COM S 474/574 Spring 2020
Introduction to Machine Learning

Homework 1

Directions:

- **Due: Sunday February 16, 2020 at 10:00pm** Late submissions will be accepted for 12 hours after that time, with a 15% penalty. (the enforcement is strict, beginning at 10:01pm, except for extreme situations; having a poor wifi connection or minor computer problems is not sufficient for the penalty to be waived.)
- Upload the homework to Canvas as a pdf file. Written responses must be typed. Text should be Times New Roman font size 12 or similar size in other fonts.

  Unless specified otherwise, plots should be computer generated (make sure axis ranges are appropriate, tick marks and labels are legible, etc.). You can use Microsoft Word (or similar) or latex, then convert to pdf.
- Any non-administrative questions must be asked in office hours or (if a brief response is sufficient) Piazza.
- We recommend using Python with the machine learning library scikit-learn. Anaconda packages that library and other useful ones (like Matplotlib, Numpy, Statsmodels, and Pandas) together with both Spyder (a nice IDE similar to Matlab or RStudio) and Jupyter notebook. We recommend getting Python 3.7:

https://www.anaconda.com/distribution/

Problems

Problem 1. [60 points] For this problem, you will model feature *y* using polynomials of *x* up to order 30,

$$y = \sum_{i=0}^{30} a_i x^i.$$

In addition to looking at issues like training error, test error, overfitting, and so on, we will also explore how the amount of data (# of samples *n*) matter.

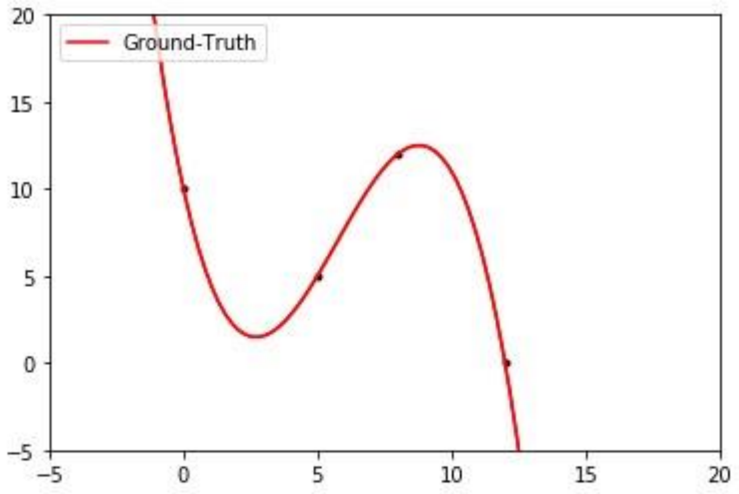
(a). First, let's generate a *ground-truth* model that the data comes from for reference (in practice we usually won't have this, but this will help us gain intuition). Recall that if we have $p + 1$ data points, we can fit that perfectly with a pth order polynomial. Fit a third order polynomial to the following four points

$$(0,10) \quad (5,5) \quad (8,12) \quad (12,0).$$

You can fit it using linear regression (such as with scikit-learn https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

and treating x^0, x^1, x^2, and x^3 as separate covariates.

(b). Make a plot of those points (big black circles) and the fit polynomial (red curve) with an x range of $[-5,20]$ and a y range of $[-5,20]$ to confirm the fit.



(c). The scikit-learn built-in linear regression function (and most built-in functions for other languages) use mean square error (MSE) as the only, or at least default, total-loss function. In 1-3 sentences, explain why the loss-function matters or not *for that fit specifically*. In other words, if you used a different total-loss function in part (a)., like mean absolute error or a weighted average of an asymmetric loss function, would you have gotten a different polynomial? Since the polynomial fits the four data points perfectly, it does not matter for this particular polynomial and data points it is fitting. This is because the residuals are all 0 meaning this polynomial would have 0 total loss for any other loss function (since any combination of 0's is 0).

(d). Now let's generate some data using that polynomial.

- First, there might be some distribution to the x values in the data. For this problem, we'll use the uniform distribution over the interval $[0,15]$. Numpy has a function for generating uniform random variables:

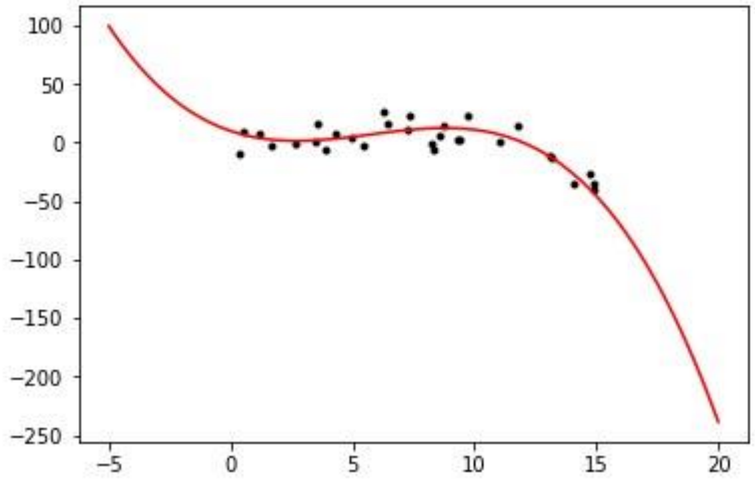
https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy. random.uniform.html

Generate n = 30 random variables. Each of these is an x value.

- Now for the y values. Generate y using the polynomial you fit in part (a). *plus* independent and identically distributed noise. Use the normal distribution, N(0,10),

  https://numpy.org/devdocs/reference/random/generated/numpy.random.Generator.normal.html#numpy.random.Generator.normal

- Now make a plot of the fit polynomial (red) and the n = 30 data points (black circles) you generated. The data should follow the curves of the polynomial but be scattered about it.

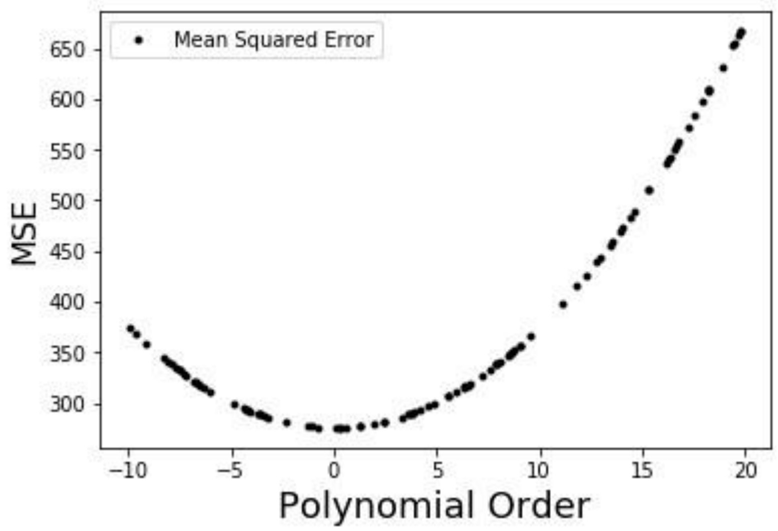


(e). We are now ready to start fitting models. First, let's look at how well constant models $y = a_0$ fit the data under different loss functions. Make a plot with a_0-values as the horizontal axis, ranging from [−10,20] with 100 values linearly spaced, and the totalloss along the vertical axis, for each of the total-loss functions

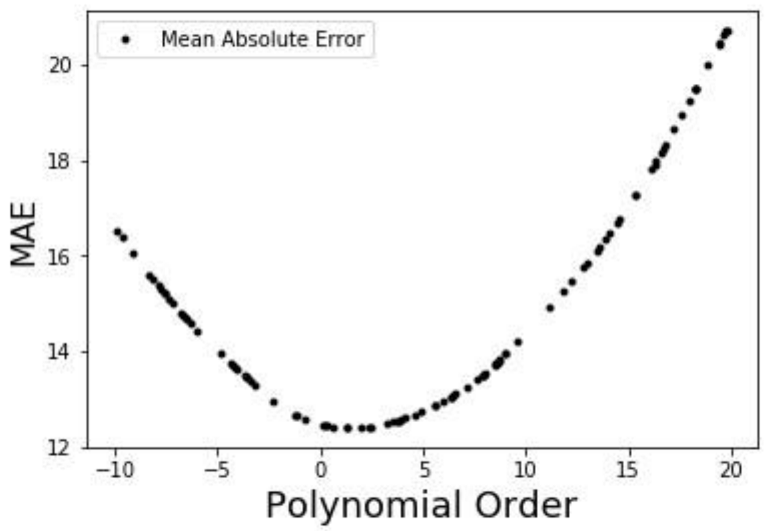
- mean squared error $\frac{1}{n}\sum_{i=1}^{n} |res(i)|^2$ (MSE)
- mean absolute error $\frac{1}{n}\sum_{i=1}^{n} |res(i)|^1$ (MAE)
- a special total-loss $\sum_{i=1}^{n} \frac{1}{|x(i)-5|+0.01} l(res(i))$ function where $x(i)$ is the x-feature value of sample i and the loss function $l(\cdot)$ is

$$l(res) = \begin{cases} -\frac{1}{5}res & \text{if } res < 0 \\ 10res & \text{if } res \geq 0 \end{cases}.$$

