MATH 6350

STATISTICAL LEARNING AND DATA MINING

PROJECT ON

PREDICTION OF STOCK MARKET PRICES OF HEALTH SECTOR COMPANIES

PART-I

QUESTION 1

1. **Download daily stockprices data at" close" time for the past 5 years in one economic sector such as health care. This gives you at each day t a vector V(t) of p stockprices X1(t) ... Xp(t)**

   Description of Dataset:

The dataset chosen is stock market closing price of following 10 companies from Health care sector:

S No	Name	Ticker
1	CVS Health	CVS
2	Danaher Corp.	DHR
3	DaVita Inc.	DVA
4	Dentsply Sirona	XRAY
5	Edwards Lifesciences	EW
6	Gilead Sciences	GILD
7	HCA Healthcare	HCA
8	Henry Schein	HSIC
9	Hologic	HOLX
10	Humana Inc.	HUM

Table 1.1. Stocks data and companies chosen

This data has been extracted from dates: '2015-01-01' to '2019-12-31' – 1258 cases.

Descriptive statistics of the data are :

	CVS	DHR	DVA	XRAY	EW	GILD	HCA	HSIC	HOLX	HUM
count	1258	1258	1258	1258	1258	1258	1258	1258	1258	1258
mean	80.92	92.4	67.32	54.74	123.9	80.4	96.05	62.01	40.32	234.56
std	15.87	24.49	8.55	7.774	46.71	15.36	24.51	5.501	5.015	56.249
min	52.13	62.08	43.42	34.32	61.93	60.54	62.83	49.76	25.75	139.09
25%	68.24	72.67	60.47	50.89	87.43	67.74	76.89	57.4	37.44	180.08
50%	78.79	85.45	67.63	56.54	114.2	74.9	84.98	62.02	39.53	234.83
75%	96.89	102.6	73.89	60.81	148.6	88.77	122.8	66.62	43.52	280.18
max	113.4	153.5	84.23	68.58	246.3	122.2	149.3	73.16	53.56	371

Table 1.2.Descriptive statistics of original dataset

2. For each stock price Xj(t) compute the recent average xj(t) = [Xj(t)+ Xj(t-1) +... +Xj(t-4)]/5 define XXj(t) = +1 if Xj(t)> xj(t) and XXj(t)= -1 otherwise for each j = 1...p

Answer:

First 5 features of recent average table x(t) is :

	Date	CVS	DHR	DVA	XRAY	EW	GILD	HCA	HSIC	HOLX	HUM
0	1/2/15	0	0	0	0	0	0	0	0	0	0
1	1/5/15	0	0	0	0	0	0	0	0	0	0
2	1/6/15	0	0	0	0	0	0	0	0	0	0
3	1/7/15	0	0	0	0	0	0	0	0	0	0
4	1/8/15	95.48	64.56	74.84	52.04	64.57	98.23	73.08	53.83	26.35	142.17

Table 2.1. First 5 features of recent average table

Descriptive statistics of above dataset are :

	CVS	DHR	DVA	XRAY	EW	GILD	HCA	HSIC	HOLX	HUM
count	1258	1258	1258	1258	1258	1258	1258	1258	1258	1258
mean	80.65	92.06	67.08	54.57	123.5	80.14	95.7	61.82	40.19	233.75
std	16.47	24.87	9.293	8.332	46.92	15.95	24.96	6.46	5.43	57.303
min	0	0	0	0	0	0	0	0	0	0
25%	67.92	72.5	60.21	51	87.24	67.76	76.88	57.33	37.45	179.6
50%	78.72	85.27	67.42	56.57	113.7	74.73	84.95	61.93	39.5	235.14
75%	96.75	102.4	73.76	60.74	148.8	88.64	122.5	66.78	43.51	279.43
max	112.7	153.1	83.98	68.01	245	121.3	148.4	72.76	53.11	369.1

Table 2.2. Descriptive statistics of recent average

Number of cases in CL1 are 652 and CL-1 are 606.

Below figure shows the timeseries plot of CVS stock values(orange) , 5 day recent moving average(blue) – x(t) and change in trends(green colour) – XX(t).

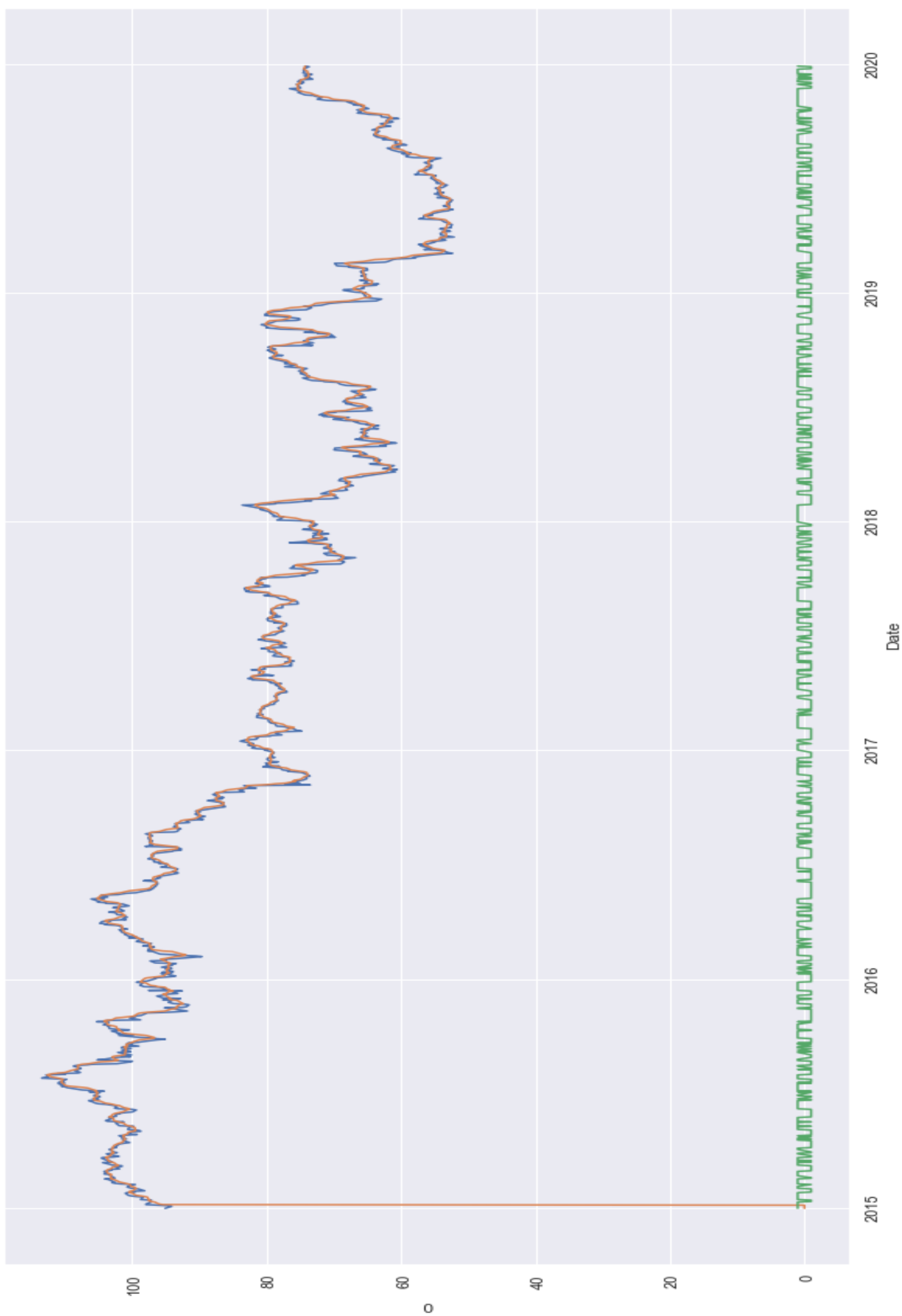


Fig 2.1. Time series plot of CVS stock prices, 5 day moving average and trend line

3. **Construct the best svm classifier SVMj to predict XXj(t+1) on the basis of V(t) V(t-1) ... V(t-9)**

Answer:

Each V(t) is a vector with 10 companies closing price. A new table has been created with 10 vectors V(t) to V(t-9) as features with XXj(t+1) for first company CVS is selected as response variable.

Top 5 rows of new dataset is :

	0	**1**	**2**	**3**	**97**	**98**	**99**	**XX_tplus1**
0	97.17	62.19	74.46	73.97	53.66	26.38	142.99	1
1	98.74	62.58	75.05	71.81	53.18	26.12	139.2	1
2	98.37	62.9	75.21	71.69	52.87	25.75	139.09	1
3	99.5	63.12	75.73	72.99	54.33	26.4	141.73	1
4	100.28	64.06	76.15	74.93	55.1	27.11	147.83	1

Table 2.1. Top 5 rows of new dataset

The proportion of cases in the dataset is maintained in class 1 and class -1 :

	CL 1	**CL -1**	**Total**
new dataset	652	606	1258
Train set	522	477	999
Test set	130	119	249

Table 2.2. Proportion of cases in new dataset, train and test datasets

Data Preprocessing is done as below :

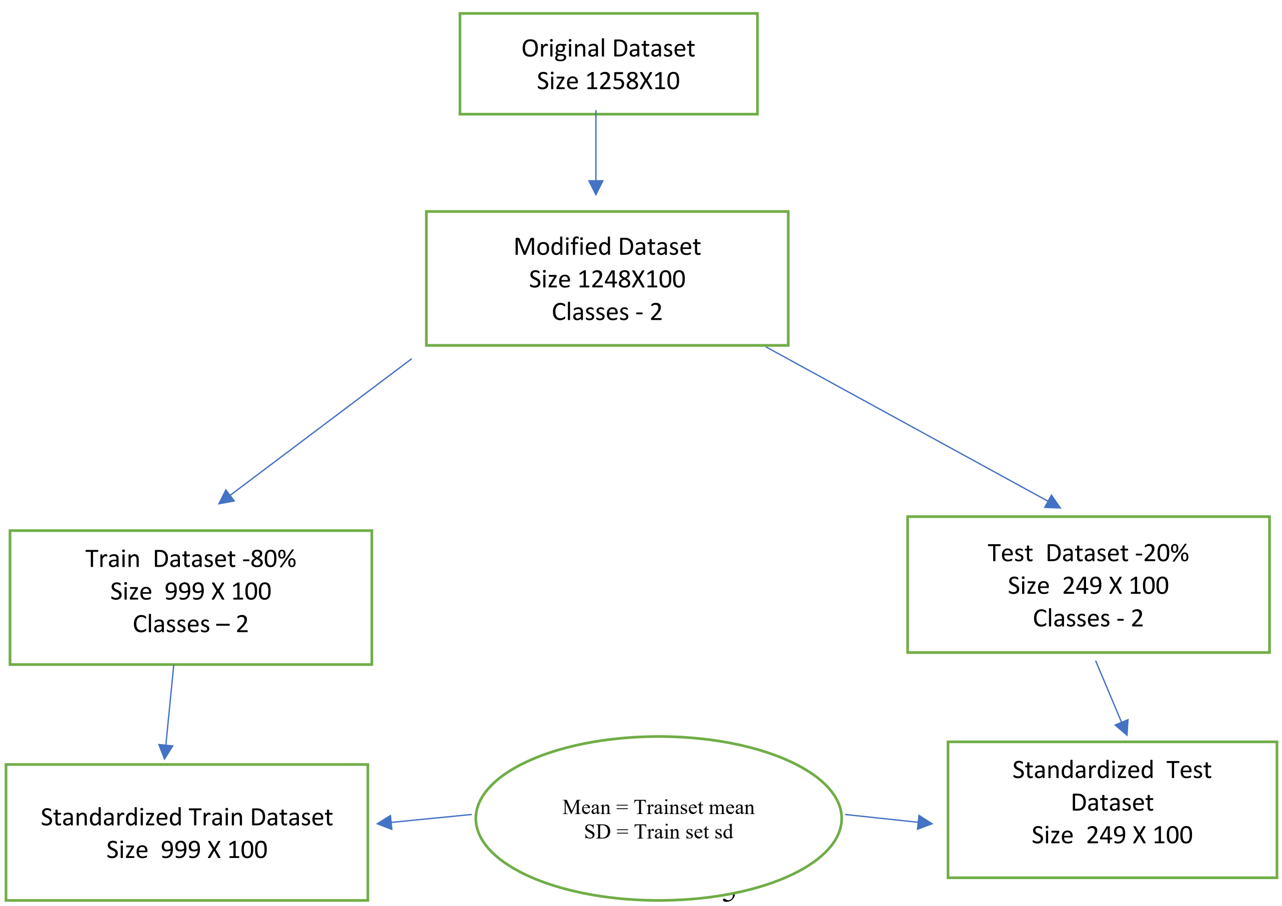


Figure 3.1. Preprocessing flowchart

a) Linear SVM best parameters

- **Linear kernel definition**

  Linear Kernel is given by

  K(x,y) = 1 + <x,y> ,
  where <x,y> refers to scalar product of vectors x and y in p dimensional space $\mathrm{R_p}$
- kernel_type: linear; best parameters: {'C': 10, 'kernel': 'linear'}
-

