MATH 6350

STATISTICAL LEARNING AND DATA MINING

FINAL PROJECT ON

KERNEL RIDGE REGRESSION ANALYSIS OF STOCK MARKET PRICES OF FINANCIAL SECTOR COMPANIES

SUBMITTED BY

KISHORE TUMARADA

PART-I

Question 1 : Download a real data set DS

1.1 DS must include N cases X(1)... X(n) with n moderately large (800 < n <1500).Each case X(j) can be viewed as vector in Rp described by p features X1(j)... Xp(j) which are the "explanatory variables". Each case X(j) is associated to an observed value Yj of the target variable Y.Each feature must be a "continuous" variable; Avoid or eliminate discrete features taking only a small number of values ; make sure that p ≥ 10 in your Data Set and make sure to include an artificial feature Xp(j) =1 for all cases j=1...n. the main technical goal is to apply Kernel Ridge Regression (KRR) to predict the value of Y whenever a new case X = [X1... Xp] is given.

1.2 Describe your data set as concretely as possible

Answer : The dataset chosen is stock market closing price of following 10 companies to predict 11th company's stock market closing price :

S No	Name	Type
1	The Bank of New York Mellon Corp.	Explanatory variable(X1(j))
2	Bank of America Corp	Explanatory variable(X2(j))
3	Capital One Financial	Explanatory variable(X3(j))
4	Citigroup Inc.	Explanatory variable(X4(j))
5	JPMorgan Chase & Co.	Explanatory variable(X5(j))
6	Morgan Stanley	Explanatory variable(X6(j))
7	S&P Global, Inc.	Explanatory variable(X7(j))
8	Charles Schwab Corporation	Explanatory variable(X8(j))
9	Discover Financial Services	Explanatory variable(X9(j))
10	Franklin Resources	Explanatory variable(X10(j))
11	Principal Financial Group	Target Variable (Y(j))

This data has been extracted from dates : '2015-01-01' to '2019-11-15' – 1228 cases.
These companies are from financial sector.

1.3. Explain the practical impact of obtaining good predictions of Y when the explanatory variables are known.

Answer:

Good predictions of Y helps in understanding the dependency of 11th stock on remaining 10 stocks on any given day.

1.4. for each feature X1 X2 ... Xp, compute and display its mean and standard deviation

Answer:

An artificial feature X11(j) = 1 has been added as a explanatory variable.

Descriptive statistics of explanatory variables are as follows:

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11
count	1228	1228	1228	1228	1228	1228	1228	1228	1228	1228	1228
mean	46.38	22.93	83.77	60.13	88.07	40.98	152.27	39.4	65.51	38.4	1
std	5.935	6.336	9.797	10.28	21.58	8.319	50.245	8.831	9.956	6.782	0
min	32.74	11.16	58.15	34.98	53.07	21.69	80.77	22.22	43.25	25.93	1
Median	46.64	23.89	83.79	60.91	90.56	42.48	145.65	40.11	64.84	37.38	1
max	58.42	33.26	105.7	80.08	130.4	58.91	267.75	59.59	92.91	55.49	1

1.5. same question for Y .

Answer:

Descriptive statistics of explanatory variables are as follows:

	Y
count	1228
mean	54.42
std	8.107
min	34.34
Median	54.27
max	75.04

1.6. Split the data set DS into a training set TRAIN and a test set TEST, with respective proportions 80% , 20%.

Answer:

Following are the descriptive statistics of Train Data:

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	Y
count	982	982	982	982	982	982	982	982	982	982	982	982
mean	46.47	22.95	83.7	60.14	88.11	41	152.2	39.48	65.43	38.35	1	54.46
std	5.946	6.363	9.845	10.33	21.56	8.414	49.596	8.851	9.875	6.616	0	8.127
min	32.74	11.16	58.15	34.98	53.07	21.69	80.77	22.22	43.25	25.93	1	34.34
Median	46.76	23.95	83.57	61	90.71	42.5	146	40.18	64.51	37.36	1	54.29
max	58.42	33.26	105.7	80.08	130.4	58.91	264.83	59.59	92.91	55.49	1	75.04

Following are the descriptive statistics for Test set:

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	Y
count	246	246	246	246	246	246	246	246	246	246	246	246
mean	46.03	22.9	84.03	60.06	87.9	40.9	152.54	39.09	65.82	38.6	1	54.28
std	5.889	6.24	9.618	10.1	21.69	7.94	52.863	8.761	10.28	7.419	0	8.043
min	33.3	11.9	60.89	37.41	54.38	22.7	82.38	22.7	44.52	26.13	1	35.38
Median	46.1	23.4	84.6	60.61	88.21	42.2	138.03	39.67	66.03	37.53	1	54.05
max	57.89	33.2	105.5	78.62	130	57.4	267.75	58.82	92	54.4	1	73.97

1.7 Compute the empirical correlations cor(X1, Y) ... cor(Xp,Y) and their absolute values C1 ... Cp

Answer:

Empirical correlations are:

	Correlation Cor(Xi,Y)
X1	0.739
X2	0.634
X3	0.720
X4	0.798
X5	0.565
X6	0.826
X7	0.380
X8	0.665
X9	0.559
X10	0.255
X11	0
Y	1

1.8 compute the 3 largest values among C1 ... Cp, to be denoted Cu > Cv > Cw which are

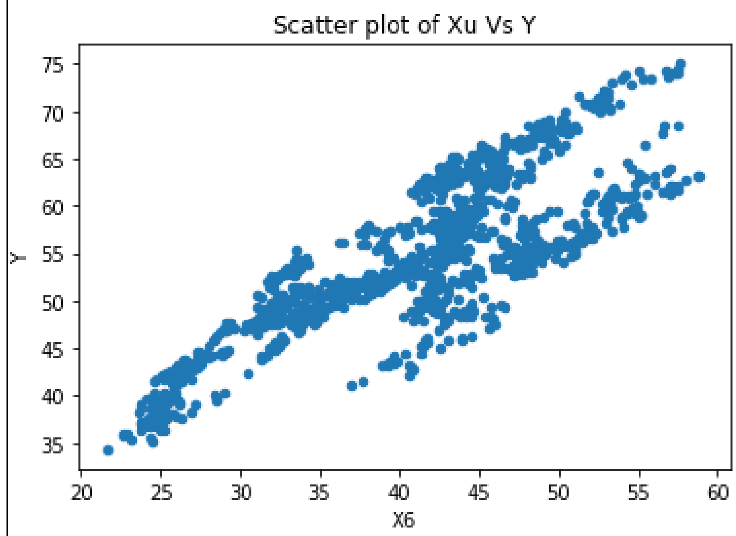
Answer:

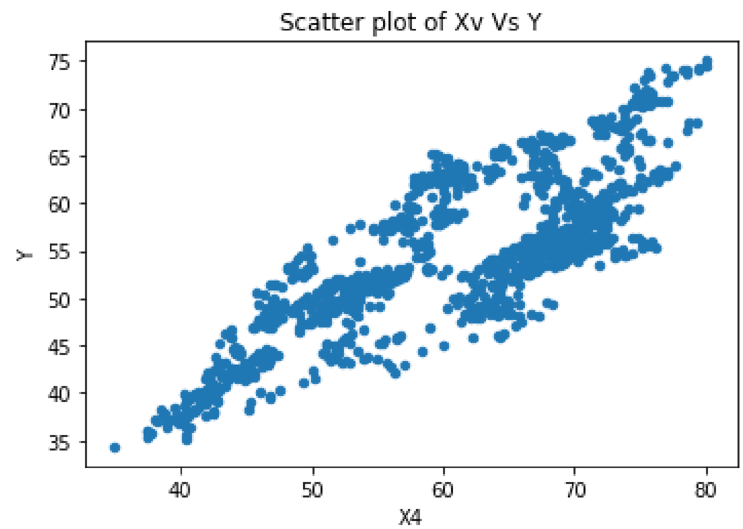
In the above table, 3 largest values of correlations are highlighted and are shown below as well :

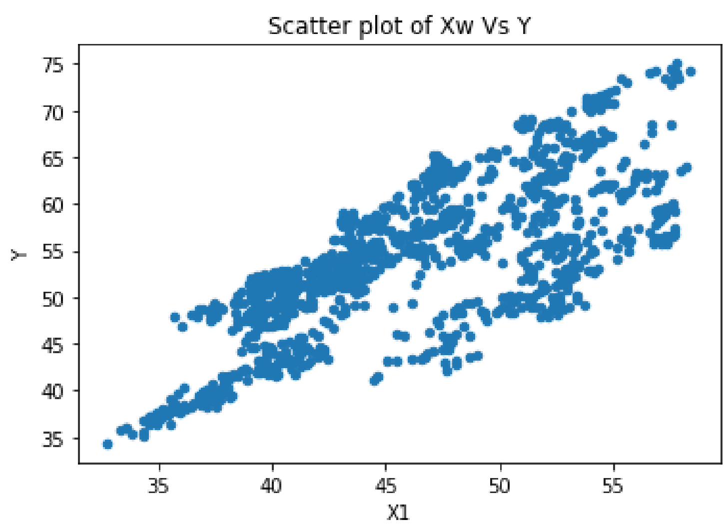
1. Cu = Y vs X6 = 0.826
2. Cv = Y vs X4 = 0.798
3. Cw =Y vs X1 = 0.739

1.9 display separately the 3 scatter plots (Xu(j), Yj) , (Xv(j), Yj) , (Xw(j), Yj) where j= 1...n

Answer:







1.10. Interpret the results

Answer:

The above scatterplots indicate that plot Xu Vs Y has relatively higher number of observations along the line of identity and less dispersion. This dispersion trend increases from Xu to Xv to Xw. This implies that there is high correlation in closing stock prices between company X6 and target company Y.

Question 2: Kernel Ridge Regression (KRR) with radial kernel

For this question we use intensively the training set TRAIN which has size m = 80% n

The m cases in TRAIN are (after reordering of their indices) denoted X(1)... X(m) to simplify notations . Select the kernel = "radial "kernel K(x,y) defined for x and y in Rp by the formula

$$K(x,y) = \exp(-\text{gamma } || x- y ||^2)$$

where gamma >0 is a parameter to be selected later

The KRR approach involves also a cost parameter 1/λ which roughly evaluates the cost of a prediction error. The parameter λ >0 will also have to be selected later

Once " λ " and "gamma" are selected , the best KRR prediction function pred(x) is defined for any input vector x in Rp by the formula

$$\text{pred}(x) = y\,(G + \lambda\, Id)^{-1}\, V(x)$$

where:
y= [Y1 ...Ym] is a line vector
Id = mxm identity matrix
V(x) is a *column* vector with m coordinates V1(x) , ... , Vm(x) given by Vj(x) = K(x, X(j))
the mxm matrix G is the kernel gramian G= [Gij] with G(i,j) = K(X(i), X(j)) for all i ,j in [1...m]

2.1. Compute the matrix G and its eigenvalues L1 >L2 > ... > Lm ≥ 0

Answer:

As sample gamma is chosen as 0.01, radial kernel formula would be as below :

$$\boldsymbol{K(x,y) = e^{-0.01*(\|x-y\|)^2}}$$

In general, Gramian matrix G of a set of vectors $x_1, x_2 \dots \dots \dots . x_m$ in an inner product space is the Hermitian matrix of inner products, whose entries are given by

$$G = [G_{ij}]\; with\; G(i,j) = \prec xi, xj \succ$$

In case of kernel gramian matrix,

$$G = [G_{ij}]\; with\; G(i,j) = K\big(X(i), X(j)\big)$$
$$for\ all\ cases\ X(k)\ in\ Train\ data\ with\ m\ cases.$$

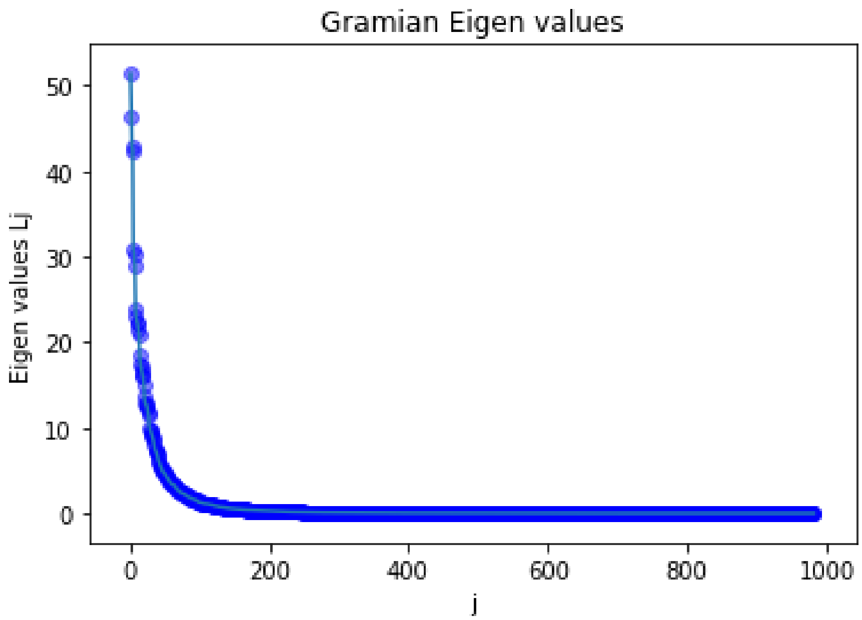
In this dataset, train data has 982 features, hence m = 982 and matrix G has 982 X 982 dimensions.
Eigen values Li of matrix G has following features :

- Max value = 51.425
- Min value = 0

- 982 eigen values

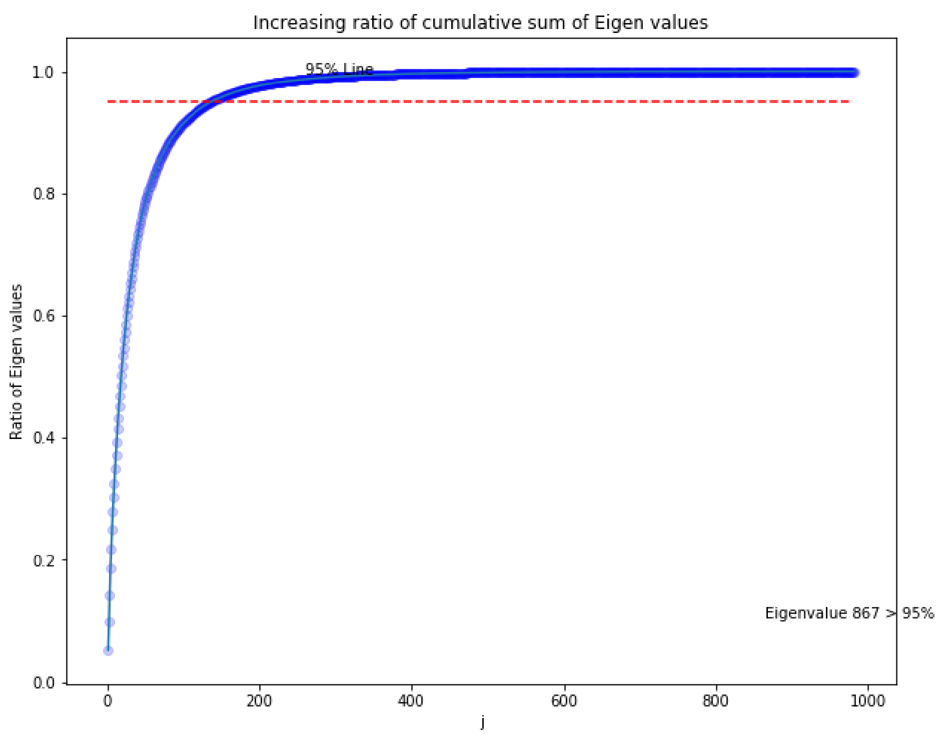
2.2. Plot Lj versus j

Answer:



2.3. Plot the increasing ratios RATj= (L1 + ... + Lj)/(L1+ ... + Lm)

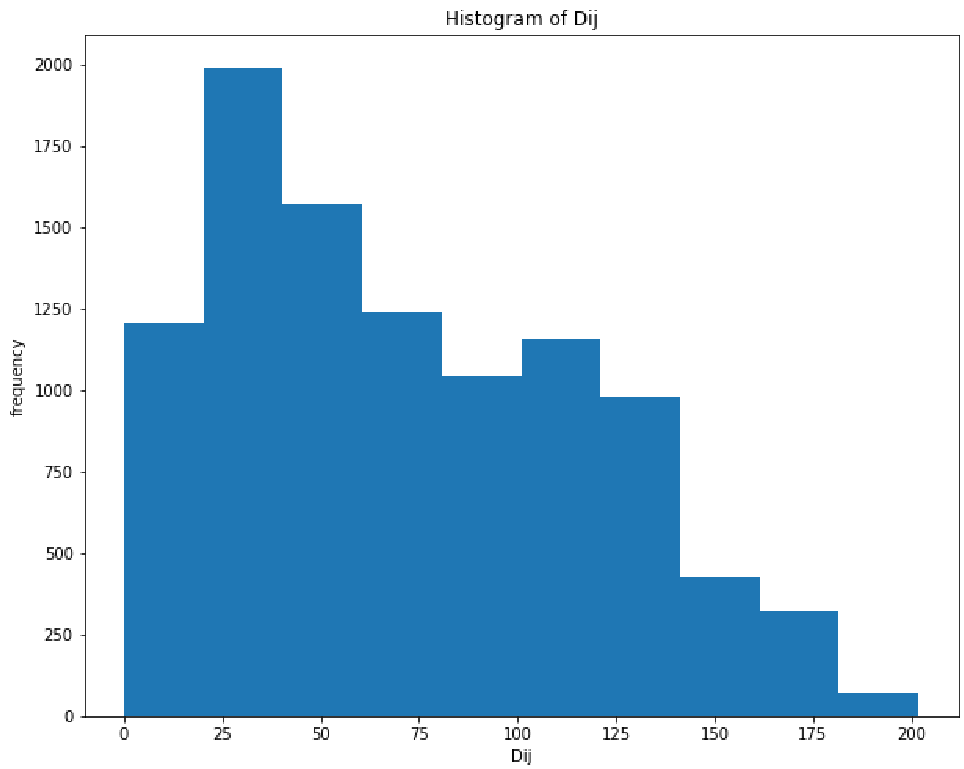
Answer:



2.4. Identify the smallest j such that RATj ≥ 95% and set λ = Lj
Answer:

For a Ratio > 95%, Smallest eigenvalue is 0.657, with j =137.
Hence lambda , $\boldsymbol{\lambda = 0.657}$.