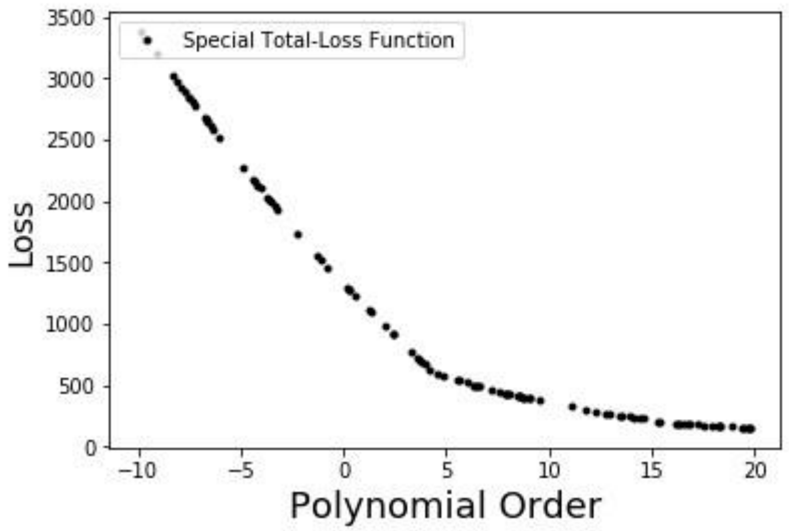
where $res(i) = y(i) - y(i)$ denotes the residual of the ith sample.



Mean Squared Error
MSE
Polynomial Order



Mean Absolute Error
MAE
Polynomial Order



Special Total-Loss Function
Loss
Polynomial Order

(f). In 2-4 sentences, explain

i. does the loss function $l(\cdot)$ prefer over-estimating or under-estimating y values?

The loss function l(.) prefers overestimating.

ii. does the loss function put more emphasis or less emphasis on points close to $x = 5$?

Yes, anything less than 5 gets weighed harder(higher loss)

(g). Calculate the mean and the median of the y-values for the data. Do you notice anything special about MAE and MSE minimizers? Explain in 2-5 sentences why that makes sense (or not) based on how residual magnitudes get penalized. Alternatively, you can use calculus to prove that they make sense (or not); i.e. you can solve for the minimizers.

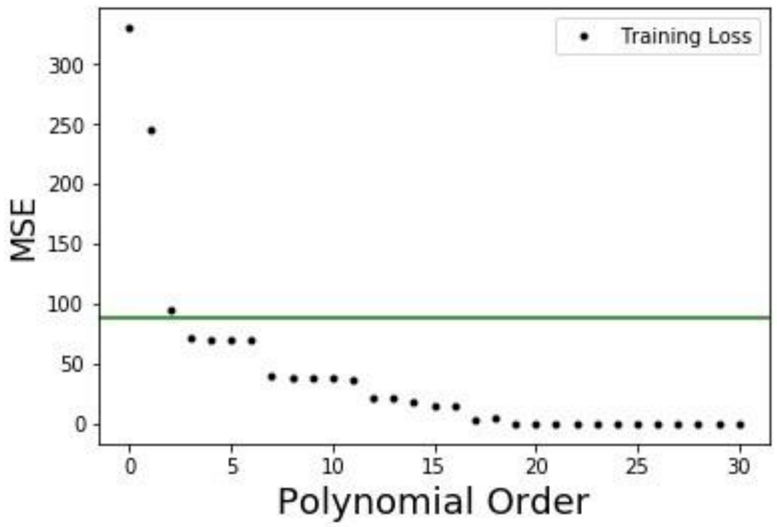
Mean: 0.0044596: The mean of the data would minimize the MSE. This can be seen when the curve is at its lowest.

Median: 1.7465 The median minimizes the sum of the absolute deviations. This can be seen when the curve is at its lowest.

(h). Now let's look at higher-order polynomials. Treating each x^i as a different covariate, we will fit the data with polynomials from order 0 to order 30. Caution: Some regression functions return wrong answers when the number of coefficients approaches the number of samples. If your results make sense for polynomials up to 10 or so, but do not make sense for higher orders, that's fine, submit what you have. Since MSE is the built-in total-loss function, we will use that (though keep in mind we could use others if someone had already coded the fitting procedure).

i. First, to evaluate over-fitting, we will use validation. Use the first 20 samples for fitting, reserving the rest as a validation set.

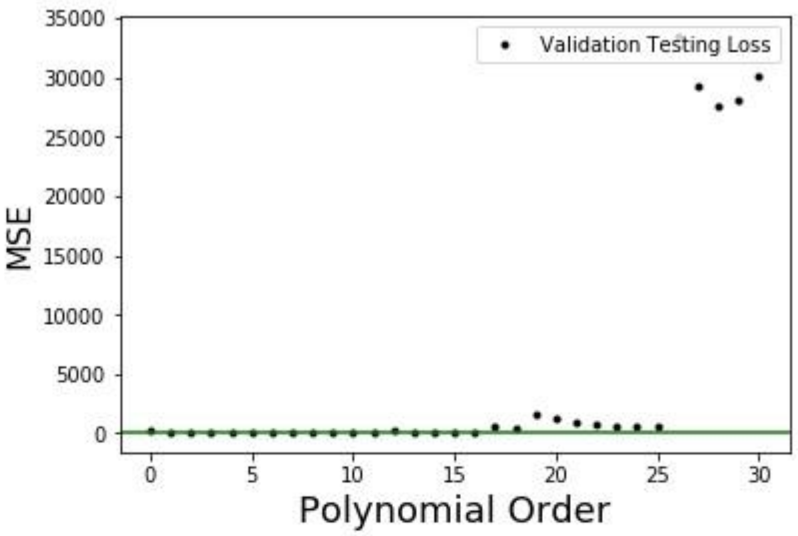
ii. Make a plot with title "Training Loss" plotting the MSE for the 20 samples used to fit for each polynomial for $p = 0$ to $p = 30$. In this plot, include a green horizontal line for the ground-truth model's MSE on the training data.



iii. In 2-5 sentences, comment on how the training loss curve behaves, such as where it has a steep slope, where it has a shallow slope, where it goes to 0. Do not simply describe the plot, but explain why it is that way, using knowledge of the ground-truth model.

The plot goes down steeply until around 15th order polynomial, then gradually is sinking towards 0. As previously stated, if there is $p + 1$ data points, they can be fit perfectly with a pth order polynomial. Because of this, a polynomial order of close to 19 or above should be approaching 0 MSE.

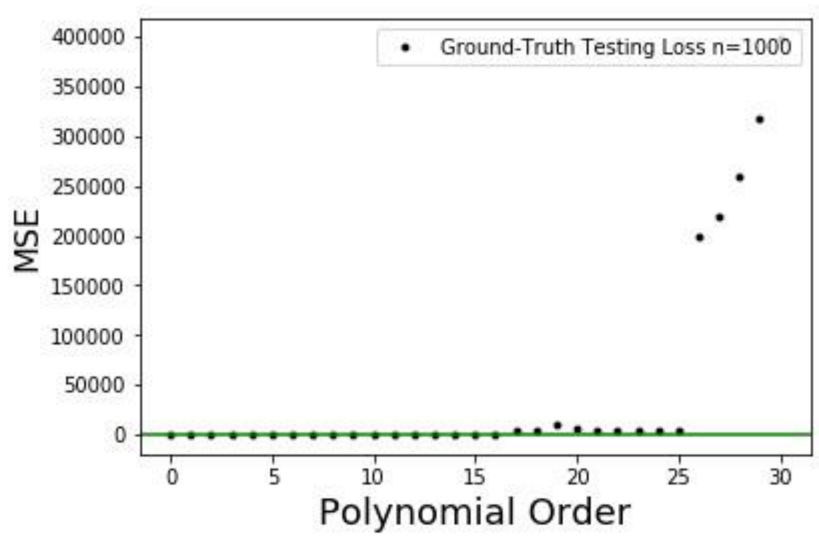
iv. Make another plot with the same axes, titled "Validation Testing Loss," plotting the MSE for the 10 samples held out. In this plot, include a green horizontal line for the ground-truth model's MSE on the validation data.



v. In 3-6 sentences, comment on how the validation testing loss curve behaves, explaining why it is that way, using knowledge of the ground-truth model. Your comment should also explain the similarities and differences between the training loss curve and the validation loss curve.

