ELL-409 Assignment

Jaskeerat Singh Saluja(2019MT60752)

September 29, 2021

Pre-requisites

Normalizing Data

While generating the design matrix we normalize the features vectors i.e $x, x^2, x^3....x^M$. The feature X $\in$ [-M,M] , then the features $X^2 \in [-M^2, M^2]$, and so on for all other powers of X. Hence the loss function (least squared error) forms contours with elliptical shapes .

Hence while optimizing error in the gradient descent algorithm , an improvement in X^2 may cause big step in X, thus instead of decreasing error ,we may cross the minima point , and error may increase . This would cause a lot of oscillation of w around the minima point.

An unnormalized data would causes:

- highly sensitive learning parameter
- high oscillation of parameter w during convergence

To prevent this we use normalization techniques like z-normal and min-max normalization. This transforms the contour plots of loss function to circular shapes , hence the gradient descent converges better

Z-Normalization: The feature vector is transformed as

$$(X) -> \frac{X - \mu}{\sigma}$$

where μ : mean of the feature vector and σ : variance of the feature.

Z-Normalizing feature vector : $x, x^2, ...x^M$is transformed as

$$x^i -> \frac{(x^i - mean(x^i)}{\sigma(x^i}$$

.

Running gradient descent yield us $w_0^*, w_1^*...w_M^*$. Which optimizes error for our normalized data points X. Thus we need to 'un-normalize' the parameters W to previous space

The **unnormalized parameter** we get are

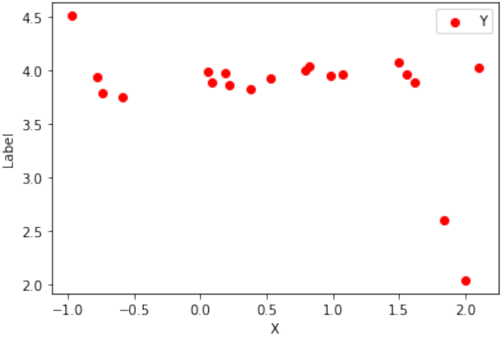
$$w_i = \frac{w_i^*}{\sigma_i}, \forall i \in \{1, M\} and$$

$$w_0 = w_0^* - \sum_{i=1}^{i=M} \frac{\mu_i}{\sigma_i}$$

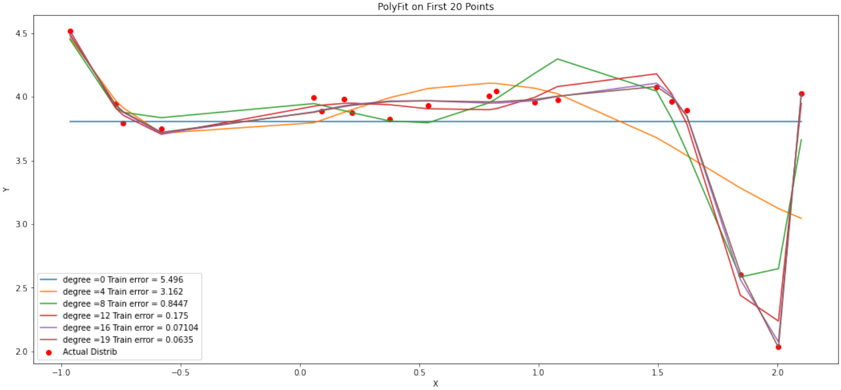
The code written for polyfit auto-normalizes the feature space before training and unnormalize the paramter w after training.
Normalization class is also implemented which is inherited by polyfit class.