------Fitting train data into svm model-----
Score for above parameters in SVC model : 0.7908

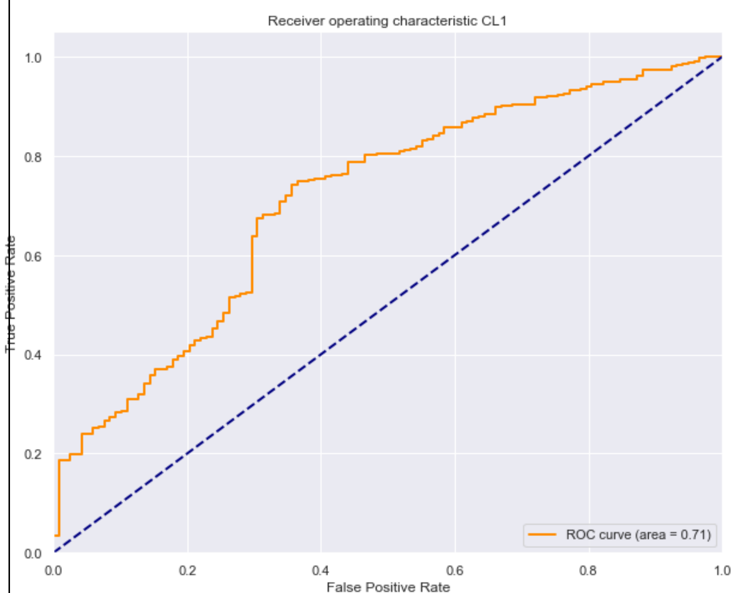


Figure 3.2.ROC of best linear SVM for CL1

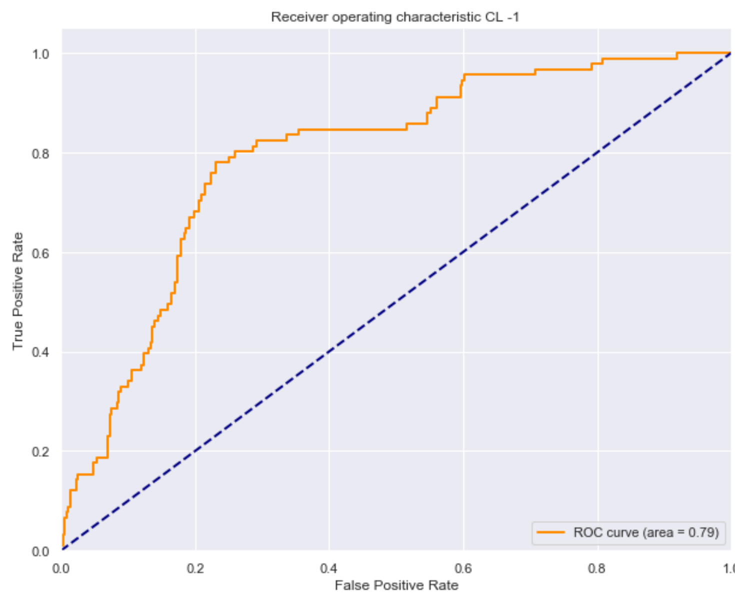


Figure 3.3.ROC of best linear SVM for CL-1

Number of Support vectors,S : 577
Ratio of Support vectors,s : 0.58

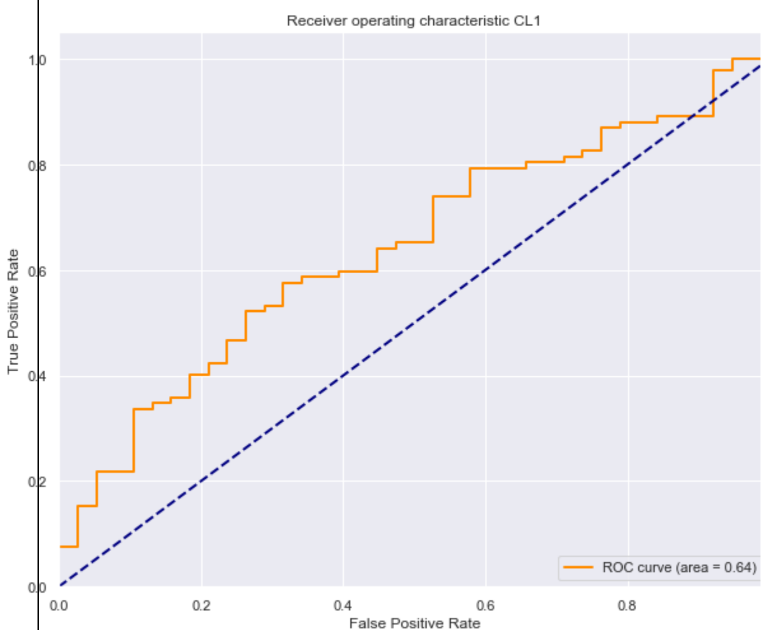
Ratio of Support vectors, $s = \frac{number\ of\ support\ vectors}{size\ of\ train\ set}$ = 0.58
Support vectors are cases that lie on the margin or one wrong side of margin for their class. Hence they affect the Support vector machine. And SVM acts as a robust classifier for remaining vectors.
Out of 999 training vectors almost 58% vectors are support vectors, which implies that the model is poor in classification of train data.

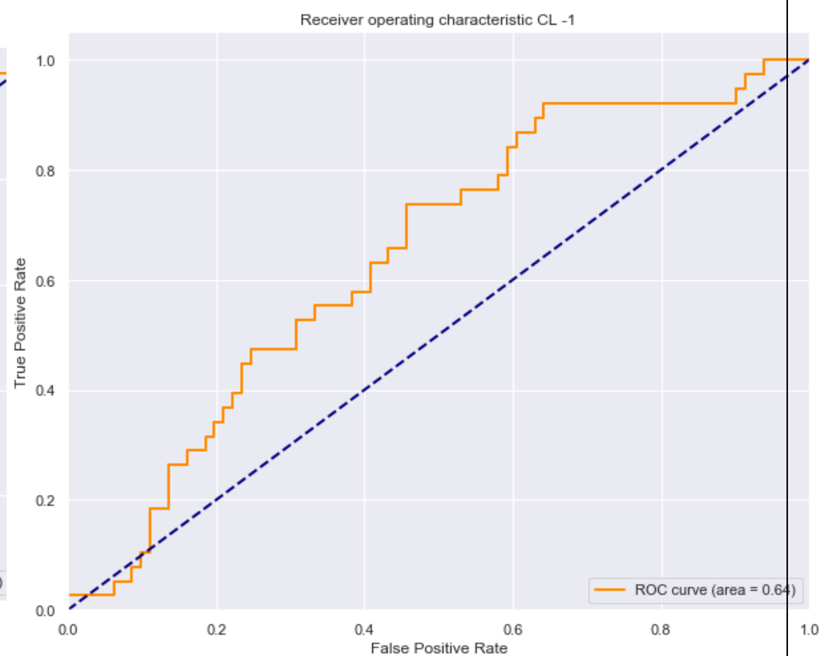
Significance of ROC curve :

1. the closer the curve follows left hand border and the top border of ROC space, the more accurate the model is.
2. The closer the curve comes to 45-degreee diagonal of ROC space , the less accurate the mode is
3. The area under ROC curve (AUC) represents the discriminating ability of model to correctly classify the cases.

ROC plot in Fig 3.2 and 3.3 for train set represents that the svm has better capacity to predict CL-1 than CL 1 as AUC is more in case of CL - 1.

------SVM applied on test set-----
Score for above parameters in SVC model for testset : 0.6948





ROC plot in above figure for test data represents that the svm has equal capacity to predict CL-1 and CL1 as AUC is same in both cases.

Confusion matrix for Train data :

	pred_CL1	**pred_CL_1**
true_CL1	82.57	17.43
true_CL_1	24.74	75.26

For 95% confidence level, confidence interval :

	pred_CL1	**pred_CL_1**
true_CL1	[79.32, 85.82]	[14.18, 20.68]
true_CL_1	[20.86, 28.62]	[71.38, 79.14]

Confusion matrix for Test data :

	pred_CL1	**pred_CL_1**
true_CL1	70.77	29.23
true_CL_1	31.93	68.07

For 95% confidence level, confidence interval :

	pred_CL1	**pred_CL_1**
true_CL1	[62.95, 78.59]	[21.41, 37.05]
true_CL_1	[23.56, 40.3]	[59.7, 76.44]

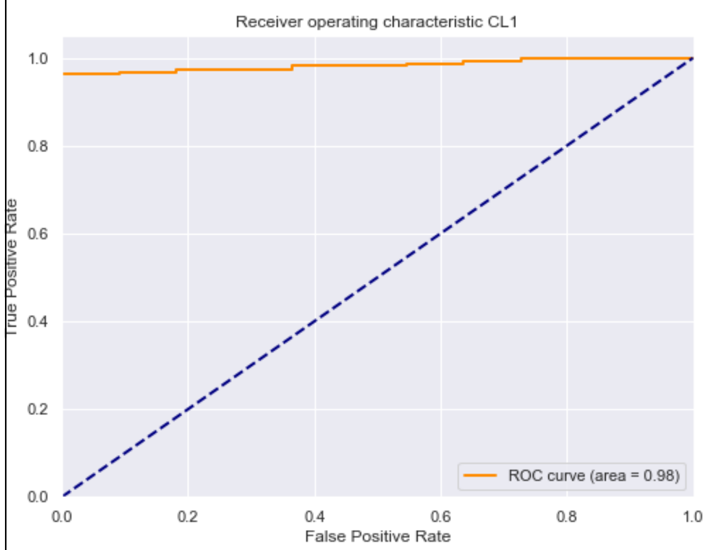
By comparing confidence intervals for train and test sets:

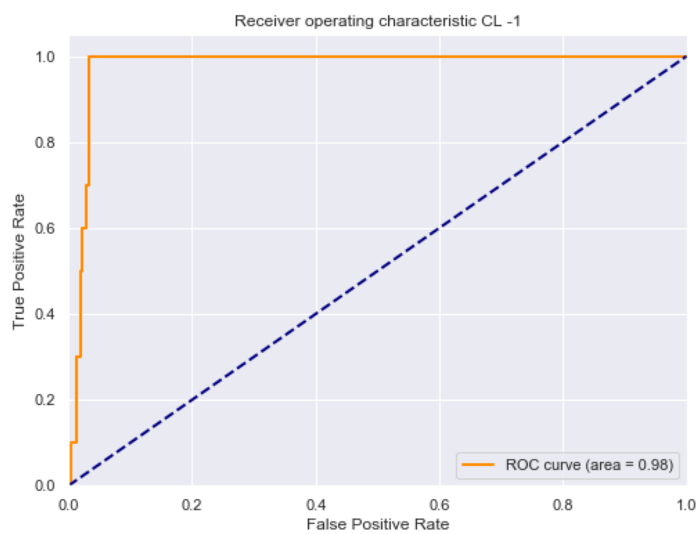
1. For CL1 class, the confidence interval is narrower in the train set than test set, hence we are more certain about the value than test set. But accuracy level is less in test set than train set.
2. For CL-1 class, confidence intervals for train and test are overlapping, hence there is no statistically significant difference between their performances.

