BIAS AND VARIANCE IN MACHINE LEARNING

The risk in following ML models is they could be based on false assumptions and skewed by noise and outliers. That could lead to making bad predictions. That is why ML cannot be a black box. The user must understand the data and algorithms if the models are to be trusted. So, here, we look at some more measures of trustworthiness.

As in the previous blog post, the way to flush out these errors is by looking at the different metrics associated with the ML model and understanding something about the data and which models work best for which types of data. You then pick different training models to find the one that has the **lowest bias versus variance tradeoff**. In an ideal world you would seek to minimize bias and variance. But as the word **tradeoff** suggests, its not that simple.

What is bias in machine learning?

Bias is the same as the mean square error (MSE).

- **Variance** shows how subject the model is to outliers, meaning those values that are far away from the **mean**.
- **Noise** is the unexplained part of the model. In terms of statistics, noise is anything that results in inaccurate data gathering, such as using measuring equipment that is not properly calibrated.

Models with high bias tend to underfit the training data while those with high variance tend to overfit:

- **Overfit** means the model is subject to outliers and noise.
- **Underfit** means the model could look at other inputs (i.e., additional features).

Models with low bias can be subject to noise.

Examples of bias and variance

Let's look at three examples. Here we take the same training and test data. The only difference is we use three different linear regression models (**least squares, ridge**, and **lasso**) then look at the bias and variance of each. (Here is the code in Zeppelin notebook format.)

First, to summarize we have these results. Below is the code and the graphs.

algorithms	coefficient	bias	variance
least squares	2	13.88	101.15
ridge	1.6	33.11	494.27
lasso	1.6	17.46	154.25

Which algorithm is best? Each has their own bias, which is why the logic in each is coded in a manner to drive out the sensitivity to outliers, noise, and flag which models are over or underfit.

You should have some understanding of what kind of data you are dealing with and hot the algorithm works. For example, are there many outliers or noise in your data. Is the model you have picked a higher order polynomial (i.e. a curve instead of a line) which can overfit the training data, meaning track it too closely, in a way that is not natural. Finally, you would need to understand the differences between each algorithm. (Here is some guidance from scikit-learn on choosing the right model.)

Anyway, below we use the same code as in the last blog post with some changes. As you can see we have made the x features and y labels perfectly correlate. That is reflected in the least square coefficient of 2, meaning for each x, f(x)=2x.

The data is the same, and the model is linear in all three cases. The only difference is the slope of the line, i.e., the **coefficient**. So all three graphs are going to look about the same.

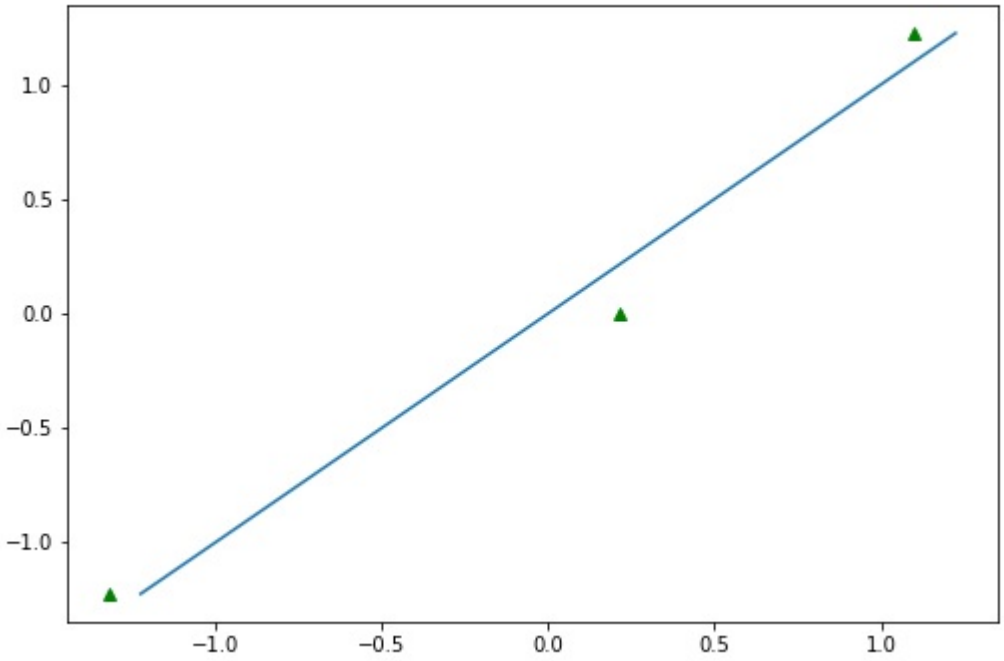
Notice that we normalize the data with **preprocessing.scale** so that the will have the same scale so that the graphs of the prediction model and the data points will roughly coincide.

```
import matplotlib.pyplot as plt
from sklearn import linear_model
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn import preprocessing

def newsample(xTest, ytest, model):
    ar = np.array(,,], ,,]])
    y = ar
    x = ar

    if model == 1:
        reg = linear_model.LinearRegression()
        reg.fit(x,y)
        print('least square Coefficients: \n', reg.coef_)
```

```
    if model == 2:
        reg = linear_model.Ridge (alpha = 0.1)
        reg.fit(x,y)
        print('ridged Coefficients: \n', ridge.coef_)
    if model == 3:
        reg = linear_model.Lasso(alpha = 0.1)
        reg.fit(x,y)
        print('lasso Coefficients: \n', ridge.coef_)
    preds = reg.predict(xTest)

    er = []
    for i in range(len(ytest)):
        print( "actual=", ytest, " preds=", preds)
        x = (ytest - preds) **2
        er.append(x)

    v = np.var(er)
    print ("variance", v)
    print("Mean squared error (bias): %.2f" %
mean_squared_error(ytest,preds))

    tst = preprocessing.scale(ytest)
    prd = preprocessing.scale(preds)
    plt.plot(tst, prd, 'g^')
    x1 = preprocessing.scale(xTest)
    fx = preprocessing.scale(xTest * reg.coef_)

    plt.plot(x1,fx )
    plt.show()


a = np.array(,,])
b = np.array(,,])
newsample(a,b, 1)
```