Part-1A

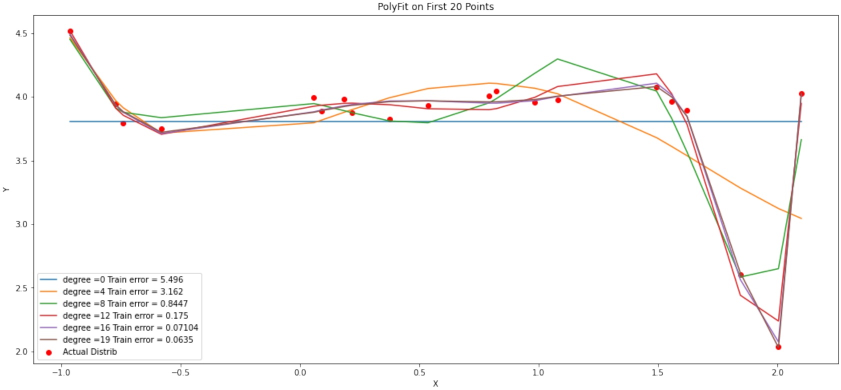
1A - (a) Fitting First 20 points



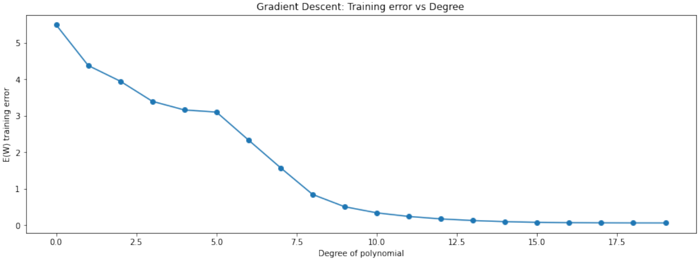
Polyfit using Gradient Descent



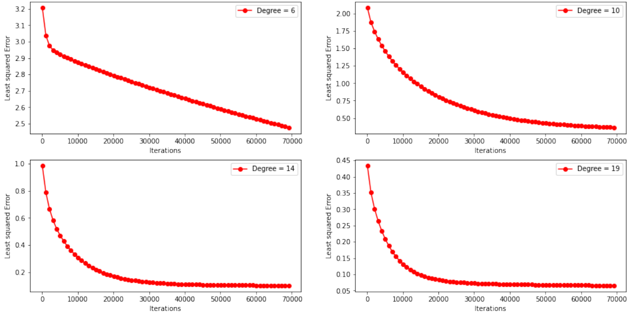
Polyfit using Gradient Descent



Training error vs Degree of Polynomial

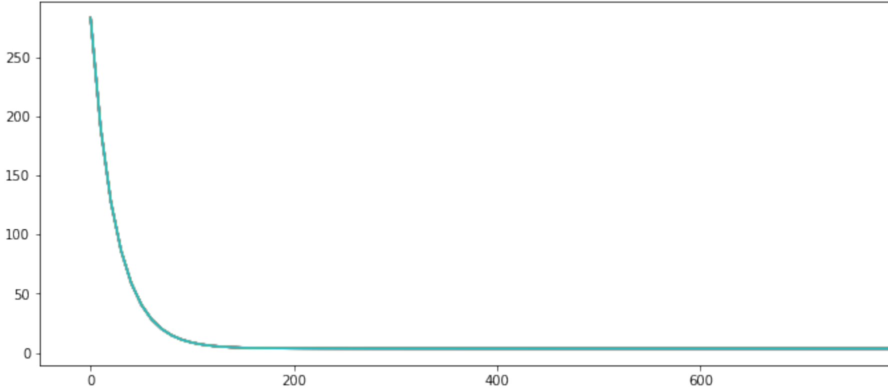


Training error vs Number of Iterations From below we can say the gradient descent algorithm converges very fast for the first 5000 iteration , after that the convergence of gradient descent is very slow in successive iteration. Here 1 iteration corresponds to 1 pass over the data set , hence in our case 1 iteration implies 1 pass over all 20 points.



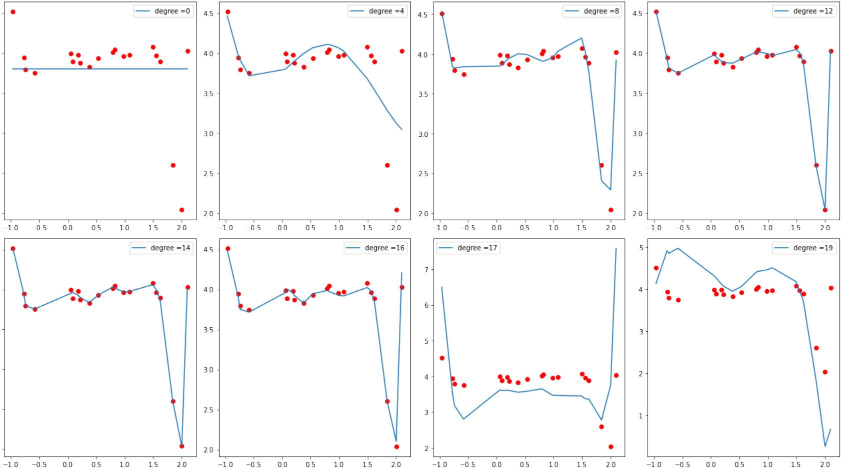
Variation of Error $E(W)$ with the batch size in gradient descent¶

Since the data set is quite small i.e only 20 points . Also the data was normalized before the grad desccent thus the batch gradient descent performed similar on all batches.