b. Radial SVM

- Radial kernel is $K(x, y) = e^{-gamma||x-y||^2}$
- Best parameters are {'C': 20, 'gamma': 0.1, 'kernel': 'rbf'}

kernel_type: rbf
------Fitting train data into svm model-----
Score for above parameters in SVC model : 0.979

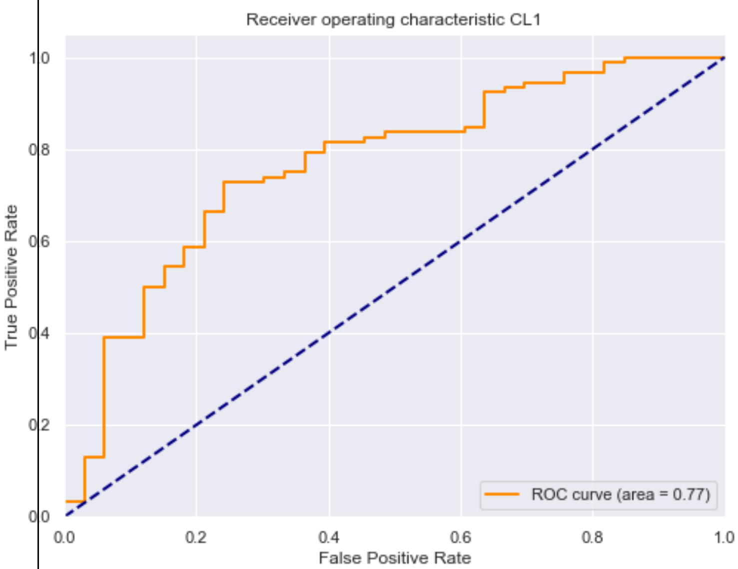


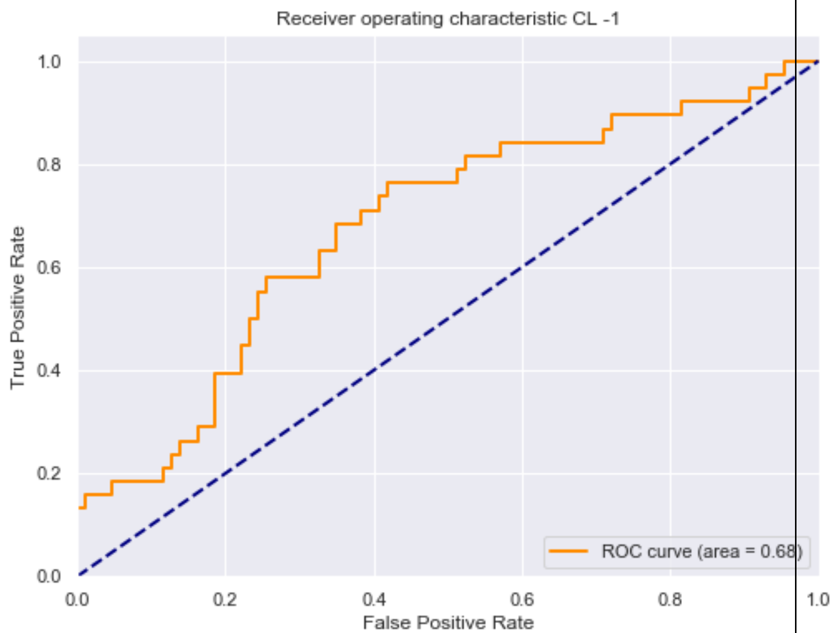


Number of Support vectors,S : 695
Ratio of Support vectors,s : 0.7
Out of 999 training vectors almost 70% vectors are support vectors, which implies that the model is poor in classification of data.
ROC plots for train set represents that the svm has equal capacity to predict CL-1 than CL 1 as AUC is same for both.

------SVM applied on test set-----

Score for above parameters in SVC model for testset : 0.7149





ROC plots for test set represents that the svm has better capacity to predict CL 1 than CL -1 as AUC is more for CL1 than CL-1.

Confusion Matrix:

Confusion matrix for Train:

	pred_CL1	**pred_CL_1**
true_CL1	98.08	1.92
true_CL_1	2.31	97.69

For 95% confidence level, confidence interval :

	pred_CL1	**pred_CL_1**
true_CL1	[96.9, 99.26]	[0.74, 3.1]
true_CL_1	[0.96, 3.66]	[96.34, 99.04]

Confusion matrix for Test:

	pred_CL1	**pred_CL_1**
true_CL1	70.77	29.23
true_CL_1	27.73	72.27

For 95% confidence level, confidence interval :

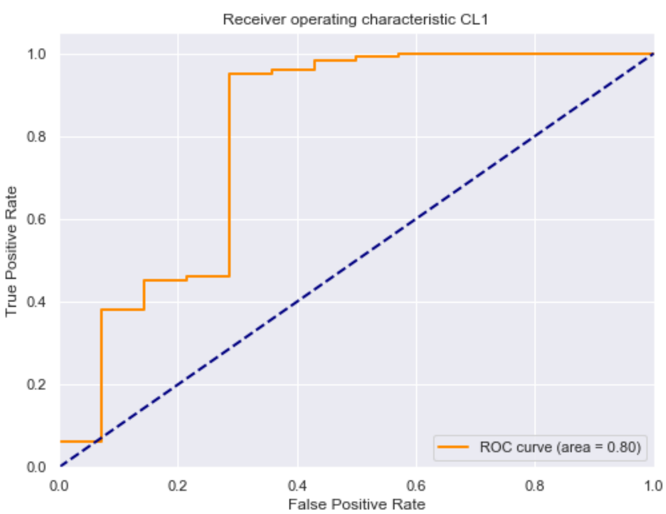
	pred_CL1	**pred_CL_1**
true_CL1	[62.95, 78.59]	[21.41, 37.05]
true_CL_1	[19.69, 35.77]	[64.23, 80.31]

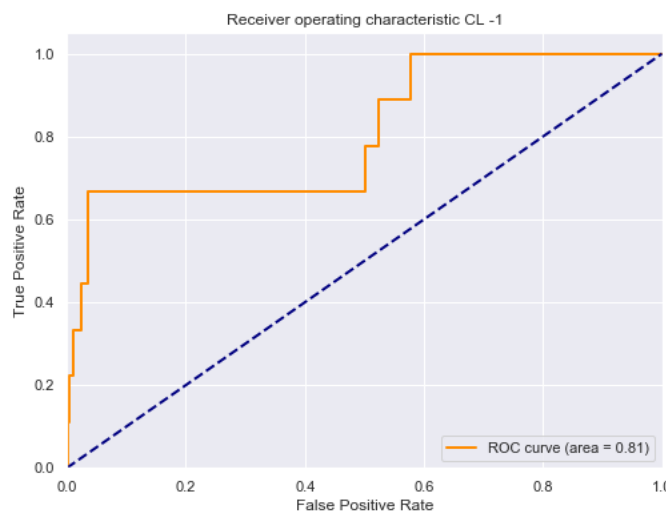
Confusion matrix for test data indicates that the radial kernel has same performance for both CL1 and CL-1, as the confidence interval is overlapping for both datasets,which indicates that there is no statistically significant difference. Moreover when compared to train data, test data has broader range hence we are more uncertain for test dataset than train dataset.

c. Polynomial SVM

- polynomial kernel K(x,y) = (a + <x,y>)^4 .
- Here a is represented as coef0.
- Best parameters after tuning are {'C': 0.01, 'coef0': 20, 'gamma': 0.05, 'kernel': 'poly'}

kernel_type: poly
------Fitting train data into svm model-----
Score for above parameters in SVC model : 0.977

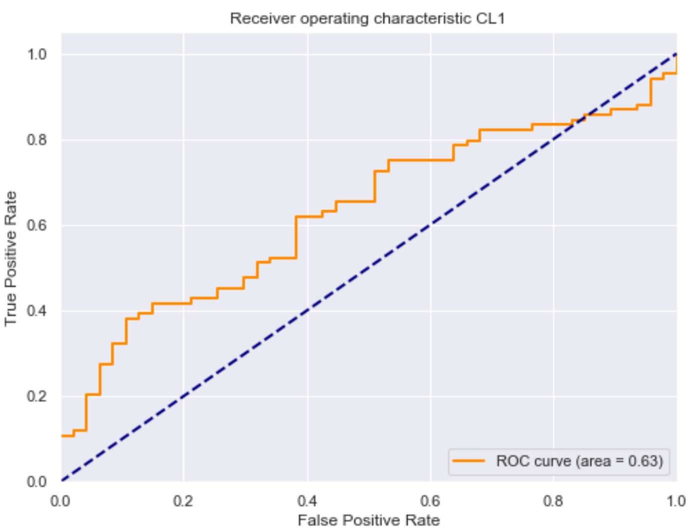


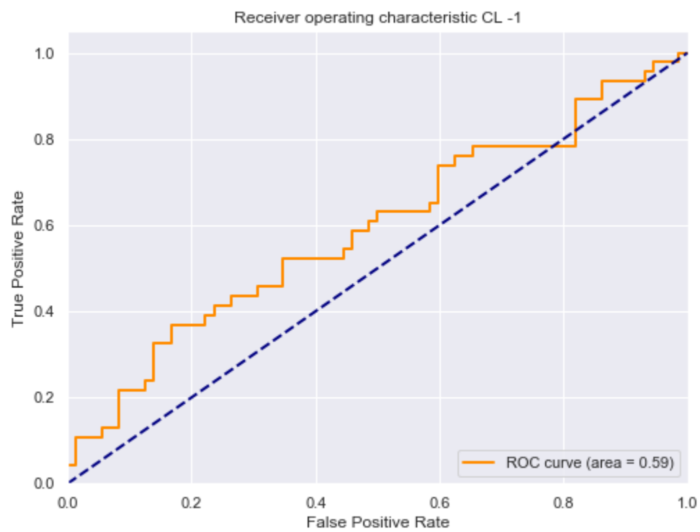


Number of Support vectors,S : 515
Ratio of Support vectors,s : 0.52

Out of 999 training vectors almost 52% vectors are support vectors, which implies that the model is poor in classification of data.
ROC plots for train set represents that the svm has equal capacity to predict CL-1 than CL 1 as AUC is same for both.

------SVM applied on test set-----





Score for above parameters in SVC model for test set : 0.6265
ROC plots for test set represents that the svm has better capacity to predict CL 1 than CL -1 as AUC is more for CL1 than CL-1.

<u>Confusion Matrix for train data :</u>
Confusion matrix for Train:

	pred_CL1	**pred_CL_1**
true_CL1	98.08	1.92
true_CL_1	2.31	97.69

For 95% confidence level, confidence interval :

	pred_CL1	**pred_CL_1**
true_CL1	[96.9, 99.26]	[0.74, 3.1]
true_CL_1	[0.96, 3.66]	[96.34, 99.04]

<u>Confusion Matrix for test data :</u>

Confusion matrix for Test:

	pred_CL1	**pred_CL_1**
true_CL1	70.77	29.23
true_CL_1	27.73	72.27

For 95% confidence level, confidence interval :

	pred_CL1	**pred_CL_1**
true_CL1	[62.95, 78.59]	[21.41, 37.05]
true_CL_1	[19.69, 35.77]	[64.23, 80.31]

Confusion matrix for test data indicates that the polynomial kernel has same performance for both CL1 and CL-1, as the confidence interval is overlapping for both datasets, which indicates that there is no statistically significant difference. Moreover when compared to train data, test data has broader range hence we are more uncertain for test dataset than train dataset.

Comparision of performance for linear, radial and polynomial kernels indicates that :

1. All three kernels have similar performance as the confidence intervals are overlapping for both classes
2. However Polynomial kernel has lesser proportion of support vectors, hence it can be considered as the best kernel of SVM classification for predicting class for 'CVS' stock.