2.5. Select at random two lists List1 and List 2 of 100 random integers each , within [1...m]
2.6. For all i in List 1 and all j in List2 compute Dij = ||X(i) -X(j)||
2.7. Plot the histogram of the 10000 numbers Dij
Answer:



2.8. Compute q =10% quantile of the 10000 numbers Dij. Set gamma = 1/q
Answer:
10% quantile value, q = 18.282.

Hence gamma value is set as :

$$\boldsymbol{\gamma = \frac{1}{q} = 0.0547}$$

2.9. Compute the matrix M = G + λ Id and its inverse M^{-1}
Answer:
New Gramian is again calculated with above gamma value, $\boldsymbol{\gamma = \frac{1}{q} = 0.0547}$

Matrix M is calculated with above lambda value calculated, $\boldsymbol{\lambda = 0.657}$.

As Gramian is 982X982 matrix(since m = 982), Matrix M and its inverse are also of same dimensions.

2.10. The prediction formula becomes pred(x) = A1 K(x, X(1)) + ... + Am K(x,X(m)). compute the line vector A = [A1 ... Am] by A= y M^{-1} .

Answer:

Here y is the matrix with target variable values in Train data and M^{-1} is also determined by train data's explanatory variables X(1) to X(982) with 11 features each.

Hence the whole line vector A is based on Train dataset.

2.11. Compute the RMSEtrain of the prediction function pred(x) by running it on all x in TRAIN set. Compute their ratios RMSE/ avy where avy = mean of the m absolute values |Y1|, ... , |Ym|.

Answer:

$$pred(x) = y * (G + \lambda * Id)^{-1} * V(x)$$

After calculation of $M^{-1} = (G + \lambda * Id)^{-1}$ and further line vector $A = y * M^{-1}$, the equation can be simply written as :

$$pred(x) = A * V(x)$$

where:

A = [A1 ...Am] is a line vector

V(x) is a *column* vector with m coordinates V1(x) , ... , Vm(x) given by

$$V_j(x) = K\big(x, x(j)\big) ---------- 2.11.1$$

with x(j) as train set cases and x is the data whose target variable has to be predicted.

$$RMSE = \sqrt{\frac{\sum_1^k (y - \hat{y})^2}{k}}$$

Where, y =true values of target variable

$\hat{y}$ = predicted values of target variable

K = number of cases

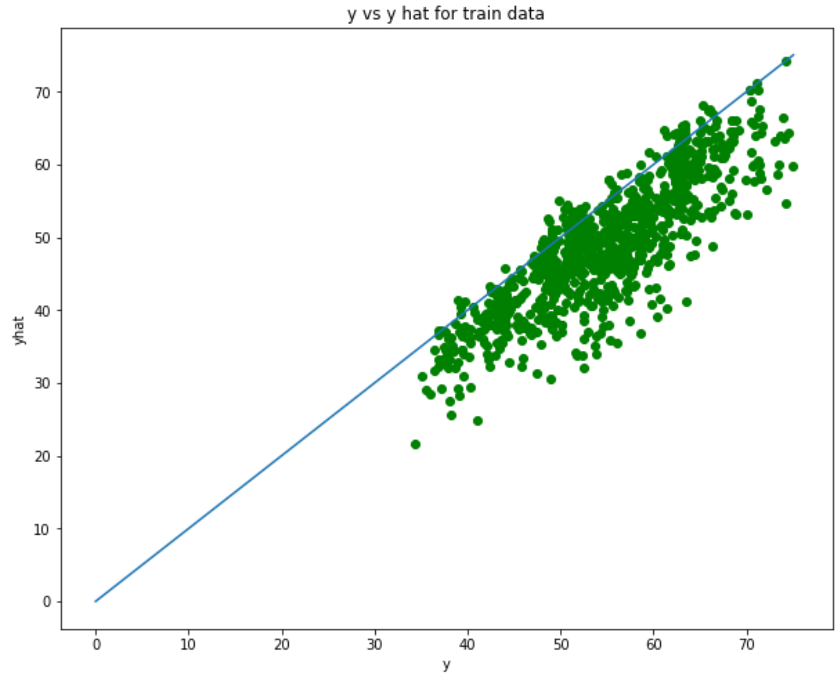
RMSE can be normalized by

$$ratio = \frac{RMSE}{avy},$$

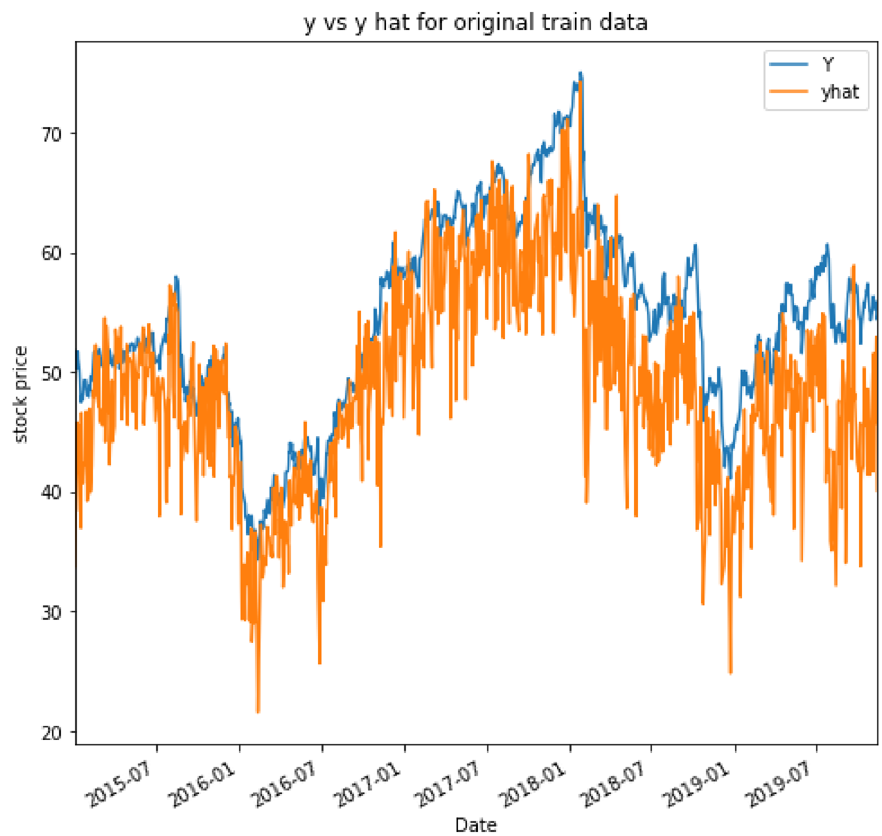
Where ,

$$avy = \frac{\sum_1^k |y_i|}{k}$$

i.e., mean of true values of target variables.



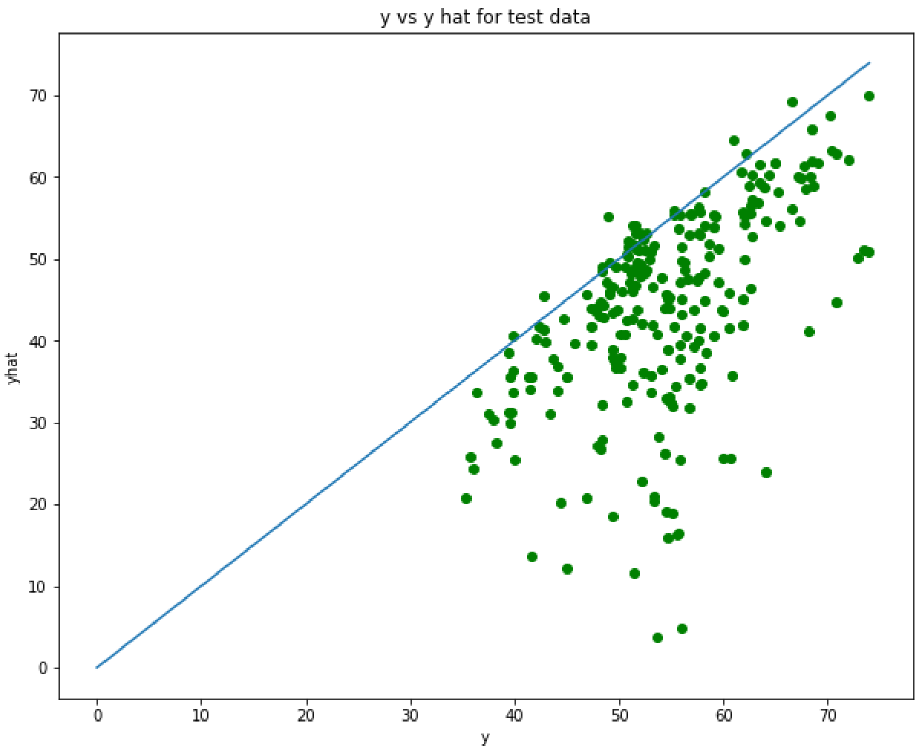
The above plot is scatter plot of true y values and predicted values of y(y hat), along with line of identity for reference. This indicates that predicted values of y are lesser in magnitude than true y. The same is reflected in the time series plot shown below for date against y and yhat.



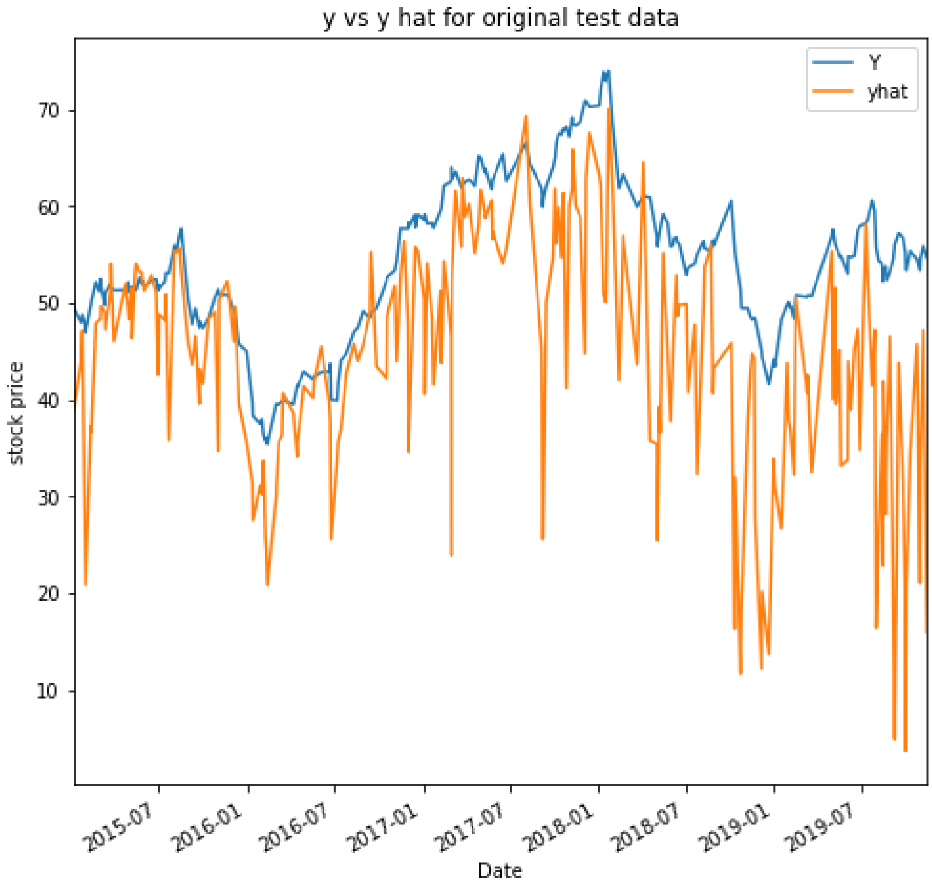
In this question, x in equation 2.11.1 is the Train data itself. Performance measures are RMSE = 7.2758 ratio rmse/avy train: 0.1336. For 95% confidence level, conifdence interval for ratio is [0.1122, 0.155]

2.12. Compute the RMSEtest of the prediction function pred(x) by running it on all x in TEST set

Answer:



In above plot Unlike Train data, test data has many observations dispersed from line of identity. This indicates that its prediction capacity is lower. The below timeseries plot shows that there is a frequent drop in predicted values compared to true y. It performance parameters are RMSE test = 14.6494, Ratio rmse/avy true = 0.2699. For 95% confidence level, confidence interval for ratio is [0.2144, 0.3254]



2.13. Compare these two RMSE values , and compute their ratios RMSE/ avy where avy = mean of the m absolute values |Y1|, ... , |Ym|.

Answer:

val_rmse_train: 7.2758
ratio rmse/avy train: 0.1336
sigma: 0.0109
For 95% confidence level, confidence interval for ratio is [0.1122, 0.155]
val_rmse_test: 14.6494
ratio rmse/avy: 0.2699
sigma: 0.0283
For 95% confidence level, confidence interval for ratio is [0.2144, 0.3254]

When compared to train data , there is rmse ratio of test is almost double, which indicates that there is an overfitting of model for train data for this set of parameters.

Question 3: Improving the results through step by step tuning

3.1. Repeat the preceding operations for other pairs of parameters gamma and λ . Suggestion: change only one parameter at a time to check in which direction to go for improved performances.