The MSE is not increasing much at first, then it increases steeply once the order of the polynomial is 25 and above. Both the training loss and validation curve change their behavior around the 20th order. The validation curve behaves this way because the data was overfit. By that I mean the high order polynomials fit the training data nice, however, the same polynomials with high order(20 and above) were not good for fitting the validation data. The tall peaks of a high order polynomials would create a big MSE of they corresponded to a data point that was far from it in the validation data.

vi. Now generate a new data-set with $n = 1000$. Make another plot, titled "GroundTruth Testing Loss $n = 1000$" with the MSE of the polynomials fitted to the $n = 30$ data set evaluated on this new, larger data-set. (This is a similar exercise to (h).iv except you are using a much more data) Use the same axes as the other plots. (keep in mind that in practice, we cannot do this since we won't know the ground truth.) In this plot, include a green horizontal line for the ground-truth model's MSE on this $n = 1000$ sample data-set.



vii. In 2-4 sentences, comment on how well (or not) the validation set testing loss approximates the ground-truth testing loss. It approximates it pretty well. Both curves are showing a big increase in MSE at around the 25th order and both are steady before that. The MSEs for the Ground truth are on a greater scale than the validation MSEs.

viii. Next, let's look at model complexity instead of using validation data. Fit polynomials using all n = 30 data points and keep track of the corresponding MSE.

Then calculate

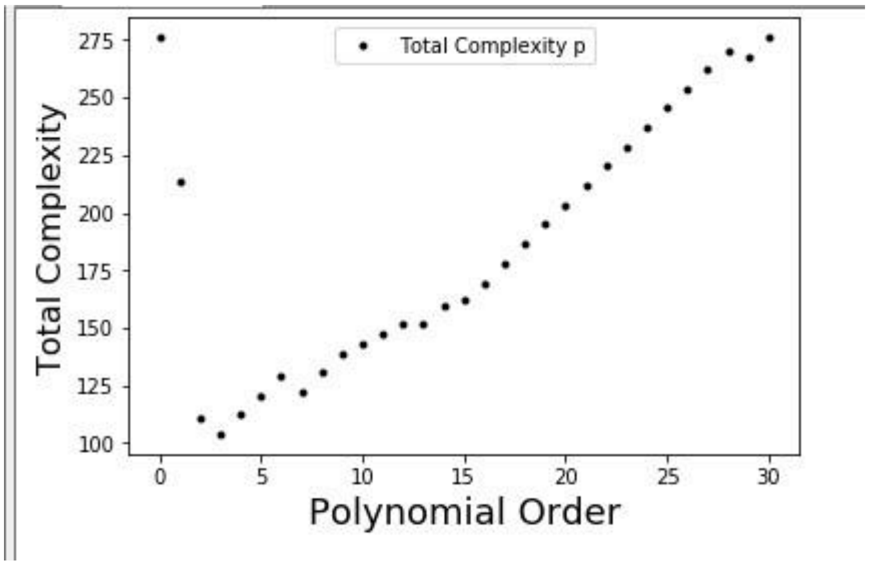
$$\begin{aligned}\text{Total Complexity} &= \text{Total Training Loss} + \lambda \cdot \text{Model Complexity} \\ &= \quad MSE \quad + \quad \lambda p\end{aligned}$$

where n refers to the number of samples used to fit (so n = 30), p is the the model order, and λ is a threshold we choose. In class, for examples, we've used λ = 1 for illustration, but in practice we need to be a bit careful. We need to convert scales, like if we want to add

3.2 miles + 17 inches,

the summed number is not 20.2. Instead we need to make sure they are on the same *scale*. (Chapter 6.1.3 of the textbook describes a few λ values that have been derived by statisticians, such as Mallow's C_p, AIC, and BIC.) For us, let's pick a λ such that the model with p = 30 has the same total-complexity as the model with p = 0.

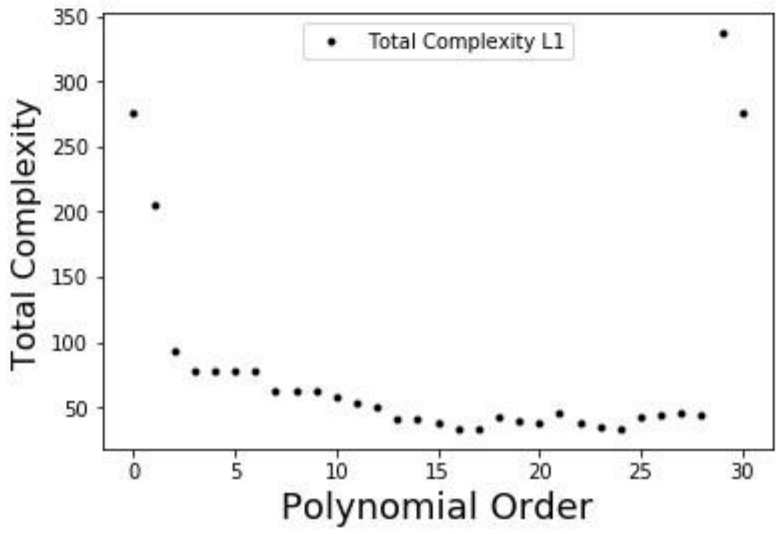
- Write what that λ should be in terms of p and the MSE of certain models. Use the notation $MSE(i)$ for the MSE for the fitted polynomial of order i. Lambda=(MSE(0)-MSE(30))/30
- Make a plot of the total-complexity as a function of the model order p. Use the title "Total Complexity p"

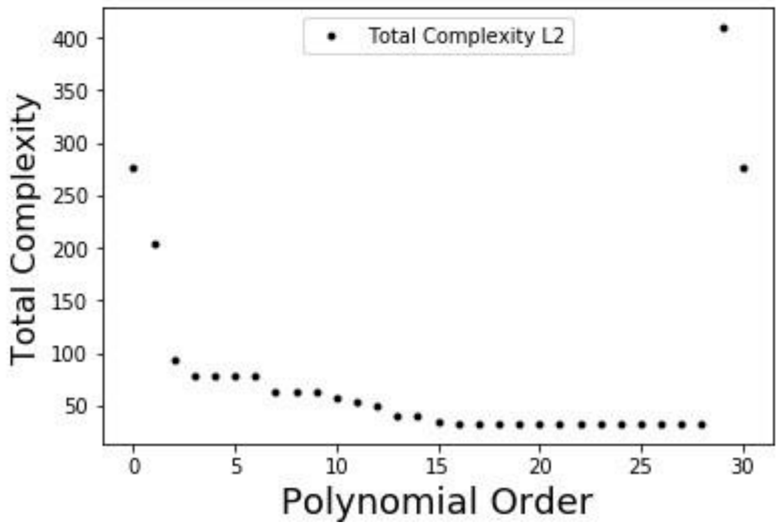


- In 2-5 sentances, explain the behavior of the total complexity curve with reference to the validation testing curve and the ground truth testing curves.

The curve decreases to till about the 4hh order then increases. This and the ground truth and validation plots show the consequences of overfitting. It results in increased complexity and increased MSE. ix. We can also use other penalties. Reusing the fitting from part (h).viii, select a new λ and make new plots for penalties other than λp

- $\lambda_1 \sum_{i=0}^{p} |a_i|^1$, with plot titled "Total Complexity L1"
- $\lambda_2 \sum_{i=0}^{p} |a_i|^2$ with plot titled "Total Complexity L2."





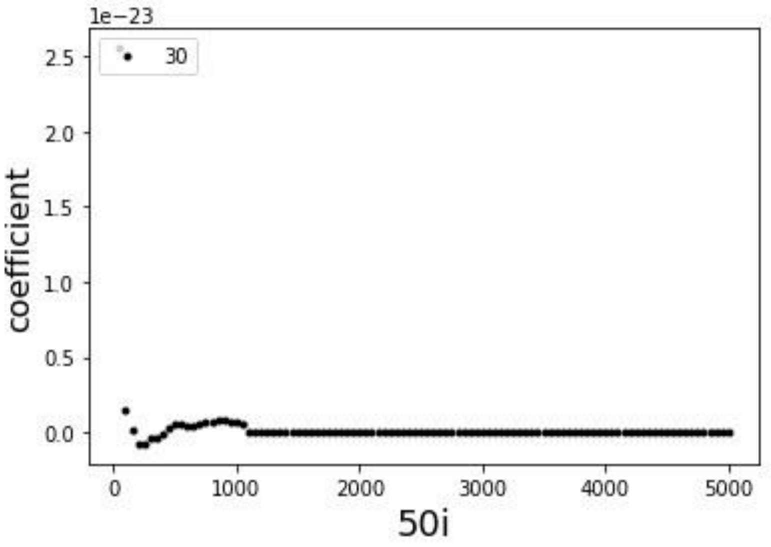
In 2-5 sentences, discuss why the curves behave the way that they do in comparison/contrast to the curves using validation testing data and groundtruth testing data. At the highest order polynomial, the complexity increases to that of the lowest. This is because of the relationahip between MSE and lambda in the equation of total complexity.