4. Construct the best Kernel Ridge Regression KRRj to estimate the price Xj(t+1) on the basis of V(t) V(t-1) ... V(t-10)

Answer:

- Select the kernel = "radial "kernel K(x,y) defined for x and y in Rp by the formula

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

where $\boldsymbol{\gamma}$ >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 " γ " 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]

- In general, Gramian matrix G of a set of vectors $x_1, x_2 \dots \dots \dots . x_m$ v 1 , ... , v n {\displaystyle v_{1},\dots ,v_{n}} 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.

- 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

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

- 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

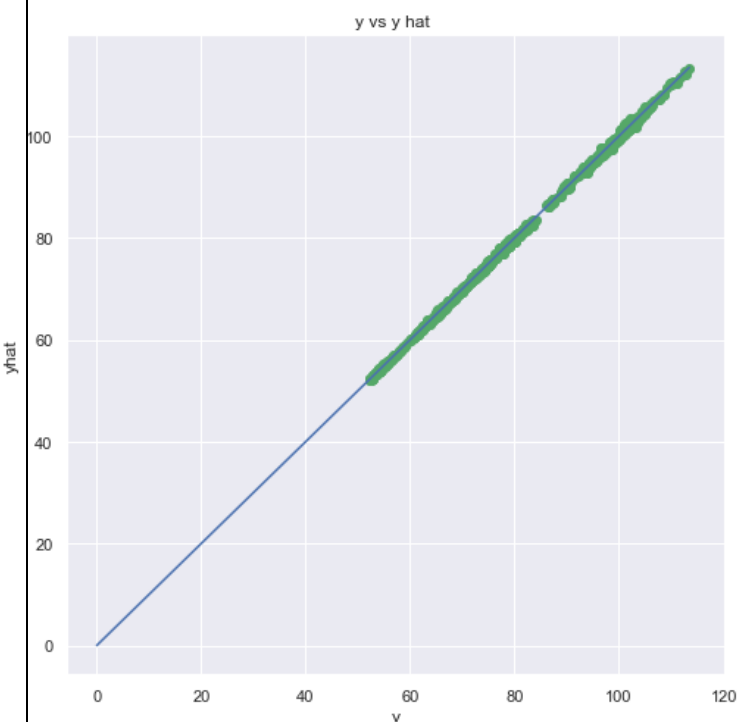
- RMSE can be normalized by

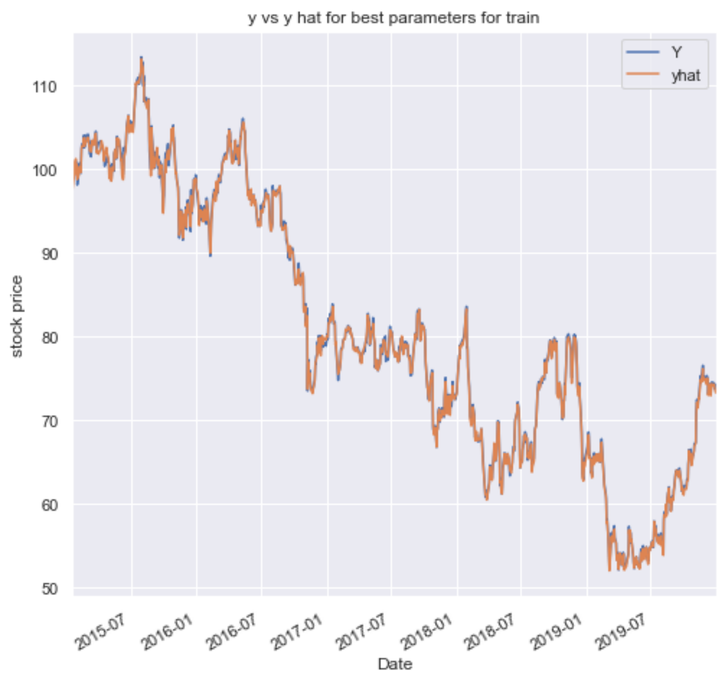
$$ratio = \frac{RMSE}{avy},$$

Where , $avy = \frac{\sum_1^k |y_i|}{k}$, i.e., mean of true values of target variables.

- Methodology followed for tuning of parameters is :
- After selecting a baseline values, in each step, only one of the parameter is tuned with 2 combinations –
  - When $\lambda = \lambda_0, \gamma = [\frac{\gamma_0}{2}, 2 * \gamma_0]$
  - Similarly when , $\gamma = \gamma_0, \lambda = [\frac{\lambda_0}{2}, \ 2 * \lambda_0]$
- 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.
- best_params after parameter tuning are : {'lambda': [0.0206], 'gamma': [0.0005]}

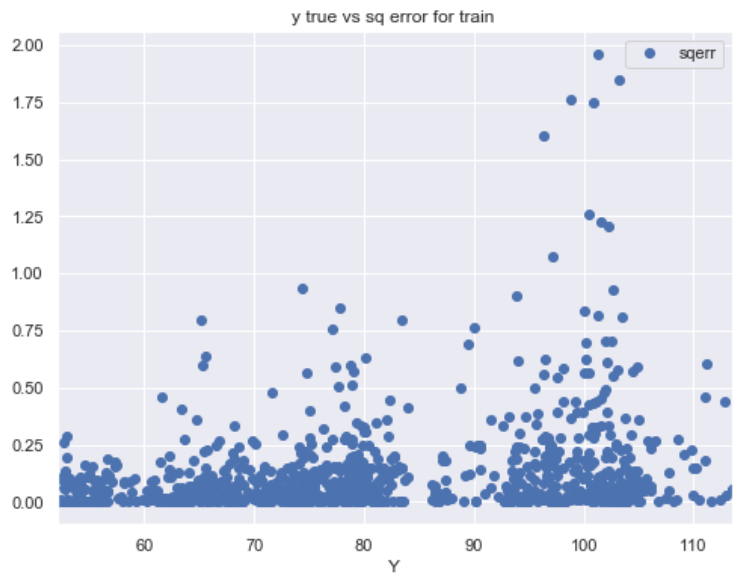
####train performance for best parameters####





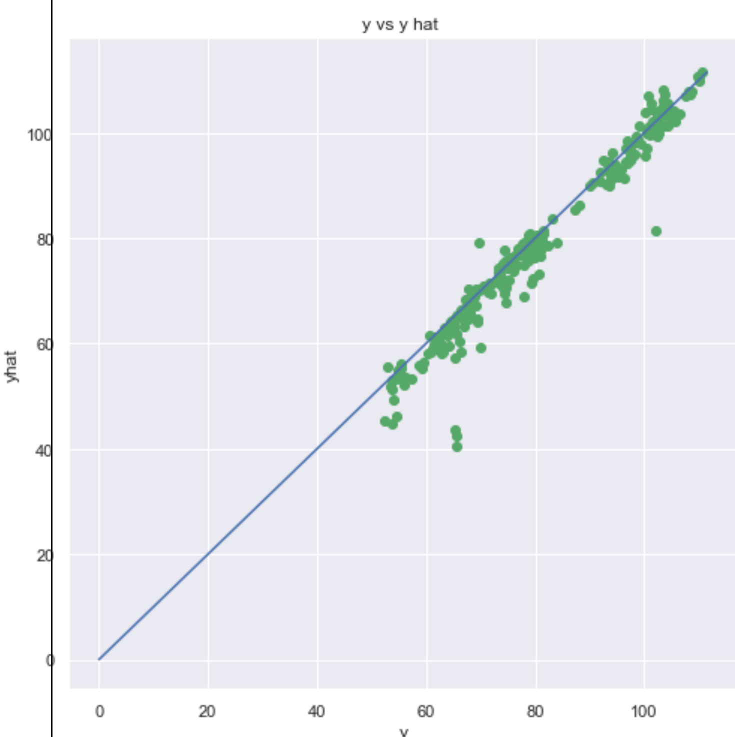
The above plot is scatter plot of true y values and predicted values of y(y hat), along with line of identity for reference. This indicates that predicted values of y are lesser in magnitude than true y. The same is reflected in the time series plot shown below for date against y and yhat.

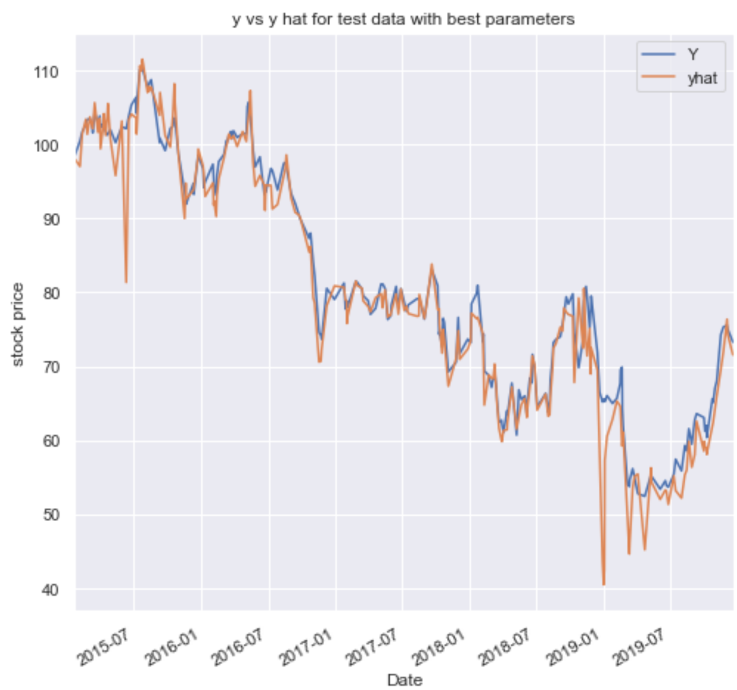
- Performance measures are
  - val_rmse: 0.3481
  - ratio rmse/avy : 0.0043
  - sigma: 0.0021
  - For 95% confidence level, confidence interval for ratio 0.0043 is [0.0002, 0.0084]
- Both of the above plots indicate that the model is able to closely predict the target variable.



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

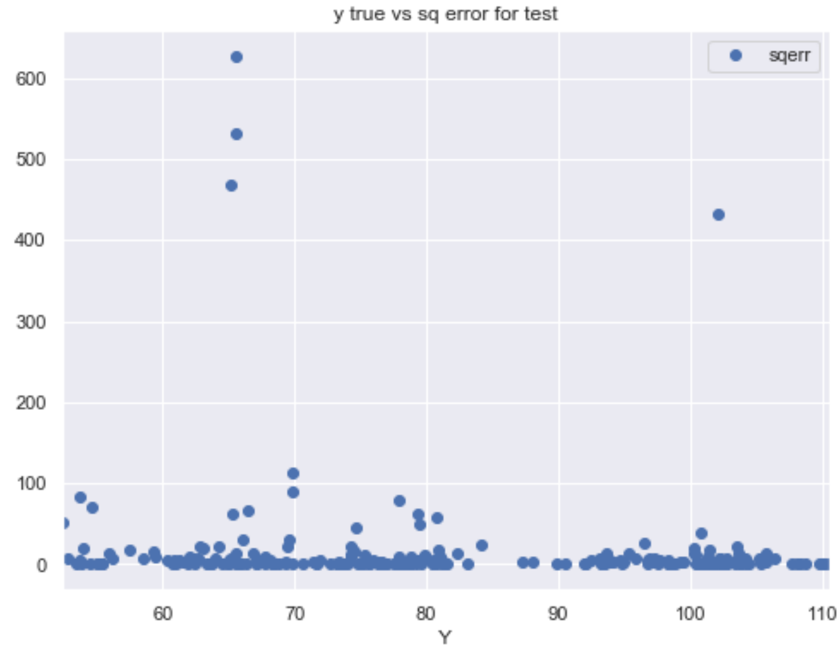
####test performance for best parameters####





- Performance measures are
    - val_rmse: 3.9934
    - ratio rmse/avy : 0.0491
    - sigma: 0.0137
    - For 95% confidence level, confidence interval for ratio 0.0491 is [0.0222, 0.076]

- 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.
- When compared to train set's performance with confidence interval of [0.0002, 0.0084] ,the performance of the model is lower for test dataset.