Answer:

Methodology followed for tuning of parameters is :

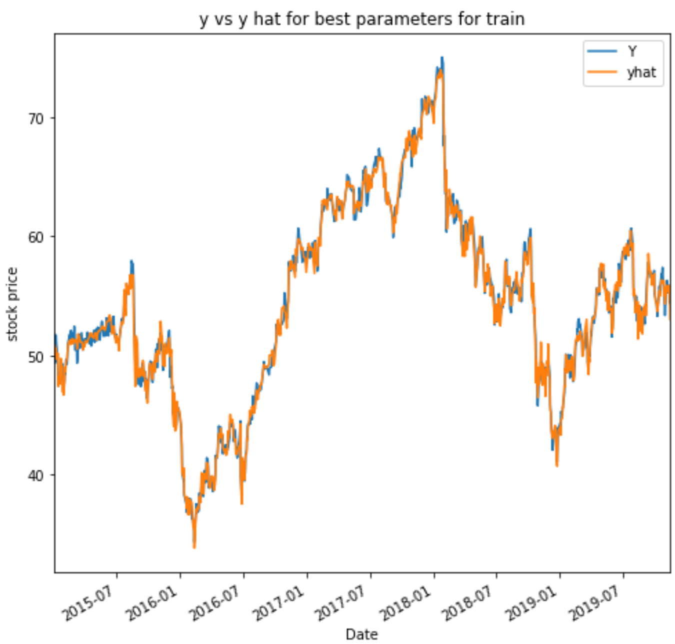
1. Parameters calculated in Question 2 is taken as baseline values, which are $\lambda = 0.657, \gamma = 0.0547$
2. In each step, only one of the parameter is tuned with 2 combinations –
   a. When $\lambda = \lambda_0, \gamma = [\frac{\gamma_0}{2}, 2 * \gamma_0]$
   b. Similarly when , $\gamma = \gamma_0, \lambda = [\frac{\lambda_0}{2},\ 2 * \lambda_0]$
3. In each step the combination of parameters, for which rmse for test and train are low and have less difference between the errors , are chosen.
4. In the following table, yellow color indicate the parameters tuned in that run.
5. **Bolded** line indicate the parameters chosen for low rmse ratio and rmse error
6. After 11 iterations, 10th combination of parameters are chosen as best parameters, since it has low rmse ratio, rmse error and least difference between test and train ratios. It is indicated in green color

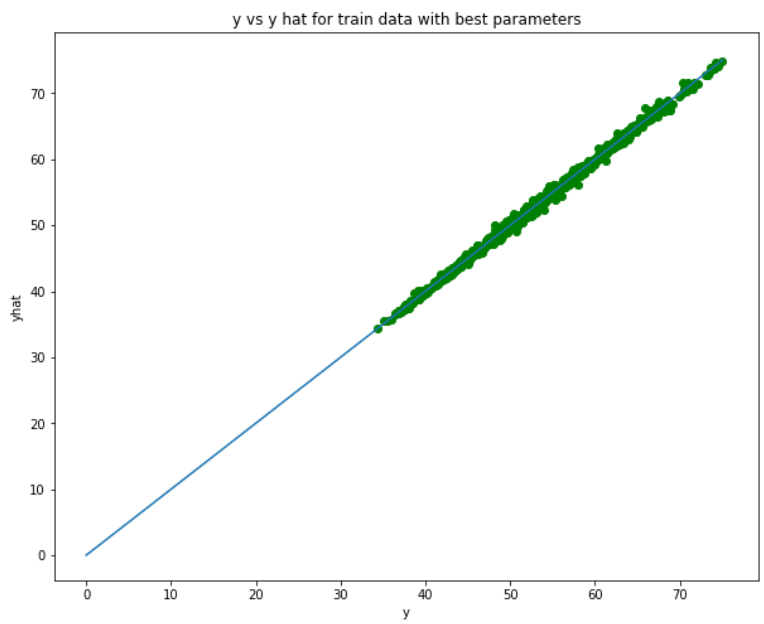
Tune no.	rmse_test	rmse_train	ratio_test	ratio_train	diff_ratio	lambda	gamma
0	14.6496	7.2759	0.2699	0.1336	0.1363	0.657	0.0547
1	**9.25095**	**5.012**	**0.1704**	**0.092**	**0.0784**	**0.657**	**0.0274**
	22.2311	10.09	0.4096	0.1853	0.2243	0.657	0.1094
2	**7.57525**	**3.1865**	**0.1396**	**0.0585**	**0.0811**	**0.3285**	**0.0274**
	11.9053	7.9414	0.2193	0.1458	0.0735	1.314	0.0274
3	**4.63263**	**2.3109**	**0.0854**	**0.0424**	**0.043**	**0.3285**	**0.0137**
	12.5096	4.4224	0.2305	0.0812	0.1493	0.3285	0.0547
4	**3.83787**	**1.6153**	**0.0707**	**0.0297**	**0.041**	**0.1643**	**0.0137**
	5.81744	3.4177	0.1072	0.0628	0.0444	0.657	0.0137
5	**2.41667**	**1.2946**	**0.0445**	**0.0238**	**0.0207**	**0.1643**	**0.0068**
	6.50919	2.0636	0.1199	0.0379	0.082	0.1643	0.0274
6	**2.01723**	**0.9785**	**0.0372**	**0.018**	**0.0192**	**0.0821**	**0.0068**
	2.99213	1.7531	0.0551	0.0322	0.0229	0.3285	0.0068
7	**1.40024**	**0.8636**	**0.0258**	**0.0159**	**0.0099**	**0.0821**	**0.0034**
	3.27669	1.1543	0.0604	0.0212	0.0392	0.0821	0.0137
8	**1.18909**	**0.6973**	**0.0219**	**0.0128**	**0.0091**	**0.0411**	**0.0034**
	1.69536	1.0946	0.0312	0.0201	0.0111	0.1643	0.0034
9	**0.97846**	**0.7291**	**0.018**	**0.0134**	**0.0046**	**0.0411**	**0.0017**
	1.73001	0.7501	0.0319	0.0138	0.0181	0.0411	0.0068

10	**0.86777**	**0.6196**	**0.016**	**0.0114**	**0.0046**	**0.0205**	**0.0017**
	1.13366	0.8723	0.0209	0.016	0.0049	0.0821	0.0017
11	**0.87051**	**0.7567**	**0.016**	**0.0139**	**0.0021**	**0.0205**	**0.0009**
	1.03681	0.5721	0.0191	0.0105	0.0086	0.0205	0.0034

3.2. Select the best choice of parameters in terms of accuracy RMSE/avy and stability of performance when one goes from TRAIN to TEST set.
Answer:

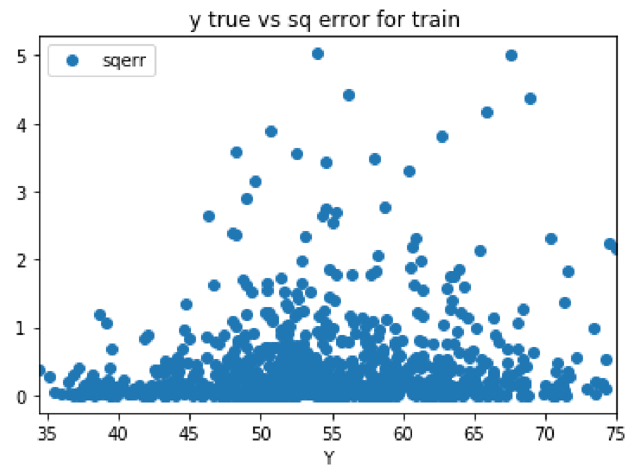
Best parameters are 'lambda': [0.0205], 'gamma': [0.0017]}
Performance on Train :
val_rmse_test: 0.6251
ratio rmse/avy: 0.0115
For 95% confidence level, confidence interval for ratio 0.0115 is [0.0048, 0.0182]



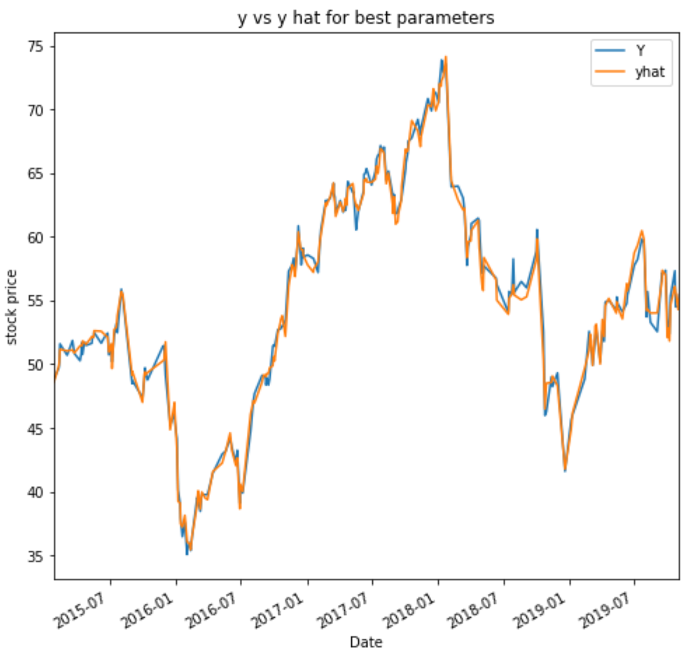


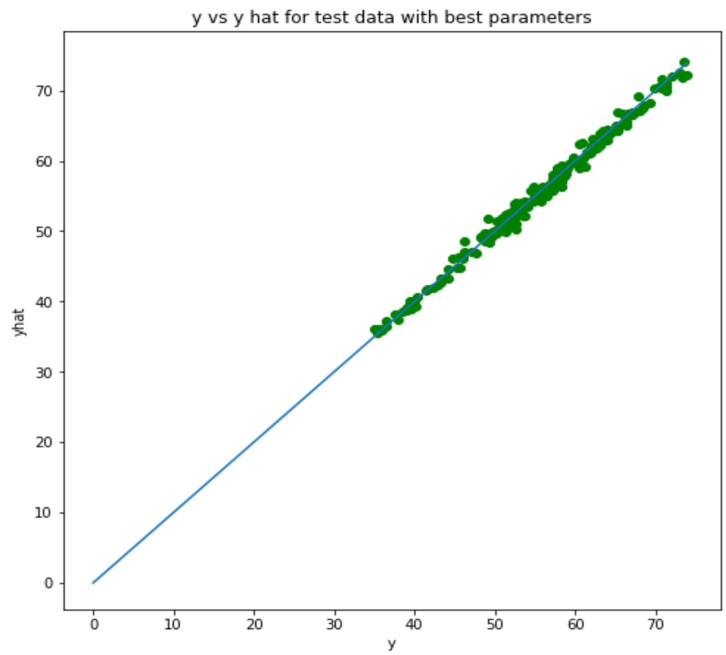
Both of the above plots indicate that the model is able to closely predict the target variable.

The adjacent plot of ytrue vs squared error indicate that error in prediction is higher for higher values of target variable y.



Test Set performance with best parameters



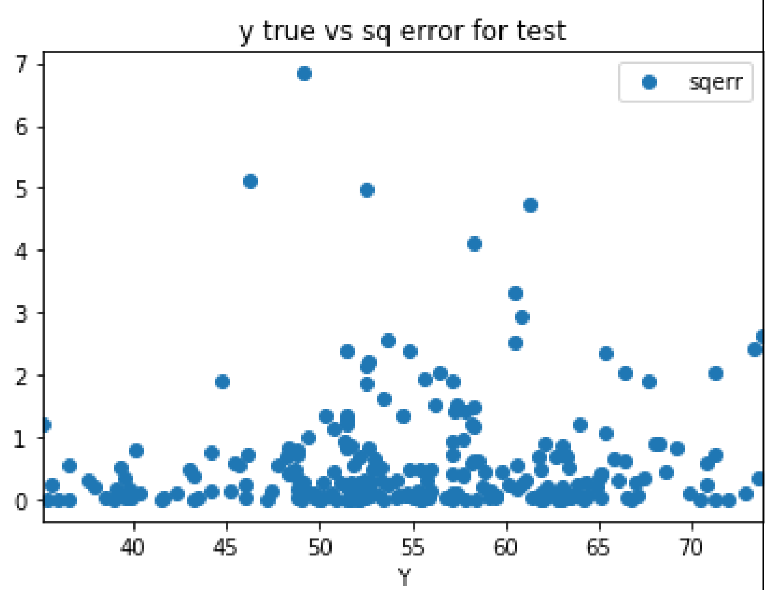


Both the plots indicate that, similar to train data, test data prediction rate is also high. But it's accuracy is low when there is a sudden fall in the price.

Performance parameters for test set:

val_rmse_test: 0.7707
ratio rmse/avy: 0.014
sigma: 0.0075
For 95% confidence level, confidence interval for ratio 0.014 is [0, 0.0287]

When compared to train set's performance with confidence interval of [0.0048, 0.0182], there is an overlap in the confidence intervals. Hence the performance of best parameters model is same on both test and train datasets. Hence, we can say the model is robust and stable performance for both train and test datasets.



3.3. Identify the 10 cases in the TEST set for which the squared prediction error is the largest .

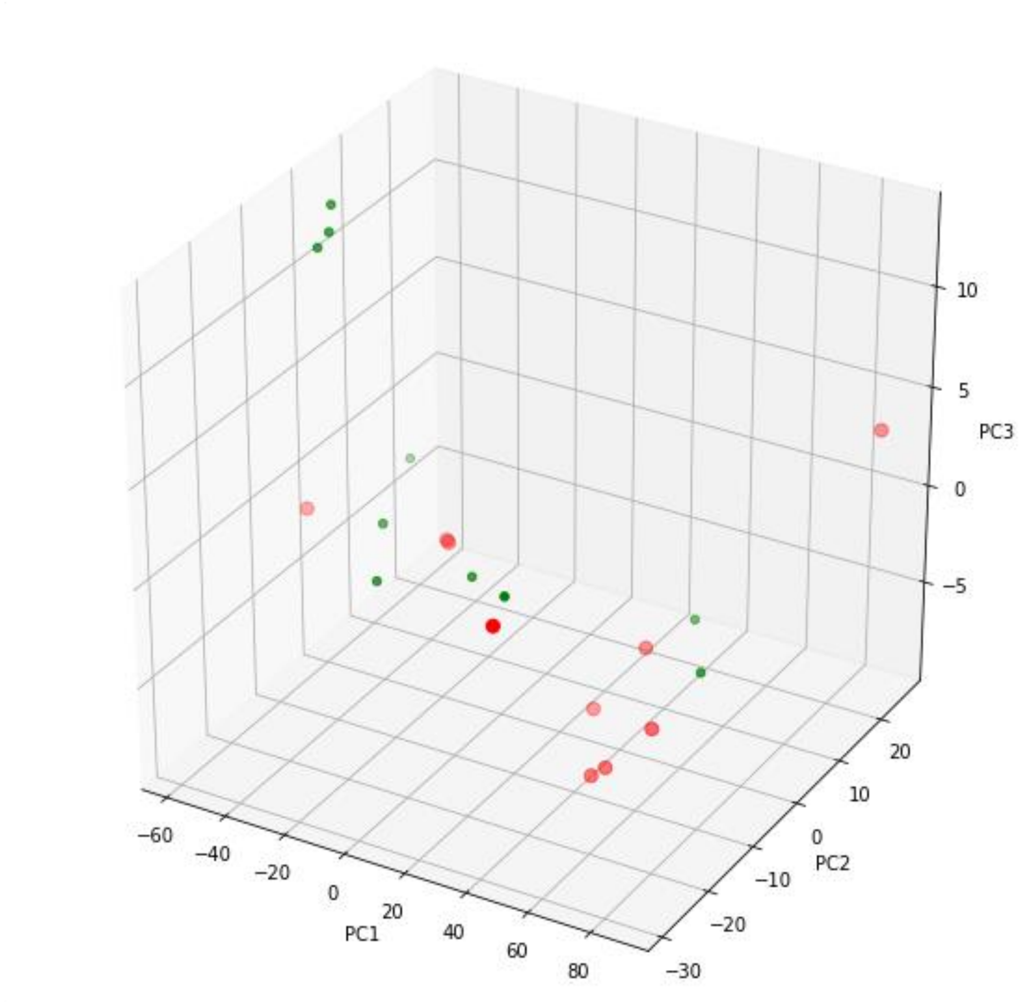
Answer:

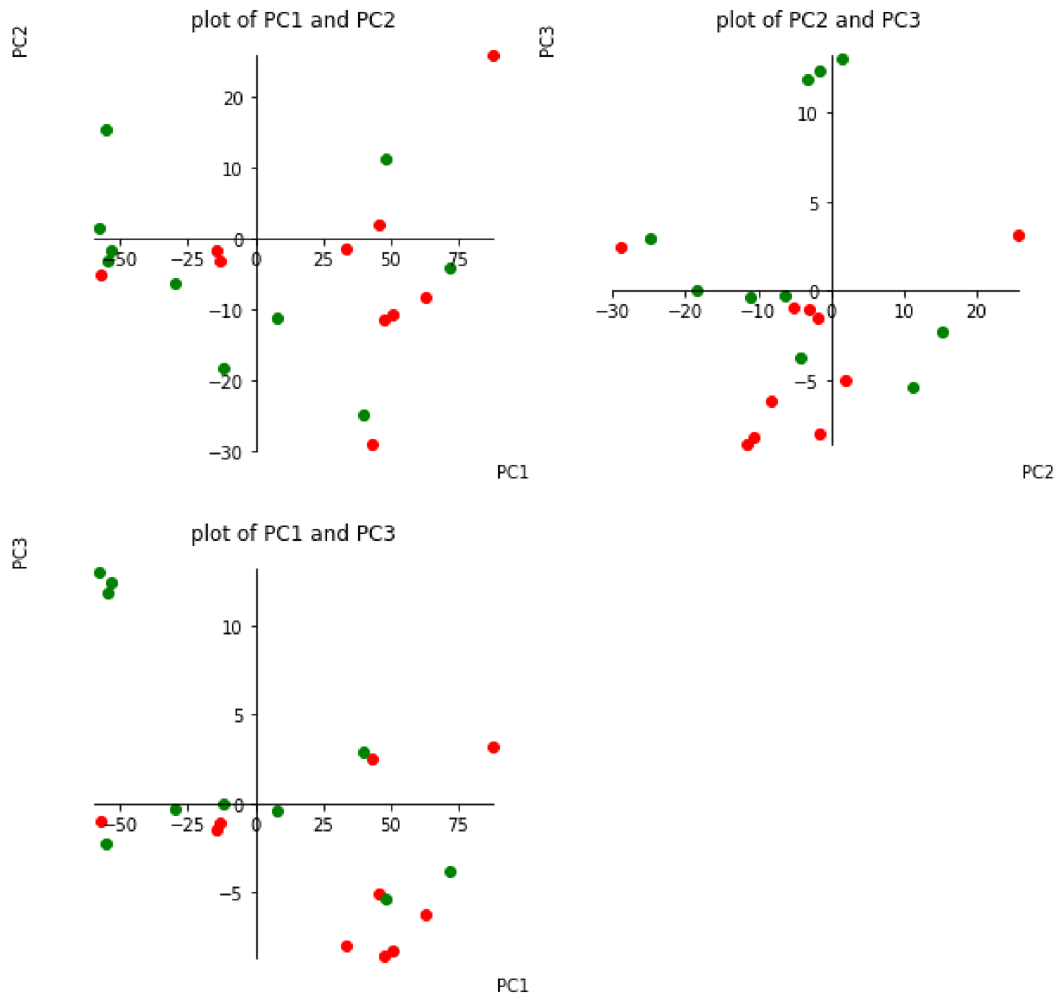
Following are the 10 worst cases with sq err in decreasing order.