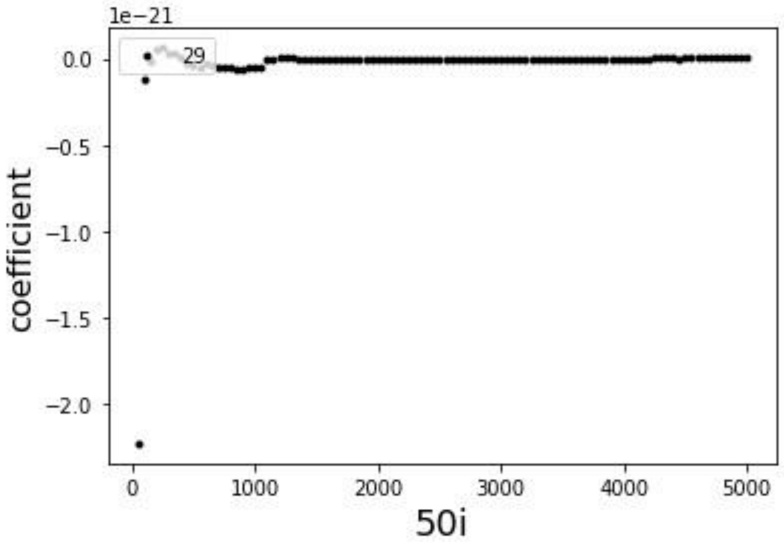
(i). Lastly, let's see how much getting more data helps to find the correct model.

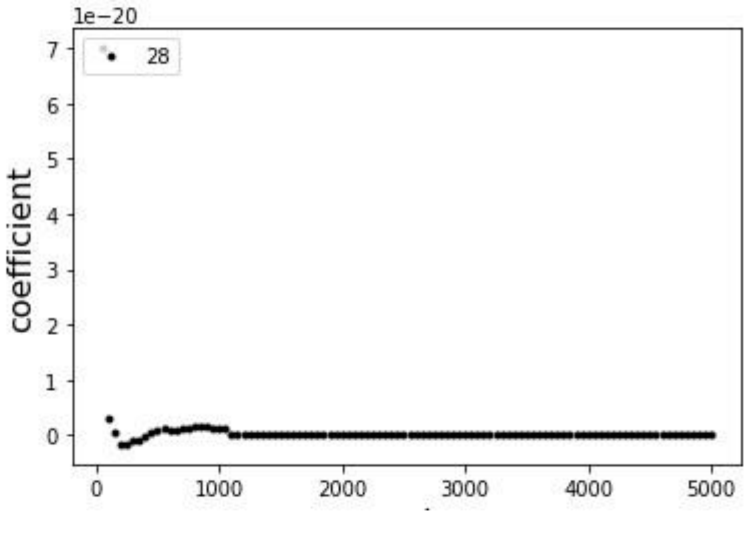
i. Generate n = 5000 samples which will be used for fitting.

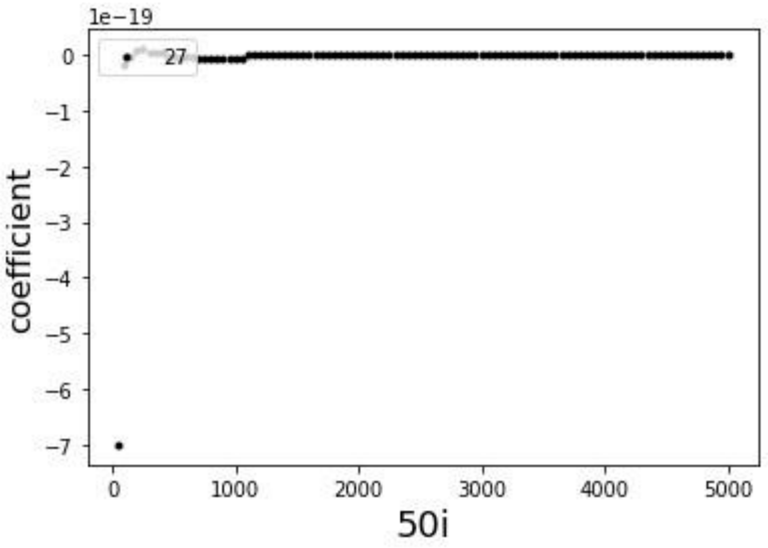
ii. For i = 1,...,100, fit the best p = 30 polynomial using n = 50i samples from the 5000 sample data set.

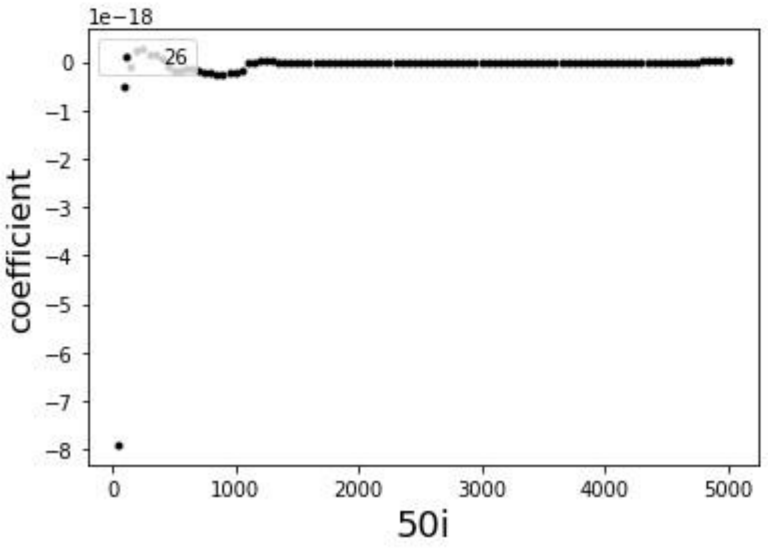
iii. Plot each coefficient $\{a_0, a_2, a_4, a_6, a_8\}$ as a function of 50i in separate plots. (so one plot for a_0, etc). In each plot, draw a green line for the corresponding coefficient in the ground-truth polynomial.