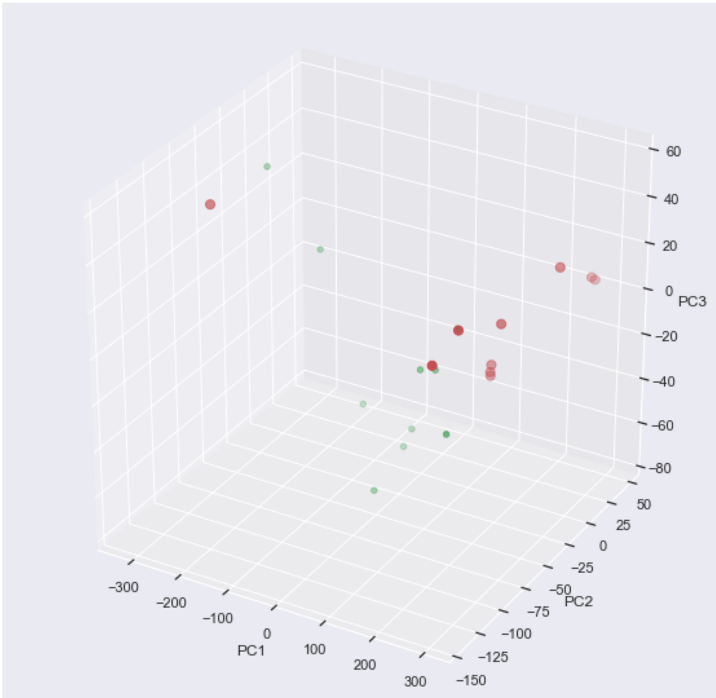


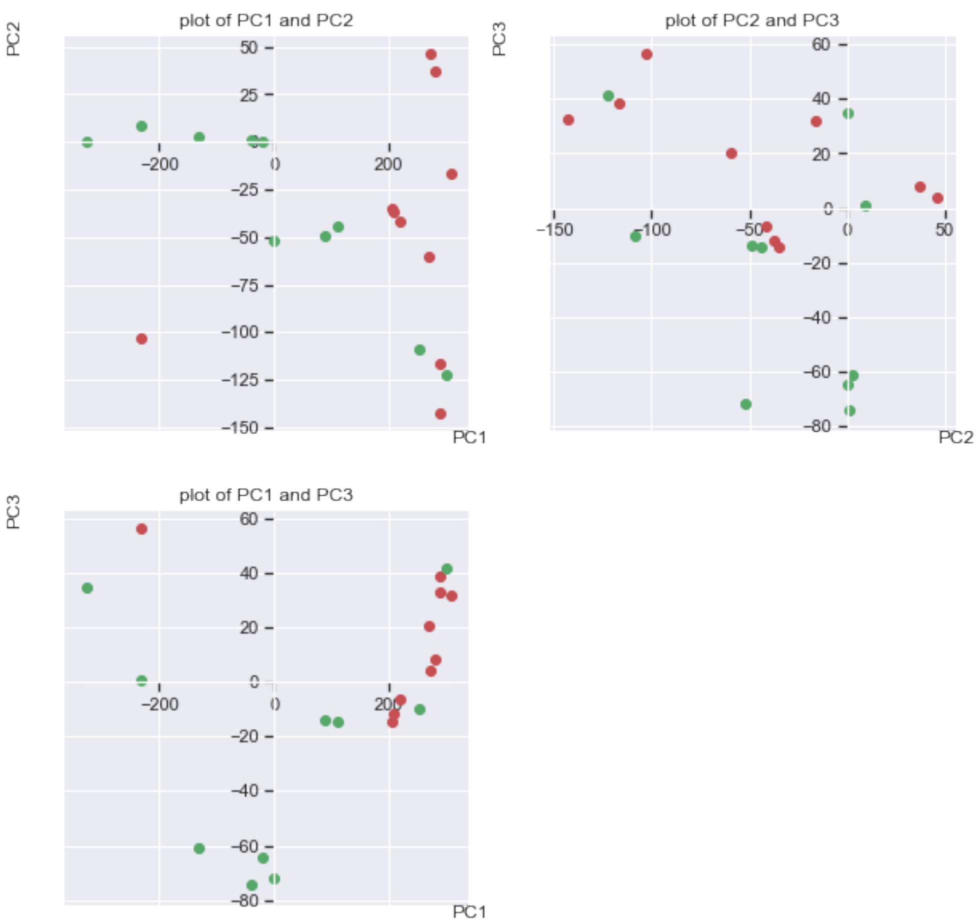
The above plot of ytrue vs squared error indicate that error in prediction is higher for y true values in between 60 to 70$ stock prices.

#Worst case analysis

Visualise 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) :
explained_variance_ratio : 0.96

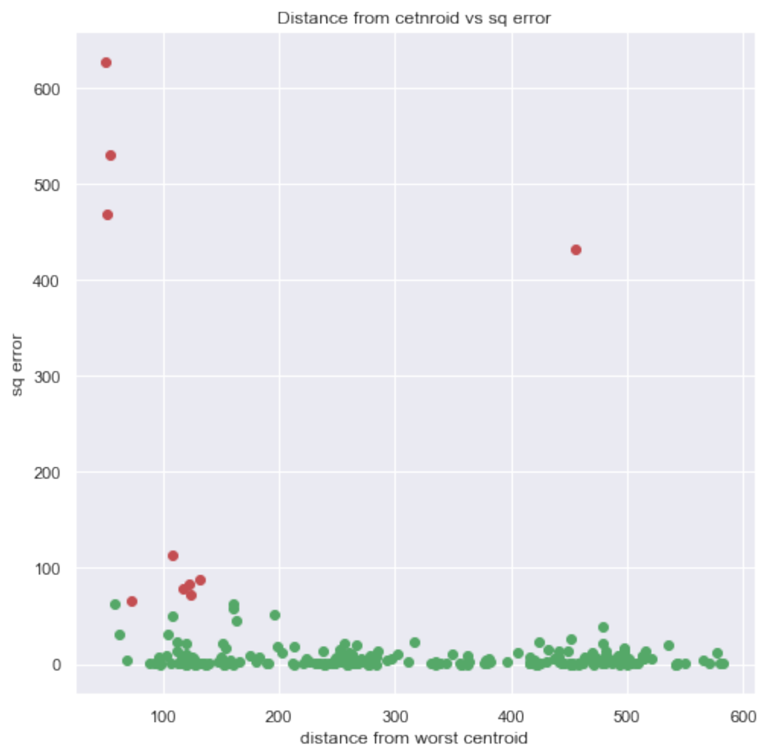




PCA visualization indicates that majority of worst test cases are concentrated with PC1 component greater than 200.

Analysis 1:

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 observations are located very near to their centroid compared to remaining cases.

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 :

[[792 793 797 795 791]
[793 792 797 795 794]
[792 791 793 789 795]
[78 77 76 79 80]
[817 818 820 819 811]
[741 728 733 731 767]
[827 828 826 829 825]
[774 775 784 786 787]
[828 827 829 826 830]
[789 788 787 791 792]]

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

mat_pred_best10:
[[487 486 491 468 471]
[230 229 223 237 232]
[630 631 627 634 639]
[526 528 529 527 525]
[708 711 709 707 712]
[774 775 773 784 772]
[28 26 27 10 22]
[491 487 486 492 489]
[450 455 410 458 460]
[630 644 639 631 634]]

On comparison of [m1 m2 m3 m4 m5] and [M1 M2 M3 M4 M5] , top 3 worst cases are influenced particularly by 789 -797 indices of pred(x) and remaining 728 – 830 indices.

5.compare the quality of results 1 versus 2

Answer:

a. SVM classification model is created for classifying test dataset data into
   i. class 1 – increasing trend of stock price compared to 5 day recent moving average
   ii. class -1 - decreasing or same price trend of stock price compared to 5 day recent moving average

   confusion matrix with 95% confidence level for best SVM classification with polynomial kernel is

	pred_CL1	**pred_CL_1**
true_CL1	[62.95, 78.59]	[21.41, 37.05]
true_CL_1	[19.69, 35.77]	[64.23, 80.31]

b. Prediction of stock price for next day (t+1) is done for CVS stock price using kernel ridge regression with radial kernel. Following are the performance measures on test set are:

   val_rmse: 3.9934
   ratio rmse/avy : 0.0491
   sigma: 0.0137
   For 95% confidence level, confidence interval for ratio 0.0491 is [0.0222, 0.076]

