0422491	0.054317
14	8.48914	0.445785	0.156428	8.04336
15	2.74943	0.0799844	0.089127	2.66945

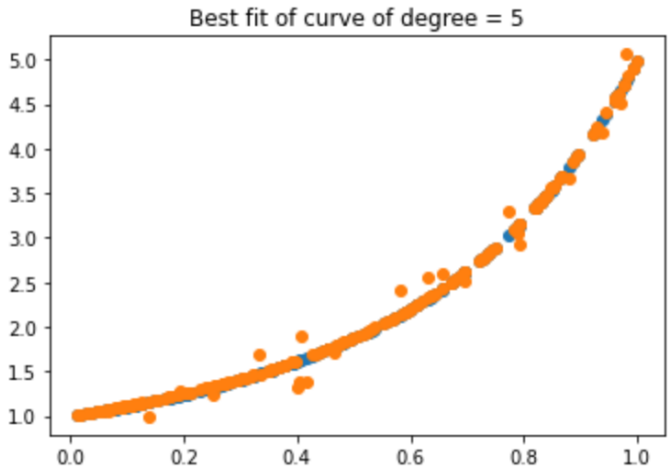
(the following code was written assuming the data set provided to be of the same length & can easily be modified to act on any given dataset)

We start with re-sampling the loaded data by re-shuffling it, we then divide the data into 20 equal random parts for training of our model. Then we proceed onto calculating the mean of bias and variance of the model over all the test points. Upon plotting the plots for our functions of predicted Y of degree(i) vs The real points it becomes very evident that:



For the lower degrees, we clearly see there is a case of underfitting, High bias and low variance can be also seen in the table, whereas the opposite happens in the case of overfitting which is observed in higher powers.





Visually we can also see that around degree 4/5/6 we can see the most amount overlap between the predicted and actual values, which can also be seen via the bias variance tradeoff why such a model could be the best fit for us. Bias refers to the model's tendency to systematically underfit or overfit the data, while variance refers to the model's sensitivity to small fluctuations in the data.

```python
#TASK 4
irr = []
for i in range(15):
    irr.append(mse[i] - (bias[i]**2 + var[i]))
    print("for degree "+str(i)+" the minimum irreducible error is "+str(irr[i]))
```

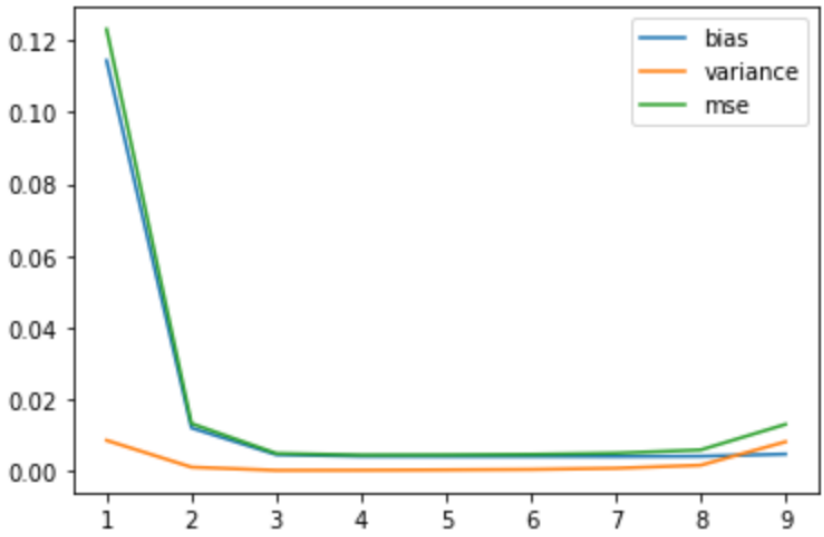
```
for degree 0 the minimum irreducible error is 0.041816754620283086
for degree 1 the minimum irreducible error is 0.004701038218762955
for degree 2 the minimum irreducible error is 0.0036010994177978627
for degree 3 the minimum irreducible error is 0.0036494537307899815
for degree 4 the minimum irreducible error is 0.003627363978191657
for degree 5 the minimum irreducible error is 0.003624545713158406
for degree 6 the minimum irreducible error is 0.00356984935645949
for degree 7 the minimum irreducible error is 0.003642341038114985
for degree 8 the minimum irreducible error is 0.003933167384490701
for degree 9 the minimum irreducible error is 0.0035927927968845584
for degree 10 the minimum irreducible error is 0.005057972077471454
for degree 11 the minimum irreducible error is 0.05190461526946233
for degree 12 the minimum irreducible error is 0.00578569102076143
for degree 13 the minimum irreducible error is 0.42131489127553934
for degree 14 the minimum irreducible error is 0.07204082150183533
```

We obtain a significant irreducible error for a linear function because of the extremely high bias, moderately high variance, and extremely high MSE. This error settles after degree=2 and remains essentially constant until degree=12. Due to the high variance and MSE introduced in these models, irreducible error rises as degree increases.

Since it represents the smallest error that can be obtained even if the model perfectly fits the data, the value of irreducible error does not change significantly as the polynomial degrees are increased. This error is a result of the data itself and is independent of the modelling approach or level of model complexity.

In [312]:
```
# Task 5

x_axis = range(1,10)
y_0 = biassq[0:9]
y_1 = var[0:9]
y_2 = mse[0:9]
plt.plot(x_axis,y_0)
plt.plot(x_axis,y_1)
plt.plot(x_axis,y_2)
plt.legend(['bias','variance','mse'])
plt.show()
```



- For degrees 1&2 we notice a very high bias, therefore underfitting.
- For degrees 8 and beyond we notice a very high bias, therefore overfitting.
- We can deduct from the plot that bias^2 and variance are at their best for 3 degrees: 4 ,5 and 6. The MSE is also the lowest among these degrees. As a result, we can say that the type of data presented can be fitted to polynomial functions of fourth, fifth & sixth degrees.