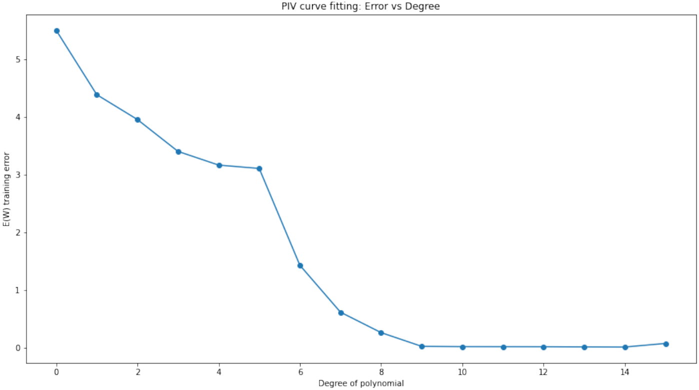
Optimization using psedo penrose Inverse

Polyfit using PIV Observations: Starting *degree* > 16 the PIV fails , thus gives poor fitting , as shown below.This is the reason we cannot rely on PIV method for fitting higher degree poynomials on our data set



Error vs degree of poynomial

Since PIV fails for *degree* ≥ 15 hence the error shoots up



Good fit of polynomial (Underfitting , overfitting) (without regularization)

We know the **train error**$->0$ as the degree of poynomial M increases but this lead to overfitting of the model over the underlying data set.

Test validation To test a model we will partition our data i.e 20 points into test and train data . The model is trained on trained data and scored on the test data.

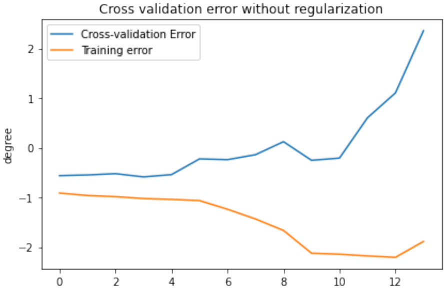
Several models are scored using above schema , one with the best score i.e in our case the least error is chosen.

I have used K-fold cross validation method to score the models with variable hyper-paramters

The k-fold cross validation method is implemented in file cross-validation.py

K-Fold Cross Validation On our data set. Observation: The polynomial of $degree = 9$ is sweet spot . Thus $degree = 9$ gives us a good fit .

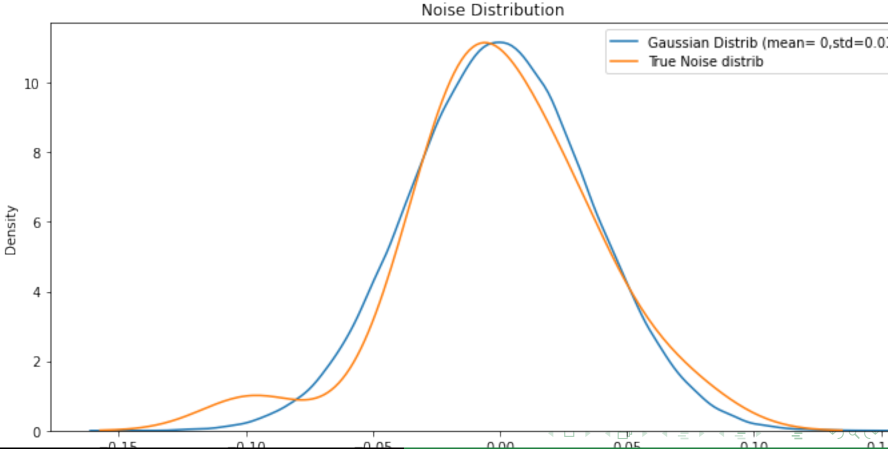
w_{ML} =[3.99399065 ,-0.56158887, -0.16833793 , 4.93082264, -2.41734063 ,-8.69985778, 6.99587172 ,3.09004417 ,-4.22105658 , 1.00545962]



Estimation of noise

$$\frac{1}{\beta_{ML}} = \frac{\sum (y-t)^2}{N}$$

. Since our best approximation comes for degree 9 . Thus $variance = \frac{1}{\beta}$. Hence Gaussian distribution $N((\mu = 0, \sigma = 0.03558)$describes the underlying noise.



Observations:

- The points 20 are too low in number thus we donot get the an expected U shaped curved depicting decrese in $(bias)^2$ and increase in *variance* ,giving an sweet-spot in between.
- The error is contributed from following:
  - test variance (= variance due to test sample size) : Since the test points are too low in our model thus the test error itsef have a high variance.
  - Model instability variance due to training sample size
  - testing bias
- We know that at lower degree of polynomial the cross validation should be dominated by bias in model ,but due to so low count of data points the bias does-not dominate that well over the data , hence the variance starts dominating soon.
- At $degree = 9$ there is dip in the CVR , hence we get a see spot where the total bias and variance sum is minimized. For $degree \geq 10$, the variance starts shooting up

Bayesian curve fitting .