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	Y	yhat	sqerr
0	44.06	17.8	79.37	55.09	67.89	35.32	98.33	34.51	56.4	41.14	49.14	51.76	6.8403
1	46.92	26.78	87.84	64.53	106.7	44.5	176.61	45.57	69.18	30.98	46.25	48.51	5.1076
2	46.47	27.38	88.54	66.59	106.4	45.11	188.93	45.6	74.32	29.04	52.54	50.31	4.9684
3	55.44	30.07	91.66	69.18	110.1	52.2	190.25	55.65	72.37	33.78	61.27	59.09	4.735
4	53.46	31.31	93.77	72.29	116.7	50.9	200.88	51.32	71.9	34.31	58.27	56.25	4.1006
5	46.2	22.74	79.27	60.08	83.96	41.31	137.45	38.33	59.28	41.87	60.55	62.37	3.3295
6	46.54	23.05	80.27	61.1	84.78	41.8	138.14	38.26	60.11	42.04	60.83	62.54	2.9279
7	57.83	30.66	104.4	75.56	110.8	54.2	176.4	54.17	80.36	43.94	73.88	72.26	2.6338
8	42.49	27.84	86.25	66.26	113.3	40.23	236.44	36.51	76.81	27.06	53.71	52.12	2.5427
9	55.32	30.15	91.72	68.99	109.4	51.86	187.07	55.99	71.22	34.28	60.46	58.88	2.5062

3.4. Vizualise the 10 cases by performing a PCA analysis and projecting all the TEST cases onto the first 3 principal eigenvectors of the PCA correlation matrix .

Answer: PCA analysis of 10 worst cases and Visualize 10 worst cases(in red color) against 10 best cases(green colors) :





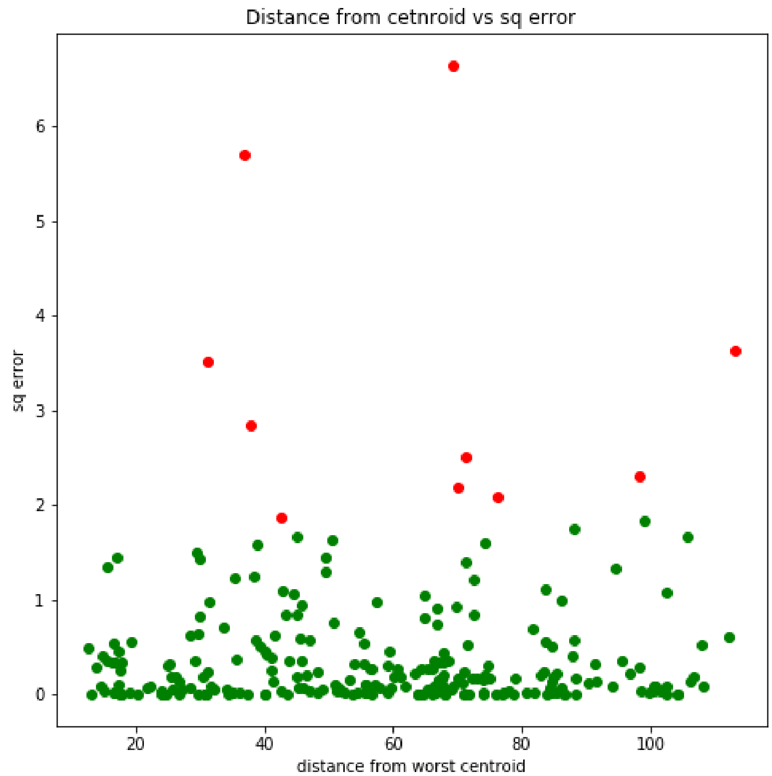
But PCA analysis doesn’t show any trend in worst cases compared to best cases.

3.4. Try to identify what went wrong with the prediction of the absolute worse case X(w) by looking at the terms involved in pred(Xw) and comparing to another case where the prediction error is really small.

<u>Answer:</u>

<u>Analysis 1:</u>

Centroid of 10 worst cases is calculated and distance of all points to centroid against squared error is plotted as below:



Above plot shows that worst points are spread from very near to their centroid to far from it. This plot also doesn't show any trend.

Analysis 2 :

For pred(x) = A1 K(x, X(1)) + ... + Am K(x,X(m)) = U1 + ... +Um , the list LIST(x) of positive numbers is V(1)= |U1| ... V(m) = |Um|.

Then the sublist LIST5(x) of the 5 largest numbers in LISTx is found as indicated below :

V(m1) > V(m2) > V(m3)) > V(m4) > V(m5)

For 10 worst cases x = worst test case to get [m1 m2 m3 m4 m5]

List of indices m1 to m5 for 10 worst cases :

S No	m1	m2	m3	m4	m5
0	131	130	194	126	195
1	778	611	771	770	762
2	771	758	762	770	772
3	640	771	758	656	611
4	736	709	640	694	771
5	492	483	476	482	493
6	492	483	476	482	493
7	606	604	605	603	587
8	892	899	893	929	921
9	771	640	611	656	758

For 10 best test cases with least error, [M1 M2 M3 M4 M5] indices have been obtained.

List of indices M1 to M5 for 10 best cases:

S No	M1	M2	M3	M4	M5
0	131	130	43	41	132
1	736	737	683	694	709
2	131	303	132	130	194
3	436	492	422	432	476
4	130	131	126	43	194
5	422	387	414	398	372
6	130	131	43	126	35
7	562	560	540	492	544
8	836	835	846	821	824
9	606	611	604	587	605

On comparision of [m1 m2 m3 m4 m5] and [M1 M2 M3 M4 M5] , best cases involve smaller indices of pred(x) and worst cases involve larger indices.

Question 4 : Analysis of the best predicting formula pred(x)

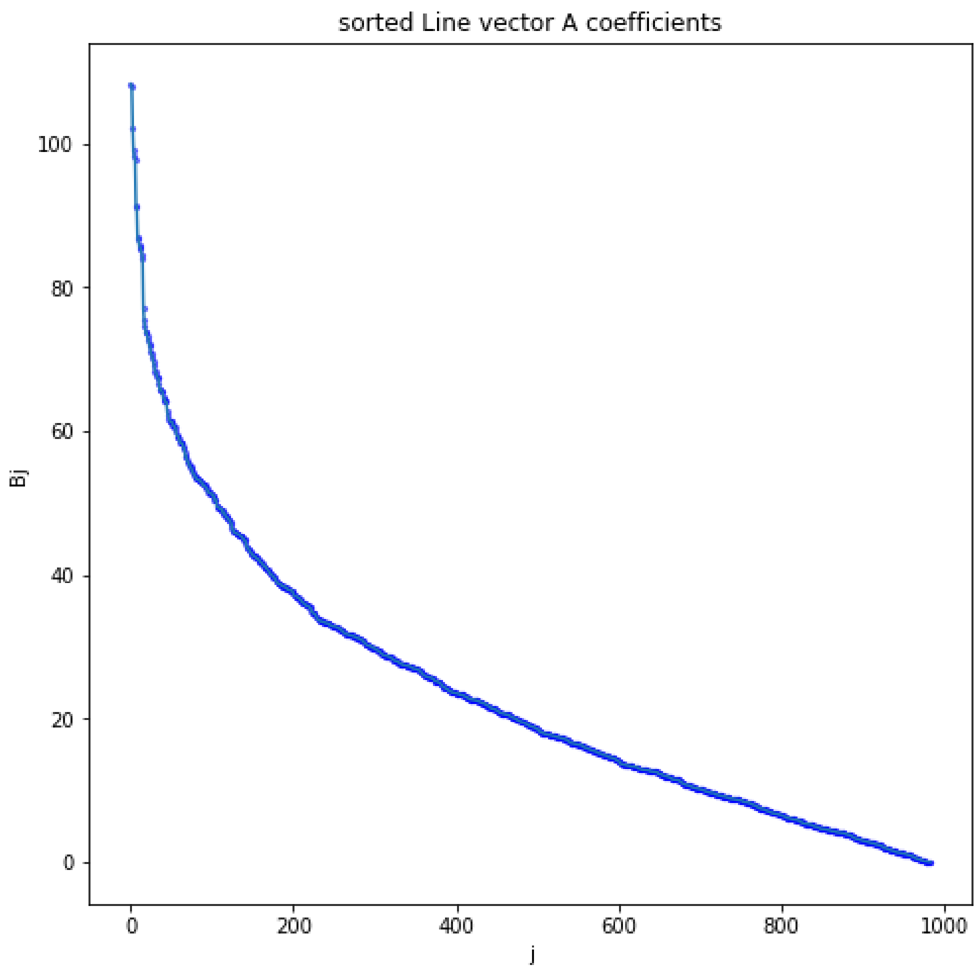
4.1. Fix the best choice of parameters as found in the preceding question.

Answer:

Best parameters chosen are {'gamma': [0.0017], 'lambda': [0.0205]}.

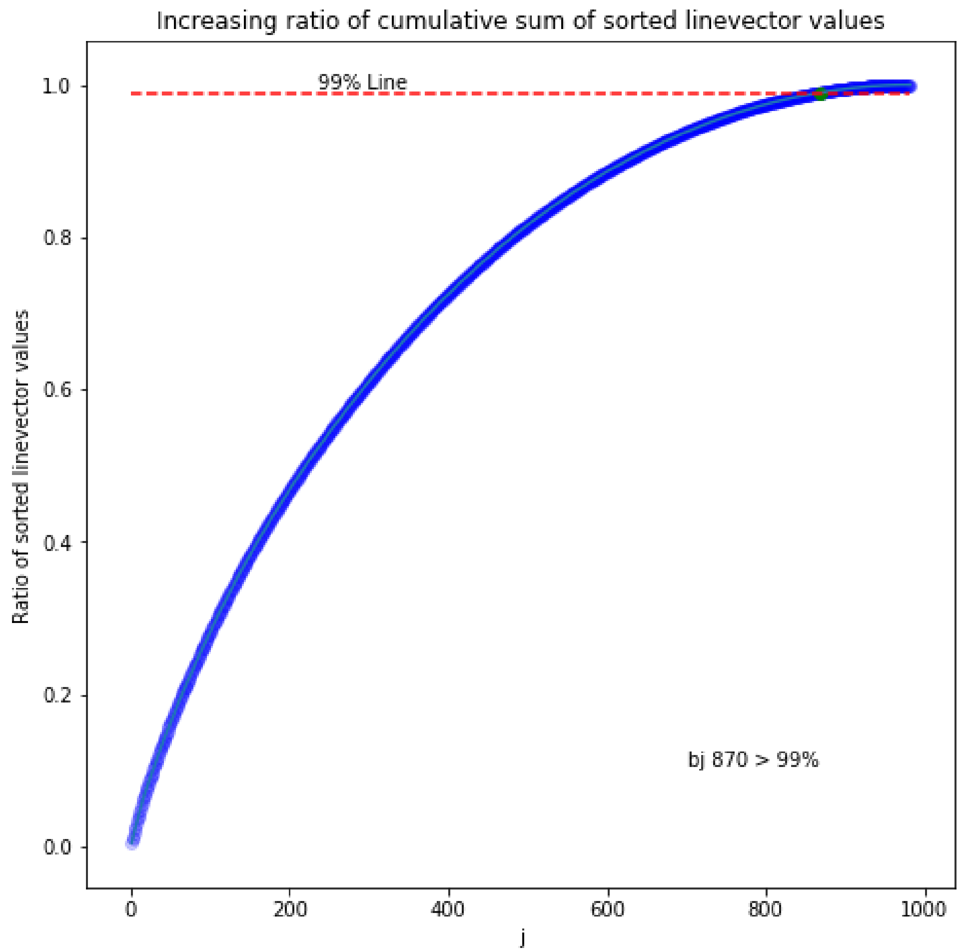
4.2. Reorder the |A1|, |A2|, ...|Am| in decreasing order , which gives a list B1 > B2 ... > Bm >0 and plot the decreasing curve Bj versus j.

Answer:



4.3. Compute the ratios bj = (B1 + ... + Bj)/(B1 + ...+Bm)and plot the increasing curve bj versus j.

Answer:

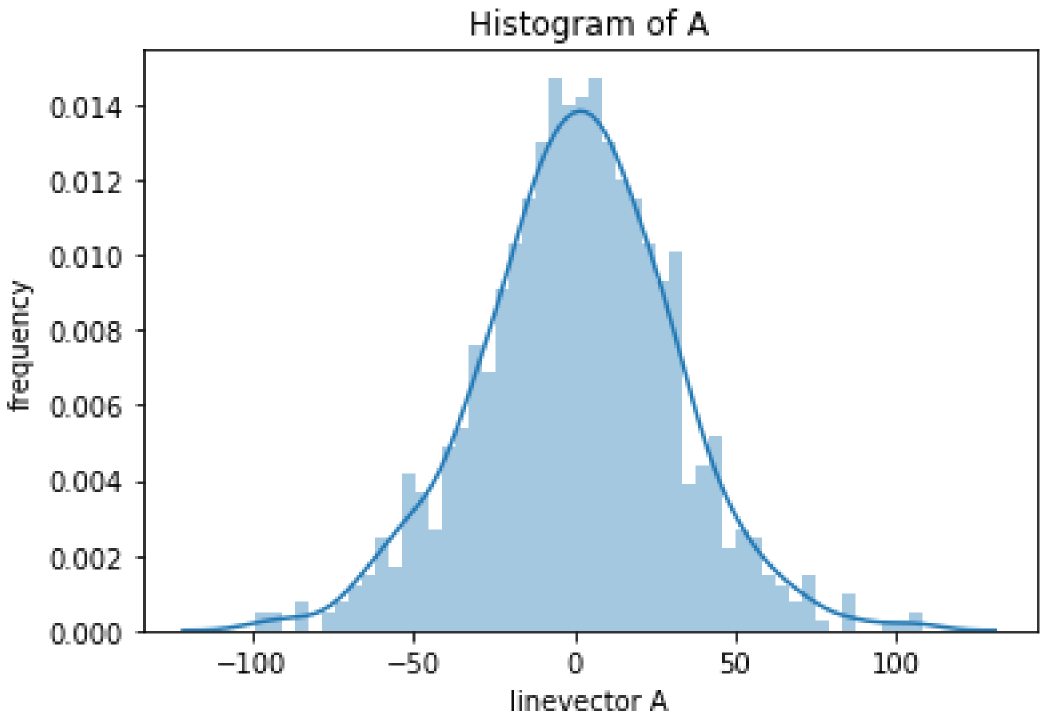


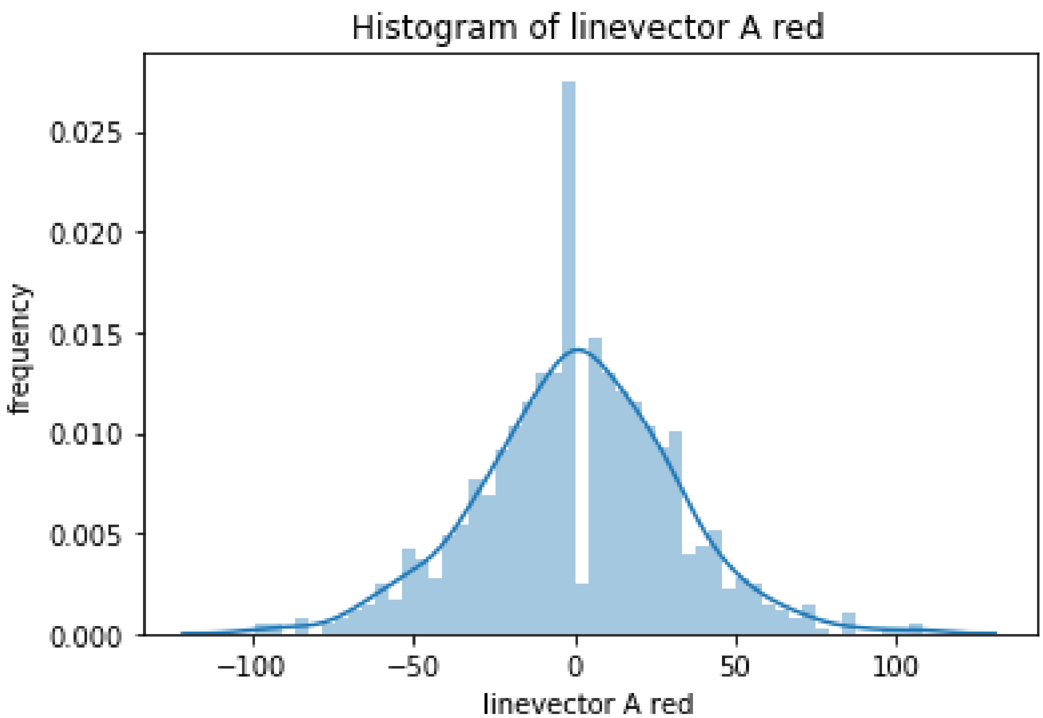
4.4. Compute the smaller j such that bj > 99%. and the corresponding threshold value THR = Bj .
Answer:

At linevector coef bj 870 THR = Bj = 4.15 we get a ratio of 99.01%.