PART – II
Python Jupyter notebook code

```
import os
import pandas as pd
import numpy as np
from scipy import io
import math
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')

from pandas import ExcelWriter
from sklearn.decomposition import PCA
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedShuffleSplit
from matplotlib.colors import Normalize

from scipy import interp
from sklearn import svm, datasets
from sklearn.metrics import roc_curve, auc

from sklearn.kernel_ridge import KernelRidge

#Read dataset from Github
get_ipython().system('pip install -q xlrd')
get_ipython().system('git clone https://github.com/kishoret04/statisticallearning_datamining.git')
##Files from the cloned git repository.
get_ipython().system('ls  statisticallearning_datamining/new_project/')

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

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

df_totaldata.describe()


# compute the recent average xj(t) = [Xj(t)+ Xj(t-1) +... +Xj(t-5)]/5
#     define  XXj(t) = +1 if Xj(t)> xj(t) and XXj(t)= -1 otherwise for each j = 1...p
#calculate recent moving average for window =5
df_rct_avg_5 = df_totaldata.rolling(window=5).mean()
df_rct_avg_5.fillna(0,inplace=True)
df_rct_avg_5.insert(0,'Date',df_totaldata['Date'])
df_rct_avg_5
```

```
df_rct_avg_5.head()


df_rct_avg_5.describe()

df_modified_data_5 = pd.DataFrame(np.where(df_totaldata.loc[:,'CVS']>df_rct_avg_5.loc[:,'CVS'],1,-1))
df_modified_data_5.insert(0,'Date',df_totaldata['Date'])
df_modified_data_5


sns.set(rc={'figure.figsize':(20,10)})
pd.plotting.register_matplotlib_converters()
sns.lineplot(data = df_totaldata,x = 'Date',y = 'CVS')
sns.lineplot(data = df_rct_avg_5,x = 'Date',y = 'CVS')

sns.lineplot(data = df_modified_data_5,x = 'Date',y = 0)



df_temp = df_totaldata.copy()
df_temp.drop(columns = 'Date',inplace=True)

#create dataset with predictor variables as v(t) to v(t-9) with XX(t+1) as class
list_columns = list(range(10*df_temp.shape[1]))
df_newdata = pd.DataFrame(columns = list_columns)
LIMIT_LOWER = 9
limit_upper = df_temp.shape[0]-1

#create new dataset
for i in range(LIMIT_LOWER,limit_upper):
    row = df_temp.loc[i-9:i][::-1].T.melt()['value']
    row.index = range(0,len(list_columns))
    df_newdata = df_newdata.append(row,ignore_index = True)


#adding class column with name XX_tplus1
XX_tplus1 = df_modified_data_5.loc[LIMIT_LOWER+1:limit_upper,0]
XX_tplus1.index = range(0,limit_upper-LIMIT_LOWER)
df_newdata['XX_tplus1'] = XX_tplus1


# In[130]:


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

with ExcelWriter(filepath) as writer:
    df_totaldata.to_excel(writer,sheet_name = 'total_data' )
    df_rct_avg_5.to_excel(writer,sheet_name = 'recent_avg_data_5' )
    df_modified_data_5.to_excel(writer,sheet_name = 'modified_data_5' )
    df_newdata.to_excel(writer,sheet_name = 'newdata' )
    writer.save()


# In[461]:
```

```
df_test[df_test['XX_tplus1'] == 1].describe()


# In[462]:


df_test[df_test['XX_tplus1'] == -1].describe()


# In[454]:


df_newdata.describe()


# In[134]:


#create train and test data with equal proportion of cl1 to cl-1
def create_traintestdata(prop,df_input):
   #calculate trainset size for cl1 and cl_1
   train_cl1_size = round(0.8*df_input[df_input['XX_tplus1'] == 1].shape[0])
   train_cl_1_size = round(0.8*df_input[df_input['XX_tplus1'] == -1].shape[0])

   #cl1 cl_1 classification
   df_cl1 = df_input[df_input['XX_tplus1'] == 1]
   df_cl_1 = df_input[df_input['XX_tplus1'] == -1]

   #extract a random sample of 80% CL1 as train and 20% CL1 as test set
   df_train = df_cl1.sample(train_cl1_size)
   df_test = df_cl1.drop(df_train.index)

   #extract a random sample of 80% CL-1 as train and 20% CL-1 as test set
   cl_1_train = df_cl_1.sample(train_cl_1_size)
   cl_1_test = df_cl_1.drop(cl_1_train.index)

   #creating train and test sets by combinining CL1 and CL-1
   df_train = df_train.append(cl_1_train,ignore_index = True)
   df_test = df_test.append(cl_1_test,ignore_index = True)

   #Reset index
   df_train.reset_index(drop=True,inplace = True)
   df_test.reset_index(drop=True,inplace = True)

   return df_train,df_test


df_train,df_test = create_traintestdata(80,df_newdata)


# In[455]:


df_train.describe()


# In[456]:
```

```
df_test.describe()


# In[136]:


df_train_std = (df_train - df_train.mean())/df_train.std()


# In[137]:


df_train_std.drop(columns='XX_tplus1',inplace=True)
df_train_std['class'] = df_train['XX_tplus1']
df_train_std


# In[138]:



df_test_std = (df_test - df_train.mean())/df_train.std()
df_test_std.drop(columns='XX_tplus1',inplace=True)
df_test_std['class'] = df_test['XX_tplus1']
df_test_std


# In[157]:


class MidpointNormalize(Normalize):
    def __init__(self, vmin=None, vmax=None, midpoint=None, clip=False):
        self.midpoint = midpoint
        Normalize.__init__(self, vmin, vmax, clip)

    def __call__(self, value, clip=None):
        x, y = [self.vmin, self.midpoint, self.vmax], [0, 0.5, 1]
        return np.ma.masked_array(np.interp(value, x, y))


class SvmClassification:

    train_data = pd.DataFrame
    train_labels = pd.DataFrame
    train_Un  = pd.DataFrame
    test_data = pd.DataFrame
    test_labels = pd.DataFrame
    test_Un  = pd.DataFrame
    svm_master= Pipeline
    pred_train = pd.DataFrame
    pred_test = pd.DataFrame
    confmat_train = pd.DataFrame
    confmat_test = pd.DataFrame
    confmat_train_error = pd.DataFrame
    confmat_train_confint = pd.DataFrame
    confmat_test_error = pd.DataFrame
    confmat_test_confint = pd.DataFrame
```

```
df_tune_data = pd.DataFrame


def __init__(self,train_data,test_data):
    #initializing train data and train labels
    self.train_data = train_data.drop(columns = ['class'])
    self.train_labels = train_data['class']
    #self.train_Un = train_data['Un']

    #initializing test data and test labels
    self.test_data = test_data.drop(columns = ['class'])
    self.test_labels = test_data['class']
    #self.test_Un = test_data['Un']

    #initializing pred_train and pred_test
    self.pred_train = pd.DataFrame(0,index = ['total','cl1','cl_1'],
                             columns = ['pred','error','conf_int'])
    self.pred_test = pd.DataFrame(0,index =['total','cl1','cl_1'],
                             columns = ['pred','error','conf_int'])
    self.confmat_train = pd.DataFrame(0,index = ['true_CL1','true_CL_1'],
                               columns = ['pred_CL1','pred_CL_1'])
    self.confmat_train_confint = pd.DataFrame(0,index = ['true_CL1','true_CL_1'],
                               columns = ['pred_CL1','pred_CL_1'])

    self.confmat_train_error =  pd.DataFrame(0,index = ['true_CL1','true_CL_1'],
                               columns = ['pred_CL1','pred_CL_1'])
    self.confmat_train_confint =  pd.DataFrame(0,index = ['true_CL1','true_CL_1'],
                               columns = ['pred_CL1','pred_CL_1'])

    self.confmat_test_error = pd.DataFrame(0,index = ['true_CL1','true_CL_1'],
                               columns = ['pred_CL1','pred_CL_1'])
    self.confmat_test_confint = pd.DataFrame(0,index = ['true_CL1','true_CL_1'],
                               columns = ['pred_CL1','pred_CL_1'])


def svm_call(self,kernel_type,parameters,is_test):

    #choosing kernel based on type
    if kernel_type == 'linear':

        print('kernel_type: ',kernel_type)
        print('parameters: ',parameters)
        self.svm_master = SVC(kernel=kernel_type, C=parameters['C'])

    elif kernel_type == 'rbf':
        print('kernel_type: ',kernel_type)
        self.svm_master = SVC(kernel=kernel_type, gamma=parameters['gamma'], C=parameters['C'])

    elif kernel_type == 'poly':
        print('kernel_type: ',kernel_type)
        self.svm_master = SVC(kernel=kernel_type,
                    degree = 4,coef0 = parameters['coef0'],
                    gamma = parameters['gamma'],C = parameters['C'])

    #######Train set fitted SVM######################
    #fitting train data into svm model
    print('------Fitting train data into svm model-----')
    self.svm_master.fit(self.train_data,self.train_labels)
```

```
    print('Score for above parameters in SVC model : ',
          self.svm_master.score(self.train_data,self.train_labels))
    size_data = self.train_data.shape[0]

    #ROC plotting
    train_score = self.svm_master.fit(self.train_data, self.train_labels).decision_function(self.train_data)
    self.plot_roc(train_score,self.train_labels)

#     #train data poly(x) vs svm(x)
#     x_values = self.train_Un
#     y_values = pd.Series(self.svm_master.decision_function(self.train_data))

#     #plot poly(x)  and svm(x)
#     self.plot_poly_svm(x_values,y_values)

    #sv count and ratio
    number_sv = sum(self.svm_master.n_support_)
    print('Number of Support vectors,S : ',number_sv)
    ratio_sv = round(number_sv/size_data,2)
    print('Ratio of Support vectors,s : ',ratio_sv)

    ########SVM applied on test set############

    print('------SVM applied on test set-----')
    print('Score for above parameters in SVC model for testset : ',
         self.svm_master.score(self.test_data,self.test_labels))
    #ROC plotting
    test_score = self.svm_master.fit(self.train_data, self.train_labels).decision_function(self.test_data)
    self.plot_roc(test_score,self.test_labels)

#     #test data poly(x) vs svm(x)
#     x_values = self.test_Un
#     y_values = pd.Series(self.svm_master.decision_function(self.test_data))

#     #plot poly(x)  and svm(x)
#     self.plot_poly_svm(x_values,y_values)


def plot_roc(self,y_score,y_true):

    # Compute ROC curve and ROC area for each class
    fpr = dict()
    tpr = dict()
    roc_auc = dict()

    # # Compute micro-average ROC curve and ROC area
    fpr["micro"], tpr["micro"], _ = roc_curve(y_true.ravel(), y_score.ravel())
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

    #For CL1
    fpr[0], tpr[0], _ = roc_curve(y_true[y_score>0], y_score[y_score>0])
    roc_auc[0] = auc(fpr[0], tpr[0])
    #Plot of a ROC curve for a CL1
    plt.figure()
    lw = 2
    plt.plot(fpr[0], tpr[0], color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[0])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
```

```
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic CL1')
    plt.legend(loc="lower right")
    plt.show()

    #For CL_1
    fpr[1], tpr[1], _ = roc_curve(y_true[y_score<0], y_score[y_score<0])
    roc_auc[1] = auc(fpr[1], tpr[1])
    #Plot of a ROC curve for a CL1
    plt.figure()
    lw = 2
    plt.plot(fpr[1], tpr[1], color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[1])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic CL -1')
    plt.legend(loc="lower right")
    plt.show()

  def plot_poly_svm(self,x_values,y_values):

    col = np.where(np.sign(x_values) == np.sign(y_values),'g','r')

    fig = plt.figure()
    ax = fig.add_subplot(1,1, 1)
    fig.set_size_inches(8,6)
    ax.scatter(x_values,y_values,c=col, s=5, linewidth=0)

    ax.spines['left'].set_position('zero')
    ax.spines['right'].set_color('none')
    ax.spines['bottom'].set_position('zero')
    ax.spines['top'].set_color('none')
    ax.spines['left'].set_smart_bounds(True)
    ax.spines['bottom'].set_smart_bounds(True)
    ax.xaxis.set_ticks_position('bottom')
    ax.yaxis.set_ticks_position('left')
    ax.set_xlabel('poly(x)')
    ax.xaxis.set_label_coords(1,0)
    ax.set_ylabel('svm(x)')
    ax.yaxis.set_label_coords(-0.1,1)
    ax.set_title("poly(x) Vs svm(x)")

  def tune(self,tuned_parameters,nfolds,is_test):

    kernel_type = tuned_parameters['kernel'][0]
    print('kernel_type : ',kernel_type)

    grid_search = GridSearchCV(SVC(), tuned_parameters, cv=nfolds)
    print('------Fitting train data into svm model-----')
    grid_search.fit(self.train_data,self.train_labels)
    size_data = self.train_data.shape[0]
    #train data scores and plots
```

```
        self.best_params  = grid_search.best_params_
        #display scores and parameters
        print("# Tuning hyper-parameters\n")
        print("\nBest parameters set found on development set:\n")
        print(grid_search.best_params_)
        print("\nGrid scores on development set:\n")
        means = grid_search.cv_results_['mean_test_score']
        stds = grid_search.cv_results_['std_test_score']
        for mean, std, params in zip(means, stds, grid_search.cv_results_['params']):
            print("%0.3f (+/-%0.03f) for %r"
                  % (mean, std * 2, params))

        #number of support vectors for train data model
        self.tune_sv_score(tuned_parameters,False)

#        #fitting test data into svm model
#        print('------calculating scores for test data for various svm models-----')
#        size_data = self.test_data.shape[0]
#        self.tune_sv_score(tuned_parameters,True)

        if kernel_type in ['rbf','poly']:
            print('in heatmap:',kernel_type)
            self.tune_heatmap(tuned_parameters,grid_search)

    def tune_sv_score(self,tuned_parameters,is_test):
        # tuned_parameters = [{'kernel': ['linear'], 'C': [0.001, 0.01, 0.1, 1, 10,100]}]
        #C = tuned_parameters[0]['C']
        # tuned_parameters[0]['kernel'][0]
        # tuned_parameters = {'kernel': ['rbf'], 'C': C_range, 'gamma': gamma_range}
        # C = tuned_parameters['C']
        kernel_type = tuned_parameters['kernel'][0]
        tune_data_col = {'cost':[],'gamma':[],'coef0':[],'number_sv':[],
                     'ratio_sv':[],'score_tr':[],'score_tt':[],'score_diff':[]}
        self.df_tune_data = pd.DataFrame(tune_data_col)


        if is_test:
          #fitting test data into svm model
          #print('------Fitting test data into svm model-----')
            select_data =  self.test_data
            select_data_labels = self.test_labels
          #size_data = self.test_data.shape[0]

        else:
          #fitting train data into svm model
            print('------Fitting train data into svm model-----')
            select_data =  self.train_data
            select_data_labels = self.train_labels
        size_data = self.train_data.shape[0]


        #choosing kernel based on type
        if kernel_type == 'linear':
            C_range = tuned_parameters['C']
            for cost in C_range:
                svm_sample = SVC(kernel=kernel_type, C=cost)
                svm_sample.fit(self.train_data,self.train_labels)
```

```
            #getting scores and SVs for each cost
            number_sv,svm_sam_tr_score,svm_sam_tt_score = self.calc_sv_score(
              svm_sample,select_data,select_data_labels)

            ratio_sv = np.around(number_sv/size_data,2)
            append_data = {'cost':cost,'number_sv':number_sv,'ratio_sv':ratio_sv,
                       'score_tr':svm_sam_tr_score,'score_tt':svm_sam_tt_score,
                       'score_diff':svm_sam_tr_score-svm_sam_tt_score}
            self.df_tune_data = self.df_tune_data.append(append_data,ignore_index = True)

            #plot ROC for each cost/parameter
            test_score = svm_sample.fit(self.train_data, self.train_labels).decision_function(self.test_data)

#           self.plot_roc(test_score,self.test_labels)

            #print('C: {0} ; Number of Support vectors,S : {1};score : {2}'.format(cost,number_sv,svm_sample_score))

        self.plot_tune_data(kernel_type)

    elif kernel_type == 'rbf':

        C_range = tuned_parameters['C']
        gamma_range = tuned_parameters['gamma']
        for cost in C_range:
            for gamma in gamma_range:
                svm_sample = SVC(kernel=kernel_type, C=cost,gamma = gamma)
                svm_sample.fit(self.train_data,self.train_labels)
                #getting scores and SVs for each cost
                number_sv,svm_sam_tr_score,svm_sam_tt_score = self.calc_sv_score(
                  svm_sample,select_data,select_data_labels)
                ratio_sv = np.around(number_sv/size_data,2)
                append_data = {'cost':cost,'gamma':gamma,'number_sv':number_sv,
                          'ratio_sv':ratio_sv,'score_tr':svm_sam_tr_score,
                          'score_tt':svm_sam_tt_score,'score_diff':svm_sam_tr_score-svm_sam_tt_score}
                self.df_tune_data = self.df_tune_data.append(append_data,ignore_index = True)
                #print('C: {0} ;gamma:{3} Number of Support vectors,S : {1};score :
{2}'.format(cost,number_sv,svm_sample_score,gamma))
                #plot ROC for each cost/parameter
                test_score = svm_sample.fit(self.train_data, self.train_labels).decision_function(self.test_data)
                #self.plot_roc(test_score,self.test_labels)

        self.plot_tune_data(kernel_type)


    elif kernel_type == 'poly':
        print('kernel_type: ',kernel_type)
        C_range = tuned_parameters['C']
        coef0_range = tuned_parameters['coef0']
        gamma = tuned_parameters['gamma'][0]
        for cost in C_range:
            for coef0 in coef0_range:
                svm_sample = SVC(kernel=kernel_type,degree = 4,coef0 = coef0,
                           gamma = gamma,C = cost)
                svm_sample.fit(self.train_data,self.train_labels)
                #getting scores and SVs for each cost
                number_sv,svm_sam_tr_score,svm_sam_tt_score = self.calc_sv_score(
                  svm_sample,select_data,select_data_labels)
                ratio_sv = np.around(number_sv/size_data,2)
```

```
                append_data = {'cost':cost,'gamma':gamma,'number_sv':number_sv,
                         'coef0':coef0,'ratio_sv':ratio_sv,'score_tr':svm_sam_tr_score,
                         'score_tt':svm_sam_tt_score,'score_diff':svm_sam_tr_score-svm_sam_tt_score}
                self.df_tune_data = self.df_tune_data.append(append_data,ignore_index = True)
                #plot ROC for each cost/parameter
                test_score = svm_sample.fit(self.train_data, self.train_labels).decision_function(self.test_data)
               # self.plot_roc(test_score,self.test_labels)

        self.plot_tune_data(kernel_type)

            #print('C:    {0}    ;coef0:{3}    Number    of    Support    vectors,S    :    {1};score    :
{2}'.format(cost,number_sv,svm_sample_score,coef0))

    def plot_tune_data(self,kernel_type):
        print('df_tune_data:\n',self.df_tune_data)
        #{'cost':cost,'gamma':gamma,'number_sv':number_sv,
                # 'ratio_sv':ratio_sv,'score_tr':svm_sam_tr_score,
                # 'score_tt':svm_sam_tt_score}

        #cost vs ratio_sv
        ax = self.df_tune_data.plot('cost','ratio_sv',kind = 'scatter',title = 'cost vs sv ratio',style='b')
        self.df_tune_data.plot('cost','ratio_sv',kind = 'line',title = 'cost vs sv ratio',
                      ax=ax,style='r',logx=True,grid=True)
        #cost vs score_Train and score_test
        # ax = self.df_tune_data.plot('cost',y = 'score_tr',kind = 'scatter',
        #                title = 'cost vs Train score',style='b',logx=True,grid=True)
        # self.df_tune_data.plot('cost',y = 'score_tr',kind = 'line',title = 'cost vs Train score',
        #               ax=ax,style='r',logx=True,grid=True)

        # ax1 = self.df_tune_data.plot('cost',y = 'score_tt',kind = 'scatter',
        #                title = 'cost vs Test score',style='b',logx=True,grid=True)
        # self.df_tune_data.plot('cost',y = 'score_tt',kind = 'line',title = 'cost vs Test score',
        #               ax=ax1,style='r',logx=True,grid=True)

        ax = self.df_tune_data.plot(x = 'cost', y =['score_tt','score_tr'],  title = 'cost vs score',
                       logx=True,grid=True)
        # self.df_tune_data.plot(x = 'cost', y =['score_tt','score_tr'],  title = 'cost vs score',
        #               ax=ax,logx=True,grid=True)

        if kernel_type == 'linear':
            pass
        elif kernel_type == 'rbf':
            #gamma vs ratio_sv
            ax = self.df_tune_data.plot('gamma','ratio_sv',kind = 'scatter',title = 'gamma vs sv ratio',style='b')
            self.df_tune_data.plot('gamma','ratio_sv',kind = 'line',title = 'gamma vs sv ratio',
                      ax=ax,style='r',logx=True,grid=True)

            #gamma vs score Tr
            ax = self.df_tune_data.plot(x = 'gamma', y =['score_tt','score_tr'],  title = 'gamma vs score',
                       logx=True,grid=True)
            # ax = self.df_tune_data.plot('gamma',y = 'score_tr',kind = 'scatter',title = 'gamma vs score Train',style='b')
            # self.df_tune_data.plot('gamma',y = 'score_tr',kind = 'line',title = 'gamma vs score Train',
            #               ax=ax,style='r',logx=True,grid=True)

            # ax1 = self.df_tune_data.plot('gamma',y = 'score_tt',kind = 'scatter',
            #                title = 'cost vs Test score',style='b',logx=True,grid=True)
            # self.df_tune_data.plot('gamma',y = 'score_tt',kind = 'line',title = 'gamma vs Test score',
            #               ax=ax,style='r',logx=True,grid=True)
```

```
        elif kernel_type == 'poly':

            #coef0 vs ratio_sv
            ax = self.df_tune_data.plot('coef0','ratio_sv',kind = 'scatter',title = 'coef0 vs sv ratio',style='b')
            self.df_tune_data.plot('coef0','ratio_sv',kind = 'line',title = 'coef0 vs sv ratio',
                        ax=ax,style='r',logx=True,grid=True)

            #coef0 vs score Train
            ax = self.df_tune_data.plot(x = 'coef0', y =['score_tt','score_tr'],  title = 'coef0 vs score',
                        logx=True,grid=True)

            # ax = self.df_tune_data.plot('coef0',y = 'score_tr',kind = 'scatter',title = 'coef0 vs score Train',style='b')
            # self.df_tune_data.plot('coef0',y = 'score_tr',kind = 'line',title = 'coef0 vs score Train',
            #             ax=ax,style='r',logx=True,grid=True)

            # ax1 = self.df_tune_data.plot('coef0',y = 'score_tt',kind = 'scatter',
            #                 title = 'cost vs Test score',style='b',logx=True,grid=True)
            # self.df_tune_data.plot('coef0',y = 'score_tt',kind = 'line',title = 'coef0 vs Test score',
            #             ax=ax,style='r',logx=True,grid=True)

    def calc_sv_score(self,svm_sample,select_data,select_data_labels):
        #train score
        svm_sample_tr_score = svm_sample.score(self.train_data,self.train_labels)

        #test score
        svm_sample_tt_score = svm_sample.score(self.test_data,self.test_labels)

        #getting decision function and identifying support vectors for train data - same for test data
        decision_function = svm_sample.decision_function(self.train_data)
        sv_indices = np.where(self.train_labels * decision_function <= 1)[0]
        support_vectors = self.train_data.loc[sv_indices]
        number_sv = sum(svm_sample.n_support_)
        return number_sv,svm_sample_tr_score,svm_sample_tt_score

    def tune_heatmap(self,tuned_parameters,grid):

        kernel_type = tuned_parameters['kernel'][0]
        if kernel_type == 'rbf':
            C_range = tuned_parameters['C']
            parameter_range = tuned_parameters['gamma']
            parameter_name = 'gamma'
        elif kernel_type == 'poly':
            C_range = tuned_parameters['C']
            parameter_range = tuned_parameters['coef0']
            parameter_name = 'a'

        scores = grid.cv_results_['mean_test_score'].reshape(len(C_range),
                    len(parameter_range))



        # Draw heatmap of the validation accuracy as a function of gamma and C
        #
        # The score are encoded as colors with the hot colormap which varies from dark
        # red to bright yellow. As the most interesting scores are all located in the
        # 0.92 to 0.97 range we use a custom normalizer to set the mid-point to 0.92 so
```

```
        # as to make it easier to visualize the small variations of score values in the
        # interesting range while not brutally collapsing all the low score values to
        # the same color.

        plt.figure(figsize=(8, 6))
        plt.subplots_adjust(left=.2, right=0.95, bottom=0.15, top=0.95)
        plt.imshow(scores, interpolation='nearest', cmap=plt.cm.hot,
            norm=MidpointNormalize(vmin=0.2, midpoint=0.92))
        plt.xlabel(parameter_name)

        plt.ylabel('C')
        plt.colorbar()
        plt.xticks(np.arange(len(parameter_range)), parameter_range, rotation=45)
        plt.yticks(np.arange(len(C_range)), C_range)
        plt.title('Validation accuracy')
        plt.show()


    def calc_correct_pred(self):
        #calculating correct prediction for train data
        self.pred_train.loc['total','pred'] = self.svm_master.score(self.train_data,self.train_labels)*100

        #Prediction accuracy on train set for CL1 and CL_1
        self.pred_train.loc['cl1','pred'] = self.svm_master.score(
            self.train_data[self.train_labels > 0],self.train_labels[self.train_labels > 0])*100

        self.pred_train.loc['cl_1','pred'] = self.svm_master.score(
            self.train_data[self.train_labels < 0],self.train_labels[self.train_labels < 0])*100

        #calculating correct prediction for test data
        self.pred_test.loc['total','pred'] = self.svm_master.score(self.test_data,self.test_labels)*100

        #Prediction accuracy on test set for CL1 and CL_1
        self.pred_test.loc['cl1','pred'] = self.svm_master.score(
            self.test_data[self.test_labels > 0],self.test_labels[self.test_labels > 0])*100

        self.pred_test.loc['cl_1','pred'] = self.svm_master.score(