Results in:

```
least square Coefficients:
 ]
actual=   preds=
actual=   preds=
actual=   preds=
variance 101.14880000000011
Mean squared error (bias): 13.88
```

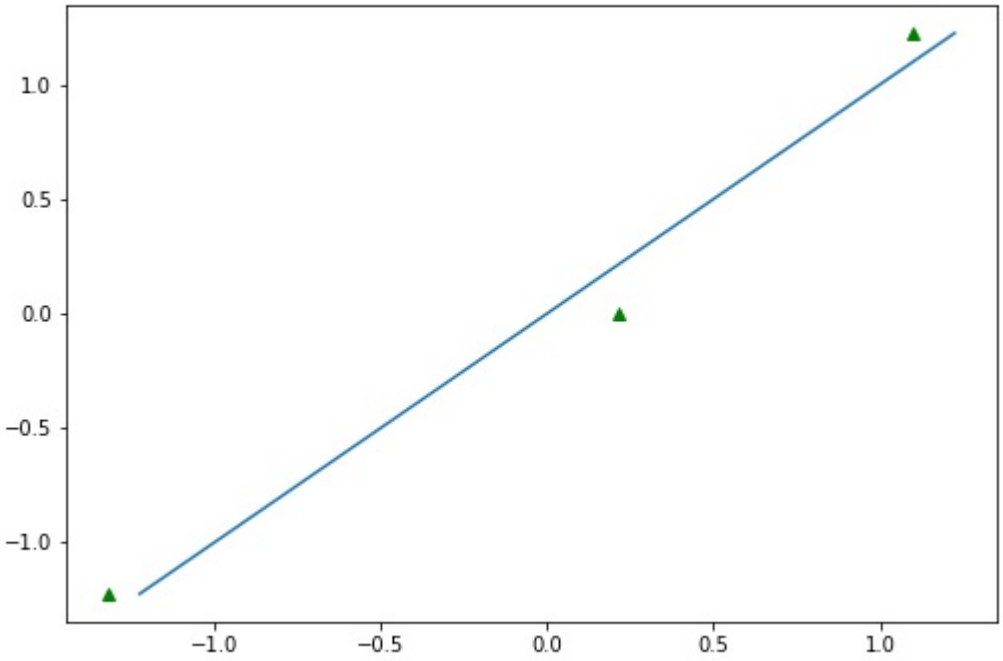


Run the program again, but this time use the Ridge algorithm.

```
newsample(a,b, 2)
```

Results in:

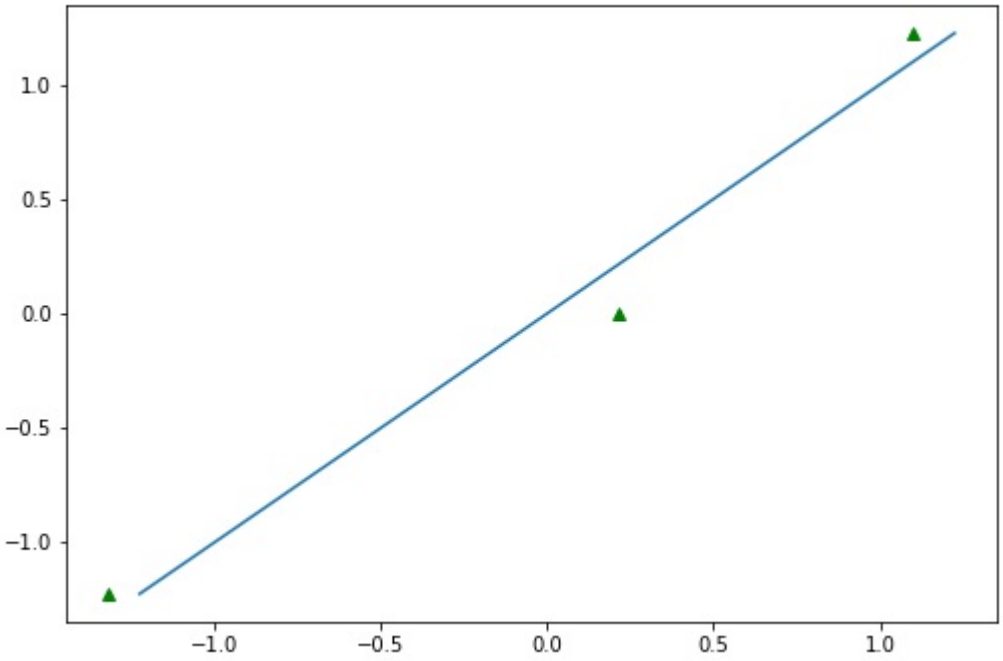
```
ridged Coefficients:
 ]
actual=   preds=
actual=   preds=
actual=   preds=
variance 494.2656834019206
Mean squared error (bias): 33.11
```



Run with the Lasso algorithm:

```
newsample(a,b, 3)
```

```
lasso Coefficients:
 ]
actual=   preds= 7.7
actual=   preds= 9.55
actual=   preds= 11.400000000000002
variance 154.25326249999975
Mean squared error (bias): 17.46
```


1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0