4.5. For i =1... m, if |Ai|> THR set AAi = Ai and otherwise set AAi = 0. This yields a reduced formula PRED(x) = AA1 K(x, X(1)) + ... + AAm K(x,X(m)).

Answer:

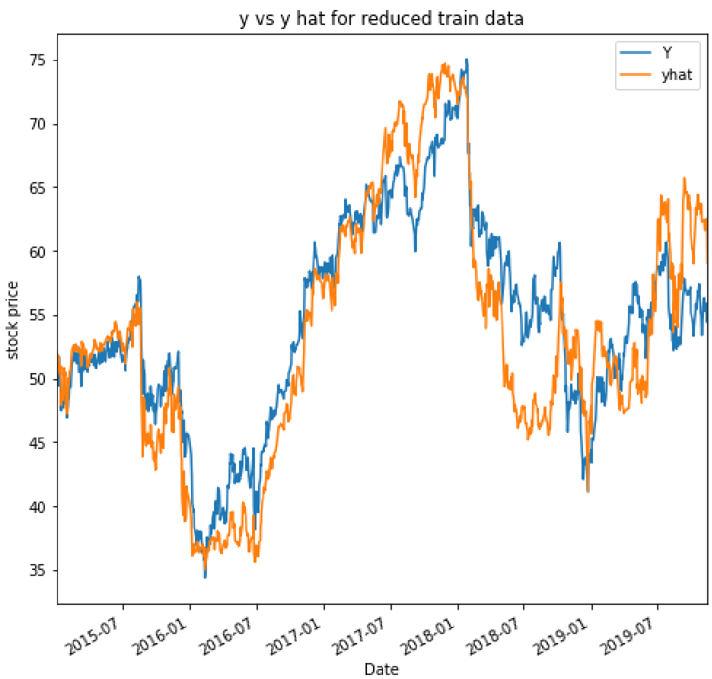


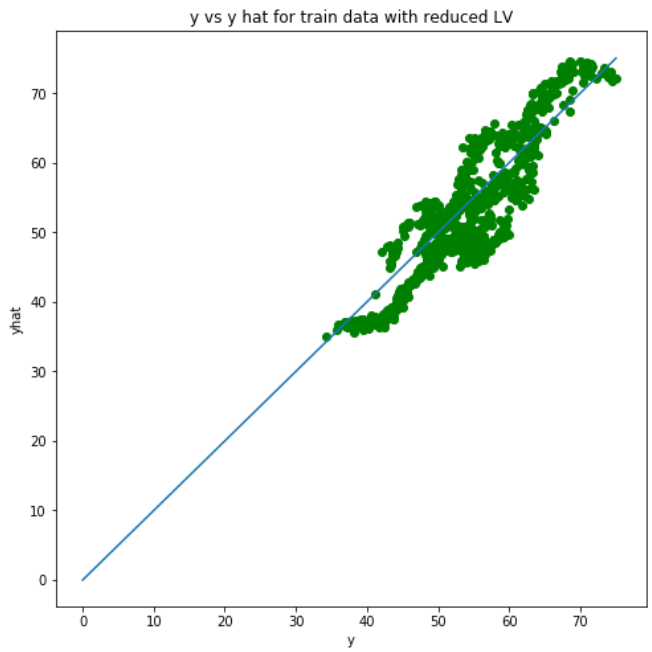


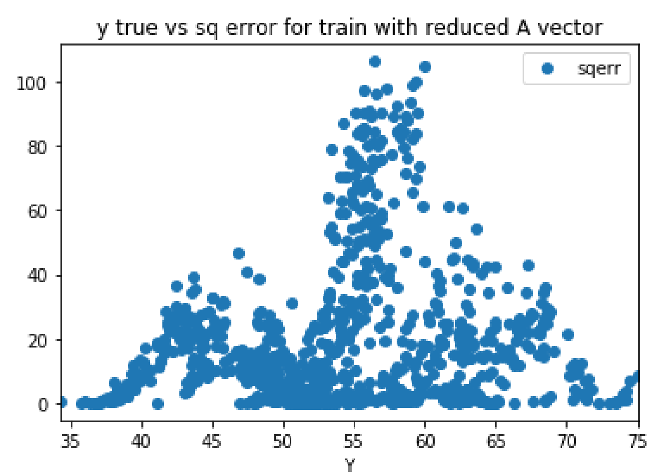
Number of non-zero Ais in reduced linevector A are: 870 out of 982.

4.6. Run this reduced formula on the TRAIN and TEST sets to evaluate its performances.

Answer:





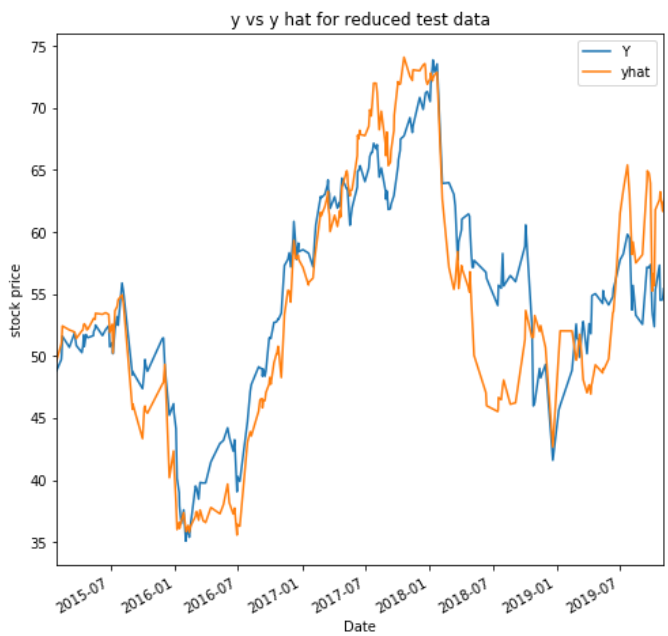


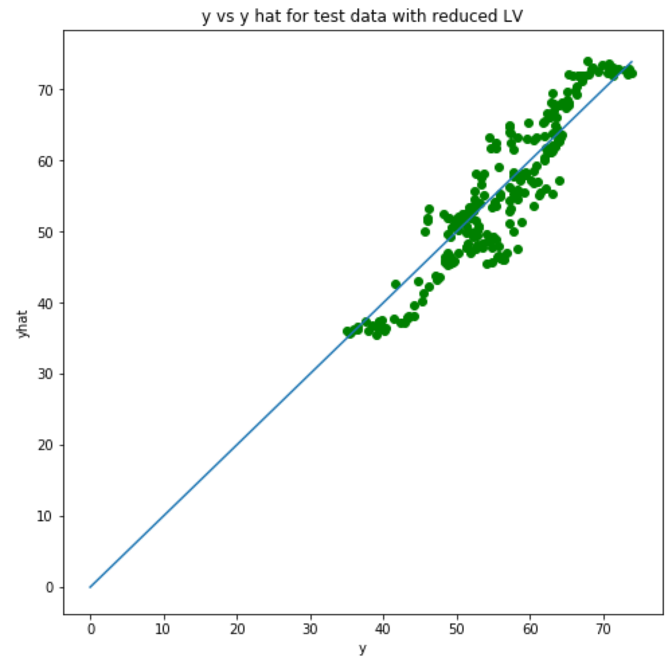
Performance of reduced line vector on Train data:

1. val_rmse_test: 4.313 ;
2. ratio rmse/avy: 0.0795
3. sigma: 0.0086

For 95% confidence level, confidence interval for ratio 0.0795 is [0.0626, 0.0964]

True y and sq error plot, adjacent figure, indicates that error is high in the target varibales in the middle region.

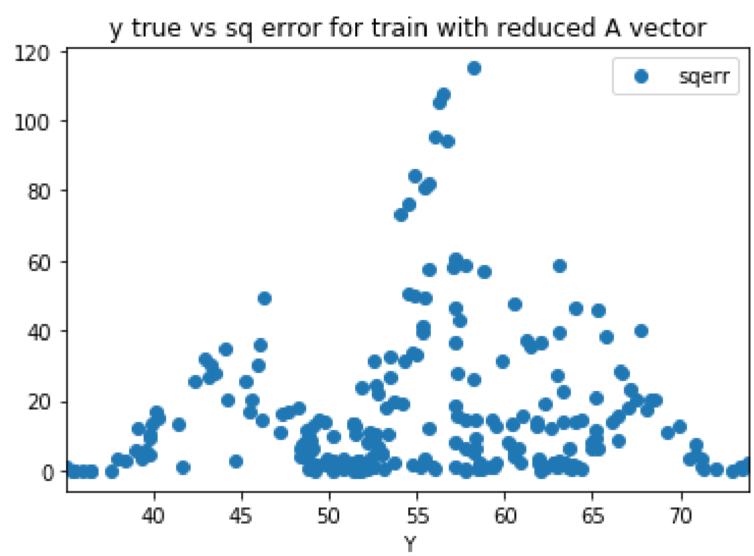




Performance of reduced linevector on Test data:

1. val_rmse_test: 4.0432
2. ratio rmse/avy: 0.0733
3. sigma: 0.0166

For 95% confidence level, confidence interval for ratio 0.0733 is [0.0408, 0.1058]



True y and sq error plot, adjacent figure, indicates that error is high in the target varibales in the middle region, similar to train data's plot.

When performance of reduced line vector model is compared between Train and test set, there is no statistically significant difference between two ratios, as there is an overlap in the confidence intervals between both. But abnormally, test set has lesser ratio compared to train.

4.7. Compare these performances to the original formula pred(x) and interpret the results

Answer:

Original formula performance is :

Performance	Train	Test
ratio rmse/avy:	0.0115	0.014
95% confidence interval	[0.0048, 0.0182]	[0, 0.0287]

Reduced Formula performance is :

Performance	Train	Test
ratio rmse/avy:	0.0795	0.0733
95% confidence interval	[0.0626, 0.0964]	[0.0408, 0.1058]

When confidence intervals are compared for Train data, performance is far higher for Original dataset.

However in case of test dataset, ratio is higher with wide confidence interval in case of reduced dataset compared to original data. Hence confidence decreases for reduced dataset.

Question 5 (optional): Implement KRR using a pre existing function

5.1. Using the best parameters found above try to use a pre-existing software function implementing the KRR technique

Answer:

By using inbuilt krr function, following are the performance parameters for

Train data :

rmse_model_train: 0.6251
ratio rmse/avy: 0.0115
sigma: 0.0034
For 95% confidence level, confidence interval for ratio 0.0115 is [0.0048, 0.0182]

Test data :

rmse: 0.7707
ratio rmse/avy: 0.014
sigma: 0.0075
For 95% confidence level, confidence interval for ratio 0.014 is [0, 0.0287]

Original formula performance is :

Performance	**Train**	**Test**
ratio rmse/avy:	0.0115	0.014
95% confidence interval	[0.0048, 0.0182]	[0, 0.0287]

Both the inbuilt function and original formula performances are similar for both train and test data.