Interested values of $\lambda = [10^{-1.0}, 10^{-1.2}, 10^{-1.4}, \ldots 10^{-4.8}]$ Interested values of *degree* $=$ [9,10,11,12,13,14]

Approach: Iterate over all (lambda ,degree) pairs , calculate the cross validation test error. One with least test error as good fit for our .

```
df = pd.DataFrame(columns=['lmda', 'degree', 'test_err', 'train_err'])
df.head()
posssible_degree =np.arange(7,15,1)
lmda= [10**i for i in np.arange(-1.0,-5.0,-0.2)]
for lm in lmda:
    for degree in posssible_degree:
        train_err,test_err=kfold_cross_validation(X,Y,degree=degree,K=10,lmda=lm,method='piv')
        df.loc[-1]={'lmda':np.log10(lm),'degree':degree,'test_err':test_err,'train_err':train_err}
        df.index = df.index + 1
        df = df.sort_index()
```

✓ 2.4s

Tuning of the hyper-parameter continued sorting the data Frame by train error gives us that $lmda = 10^{-4.6}$and $degree = 14$ fits our model with least test error.Hence we found sweet spot of our model using regularization.

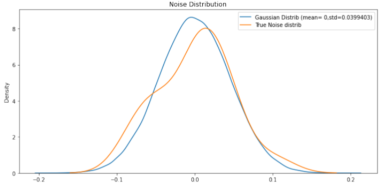
```python
dd = pd.DataFrame(columns=['lmda', 'degree', 'test_err', 'train_err'])
# df.head()
posssible_degree = np.arange(8, 13, 1)
lmda = [10**i for i in range(-2, -8, -1)]
# lmda.append(0)
for lm in lmda:
    for degree in posssible_degree:
        train_err, test_err = shuffled_cvr(
            X, Y, degree=degree, K=10, lmda=lm, method='piv')
        dd.loc[-1] = {'lmda': (lm), 'degree': degree,
                      'test_err': test_err, 'train_err': train_err}
        dd.index = dd.index + 1
        dd = dd.sort_index()

dd = dd.sort_values('test_err')

dd.head(10)

```
✓ 3.6s

Estimation of noise (we derived that β parameter for both bayesian and maximum likelihood approach gives us same noise distribution, i.e

$$\frac{1}{\beta_{MAP}} = \frac{\sum (y(x,w) - t)^2}{N}$$

.

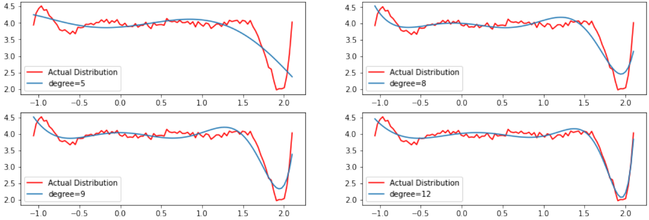
Since our best approximation comes for degree 14 and $\lambda = 10^{-4.6}$. Thus $variance = \frac{1}{\beta_{MAP}}$. Hence Gaussian distribution$N(\mu = 0, \sigma = 0.0460762419)$describes the underlying noise. w_{MAP}=[4.00508911e+00 -2.32331032e-02 -7.08705280e-01 1.34058586e+00 , 1.09848573e+00, -3.16688496e+00 , ,4.49931451e-01 ,1.58108297e+00 ,-5.82997950e-01 , 6.70898630e-02 ,-1.18962016e-01 ,1.58515472e-02 , 2.62794355e-03 , 7.11685818e-03 ,-1.34651068e-03]



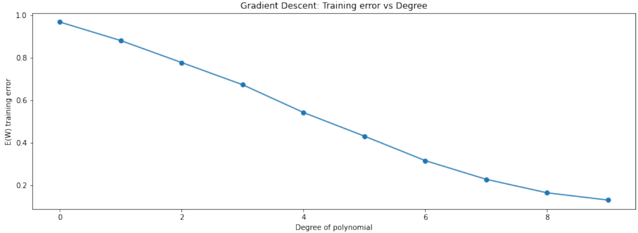
Using Full 100 points dataset

Polynomial fit over our dataset

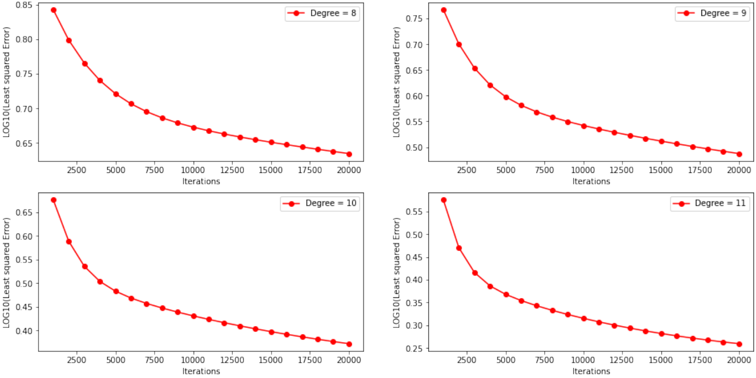
Optimization using Gradient descent Even after **20,000 iteration iterations per model** (took minutes to just train) , gradient descent could not optimize the polynomial fit , very well. I have used moment based gradient descent which is faster than batch gradient descent , but not fast enough to converge at a faster rate. In the next slide we see that better fit by pseudo penrose inverse formulation