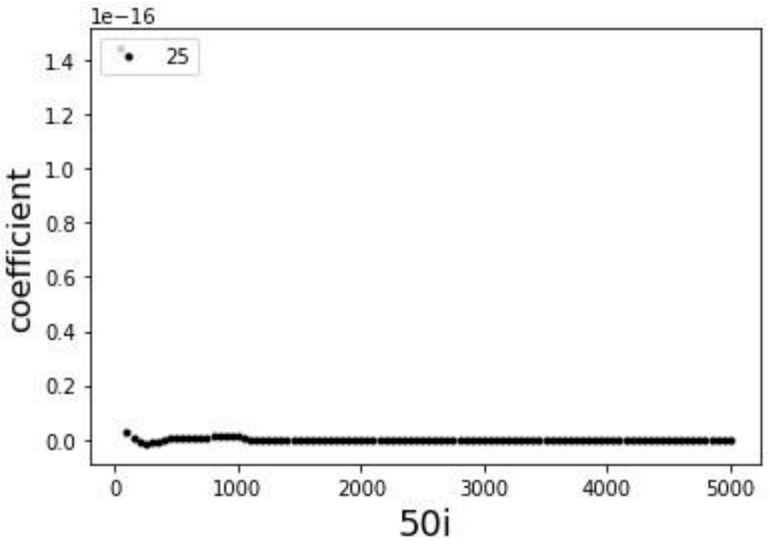

1e−23
30
coefficient
50i
2.5
2.0
1.5
1.0
0.5
0.0
0
1000
2000
3000
4000
5000

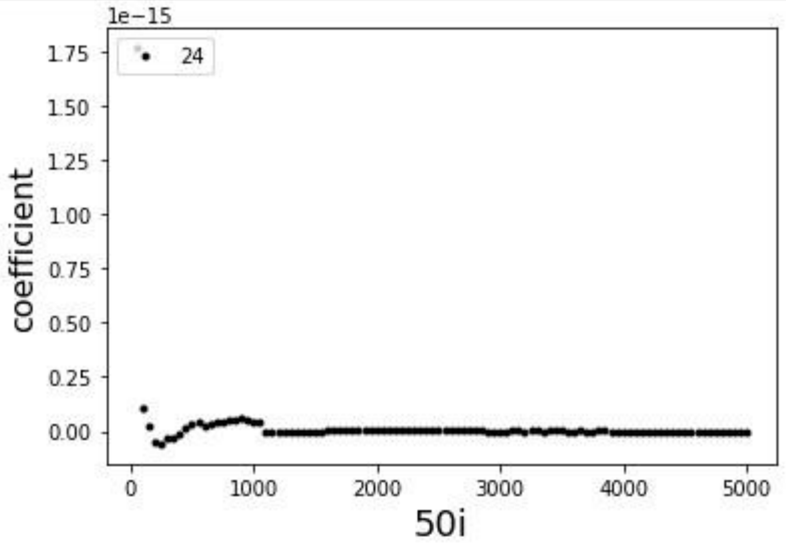

1e−21
29
coefficient
50i
0.0
−0.5
−1.0
−1.5
−2.0
0
1000
2000
3000
4000
5000

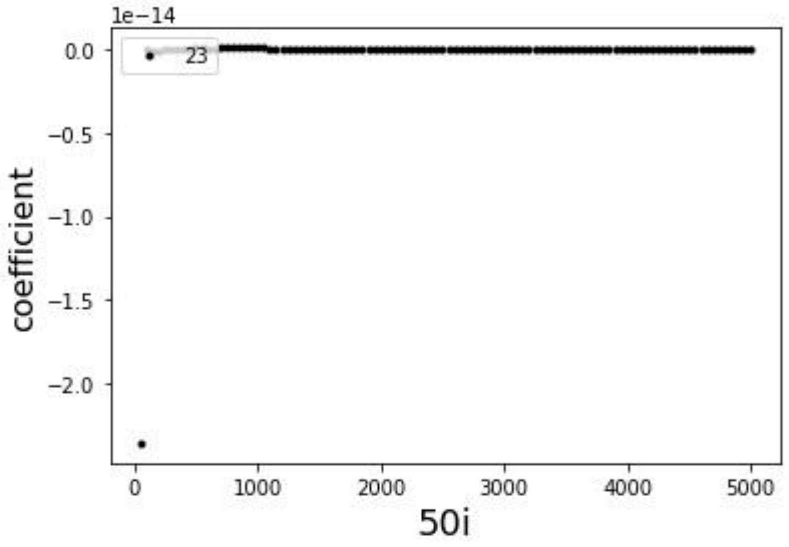

1e−20
28
coefficient
7
6
5
4
3
2
1
0
0
1000
2000
3000
4000
5000

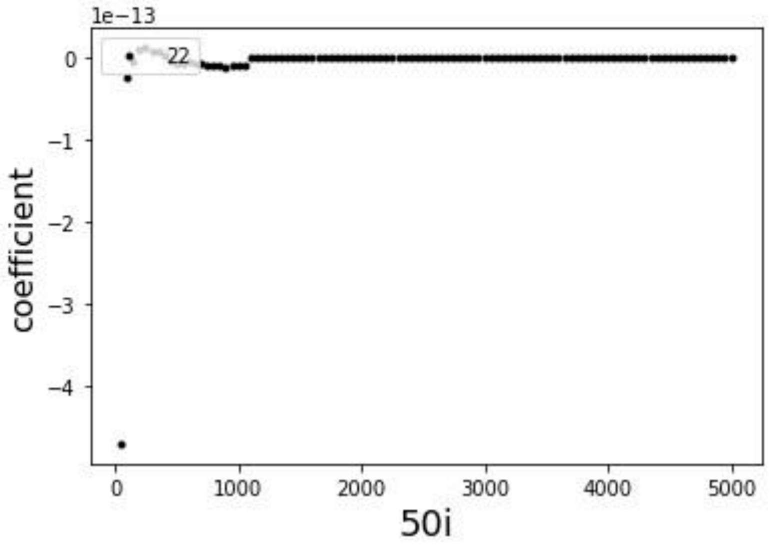

1e−19
27
coefficient
50i
0
−1
−2
−3
−4
−5
−6
−7
0
1000
2000
3000
4000
5000

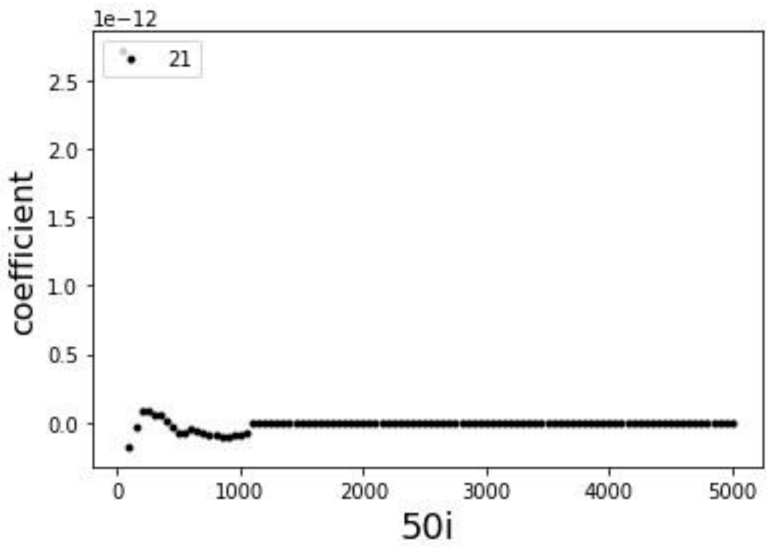

1e−18
26
coefficient
50i
0
−1
−2
−3
−4
−5
−6
−7
−8
0
1000
2000
3000
4000
5000

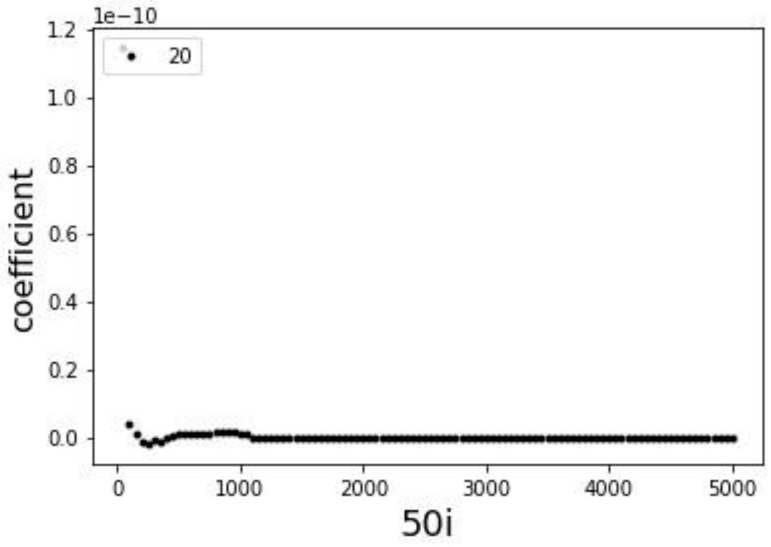

1e−16
25
coefficient
50i
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
1000
2000
3000
4000
5000