PART – II

PYTHON CODE:

```
# Commented out IPython magic to ensure Python compatibility.
import os
import pandas as pd
import numpy as np
from scipy import io
import math
import matplotlib.pyplot as plt
# %matplotlib inline

from pandas import ExcelWriter
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

#Read dataset from Github
!pip install -q xlrd
!git clone https://github.com/kishoret04/statisticallearning_datamining.git
##Files from the cloned git repository.
!ls  statisticallearning_datamining/datasets

#Copy data into a dataframe
ds_filename = 'statisticallearning_datamining/datasets/stocks_dataset.xlsx'

#ds_filename = r'C:\Users\kisho\Desktop\python_jupyter\stocks_dataset.xlsx'
df_totaldata = pd.read_excel(ds_filename)
df_totaldata
```

```
"""# Functions"""

#missing value count
def missingvalue_count(df_train):

  #initiaizing severity dataframe with NaN values
  df_severity = pd.DataFrame(data = np.NaN,index = df_train.columns,
                      columns = ['Missing_Values','%_of_MV'])

  df_missingrec = pd.DataFrame()
  #iterating through columns to find null values count and percentage
  for column in df_train:
    if df_train[column].isnull().values.any():
      null_rows = df_train[column].isnull()
      null_count = sum(null_rows)
      nullcount_percent = round(null_count/df_train[column].size*100,2)
      df_missingrec = df_missingrec.append(df_train[null_rows])
    else:
      null_count = 0
      nullcount_percent = 0
  #saving missing value counts in dataframe
  df_severity.loc[column,'Missing_Values'] = null_count
  df_severity.loc[column,'%_of_MV'] = nullcount_percent

  return df_severity,df_missingrec

def create_traintestdata(prop,df_input):
```

```
    #calculate trainset size for cl1
    train_size = int(prop*df_input.shape[0]/100)
    #extract a random sample of 80% as train and 20% as test set
    df_train = df_input.sample(train_size)
    df_test = df_input.drop(df_train.index)

#    #separating Target variable from train and test
#    df_train_notarget = df_train.drop(columns = ['Y'])
#    df_train_target = df_train['Y']

#    df_test_notarget = df_test.drop(columns = ['Y'])
#    df_test_target = df_test['Y']

    #sort train and test by date
    df_train.sort_values(by='Date',ascending=True,axis =0,inplace=True)
    df_test.sort_values(by='Date',ascending=True,axis =0,inplace=True)
    #Reset index
    df_train.reset_index(drop=True,inplace = True)
    df_test.reset_index(drop=True,inplace = True)
#    df_train_notarget.reset_index(drop=True,inplace = True)
#    df_train_target.reset_index(drop=True,inplace = True)
#    df_test_notarget.reset_index(drop=True,inplace = True)
#    df_test_target.reset_index(drop=True,inplace = True)
    #return df_train,df_test,df_train_notarget,df_train_target,df_test_notarget,df_test_target
    return df_train,df_test

def calc_kl_radial(gamma,x,y):
    val_k = math.exp(-1*gamma*np.power(np.linalg.norm(x-y),2))
    return val_k
```

```
####Line vector calculation-----------
def cal_linevector_A(val_gamma,val_lambda,df_train):
    #gramian matrix
    mat_train = np.matrix(df_train.loc[:,'X1':'X11'])
    val_m =mat_train.shape[0]

    for i in np.arange(val_m):
        for j in np.arange(val_m):
            mat_gramian[i,j] = calc_kl_radial(val_gamma,mat_train[i],mat_train[j])

    #Calculation of M = G + lambda*Identity
    mat_M = mat_gramian + val_lambda*np.eye(mat_gramian.shape[0])
    mat_Minv = np.linalg.inv(mat_M)

    #line vector A calculation
    mat_y = np.asmatrix(df_train['Y'])
    linevector_A = mat_y* mat_Minv

    return linevector_A

#calculate prediction for input vector input_X
def cal_predx(val_gamma,linevector_A,mat_input_X,mat_train_X):
    val_trainsize = mat_train_X.shape[0]
    mat_vx = np.zeros(val_trainsize).reshape(val_trainsize,1)

    for i in np.arange(val_trainsize):
        #print('sizes: {0},{1}'.format(mat_input_X.shape,mat_train_X[i].shape))
        mat_vx[i,0] = calc_kl_radial(val_gamma,mat_input_X,mat_train_X[i])
```

```
    #print(' cal_predx variable mat_vx:\n',mat_vx)
    pred_x = linevector_A*mat_vx
    #print('cal_predx variable pred_x: ',pred_x)
    return np.around(pred_x[0,0],4)

def cal_rmse(y_true,y_pred):
    val_rmse = np.sqrt(np.mean(np.square(y_true-y_pred)))
    return val_rmse

###function to calculate performance for given test set and linevector_A
def cal_performance_params(val_gamma,linevector_A,df_test,df_train):
    #matrix formulation
    mat_train = np.matrix(df_train.loc[:,'X1':'X11'])
    val_m =mat_train.shape[0]
    mat_test = np.matrix(df_test.loc[:,'X1':'X11'])
    val_testsize = mat_test.shape[0]

    #calculate predictions for test data
    mat_pred_x = np.zeros(val_testsize).reshape(val_testsize,1)
    for i in np.arange(val_testsize):
        mat_pred_x[i,0] = cal_predx(val_gamma,linevector_A,mat_test[i],mat_train)

    #print('cal_performance_params variable mat_pred_x:\n',mat_pred_x)
    #reshaping arguments for RMSE
    mat_y_true = np.asmatrix(df_test['Y'])
    mat_y_pred = np.asmatrix(np.array(mat_pred_x[:,0])).reshape(mat_y_true.shape)
    #print('cal_performance_params variable mat_y_pred:\n ',mat_y_pred)
    #calculate RMSE
```

```
    val_rmse_test = cal_rmse(mat_y_true,mat_y_pred)
    print('val_rmse_test: ',np.around(val_rmse_test,4))

    #calculate ratio RMSE/avy for test
    val_avy_test = np.mean(np.abs(mat_y_true))
    ratio_rmse_avy_test = np.around(val_rmse_test/val_avy_test,4)
    print('ratio rmse/avy: ',ratio_rmse_avy_test)

    return val_rmse_test,ratio_rmse_avy_test,mat_y_pred
####---------------------------------
def tune_krr(tuned_parameters,df_train,df_test):
  lambda_range = tuned_parameters['lambda']
  gamma_range = tuned_parameters['gamma']
  df_grid_scores = pd.DataFrame(0,index = [],columns =
                              ['rmse_test','rmse_train','ratio_test','ratio_train',
                               'diff_ratio','params'])

  #tuning output
  for val_lambda in lambda_range:
    for val_gamma in gamma_range:
      print('tuning parameters: lambda ={0},gamma = {1}'.format(val_lambda,val_gamma))
      linevector_A = cal_linevector_A(val_gamma,val_lambda,df_train)
      #train performance
      val_rmse_train,ratio_rmse_avy_train,mat_train_pred =
cal_performance_params(val_gamma,linevector_A,df_train,df_train)
      #test performance
      val_rmse_test,ratio_rmse_avy_test,mat_test_pred =
cal_performance_params(val_gamma,linevector_A,df_test,df_train)
      params = 'lambda = '+ str(val_lambda)+ '; gamma = '+ str(val_gamma)
      row = {'rmse_test':val_rmse_test,'rmse_train':val_rmse_train,
```

```
            'ratio_test':ratio_rmse_avy_test,'ratio_train':ratio_rmse_avy_train,
            'diff_ratio': ratio_rmse_avy_test -ratio_rmse_avy_train ,
            'params':params }
      df_grid_scores = df_grid_scores.append(row,ignore_index = True)

  df_grid_scores.to_excel('grid_scores.xlsx')

def err_est_element(term,size,return_sigma):

  sigma = np.around(math.sqrt(term*(1-term)/size),4)
  print('sigma: ',sigma)
  #for 95% confidence level
  Z_VAL = 1.96
  limit_lower = np.around((term - Z_VAL*sigma),4)

  if limit_lower < 0:
    #print('before if:',limit_lower)
    limit_lower = 0
    #print('after if:',limit_lower)

  limit_upper = np.around((term + Z_VAL*sigma),4)

  if limit_upper > 100:
    #print('before if:',limit_upper)
    limit_upper = 100
    #print('after if:',limit_upper)

  conf_int = [limit_lower,limit_upper]
```

```
  if return_sigma:
    return sigma
  else:
    return str(conf_int)


def plot_results(data,title):
  f_size = (8,8)
  fig, ax = plt.subplots(figsize=f_size)
  # ax.plot(data['Date'],data['Y'],  'b-')
  # ax.plot(data['Date'],data['yhat'],  'g-')
  data.plot('Date',['Y','yhat'],ax=ax)
  #ax.plot(np.arange(1,sort_linevect_A.shape[0]+1),sort_linevect_A, c='b',s=5, alpha=.5)
  ax.set(title = title , xlabel = 'Date', ylabel = 'stock price')
  plt.show()


"""# 1.1 make sure that p ≥ 10 in your Data Set and make sure to include an artificial feature Xp(j) =1 for all cases j=1...n
each feature must be a "continuous" variable;
avoid or eliminate discrete features taking only a small number of values ;
"""


df_totaldata_orig = df_totaldata.copy()
#drop Date column
#df_totaldata.drop(columns = ['Date'],inplace = True)


#add Xp(j)=1 column
df_totaldata.insert(11,'X11',np.repeat(1.,df_totaldata.shape[0]))


df_totaldata.describe()
```

```
"""# 1.4.for each feature X1 X2 ... Xp, compute and display its  mean and standard deviation"""

df_totaldata.describe().to_excel('df_totaldata.describe.xlsx')

"""# 1.5. compute and display its  mean and standard deviation for Y"""

df_totaldata.describe()['Y']

"""# 1.6. Split the data set DS into a training set TRAIN and a test set TEST, with respective proportions 80% ,  20%"""

#missing values in each column
df_missingstats,df_missingrec = missingvalue_count(df_totaldata)
print('Data Severity\n ', df_missingstats)

#create train and test datasets
PROP = 80
#df_train,df_test,df_train_notarget,df_train_target,df_test_notarget,df_test_target = create_traintestdata(prop,df_totaldata.copy())
df_train,df_test = create_traintestdata(PROP,df_totaldata.copy())

#describe train and test data
print('############# Train data #######')

#write to excel dataset
filepath = 'preprocessed_data.xlsx'

with ExcelWriter(filepath) as writer:
    df_train.to_excel(writer,sheet_name = 'train_data' )
```

```
    df_test.to_excel(writer,sheet_name = 'test_data' )
    df_train.loc[:,'X1':'X11'].to_excel(writer,sheet_name = 'train_data_nolabel' )
    df_train['Y'].to_excel(writer,sheet_name = 'train_labels_all' )
    df_test.loc[:,'X1':'X11'].to_excel(writer,sheet_name = 'test_data_nolabel' )
    df_test['Y'].to_excel(writer,sheet_name = 'test_labels_all' )
    writer.save()

df_train.describe().to_excel('df_train.describe.xlsx')

df_test.describe().to_excel('df_test.describe.xlsx')

"""# 1.7.Compute the empirical correlations cor(X1, Y) ... cor(Xp,Y) and their absolute values C1 ... Cp"""

sr_corr_XY = pd.DataFrame(np.abs(df_totaldata.corrwith(df_totaldata['Y'])))
sr_corr_XY

sr_corr_XY.to_excel('sr_corr_XY.xlsx')

"""# 1.8.compute  the 3 largest values among C1 ... Cp, to be denoted  Cu > Cv > Cw which are"""

sr_corr_XY.sort_values(by=0,ascending=False)[1:4]

"""# 1.9.display separately the 3 scatter plots (Xu(j), Yj)  ,  (Xv(j), Yj)  , (Xw(j), Yj)  where j= 1...n"""

df_totaldata.plot(x='X6',y='Y',kind = 'scatter',title = 'Scatter plot of Xu Vs Y')

df_totaldata.plot(x='X4',y='Y',kind = 'scatter',title = 'Scatter plot of Xv Vs Y')

df_totaldata.plot(x='X1',y='Y',kind = 'scatter',title = 'Scatter plot of Xw Vs Y')
```

```
"""# Question 2: Kernel Ridge Regression (KRR) with radial kernel.For this question we use intensively
the training set TRAIN which has size m = 80% n

# 2.1.Compute the matrix G and its eigenvalues L1 >L2 > ... > Lm ≥ 0
"""

mat_train = np.matrix(df_train.loc[:,'X1':'X11'])
val_m =mat_train.shape[0]
mat_gramian = np.zeros((val_m,val_m))
VAL_SAMPLE_GAMMA = 0.01
for i in np.arange(val_m):
   for j in np.arange(val_m):
      mat_gramian[i,j] = calc_kl_radial(VAL_SAMPLE_GAMMA,mat_train[i],mat_train[j])

print('Gramian: \n',mat_gramian)
#count of negative values in gramian
sum(sum(mat_gramian<0))

gramian_eig_val,gramian_eig_vec = np.linalg.eig(mat_gramian)
gramian_eig_val = gramian_eig_val.real
gramian_eig_vec = gramian_eig_vec.real

print('eigen values:\n{0} \n eigen vectors:\n{1}'.format(gramian_eig_val,gramian_eig_vec))

pd.DataFrame(gramian_eig_val).describe()

gramian_eig_val[::-1].sort()
gramian_eig_val
```

```
"""# 2.2.Plot Lj versus j"""

fig, ax = plt.subplots()
ax.plot(np.arange(1,gramian_eig_val.shape[0]+1),gramian_eig_val,  '-')
ax.scatter(np.arange(1,gramian_eig_val.shape[0]+1),gramian_eig_val, c='b', alpha=.5)
ax.set(title = 'Gramian Eigen values', xlabel = 'j', ylabel = 'Eigen values Lj')
    #xticks = np.arange(1,gramian_eig_val.shape[0]+1))

plt.show()

"""#2.3. Plot the increasing ratios RATj= (L1 + ... + Lj)/(L1+ ... + Lm)
#2.4.Identify the smallest j such that RATj ≥ 95% and set λ = Lj
"""

ratio_eig_val = gramian_eig_val.cumsum()/gramian_eig_val.sum()
ratio_eig_val

# Where does Rj>.95
a_idx = np.where(ratio_eig_val>.95)[0][0] #gets the index
a = ratio_eig_val[a_idx] #gets the value at index
print('At eigenvalue', (a_idx+1), format(gramian_eig_val[a_idx], '.2f'), 'we get a ratio of', format(a, '.2%'))

val_lambda = gramian_eig_val[a_idx]

val_lambda

f_size=(10,8)
fig, ax = plt.subplots(figsize=f_size)
```

```
ax.plot(np.arange(1,gramian_eig_val.shape[0]+1),ratio_eig_val,  '-')
ax.scatter(np.arange(1,gramian_eig_val.shape[0]+1),ratio_eig_val, c='b', alpha=.2)

#ax.scatter(a_idx, a, c='green', alpha=1)
ax.plot(range(len(ratio_eig_val)), [.95]*gramian_eig_val.shape[0], 'r--', alpha=1)
ax.text(350,a,'95% Line', horizontalalignment = 'right', verticalalignment = 'bottom')
ax.text(a_idx,.1,'Eigenvalue {0} > 95%'.format(a_idx+1),
        horizontalalignment = 'left', verticalalignment = 'bottom')

ax.set(title = 'Increasing ratio of cumulative sum of Eigen values',
       xlabel = 'j', ylabel = 'Ratio of Eigen values')



plt.show()

"""# 2.5.Select at random two lists  List1 and List 2 of 100 random integers each , within   [1...m]

# 2.6.For all i in List 1 and all j in List2  compute Dij = ||X(i) -X(j)||
"""

SIZE_SAMPLE = 100
df_list1 = df_train.loc[:,'X1':'X11'].sample(SIZE_SAMPLE).reset_index(drop=True)
df_list2 = df_train.loc[:,'X1':'X11'].sample(SIZE_SAMPLE).reset_index(drop=True)
df_list1

mat_diff_dij = np.zeros(SIZE_SAMPLE*SIZE_SAMPLE).reshape(SIZE_SAMPLE,SIZE_SAMPLE)
for i in np.arange(df_list1.shape[0]):
    for j in np.arange(df_list2.shape[0]):
        mat_diff_dij[i,j] = np.around(np.linalg.norm(df_list1.loc[i,:] - df_list2.loc[j,:]),4)
```

```
mat_diff_dij

arr_dij = mat_diff_dij.reshape(1,SIZE_SAMPLE*SIZE_SAMPLE)[0]
arr_dij

"""# 2.7.Plot the histogram of the 10000 numbers Dij

# 2.8.Compute q =10% quantile of the 10000 numbers Dij

# 2.9.Set    gamma = 1/q
"""

fig, ax = plt.subplots(figsize=f_size)
ax.hist(arr_dij)
ax.set(title = 'Histogram of Dij',
    xlabel = 'Dij', ylabel = 'frequency')
    #  xticks = np.arange(1,gramian_eig_val.shape[0]+1))

plt.show()

val_q = np.quantile(arr_dij,0.10)
val_gamma = 1/val_q
val_gamma

np.quantile(arr_dij,0.50)

val_q
```

```
"""# 2.9 Compute the matrix M = G + λ Id  and its inverse M-1

# 2.10 As seen in class the prediction formula becomes pred(x) = A1 K(x, X(1)) + ... + Am
K(x,X(m))compute the line vector A  = [A1 ... Am] by  A= y M-1

"""


mat_train = np.matrix(df_train.loc[:,'X1':'X11'])

val_m =mat_train.shape[0]

mat_gramian = np.zeros((val_m,val_m))

VAL_SAMPLE_GAMMA = val_gamma

for i in np.arange(val_m):

    for j in np.arange(val_m):

        mat_gramian[i,j] = calc_kl_radial(VAL_SAMPLE_GAMMA,mat_train[i],mat_train[j])


mat_gramian


val_lambda


mat_M = mat_gramian + val_lambda*np.eye(mat_gramian.shape[0])

mat_M


mat_Minv = np.linalg.inv(mat_M)

mat_Minv


mat_y = np.asmatrix(df_train['Y'])

mat_y.shape


linevector_A = mat_y* mat_Minv

linevector_A
```

```
"""# 2.11 Compute the RMSEtrain  of the prediction function pred(x) by running it on all x in TRAIN
set"""


#calculate predictions for train data
val_trainsize = mat_train.shape[0]
mat_pred_x = np.zeros(val_trainsize).reshape(val_trainsize,1)
for i in np.arange(val_trainsize):