Least squared error vs degree Clearly *degree* ≥ 9 we get a good fit over our dataset.
Still , the convergence of gradient descent is quite slow, need high speed computers to train. But gradient descent converges to minima and does not have issue when design matrix is singular

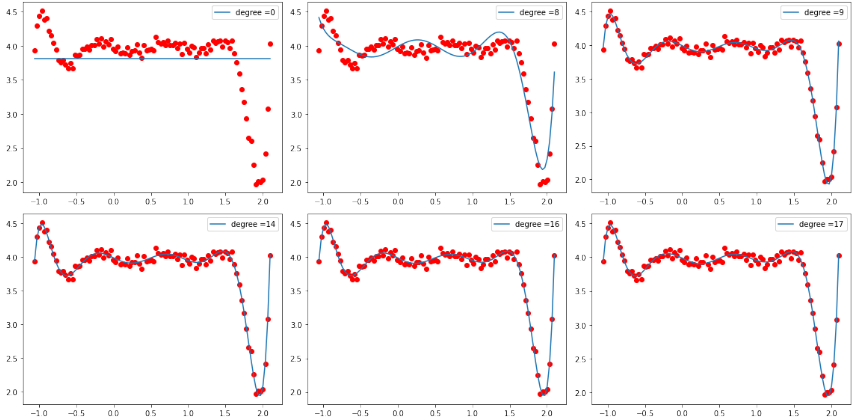


Errors vs number of iterations in gradient descent. The below picture clearly depicts the rate at which gradient descent converges.

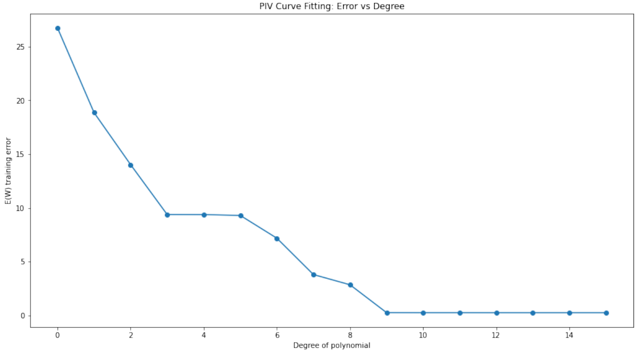


Polynomial fit over our dataset Optimization using PIV (pen-rose inverse matrix)

Degree = 9**(least complexity) polynomial fits our curve nicely.**



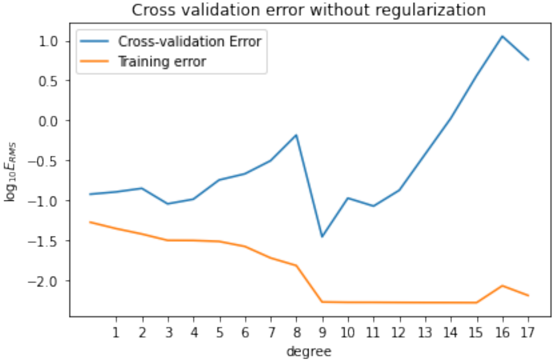
Training error vs degree Clearly *degree* ≥ 9 we get a good fit over our dataset



Maximum Likelihood

I have used K-fold cross validation method to score the models with variable hyper-paramters paramter

Clearly $M = 9$ is polynomial with least complexity that gives us the sweet spot over our data set.



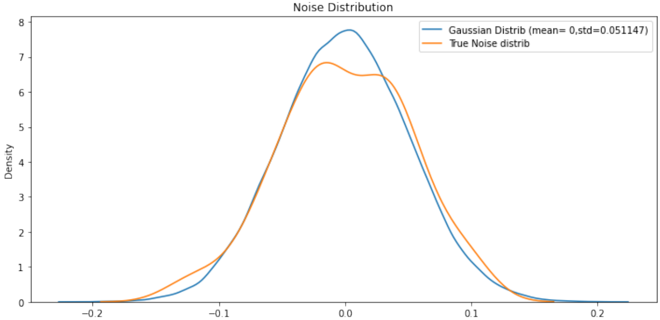
Observations:

- The points 100 are good in number thus we have smoother training and test error over our data set.
- Due to enough points the cross validation error is smooth as function of degree of the polynomial.
- $degree = 9$ is the lowest degree which fits our data set nicely.
- Hence polynomial of $degree = 9$ fits our data well (low train and test error)
- region ($degree < 9$ high training error $- >$ Under-fitting)
- region ($degree \geq 10$ high testing error $- >$ over fitting)

Estimation of noise

$$\frac{1}{\beta_{ML}} = \frac{\sum (y - t)^2}{N}$$

. Since our best approximation comes for degree 9 . Thus $variance = \frac{1}{\beta}$. Hence Gaussian distribution $N((\mu = 0, \sigma = 0.0511478)$describes the underlying noise.



Bayesian curve fitting .

Interested values of $\lambda = [10^{-1}, 10^{-2}, 10^{-3}, 10^{-4} ... 10^{-20}]$ Interested values of $degree$ = [9,10,11,12,13,14]

Approach: Iterate over all (lambda ,degree) pairs , calculate the cross validation test error. One with least test error as good fit for our .

```
df = pd.DataFrame(columns=['lmda', 'degree', 'test_err', 'train_err'])
df.head()
posssible_degree =np.arange(7,15,1)
lmda= [10**i for i in range(-1,-20,-1)]
for lm in lmda:
    for degree in posssible_degree:
        train_err,test_err=kfold_cross_validation(X,Y,degree=degree,K=10,lmda=lm,method='piv')
        df.loc[-1]={'lmda':lm,'degree':degree,'test_err':test_err,'train_err':train_err}
        df.index = df.index + 1
        df = df.sort_index()


```

✓ 1.6s

```
df.sort_values('test_err')
```

✓ 0.2s

Tuning of the hyper-parameter continued sorting the data Frame by train error gives us that $lmda = 10^{-17}$and $degree = 9$ fits our model with least test error.Hence we found sweet spot of our model using regularization.

...

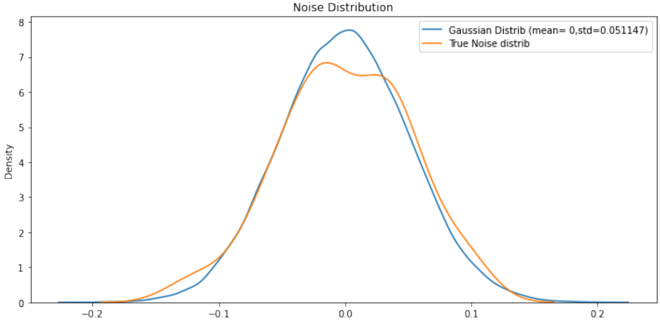
	lmda	degree	test_err	train_err
21	1.000000e-17	9	0.035063	0.005369
29	1.000000e-16	9	0.035063	0.005369
37	1.000000e-15	9	0.035063	0.005369
13	1.000000e-18	9	0.035063	0.005369
5	1.000000e-19	9	0.035063	0.005369
...	...	...	...	...
8	1.000000e-18	14	1.065577	0.005275
32	1.000000e-15	14	1.065577	0.005275
24	1.000000e-16	14	1.065577	0.005275
16	1.000000e-17	14	1.065577	0.005275
0	1.000000e-19	14	1.065577	0.005275

152 rows × 4 columns

Estimation of noise

$$\frac{1}{\beta_{MAP}} = \frac{\sum (y-t)^2}{N}$$

. Since our best approximation comes for degree 9 and $\lambda = 10^{-17}$. Thus $variance = \frac{1}{\beta}$. Hence Gaussian distribution $N((\mu = 0, \sigma = 0.0511478958)$describes the underlying noise.



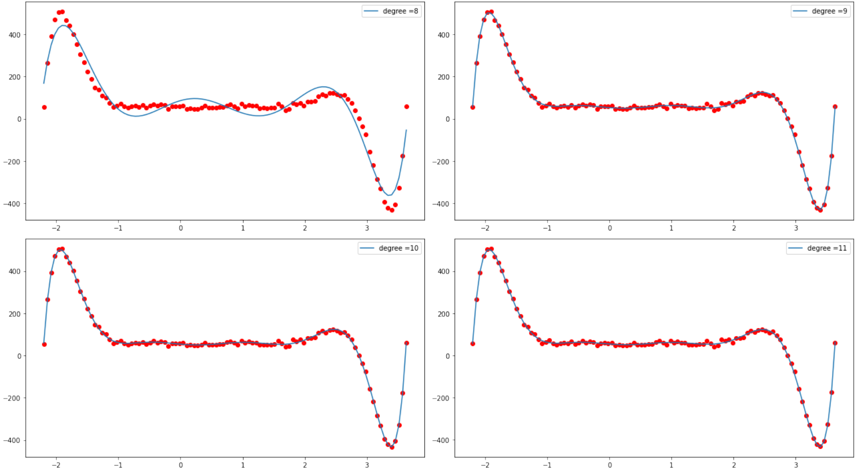
Final estimate of polynomial is poynomial with degree=9 and regularization $\lambda = 10^{-17}$. Since it is one with minimum cross validation error. Also it prevents over fitting by penalizing higher values.