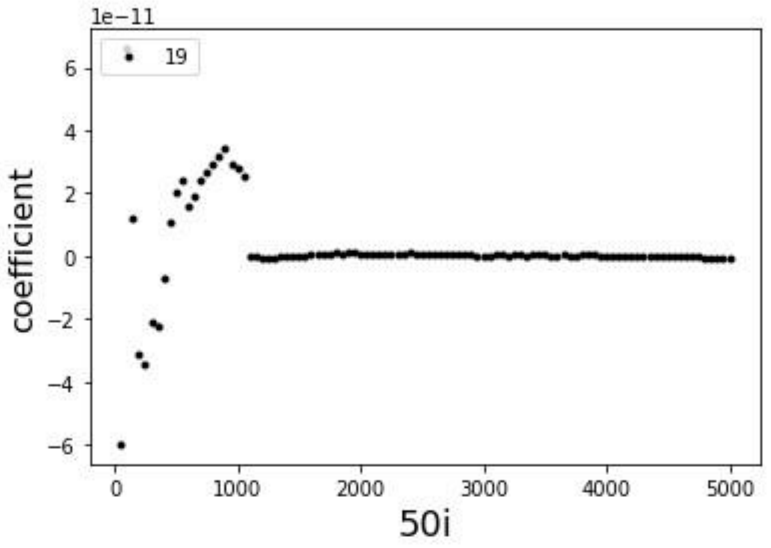

1e−15
24
coefficient
50i

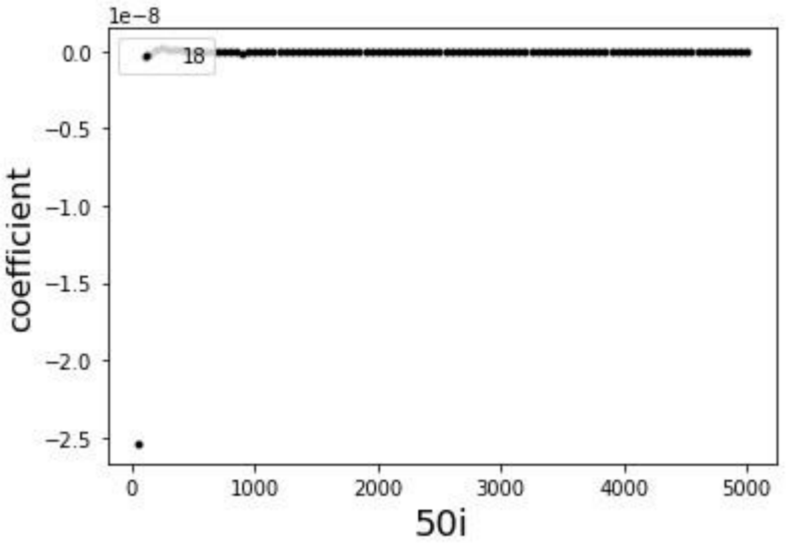

1e−14
23
coefficient
50i

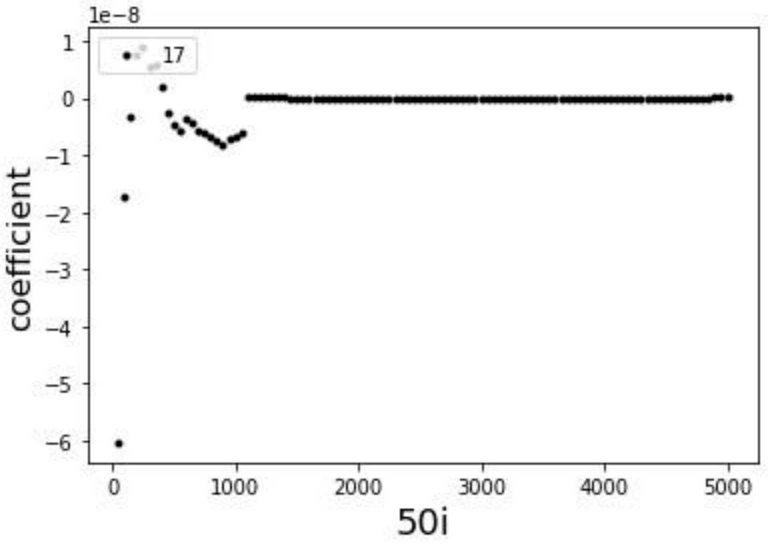

1e−13
22
coefficient
50i

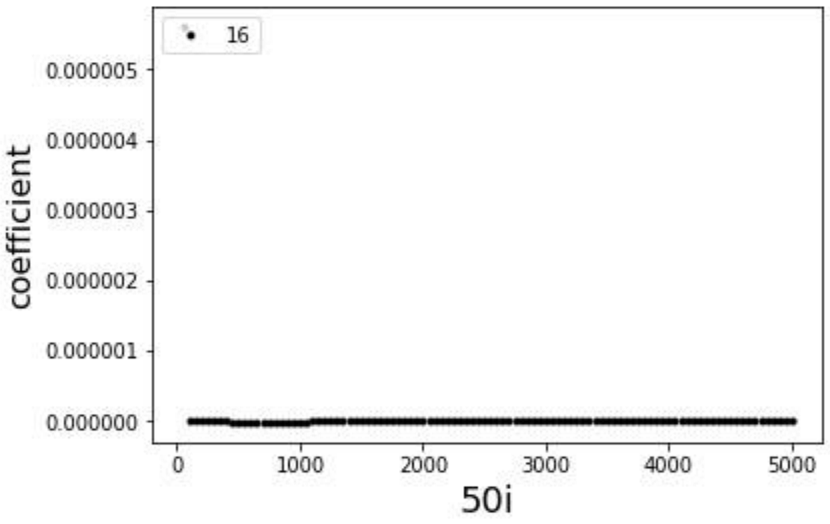

1e−12
21
2.5
2.0
1.5
1.0
0.5
0.0
coefficient
0
1000
2000
3000
4000
5000
50i

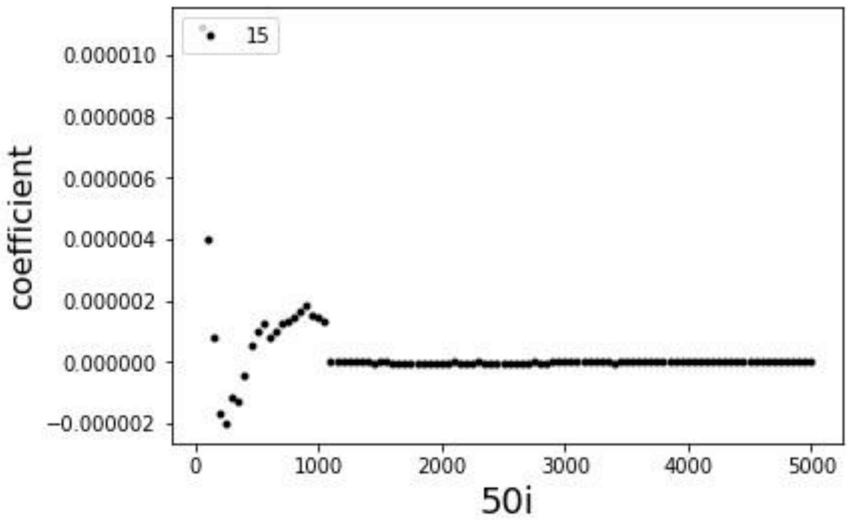

1e−10
20
1.2
1.0
0.8
0.6
0.4
0.2
0.0
coefficient
0
1000
2000
3000
4000
5000
50i


1e−11
19
6
4
2
0
−2
−4
−6
coefficient
0
1000
2000
3000
4000
5000
50i

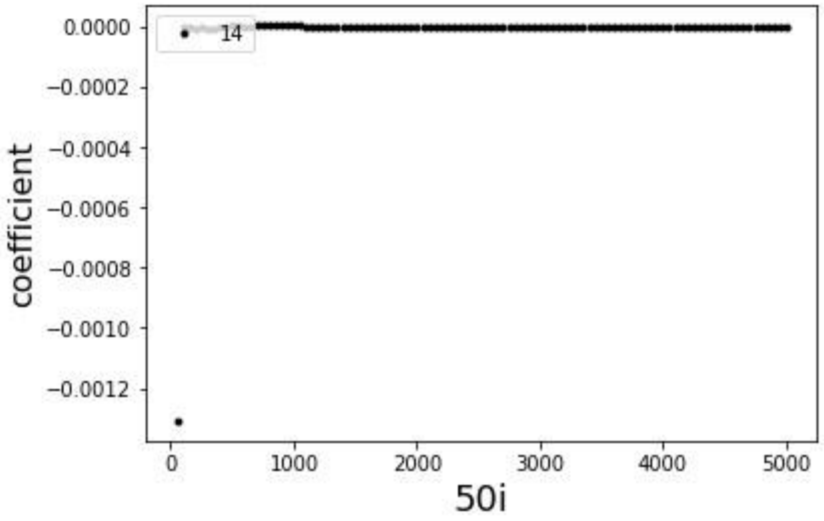

1e−8
18
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
coefficient
0
1000
2000
3000
4000
5000
50i


1e−8
17
1
0
−1
−2
−3
−4
−5
−6
coefficient
0
1000
2000
3000
4000
5000
50i

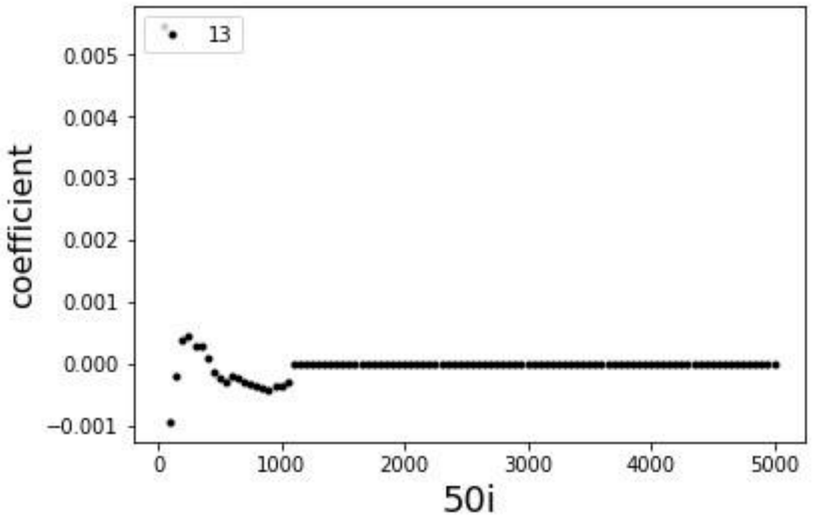

16
0.000005
0.000004
0.000003
0.000002
0.000001
0.000000
coefficient
0
1000
2000
3000
4000
5000
50i



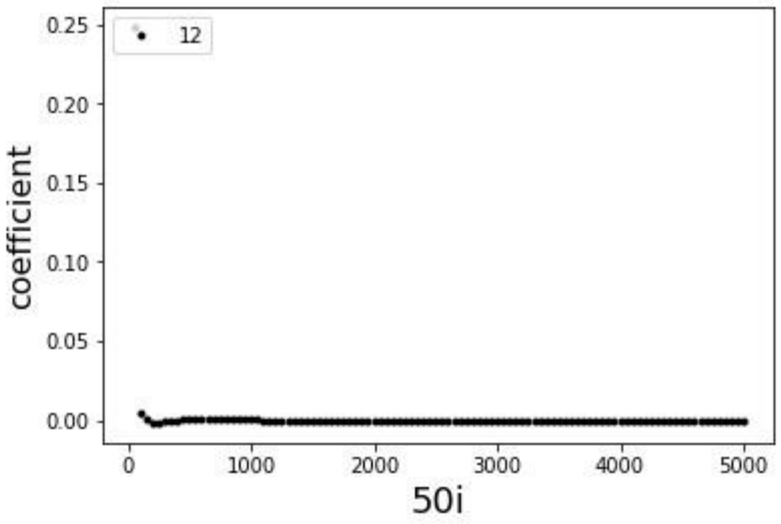
15
coefficient
0.000010
0.000008
0.000006
0.000004
0.000002
0.000000
−0.000002
0
1000
2000
3000
4000
5000
50i