   mat_pred_x[i,0] = cal_predx(val_gamma,linevector_A,mat_train[i],mat_train)


mat_pred_x


#reshaping arguments for RMSE
mat_y_true = np.asmatrix(df_train['Y'])
mat_y_pred = np.asmatrix(np.array(mat_pred_x[:,0])).reshape(mat_y_true.shape)


#calculate RMSE
val_rmse_train = cal_rmse(mat_y_true,mat_y_pred)
print('val_rmse_train: ',np.around(val_rmse_train,4))


#calculate ratio RMSE/avy for train
val_avy_train = np.mean(np.abs(mat_y_true))
ratio_rmse_avy_train = np.around(val_rmse_train/val_avy_train,4)
print('ratio rmse/avy train: ',ratio_rmse_avy_train)


fig, ax = plt.subplots(figsize=f_size)
ax.plot( [0,np.max(mat_y_true)],[0,np.max(mat_y_true)] )
ax.scatter(np.asarray(mat_y_true),np.asarray(mat_y_pred),c = 'g')
ax.set(title = 'y vs y hat for train data' , xlabel = 'y', ylabel = 'yhat')
plt.show()
```

```
"""2.12.Compute the RMSEtest   of the prediction function pred(x) by running it on all x in TEST set"""

#calculate predictions for test data
mat_test = np.matrix(df_test.loc[:,'X1':'X11'])
val_testsize = mat_test.shape[0]
mat_pred_x = np.zeros(val_testsize).reshape(val_testsize,1)
for i in np.arange(val_testsize):
    mat_pred_x[i,0] = cal_predx(val_gamma,linevector_A,mat_test[i],mat_train)

mat_pred_x

#reshaping arguments for RMSE
mat_y_true = np.asmatrix(df_test['Y'])
mat_y_pred = np.asmatrix(np.array(mat_pred_x[:,0])).reshape(mat_y_true.shape)

#calculate RMSE
val_rmse_test = cal_rmse(mat_y_true,mat_y_pred)
print('val_rmse_test: ',np.around(val_rmse_test,4))

#calculate ratio RMSE/avy for test
val_avy_test = np.mean(np.abs(mat_y_true))
ratio_rmse_avy_test = np.around(val_rmse_test/val_avy_test,4)
print('ratio rmse/avy: ',ratio_rmse_avy_test)

fig, ax = plt.subplots(figsize=f_size)
ax.plot( [0,np.max(mat_y_true)],[0,np.max(mat_y_true)] )
ax.scatter(np.asarray(mat_y_true),np.asarray(mat_y_pred),c = 'g')
ax.set(title = 'y vs y hat for test data' , xlabel = 'y', ylabel = 'yhat')
```

```
plt.show()


"""2.13 Compare these two RMSE values , and compute their ratios RMSE/ avy where

avy = mean of the m absolute values |Y1|,  ... , |Ym|

"""


print('val_rmse_train: ',np.around(val_rmse_train,4))

print('ratio rmse/avy train: ',ratio_rmse_avy_train)

confint_ratio_train = err_est_element(ratio_rmse_avy_train,df_train['Y'].shape[0],False)

print('For 95% confidence level, conifdence interval for ratio is ',confint_ratio_train)


print('val_rmse_test: ',np.around(val_rmse_test,4))

print('ratio rmse/avy: ',ratio_rmse_avy_test)

confint_ratio_test = err_est_element(ratio_rmse_avy_test,df_test['Y'].shape[0],False)

print('For 95% confidence level, conifdence interval for ratio is ',confint_ratio_test)


"""Question 3: Improving the results through step by step tuning


# 3.1.Repeat the preceding operations for other pairs of parameters gamma and λ

Suggestion: change only one parameter at a time to check in which direction to go for improved performances



# 3.2.Select the best choice of parameters in terms of accuracy RMSE/avy and stability of performance when one goes from TRAIN to TEST set

"""


#results from question 2 params

print('gamma : ',val_gamma)

print('lambda: ',val_lambda)
```

```
print('tuning parameters: gamma = {0},lambda ={1}'.format(val_gamma,val_lambda))
linevector_A_orig = cal_linevector_A(val_gamma,val_lambda,df_train)

#train performance
val_rmse_train,ratio_rmse_avy_train,mat_y_pred_train = cal_performance_params(
   val_gamma,linevector_A_orig,df_train,df_train)

df_train_orig_lv = df_train.copy()
df_train_orig_lv['yhat'] = np.asarray(mat_y_pred_train.T)
df_train_orig_lv

df_train_orig_lv

#plotting y vs yhat
xlim = np.min(df_train_orig_lv['Y'])
ylim = np.max(df_train_orig_lv['Y'])
fig, ax = plt.subplots(figsize=f_size)
ax.plot( [0,ylim],[0,ylim] )
ax.scatter(df_train_orig_lv['Y'],df_train_orig_lv['yhat'],c = 'g')
ax.set(title = 'y vs y hat for train data' , xlabel = 'y', ylabel = 'yhat')
plt.show()

# f_size = (10,8)
# fig, ax = plt.subplots(figsize=f_size)
# ax.plot(df_train_orig_lv['Date'],df_train_orig_lv['Y'],  'b-')
# ax.plot(df_train_orig_lv['Date'],df_train_orig_lv['yhat'],  'g-')
# #ax.plot(np.arange(1,sort_linevect_A.shape[0]+1),sort_linevect_A, c='b',s=5, alpha=.5)
# ax.set(title = 'y vs y hat for train data', xlabel = 'Date', ylabel = 'stock price')
```

```
# plt.show()
plot_results(df_train_orig_lv,title= 'y vs y hat for original train data')

#test performance
val_rmse_test,ratio_rmse_avy_test,mat_y_pred_test = cal_performance_params(
    val_gamma,linevector_A_orig,df_test,df_train)

df_test_orig_lv = df_test.copy()
df_test_orig_lv['yhat'] = np.asarray(mat_y_pred_test.T)
df_test_orig_lv



plot_results(df_test_orig_lv,title= 'y vs y hat for original test data')

#first set  of values
gamma_0 = round(val_gamma,4)
lambda_0 = round(val_lambda,4)
#results from question 2 params
lambda_range = [lambda_0]
gamma_range = [gamma_0]
tuned_parameters = {'lambda': lambda_range, 'gamma': gamma_range}
tune_krr(tuned_parameters,df_train,df_test)

#tuning with lambda fixed and changing gamma
lambda_range = [lambda_0]
gamma_range = [gamma_0/2,2*gamma_0]
tuned_parameters = {'lambda': lambda_range, 'gamma': gamma_range}
tune_krr(tuned_parameters,df_train,df_test)
```

```
gamma_1 = gamma_0/2
#tuning with gamma fixed and changing lambda
lambda_range = [lambda_0/2,2*lambda_0]
gamma_range = [gamma_1]
tuned_parameters = {'lambda': lambda_range, 'gamma': gamma_range}
tune_krr(tuned_parameters,df_train,df_test)

lambda_1 = lambda_0/2
#tuning with gamma fixed and changing lambda
lambda_range = [lambda_1]
gamma_range = [gamma_1/2,2*gamma_1]
tuned_parameters = {'lambda': lambda_range, 'gamma': gamma_range}
tune_krr(tuned_parameters,df_train,df_test)

gamma_2 = gamma_1/2
#tuning with gamma fixed and changing lambda
lambda_range = [lambda_1/2,2*lambda_1]
gamma_range = [gamma_2]
tuned_parameters = {'lambda': lambda_range, 'gamma': gamma_range}
tune_krr(tuned_parameters,df_train,df_test)

lambda_2 = lambda_1/2
#tuning with lambda fixed and changing gamma
lambda_range = [lambda_2]
gamma_range = [gamma_2/2,2*gamma_2]
tuned_parameters = {'lambda': lambda_range, 'gamma': gamma_range}
tune_krr(tuned_parameters,df_train,df_test)
```

```
gamma_3 = gamma_2/2
#tuning with gamma fixed and changing lambda
lambda_range = [lambda_2/2,2*lambda_2]
gamma_range = [gamma_3]
tuned_parameters = {'lambda': lambda_range, 'gamma': gamma_range}
tune_krr(tuned_parameters,df_train,df_test)

lambda_3 = lambda_2/2
#tuning with lambda fixed and changing gamma
lambda_range = [lambda_3]
gamma_range = [gamma_3/2,2*gamma_3]
tuned_parameters = {'lambda': lambda_range, 'gamma': gamma_range}
tune_krr(tuned_parameters,df_train,df_test)

gamma_4 = gamma_3/2
#tuning with gamma fixed and changing lambda
lambda_range = [lambda_3/2,2*lambda_3]
gamma_range = [gamma_4]
tuned_parameters = {'lambda': lambda_range, 'gamma': gamma_range}
tune_krr(tuned_parameters,df_train,df_test)

lambda_4 = lambda_3/2
#tuning with lambda fixed and changing gamma
lambda_range = [lambda_4]
gamma_range = [gamma_4/2,2*gamma_4]
tuned_parameters = {'lambda': lambda_range, 'gamma': gamma_range}
tune_krr(tuned_parameters,df_train,df_test)

gamma_5 = gamma_4/2
```

```
#tuning with gamma fixed and changing lambda
lambda_range = [lambda_4/2,2*lambda_4]
gamma_range = [gamma_5]
tuned_parameters = {'lambda': lambda_range, 'gamma': gamma_range}
tune_krr(tuned_parameters,df_train,df_test)

lambda_5 = lambda_4/2
#tuning with lambda fixed and changing gamma
lambda_range = [lambda_5]
gamma_range = [gamma_5/2,2*gamma_5]
tuned_parameters = {'lambda': lambda_range, 'gamma': gamma_range}
tune_krr(tuned_parameters,df_train,df_test)

"""# 3.3. Identify the 10  cases in the TEST set for which the squared prediction error is the largest"""

best_params = {'lambda': [0.0205], 'gamma': [0.0017]}

#calc linevector_A for  best params
linevector_A_best = cal_linevector_A(best_params['gamma'][0],best_params['lambda'][0],df_train)

#calc y_predictions
val_rmse_train_best,ratio_rmse_avy_train_best,mat_y_pred_train_best = cal_performance_params(
    best_params['gamma'][0],linevector_A_best,df_train,df_train)

df_train_best_lv = df_train.copy()
df_train_best_lv['yhat'] = np.asarray(mat_y_pred_train_best.T)

plot_results(df_train_best_lv,title= 'y vs y hat for best parameters for train')
```

```
confint_ratio_train_best =
err_est_element(ratio_rmse_avy_train_best,df_train_best_lv['Y'].shape[0],False)

print('For 95% confidence level, confidence interval for ratio {0} is
{1}'.format(ratio_rmse_avy_train_best,
                                                            confint_ratio_train_best))


#plotting y vs yhat for train for best parameters

xlim = np.min(df_train_best_lv['Y'])

ylim = np.max(df_train_best_lv['Y'])

fig, ax = plt.subplots(figsize=f_size)

ax.plot( [0,ylim],[0,ylim] )

ax.scatter(df_train_best_lv['Y'],df_train_best_lv['yhat'],c = 'g')

ax.set(title = 'y vs y hat for train data with best parameters' , xlabel = 'y', ylabel = 'yhat')

plt.show()


#train data - true vs error plot

df_train_best_lv['sqerr'] = np.asarray(np.square(df_train_best_lv['Y']-df_train_best_lv['yhat']))

df_train_best_lv.plot('Y','sqerr',style='o',title = 'y true vs sq error for train ')


#calc y_predictions for test data for best params

val_rmse_test_best,ratio_rmse_avy_test_best,mat_y_pred_test_best = cal_performance_params(

   best_params['gamma'][0],linevector_A_best,df_test,df_train)


df_test_best_lv = df_test.copy()

df_test_best_lv['yhat'] = np.asarray(mat_y_pred_test_best.T)

df_test_best_lv


plot_results(df_test_best_lv,title= 'y vs y hat for best parameters')


confint_ratio_test_best = err_est_element(
```

```
    ratio_rmse_avy_test_best,df_test_best_lv['Y'].shape[0],False)
print('For 95% confidence level, confidence interval for ratio {0} is {1}'.format(
    ratio_rmse_avy_test_best,confint_ratio_test_best))

#plotting y vs yhat for test for best parameters
xlim = np.min(df_test_best_lv['Y'])
ylim = np.max(df_test_best_lv['Y'])
fig, ax = plt.subplots(figsize=f_size)
ax.plot( [0,ylim],[0,ylim] )
ax.scatter(df_test_best_lv['Y'],df_test_best_lv['yhat'],c = 'g')
ax.set(title = 'y vs y hat for test data with best parameters' , xlabel = 'y', ylabel = 'yhat')
plt.show()

#test data - true vs error plot
df_test_best_lv['sqerr'] = np.asarray(np.square(df_test_best_lv['Y']-df_test_best_lv['yhat']))
df_test_best_lv.plot('Y','sqerr',style='o',title = 'y true vs sq error for test')

#calc squared pred error and top 10 largest error cases
#reshaping arguments for RMSE
mat_y_true = np.asmatrix(df_test['Y'])
df_test_error = df_test.copy()
df_test_error['yhat'] = df_test_best_lv['yhat']
df_test_error['sqerr'] = np.asarray(np.square(mat_y_pred_test_best-mat_y_true))[0]
#sqerror_top10 = np.asarray(-np.sort(-np.square(mat_y_pred-mat_y_true))[0,:10])[0]
df_test_error.sort_values(by = ['sqerr'],ascending =False,axis=0,inplace=True)

df_test_error.reset_index(inplace=True,drop=True)
df_test_error
```

```
df_test_error.to_excel('df_test_error.xlsx')


"""# 3.4.Vizualise the 10 cases by performing a PCA analysis and projecting all the TEST cases onto the
first 3 principal eigenvectors of the PCA correlation matrix"""


#PCA analysis and projection to 3 components
pca = PCA(n_components=3)
#fit train data
print("--------PCA Train data--------")
df_test_error_reduced = pca.fit_transform(df_test_error.loc[:,'X1':'X11'])
df_test_error_reduced = pd.DataFrame(df_test_error_reduced)
print('explained_variance_ratio :', np.around(np.sum(pca.explained_variance_ratio_),2) )
filepath = 'df_test_error_reduced.xlsx'
with ExcelWriter(filepath) as writer:
   df_test_error_reduced.to_excel(writer,sheet_name = 'df_test_error_reduced' )
   df_test_error.to_excel(writer,sheet_name = 'df_test_error' )
   writer.save()

size_testerr = df_test_error_reduced.shape[0]

cases_lowerror = np.arange((size_testerr-10),size_testerr)
cases_lowerror

size_testerr = df_test_error_reduced.shape[0]
cases_lowerror = np.arange((size_testerr-10),size_testerr)

threedee = plt.figure(figsize=(10,10)).gca(projection='3d')
threedee.scatter(df_test_error_reduced.loc[0:9,0],
            df_test_error_reduced.loc[0:9,1],
```

```
        df_test_error_reduced.loc[0:9,2],c='r',s=50)
threedee.scatter(df_test_error_reduced.loc[cases_lowerror,0],
        df_test_error_reduced.loc[cases_lowerror,1],
        df_test_error_reduced.loc[cases_lowerror,2],c='g')
threedee.set_xlabel('PC1')
threedee.set_ylabel('PC2')
threedee.set_zlabel('PC3')
plt.show()

def plot_setspines(ax1):
  ax1.spines['left'].set_position('zero')
  ax1.spines['right'].set_color('none')
  ax1.spines['bottom'].set_position('zero')
  ax1.spines['top'].set_color('none')
  ax1.spines['left'].set_smart_bounds(True)
  ax1.spines['bottom'].set_smart_bounds(True)
  ax1.xaxis.set_ticks_position('bottom')
  ax1.yaxis.set_ticks_position('left')
  ax1.xaxis.set_label_coords(1,0)
  ax1.yaxis.set_label_coords(-0.1,1)
  return ax1


fig = plt.figure(figsize=(10,10))
#plot of PC1 and PC2
ax1 = fig.add_subplot(2,2,1)
ax1 = plot_setspines(ax1)
ax1.scatter(df_test_error_reduced.loc[0:9,0],df_test_error_reduced.loc[0:9,1],color='r')
```

```
ax1.scatter(df_test_error_reduced.loc[cases_lowerror,0],df_test_error_reduced.loc[cases_lowerror,1],color='g')

ax1.set(title = 'plot of PC1 and PC2',xlabel = 'PC1', ylabel = 'PC2')



#plot of PC2 and PC3

ax2 = fig.add_subplot(2,2,2)

ax2 = plot_setspines(ax2)

ax2.scatter(df_test_error_reduced.loc[0:9,1],df_test_error_reduced.loc[0:9,2],color='r')

ax2.scatter(df_test_error_reduced.loc[cases_lowerror,1],df_test_error_reduced.loc[cases_lowerror,2],color='g')

ax2.set(title = 'plot of PC2 and PC3',xlabel = 'PC2', ylabel = 'PC3')



#plot of PC1 and PC3

ax3 = fig.add_subplot(2,2,3)

ax3 = plot_setspines(ax3)

ax3.scatter(df_test_error_reduced.loc[0:9,0],df_test_error_reduced.loc[0:9,2],color='r')

ax3.scatter(df_test_error_reduced.loc[cases_lowerror,0],df_test_error_reduced.loc[cases_lowerror,2],color='g')

ax3.set(title = 'plot of PC1 and PC3',xlabel = 'PC1', ylabel = 'PC3')


plt.show()


#calculate centroid of 10 worst cases

centroid_worst = np.sum(df_test_error.loc[:9,'X1':'X11' ])/10

centroid_worst


#distance between centroid and each case

df_test_error['Dn_centr_w'] = np.linalg.norm(df_test_error.loc[:,'X1':'X11']- centroid_worst, axis=1)
```

```
df_test_error


#scatterplot of distances

f_size=(8,8)

fig, ax = plt.subplots(figsize=f_size)

ax.plot(df_test_error.loc[0:9,'Dn_centr_w'],df_test_error.loc[0:9,'sqerr'],'ro')

ax.plot(df_test_error.loc[cases_lowerror,'Dn_centr_w'],df_test_error.loc[cases_lowerror,'sqerr'],'go')

ax.plot(df_test_error.loc[10:,'Dn_centr_w'],df_test_error.loc[10:,'sqerr'],'go')

ax.set(title = 'Distance from cetnroid vs sq error',xlabel = 'distance from worst centroid',
      ylabel = 'sq error')


plt.show()


"""# 3.5.Try to identify what went wrong with the prediction of the absolute worse case X(w) by looking
at the terms involved in pred(Xw) and comparing to another case where the prediction erro is really
small"""


print(df_test_error.loc[0:9])

print('least error: \n',df_test_error.loc[(df_test_error.shape[0]-2) :])


df_test_error.to_excel('df_test_error.xlsx')


"""# 3.6 worst case analysis

pred(x) = A1 K(x, X(1)) + ... + Am K(x,X(m)) = U1 + ... +Um


consider the list LIST(x) of positive  numbers V(1)= |U1| ... V(m) = |Um|


find the sublist LIST5(x) of the 5 largest numbers in LISTx , and denote them


V(m1) > V(m2) > V(m3)) > V(m4) > V(m5)
```