The parameters w is

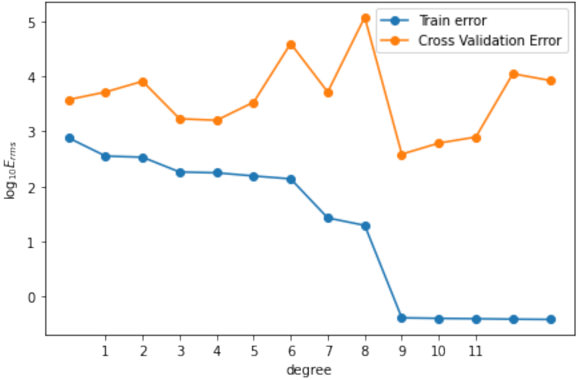
$[w_0 = 3.99399065, w_1 = -0.56158887, -0.16833793,$
$4.93082264, -2.41734063, -8.69985778, 6.99587172$
$, 3.09004417, -4.22105658, w_9 = 1.00545962]$

Part-1B

Polynomial Fit over the given points.



Maximum Likelihood Approach The below plots indicated that *degree* $= 9$ is the best fit for our model as it attains low training error and test error(cross validation).



Bayesian Curve Fitting

Regularizing on various values of λ. Interested values are $\lambda = [10, 1, 10^{-1}, 10^{-2}, 10^{-3} ... 10^{-15}]$ and *degree* $= [9, 10, 11, 12, 13]$. The below approach trains over all combinations of λ and *degree*. The combination with least test error(validation) is chosen.

```
dd = pd.DataFrame(columns=['lmda', 'degree', 'test_err', 'train_err'])
# df.head()
posssible_degree = np.arange(8, 13, 1)
lmda = [10**i for i in range(1, -4, -1)]
# lmda.append(0)
for lm in lmda:
    for degree in posssible_degree:
        train_err, test_err = shuffled_cvr(
            X, Y, degree=degree, K=10, lmda=lm, method='piv')
        dd.loc[-1] = {'lmda': (lm), 'degree': degree,
                      'test_err': test_err, 'train_err': train_err}
        dd.index = dd.index + 1
        dd = dd.sort_index()

dd = dd.sort_values('test_err')

dd.head(10)
```

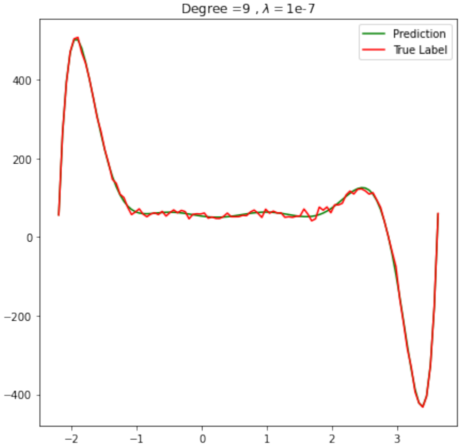
Bayesian Curve Fitting

Sorting the dataframe on basis of the "training error" , we get a good regularization for *degree* $= 9$ and $\lambda = 10^{-7}$. **Choosing low degree and high regularization when similar results over all combinations.**

...

	lmda	degree	test_err	train_err
18	1.000000e-11	9	2.187783	0.686303
23	1.000000e-10	9	2.198363	0.686405
33	1.000000e-08	9	2.200270	0.686434
3	1.000000e-14	9	2.205133	0.686337
28	1.000000e-09	9	2.207801	0.686381
13	1.000000e-12	9	2.208485	0.686137
38	1.000000e-07	9	2.213284	0.686350
8	1.000000e-13	9	2.218115	0.686225
35	1.000000e-07	12	2.222088	0.665190
43	1.000000e-06	9	2.224840	0.689963

Noise: Since the true data has inherent noise in it's data , the below plot helps visualizing noise. Even if we fit the below data with high degree, we may end up fitting the noise , thus leads to over fitting. $noise = h(x) - t$