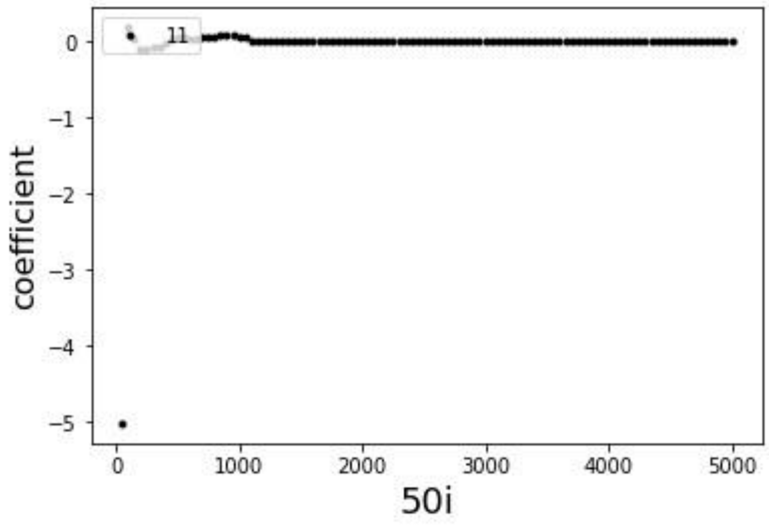


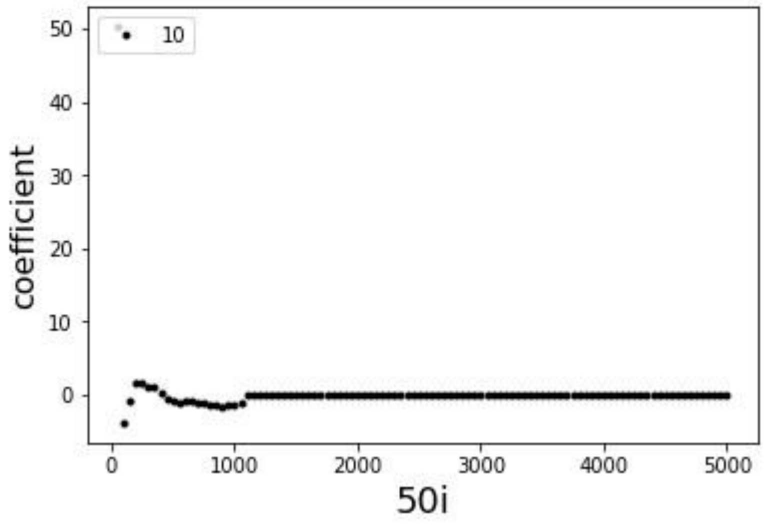
14
coefficient
0.0000
−0.0002
−0.0004
−0.0006
−0.0008
−0.0010
−0.0012
0
1000
2000
3000
4000
5000
50i

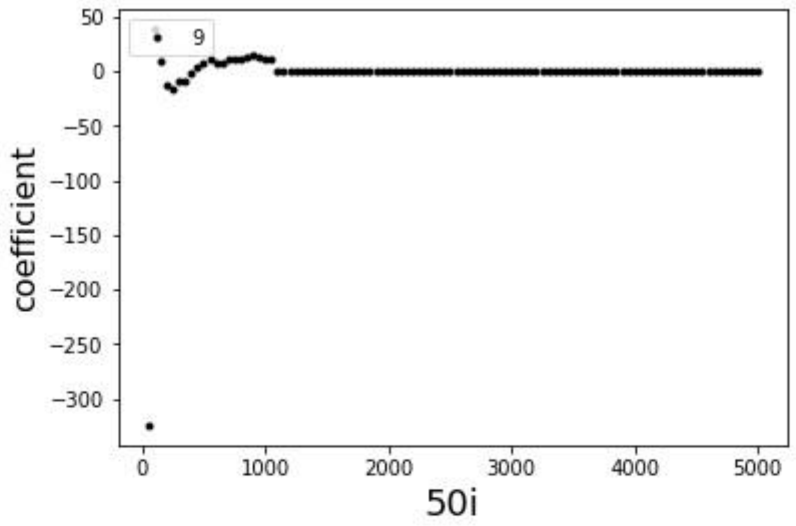


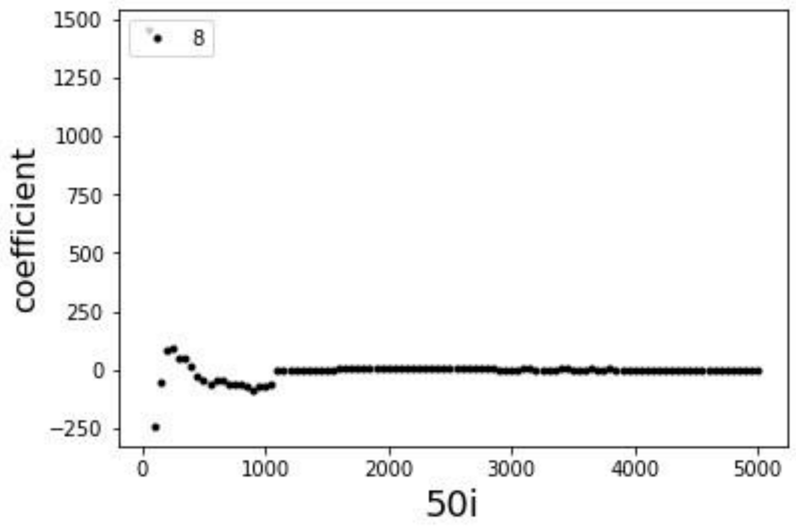
13
coefficient
0.005
0.004
0.003
0.002
0.001
0.000
−0.001
0
1000
2000
3000
4000
5000
50i