Do this for x = worst test case to get [m1 m2 m3 m4 m5]

Do this for x = good test case to get [M1 M2 M3 M4 M5]

compare

[m1 m2 m3 m4 m5] and [M1 M2 M3 M4 M5]

repeat this comparison for a few more good cases to check if you find interpretable patterns of indices

you can also apply the same method of sublists extraction to the list of positive numbers W(1)= K(x,X(1)) ... W(m) = K(x,X(m))

"""

```
#calculate prediction coefficients list for input vector input_X
def cal_predx_list(val_gamma,linevector_A,mat_input_X,mat_train_X):
    val_trainsize = mat_train_X.shape[0]
    #reshaping with same
    mat_vx = np.zeros(val_trainsize).reshape(linevector_A.shape)

    for i in np.arange(val_trainsize):
        #print('sizes: {0},{1}'.format(mat_input_X.shape,mat_train_X[i].shape))
        mat_vx[0,i] = calc_kl_radial(val_gamma,mat_input_X,mat_train_X[i])

    #print(' cal_predx variable mat_vx:\n',mat_vx)
    pred_x = np.multiply(linevector_A, mat_vx)
    #print('cal_predx variable pred_x: ',pred_x)
    return np.around(pred_x,4)

#function that returns list of elements in pred(x) summations in test set
```

```
def cal_indices_pred(val_gamma,linevector_A,df_train,df_test):


    #matrix formulation
    mat_train = np.matrix(df_train.loc[:,'X1':'X11'])
    val_m =mat_train.shape[0]
    #mat_test_worst10 = np.matrix(df_test.loc[0:9,'X1':'X11'])
    mat_test_worst10 = np.matrix(df_test.loc[:,'X1':'X11'])
    val_testsize = mat_test_worst10.shape[0]


    #calculate predictions for test data
    #mat_pred_x = np.zeros(val_testsize).reshape(val_testsize,linevector_A_best.shape[1])
    mat_pred_x = np.zeros((val_testsize,linevector_A.shape[1]))
    for i in np.arange(val_testsize):
        mat_pred_x[i] = cal_predx_list(val_gamma,linevector_A,mat_test_worst10[i],mat_train)


    #print('cal_performance_params variable mat_pred_x:\n',mat_pred_x)


    return mat_pred_x


best_params


#worst 10 cases indices in pred(x)
mat_pred_worst10 = cal_indices_pred(best_params['gamma'][0],linevector_A_best,
                        df_train,df_test_error.loc[0:9])


#calculate indices of highest Ui values in pred_x summation for worst 10 cases
mat_list_indices_worst10 = np.argsort(-np.abs(mat_pred_worst10))[:,0:5]
print('mat_list_indices_worst:\n',mat_list_indices_worst10)
```

```
pd.DataFrame(mat_list_indices_worst10).to_excel('mat_list_indices_worst10.xlsx')


#best 10 cases indices in pred(x)

req_index = (df_test_error.shape[0]-10)

mat_pred_best10 = cal_indices_pred(best_params['gamma'][0],linevector_A_best,

                         df_train,df_test_error.loc[ req_index: ])


#calculate indices of highest Ui values in pred_x summation for worst 10 cases

mat_list_indices_best10 = np.argsort(-np.abs(mat_pred_best10))[:,0:5]

print('mat_pred_best10:\n',mat_list_indices_best10)


pd.DataFrame(mat_list_indices_best10).to_excel('mat_list_indices_best10.xlsx')


"""# Question 4 : Analysis of the best predicting formula pred(x)


# 4.1. Fix the best choice of parameters as found in the preceding question.


# 4.2. reorder the |A1|, |A2|, ....|Am| in decreasing order , which gives a list B1 > B2 ... > Bm  >0 and
plot the decreasing curve Bj versus j
"""


pd.DataFrame(np.asarray(linevector_A_best[0:10])[0]).to_excel('linevector_A_best.xlsx')


(linevector_A_best[0:10])[0].shape


best_params


#sorting in descending order

sort_linevector_A_best = np.asarray(np.abs(linevector_A_best))[0]
```

```
sort_linevector_A_best[::-1].sort()


sort_linevector_A_best[0:10]


fig, ax = plt.subplots(figsize=f_size)


ax.plot(np.arange(1,sort_linevector_A_best.shape[0]+1),sort_linevector_A_best,  '-')
ax.scatter(np.arange(1,sort_linevector_A_best.shape[0]+1),sort_linevector_A_best, c='b',s=5, alpha=.5)
ax.set(title = 'sorted Line vector A coefficients', xlabel = 'j', ylabel = 'Bj')


plt.show()


"""# 4.3.Compute the ratios bj = (B1 + ... + Bj)/(B1 + ...+Bm)and plot the increasing curve bj versus j"""


ratio_sorted_lv_A = sort_linevector_A_best.cumsum()/sort_linevector_A_best.sum()
# Where does Bj>.99
a_idx = np.where(ratio_sorted_lv_A>.99)[0][0] #gets the index
a = ratio_sorted_lv_A[a_idx] #gets the value at index
print('At linevector coef bj', (a_idx+1),'THR = Bj = ', format(sort_linevector_A_best[a_idx], '.2f'),
    'we get a ratio of', format(a, '.2%'))


val_threshold = sort_linevector_A_best[a_idx]
val_threshold


"""# 4.4.Compute the smaller j such that bj > 99%. and the corresponding threshold value THR = Bj"""


f_size=(8,8)
fig, ax = plt.subplots(figsize=f_size)
ax.plot(np.arange(1,sort_linevector_A_best.shape[0]+1),ratio_sorted_lv_A,  '-')
```

```
ax.scatter(np.arange(1,sort_linevector_A_best.shape[0]+1),ratio_sorted_lv_A, c='b', alpha=.2)


ax.scatter(a_idx, a, c='green', alpha=1)
ax.plot(range(len(ratio_sorted_lv_A)), [.99]*sort_linevector_A_best.shape[0], 'r--', alpha=1)
ax.text(350,a,'99% Line', horizontalalignment = 'right', verticalalignment = 'bottom')
ax.text(a_idx,.1,'bj {0} > 99%'.format(a_idx+1),
        horizontalalignment = 'right', verticalalignment = 'bottom')

ax.set(title = 'Increasing ratio of cumulative sum of sorted linevector values',
       xlabel = 'j', ylabel = 'Ratio of sorted linevector values')

plt.show()

"""# 4.5. For i =1... m, if |Ai|> THR set AAi = Ai   and otherwise set AAi = 0. This yields a reduced formula
# PRED(x)  = AA1 K(x, X(1)) + ... + AAm K(x,X(m))
"""

#linevector_A_red = np.where(np.abs(linevector_A_best)>val_threshold,linevector_A_best,0)
linevector_A_red = linevector_A_best.copy()
linevector_A_red[np.abs(linevector_A_red)<val_threshold] = 0

import seaborn as sns

ax = sns.distplot(linevector_A_best,label= 'linevector A ', bins =50)
ax.set(title = 'Histogram of A',
      xlabel = 'linevector A', ylabel = 'frequency')
      #  xticks = np.arange(1,gramian_eig_val.shape[0]+1))

plt.show()
```

```
ax = sns.distplot(linevector_A_red, bins =50)
ax.set(title = 'Histogram of linevector A red',
      xlabel = 'linevector A red', ylabel = 'frequency')
      #  xticks = np.arange(1,gramian_eig_val.shape[0]+1))

plt.show()

pd.DataFrame(np.asarray(linevector_A_red)[0]).to_excel('linevector_A_red.xlsx')

filepath = 'linevector_comp.xlsx'
with ExcelWriter(filepath) as writer:
    pd.DataFrame(linevector_A_best.T).to_excel(writer,sheet_name = 'linevector_A_best' )
    pd.DataFrame(linevector_A_red.T).to_excel(writer,sheet_name = 'linevector_A_red' )
    writer.save()

sum(sum(np.asarray(linevector_A_red!=0)))

print('number of non-zero Ais in reduced linevector A are: ',sum(sum(np.asarray(linevector_A_red!=0))),
      ' out of ',linevector_A_red.shape[1])

"""# 4.6. Run this reduced formula on the TRAIN and TEST sets to evaluate its performances"""

best_params

# A_sort = np.flip(np.sort(abs(linevector_A_best)))
# A_rat = np.cumsum(A_sort)/np.sum(A_sort)
# A_thresh = A_sort[:,np.sum(np.where(A_rat>.99,0,1))+1]
# A_thresh[0,0]
```

```
# linevector_A_red = linevector_A_best.copy()
# linevector_A_red[np.abs(linevector_A_red)<23] = 0
# linevector_A_red

#calc y_predictions for train set for best parameters with reduced linevector A
print('#####performance of reduced linevector on Train data#####')
val_rmse_test,ratio_rmse_avy_test,mat_y_pred_train_red = cal_performance_params(
    best_params['gamma'][0],linevector_A_red,df_train,df_train)

df_train_reduced_lv = df_train.copy()
df_train_reduced_lv['yhat'] = np.asarray(mat_y_pred_train_red.T)
df_train_reduced_lv

plot_results(df_train_reduced_lv,title= 'y vs y hat for reduced train data')

confint_ratio_test = err_est_element(ratio_rmse_avy_test,df_train['Y'].shape[0],False)

print('For 95% confidence level, confidence interval for ratio {0} is {1}'.format(ratio_rmse_avy_test,
                                                        confint_ratio_test))

#plotting y vs yhat for train for best parameters for reduced LV
xlim = np.min(df_train_reduced_lv['Y'])
ylim = np.max(df_train_reduced_lv['Y'])
fig, ax = plt.subplots(figsize=f_size)
ax.plot( [0,ylim],[0,ylim] )
ax.scatter(df_train_reduced_lv['Y'],df_train_reduced_lv['yhat'],c = 'g')
ax.set(title = 'y vs y hat for train data with reduced LV' , xlabel = 'y', ylabel = 'yhat')
plt.show()
```

```
df_train_reduced_lv.to_excel('df_train_reduced_lv.xlsx')

df_train_best_lv.to_excel('df_train_best_lv.xlsx')


#train data wiht reduced linevector A - true vs error plot

df_train_reduced_lv['sqerr'] = np.asarray(np.square(df_train_reduced_lv['Y']-
df_train_reduced_lv['yhat']))

df_train_reduced_lv.plot('Y','sqerr',style='o',title = 'y true vs sq error for train with reduced A vector')


df_train_reduced_lv


print('#####performance of reduced linevector on Test data#####')

val_rmse_test,ratio_rmse_avy_test,mat_y_pred_test_red = cal_performance_params(

   best_params['gamma'][0],linevector_A_red,df_test,df_train)


df_test_reduced = df_test.copy()

df_test_reduced['yhat'] = np.asarray(mat_y_pred_test_red.T)

df_test_reduced


plot_results(df_test_reduced,title= 'y vs y hat for reduced test data')


confint_ratio_test = err_est_element(ratio_rmse_avy_test,df_test['Y'].shape[0],False)


print('For 95% confidence level, confidence interval for ratio {0} is {1}'.format(ratio_rmse_avy_test,

                                                            confint_ratio_test))


#plotting y vs yhat for test for best parameters for reduced LV

xlim = np.min(df_test_reduced['Y'])

ylim = np.max(df_test_reduced['Y'])
```

```
fig, ax = plt.subplots(figsize=f_size)
ax.plot( [0,ylim],[0,ylim] )
ax.scatter(df_test_reduced['Y'],df_test_reduced['yhat'],c = 'g')
ax.set(title = 'y vs y hat for test data with reduced LV' , xlabel = 'y', ylabel = 'yhat')
plt.show()

#test data with reduced linevector A - true vs error plot
df_test_reduced['sqerr'] = np.asarray(np.square(df_test_reduced['Y']-df_test_reduced['yhat']))
df_test_reduced.plot('Y','sqerr',style='o',title = 'y true vs sq error for train with reduced A vector')

"""# 4.7.Compare these performances to the original formula pred(x) and interpret the results"""

#results from question 2 params
print('gamma : ',val_gamma)
print('lambda: ',val_lambda)

print('tuning parameters: gamma = {0},lambda ={1}'.format(val_gamma,val_lambda))
linevector_A_orig = cal_linevector_A(val_gamma,val_lambda,df_train)

vect_A = cal_linevector_A(val_gamma,val_lambda,df_train)

#matrix formulation
mat_train = np.matrix(df_train.loc[:,'X1':'Y'])
val_m =mat_train.shape[0]
mat_test = np.matrix(df_test.loc[:,'X1':'Y'])
val_testsize = mat_test.shape[0]

mat_input_X = mat_test[0]
mat_train_X = mat_train
```

```
val_trainsize = mat_train_X.shape[0]
mat_vx = np.zeros(val_trainsize).reshape(val_trainsize,1)

for i in np.arange(val_trainsize):
    #print('sizes: {0},{1}'.format(mat_input_X.shape,mat_train_X[i].shape))
    mat_vx[i,0] = calc_kl_radial(val_gamma,mat_input_X,mat_train_X[i])

print('mat_vx:\n',mat_vx)
pred_x = linevector_A*mat_vx
#print('pred_x: ',pred_x)
#return np.around(pred_x[0,0],4)
#return np.around(pred_x,4)

#calculate predictions for test data
mat_pred_x = np.zeros(val_testsize).reshape(val_testsize,1)
for i in np.arange(val_testsize):
    mat_pred_x[i,0] = cal_predx(val_gamma,vect_A,mat_test[i],mat_train)

print('mat_pred_x:\n',mat_pred_x)

#reshaping arguments for RMSE
mat_y_true = np.asmatrix(df_test['Y'])
mat_y_pred = np.asmatrix(np.array(mat_pred_x[:,0])).reshape(mat_y_true.shape)

mat_y_true

"""Question 5 (optional): Implement KRR using a pre existing function
```

5.1. using the best parameters found above try to use a pre-existing software function implementing the KRR technique

"""

```
best_params

from sklearn.kernel_ridge import KernelRidge

clf = KernelRidge(alpha=best_params['lambda'][0],kernel = 'rbf',gamma = best_params['gamma'][0])
clf.fit(df_train.loc[:,'X1':'X11'], df_train.loc[:,'Y'])

#model prediction for train data
pred_y = clf.predict(df_train.loc[:,'X1':'X11'])
pred_y
rmse_model_train = cal_rmse(df_train.loc[:,'Y'],pred_y)
print('rmse_model_train: ',rmse_model_train)
#print('cal_performance_params variable mat_pred_x:\n',mat_pred_x)
#reshaping arguments for RMSE
mat_y_true = np.asmatrix(df_train['Y'])
mat_y_pred = np.asmatrix(pred_y).reshape(mat_y_true.shape)

#calculate ratio RMSE/avy for test
val_avy_test = np.mean(np.abs(mat_y_true))
ratio_rmse_avy_test = np.around(rmse_model_train/val_avy_test,4)
print('ratio rmse/avy: ',ratio_rmse_avy_test)

confint_ratio_test = err_est_element(ratio_rmse_avy_test,df_train['Y'].shape[0],False)

print('For 95% confidence level, confidence interval for ratio {0} is {1}'.format(ratio_rmse_avy_test,
```

```
                                                    confint_ratio_test))

df_train_model = df_train.copy()
df_train_model['yhat'] = np.asarray(pred_y)

plot_results(df_train_model,title= 'y vs y hat for train data for model')

#plotting y vs yhat for test for best parameters for reduced LV
xlim = np.min(df_train_model['Y'])
ylim = np.max(df_train_model['Y'])
fig, ax = plt.subplots(figsize=f_size)
ax.plot( [0,ylim],[0,ylim] )
ax.scatter(df_train_model['Y'],df_train_model['yhat'],c = 'g')
ax.set(title = 'y vs y hat for train data for model' , xlabel = 'y', ylabel = 'yhat')
plt.show()

#test data with reduced linevector A - true vs error plot
df_train_model['sqerr'] = np.asarray(np.square(df_train_model['Y']-df_train_model['yhat']))
df_train_model.plot('Y','sqerr',style='o',title = 'y true vs sq error for train model')

#model prediction for test data
pred_y = clf.predict(df_test.loc[:,'X1':'X11'])
pred_y
rmse_model_test = cal_rmse(df_test.loc[:,'Y'],pred_y)

#print('cal_performance_params variable mat_pred_x:\n',mat_pred_x)
#reshaping arguments for RMSE
mat_y_true = np.asmatrix(df_test['Y'])
mat_y_pred = np.asmatrix(pred_y).reshape(mat_y_true.shape)
```

```
#calculate ratio RMSE/avy for test
val_avy_test = np.mean(np.abs(mat_y_true))
ratio_rmse_avy_test = np.around(rmse_model_test/val_avy_test,4)
print('rmse: ',rmse_model_test)
print('ratio rmse/avy: ',ratio_rmse_avy_test)

confint_ratio_test = err_est_element(ratio_rmse_avy_test,df_test['Y'].shape[0],False)

print('For 95% confidence level, confidence interval for ratio {0} is {1}'.format(ratio_rmse_avy_test,
                                                        confint_ratio_test))

df_test_model = df_test.copy()
df_test_model['yhat'] = np.asarray(pred_y)

plot_results(df_test_model,title= 'y vs y hat for train data for model')

#plotting y vs yhat for test for best parameters for reduced LV

ylim = np.max(df_test_model['Y'])
fig, ax = plt.subplots(figsize=f_size)
ax.plot( [0,ylim],[0,ylim] )
ax.scatter(df_test_model['Y'],df_test_model['yhat'],c = 'g')
ax.set(title = 'y vs y hat for train data for model' , xlabel = 'y', ylabel = 'yhat')
plt.show()

#test data with reduced linevector A - true vs error plot
df_test_model['sqerr'] = np.asarray(np.square(df_test_model['Y']-df_test_model['yhat']))
df_test_model.plot('Y','sqerr',style='o',title = 'y true vs sq error for train model')
```