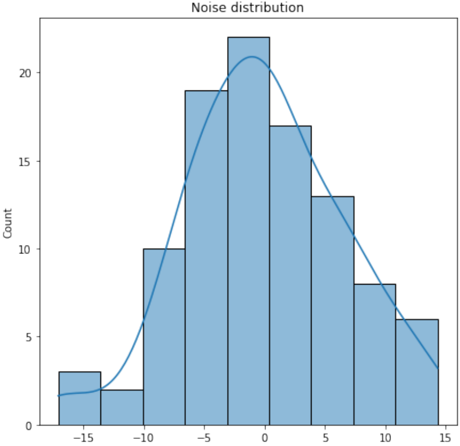


The noise in underlying distribution has a mean of $-6.65686718797076 * 10^{-7}$, which is quite close to 0. Also noise lies in interval $[-20, 20]$. We can test if the underlying noise is beta distribution.
Since beta distribution (α, β) lie in $[0, 1]$ with mean of $\frac{\alpha}{\alpha+\beta}$.
We scale our noise distribution to lie in interval [-0.5,0.5] by dividing whole by noise values by 40. Since mean of our $noise_{normalized}$ distribution is close to 0 .The shifted noise distribution by factor of 0.5 would lies in [0,1] ,which gives a mean close to 0.5.
We can assume mean of $noise_{normalized}$ to be 0.5 . Comparing it with beta distribution gives us

$$\frac{\alpha}{\alpha + \beta} = \frac{1}{2}$$

$$\alpha = \beta$$

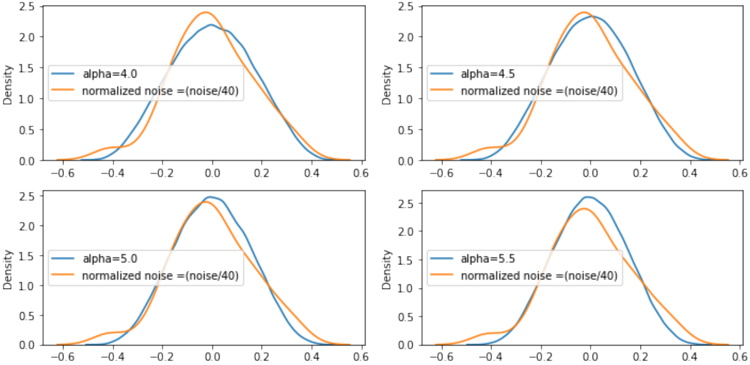
Thus we plot various $\beta(\alpha, \alpha)$ distribution in comparison to our normalized noise distribution , to get best estimate for our normalized noise.



below comparative plot gives us good estimation of our normalized distribution . Since by inspection $\alpha = 4.5$ gives us good estimation. Thus we have

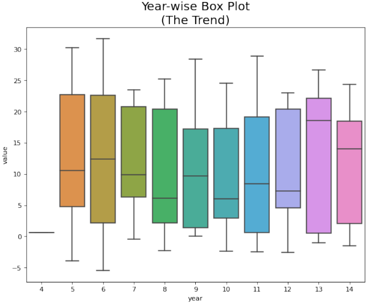
$$noise_{normalized} = \beta(4.5, 4.5)$$

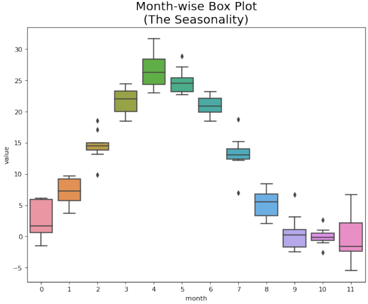
$$noise = 40 * \beta(4.5, 4.5)$$



Part-2 (estimating time series)

Visualizing the trends and seasonality of the series. The box plot tells that there is not much variation of the series over the years, but is periodic every 12 months. Observation : The range of values in a month is of length at max 5 . Thus we can model each month separately assuming Gaussian distribution of hypothesis error.





Since periodic ,the prediction of value would be linear combination of given month's value. We train a model for each month , by fitting a polynomial over the index value of dates with same month.

$$indexval_{date} = (month - 1) + (year - 2000)$$

for example 1 jan 2020 , we will have $0 + 20 = 20$ as its index value.

Training Using above index value for each date is calculated . Then we train the dates with same months with input as their index value and output as the label. Polyfit is implemeted on these models.

Predictions

Depending on the month , the value is predicted on date's month

The models were trained using regularization , coupled with cross-validation. Copying all images for all 12 models to this presentation would be an overkill , thus I haven't included them in the presentation. Although there is Jupyter notebook with name **P2.ipynb** , where the models trained are represented. The final hyper paramter used were

month	degree	lmda
jan	2	1e1
feb	1	1e0
march	1	1e-1
april	2	1e-1
may	3	1e-3
june	1	1e-1
july	1	1e-1
august	1	1e-3
sept	1	1e0
oct	2	1e4
nov	2	1e2
dec	1	1e3

The End