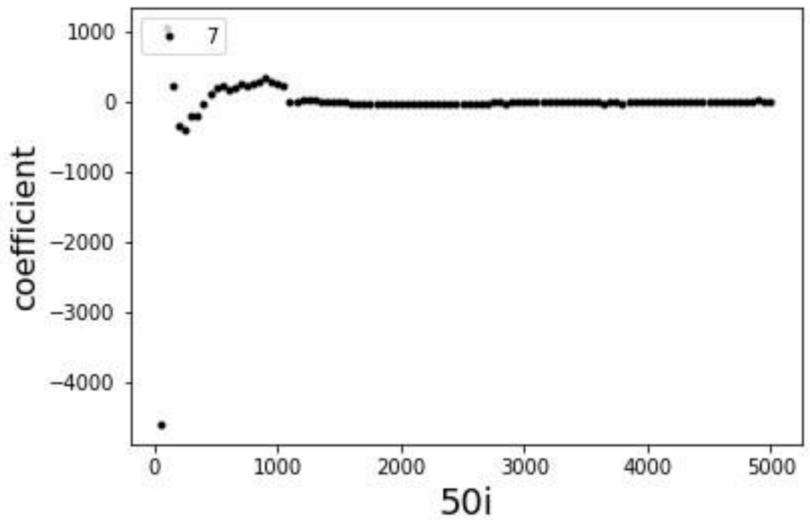

12
coefficient
50i
0.25
0.20
0.15
0.10
0.05
0.00
0
1000
2000
3000
4000
5000

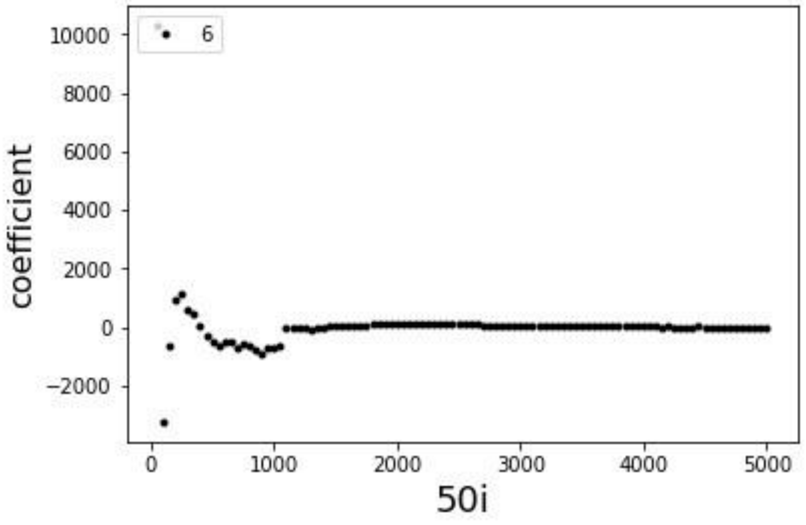

11
coefficient
50i
0
−1
−2
−3
−4
−5
0
1000
2000
3000
4000
5000

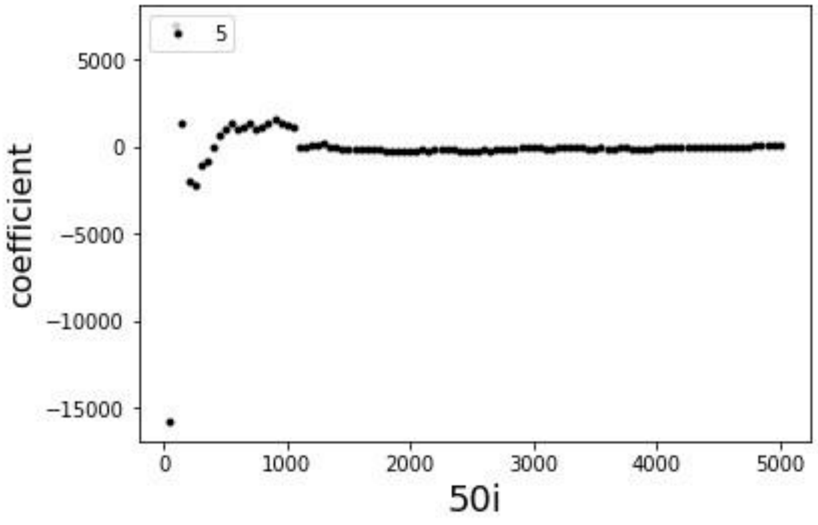

10
coefficient
50i
50
40
30
20
10
0
0
1000
2000
3000
4000
5000

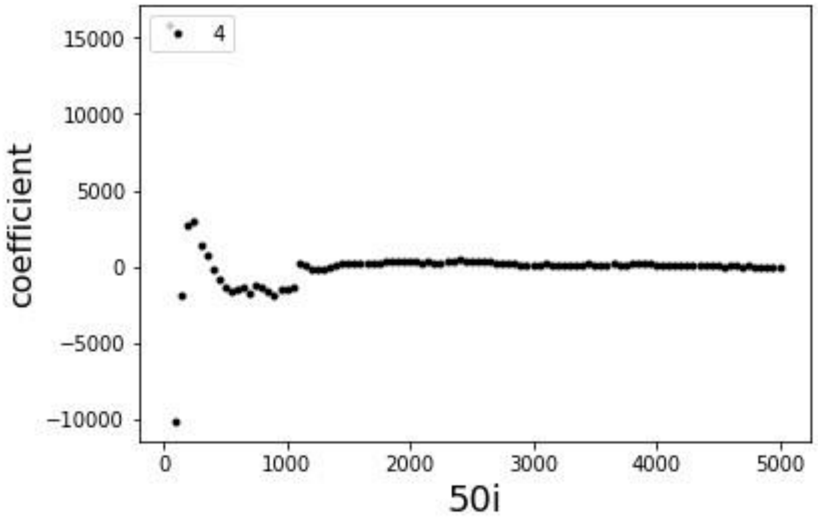

9
coefficient
50i

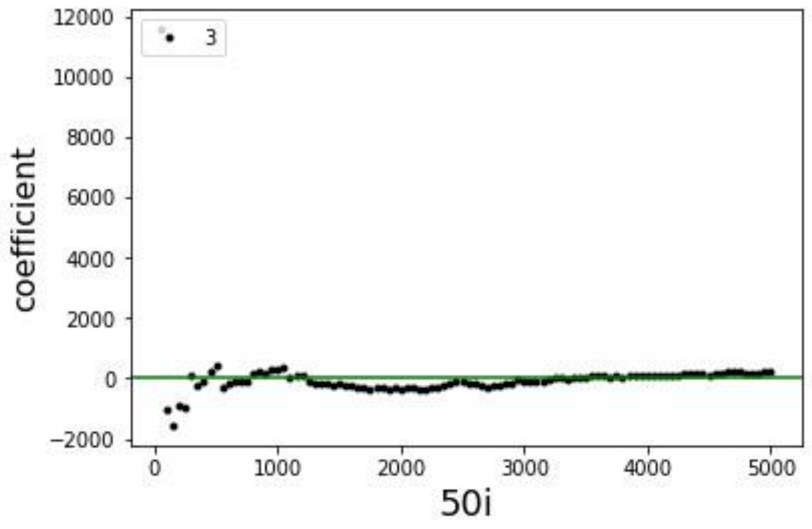

8
coefficient
50i

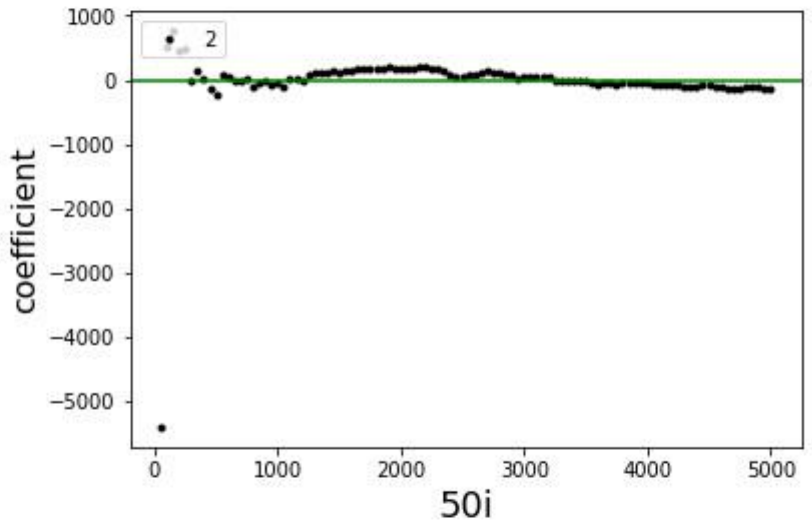

7
coefficient
50i

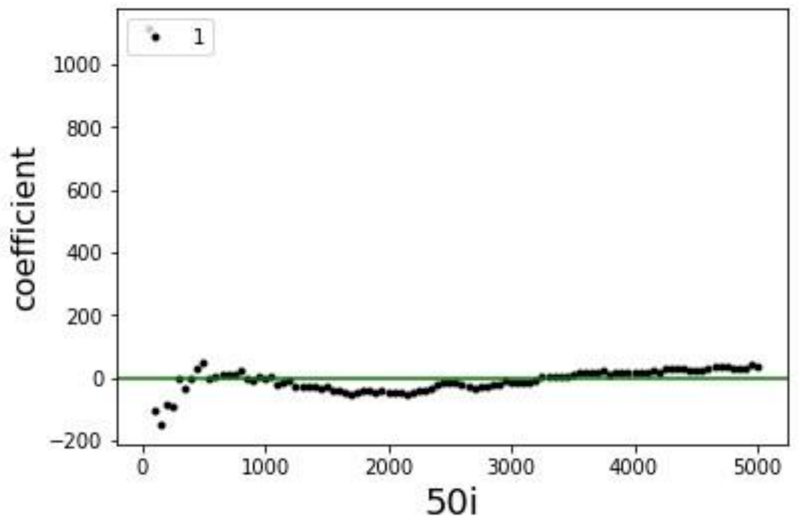

6
coefficient
50i

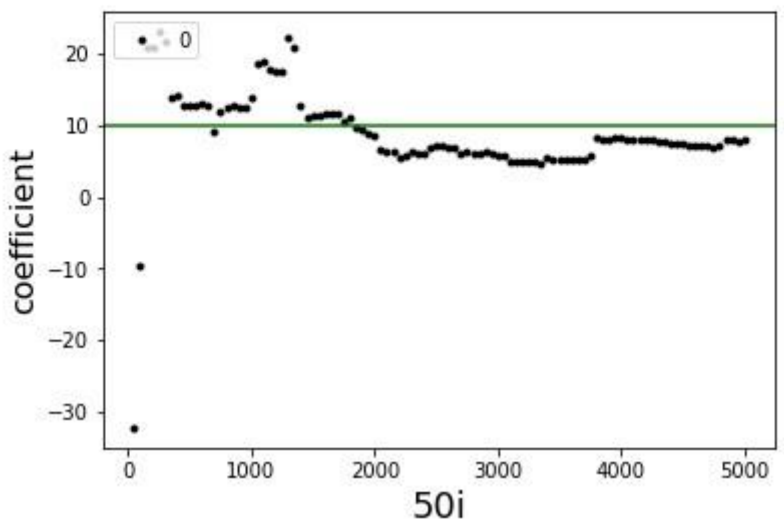

5
coefficient
50i


4
coefficient
50i


3
coefficient
50i
12000
10000
8000
6000
4000
2000
0
−2000
0
1000
2000
3000
4000
5000


2
coefficient
50i
1000
0
−1000
−2000
−3000
−4000
−5000
0
1000
2000
3000
4000
5000


1
coefficient
50i
1000
800
600
400
200
0
−200
0
1000
2000
3000
4000
5000



iv. In 2-5 sentences, discuss what is happening to the coefficients and comment on how the amount of over-fitting depends on the number of samples used.

The coefficients are approaching 0 as they increase 0-30. As the number of samples used increases, all coefficients approach 0(fluctuate around it). The amount of overfitting will decrease as sample size increases. A large sample size isn't as affected by outliers or noise as a small one is.