            self.test_data[self.test_labels < 0],self.test_labels[self.test_labels < 0])*100

    def calc_confusionmatrix(self):
        #reading predictions for cl1 and cl_1
        pred_train_cl1 = self.pred_train.loc['cl1','pred']
        pred_train_cl_1 = self.pred_train.loc['cl_1','pred']

        #creating confusion matrix for train data
        confmat_train_data = np.array([pred_train_cl1,100-pred_train_cl1,100-pred_train_cl_1,pred_train_cl_1]).reshape(2,2)
        self.confmat_train = pd.DataFrame(np.around(confmat_train_data,2),index = ['true_CL1','true_CL_1'],columns =
['pred_CL1','pred_CL_1'])

        #reading predictions for cl1 and cl_1
        pred_test_cl1 = self.pred_test.loc['cl1','pred']
        pred_test_cl_1 = self.pred_test.loc['cl_1','pred']

        #creating confusion matrix for test data
        confmat_test_data = np.array([pred_test_cl1,100-pred_test_cl1,100-pred_test_cl_1,pred_test_cl_1]).reshape(2,2)
        self.confmat_test = pd.DataFrame(np.around(confmat_test_data,2),index = ['true_CL1','true_CL_1'],columns =
['pred_CL1','pred_CL_1'])
```

```
    def error_estimate(self):
        #error estimation for train data
        size_train_cl1 = sum(self.train_labels > 0)
        size_train_cl_1 = sum(self.train_labels < 0)
        #sigma estimation
        self.pred_train.loc['total','error'] = self.err_est_element(self.pred_train.loc['total','pred'],
                                        self.train_data.shape[0],True)
        self.pred_train.loc['cl1','error'] = self.err_est_element(self.pred_train.loc['cl1','pred'],
                                        size_train_cl1,True)
        self.pred_train.loc['cl_1','error'] = self.err_est_element(self.pred_train.loc['cl_1','pred'],
                                        size_train_cl_1,True)
        #confidence interval calculation
        self.pred_train.loc['total','conf_int'] = self.err_est_element(self.pred_train.loc['total','pred'],
                                        self.train_data.shape[0],False)
        self.pred_train.loc['cl1','conf_int'] = self.err_est_element(self.pred_train.loc['cl1','pred'],
                                        size_train_cl1,False)
        self.pred_train.loc['cl_1','conf_int'] = self.err_est_element(self.pred_train.loc['cl_1','pred'],
                                        size_train_cl_1,False)
        #sigma and confid interval estimation for confmat_train
        #calculations for first row
        #size = self.confmat_train.loc['true_CL1','pred_CL1']*size_train_cl1/100
        size = size_train_cl1
        self.confmat_train_error.loc['true_CL1','pred_CL1']                                    =
self.err_est_element(self.confmat_train.loc['true_CL1','pred_CL1'],size,True)
        self.confmat_train_confint.loc['true_CL1','pred_CL1']                                  =
self.err_est_element(self.confmat_train.loc['true_CL1','pred_CL1'],size,False)
        #size = self.confmat_train.loc['true_CL1','pred_CL_1']*size_train_cl1/100
        self.confmat_train_error.loc['true_CL1','pred_CL_1']                                   =
self.err_est_element(self.confmat_train.loc['true_CL1','pred_CL_1'],size,True)
        self.confmat_train_confint.loc['true_CL1','pred_CL_1']                                 =
self.err_est_element(self.confmat_train.loc['true_CL1','pred_CL_1'],size,False)

        #calculations for second row
        #size = self.confmat_train.loc['true_CL_1','pred_CL1']*size_train_cl_1/100
        size = size_train_cl_1
        self.confmat_train_error.loc['true_CL_1','pred_CL1']                                   =
self.err_est_element(self.confmat_train.loc['true_CL_1','pred_CL1'],size,True)
        self.confmat_train_confint.loc['true_CL_1','pred_CL1']                                 =
self.err_est_element(self.confmat_train.loc['true_CL_1','pred_CL1'],size,False)
        #size = self.confmat_train.loc['true_CL_1','pred_CL_1']*size_train_cl_1/100
        self.confmat_train_error.loc['true_CL_1','pred_CL_1']                                  =
self.err_est_element(self.confmat_train.loc['true_CL_1','pred_CL_1'],size,True)
        self.confmat_train_confint.loc['true_CL_1','pred_CL_1']                                =
self.err_est_element(self.confmat_train.loc['true_CL_1','pred_CL_1'],size,False)

        #error estimation for test data
        size_test_cl1 = sum(self.test_labels > 0)
        size_test_cl_1 = sum(self.test_labels < 0)
        #sigma estimation
        self.pred_test.loc['total','error'] = self.err_est_element(self.pred_test.loc['total','pred'],
                                        self.test_data.shape[0],True)
        self.pred_test.loc['cl1','error'] = self.err_est_element(self.pred_test.loc['cl1','pred'],
                                        size_test_cl1,True)
        self.pred_test.loc['cl_1','error'] = self.err_est_element(self.pred_test.loc['cl_1','pred'],
                                        size_test_cl_1,True)
        #confidence interval calculation
        self.pred_test.loc['total','conf_int'] = self.err_est_element(self.pred_test.loc['total','pred'],
```

```
                                        self.test_data.shape[0],False)
        self.pred_test.loc['cl1','conf_int'] = self.err_est_element(self.pred_test.loc['cl1','pred'],
                                        size_test_cl1,False)
        self.pred_test.loc['cl_1','conf_int'] = self.err_est_element(self.pred_test.loc['cl_1','pred'],
                                        size_test_cl_1,False)

        #sigma and confid interval estimation for confmat_test
        #calculations for first row for true_cl1
        #size = self.confmat_test.loc['true_CL1','pred_CL1']*size_test_cl1/100
        size = size_test_cl1
        self.confmat_test_error.loc['true_CL1','pred_CL1'] =
self.err_est_element(self.confmat_test.loc['true_CL1','pred_CL1'],size,True)
        self.confmat_test_confint.loc['true_CL1','pred_CL1'] =
self.err_est_element(self.confmat_test.loc['true_CL1','pred_CL1'],size,False)
        #size = self.confmat_test.loc['true_CL1','pred_CL_1']*size_test_cl1/100
        self.confmat_test_error.loc['true_CL1','pred_CL_1'] =
self.err_est_element(self.confmat_test.loc['true_CL1','pred_CL_1'],size,True)
        self.confmat_test_confint.loc['true_CL1','pred_CL_1'] =
self.err_est_element(self.confmat_test.loc['true_CL1','pred_CL_1'],size,False)

        #calculations for second row for true_cl_1
        #size = self.confmat_test.loc['true_CL_1','pred_CL1']*size_test_cl_1/100
        size = size_test_cl_1
        self.confmat_test_error.loc['true_CL_1','pred_CL1'] =
self.err_est_element(self.confmat_test.loc['true_CL_1','pred_CL1'],size,True)
        self.confmat_test_confint.loc['true_CL_1','pred_CL1'] =
self.err_est_element(self.confmat_test.loc['true_CL_1','pred_CL1'],size,False)
        #size = self.confmat_test.loc['true_CL_1','pred_CL_1']*size_test_cl_1/100
        self.confmat_test_error.loc['true_CL_1','pred_CL_1'] =
self.err_est_element(self.confmat_test.loc['true_CL_1','pred_CL_1'],size,True)
        self.confmat_test_confint.loc['true_CL_1','pred_CL_1'] =
self.err_est_element(self.confmat_test.loc['true_CL_1','pred_CL_1'],size,False)

        # print('For 95% confidence level, Errors of estimation on {0} = {1} is {2} and confidence interval is {3}'.
        #     format(term_name,term,sigma,conf_int))

    def err_est_element(self,term,size,return_sigma):
        sigma = np.around(math.sqrt(term*(100-term)/size),2)

        #for 95% confidence level
        Z_VAL = 1.96
        limit_lower = np.around((term - Z_VAL*sigma),2)
        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),2)
        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)


# # Run the svm() function on the set TRAIN
# Fix arbitrarily the "cost" parameter in the svm() function, for instance cost = 5; Select the kernel parameter kernel = "linear "
#
#

# In[144]:


#create linear SVM with parameter C
svm_linear = SvmClassification(df_train_std.copy(),df_test_std.copy())


# In[145]:


#compute the number S of support vectors and the ratio s = S/4000
#creating dicitionary for parameters
parameters = {'C':5}
kernel_type = 'linear'
svm_linear.svm_call(kernel_type,parameters,False)


# In[146]:


#compute the percentages of correct prediction PredTrain and PredTest on the sets TRAIN and TEST
svm_linear.calc_correct_pred()
print(svm_linear.pred_train['pred'])
print(svm_linear.pred_test['pred'])

# compute two confusion matrices (one for the set TRAIN and one for the test set
# confusion matrices must be converted in terms of frequencies of correct predictions within each class
svm_linear.calc_confusionmatrix()
print(svm_linear.confmat_train)
print(svm_linear.confmat_test)

# compute the errors of estimation on PredTRAIN, PredTEST, and on the terms of the confusion matrices
svm_linear.error_estimate()
print(svm_linear.pred_train)
print(svm_linear.pred_test)

#print('confmat_train_error:\n',svm_linear.confmat_train_error)
print('confmat_train_confint:\n',svm_linear.confmat_train_confint)
#print('confmat_test_error:\n',svm_linear.confmat_test_error)
print('confmat_test_confint:\n',svm_linear.confmat_test_confint)

# interpret your results


# In[147]:


# **Question 3 : optimize the parameter "cost"**

# Select a list of 6 values for the "cost " parameter
```

```
# Run the tuning function tune() for the linear svm() to identify the best value of "cost"


# Set the parameters by cross-validation
tuned_parameters = {'kernel': ['linear'], 'C': [0.001, 0.01, 0.1, 1, 10,100]}
nfolds = 10
#building SVMs with train dataset and calculating SV ratio and scores
svm_linear.tune(tuned_parameters,nfolds,False)

# Evaluate the performance characteristics of the "best" linear svm as in question 2

svm_linear.best_params


# In[148]:


# Set the parameters by cross-validation for finetuning 0.1 to 20
tuned_parameters = {'kernel': ['linear'], 'C': [0.05,0.1, 1, 5,10]}
nfolds = 10
#building SVMs with train dataset and calculating SV ratio and scores
svm_linear.tune(tuned_parameters,nfolds,False)


# In[149]:


#creating svm_master for best
kernel_type = svm_linear.best_params['kernel']
parameters = svm_linear.best_params
svm_linear.svm_call(kernel_type,parameters,False)

#calculating PredTrain and PredTest for best SVM
svm_linear.calc_correct_pred()
print(svm_linear.pred_train['pred'])
print(svm_linear.pred_test['pred'])

#calulating confusion matrix for best params
svm_linear.calc_confusionmatrix()
print(svm_linear.confmat_train)
print(svm_linear.confmat_test)

#error estimation for best params
svm_linear.error_estimate()
print(svm_linear.pred_train)
print(svm_linear.pred_test)
#print('confmat_train_error:\n',svm_linear.confmat_train_error)
print('confmat_train_confint:\n',svm_linear.confmat_train_confint)
#print('confmat_test_error:\n',svm_linear.confmat_test_error)
print('confmat_test_confint:\n',svm_linear.confmat_test_confint)


# In[163]:


# Question 4: SVM classification by radial kernel
# Fix the "cost" parameter in the svm() function to the best cost value identified in question 3
# Select the kernel parameter kernel = "radial " which means that the kernel ks given by the formula
```

```
# K(x,y) = exp(-gamma || x- y ||2)
# Select arbitrarily the gamma parameter "gamma" = 1
# Run the svm() function on the set TRAIN

svm_radial = SvmClassification(df_train_std.copy(),df_test_std.copy())


# In[164]:


#as in question 2 compute the number S and the ratio s = S/4000
#Build Radial kernel SVM with cost as linear's best cost and gamma = 1
parameters = {'kernel': ['rbf'], 'C': svm_linear.best_params['C'],'gamma' : 1}

#build SVM with Training set
svm_radial.svm_call("rbf",parameters,False)


# In[165]:


#the percentages of correct predictions PredTrain and PredTest and the two confusion matrices
svm_radial.calc_correct_pred()
print('Predictions Train:\n',svm_radial.pred_train)
print('Predictions Test:\n',svm_radial.pred_test)

svm_radial.calc_confusionmatrix()
print('Confusion matrix for Train:\n',svm_radial.confmat_train)
print('Confusion matrix for Test:\n',svm_radial.confmat_test)

svm_radial.error_estimate()
print('prediction errors train:\n',svm_radial.pred_train)
print('prediction errors test:\n',svm_radial.pred_test)

#print('confmat_train_error:\n',svm_radial.confmat_train_error)
print('confmat_train_confint:\n',svm_radial.confmat_train_confint)
#print('confmat_test_error:\n',svm_radial.confmat_test_error)
print('confmat_test_confint:\n',svm_radial.confmat_test_confint)

#interpret your results


# In[166]:


# Question 5 : optimize the parameter "cost"and "gamma"
# Select a list of 5 values for the "cost " parameter and a list of 5 values for the parameter "gamma"
# On the TRAIN set , run the tuning function tune() for the radial svm() to identify the best value of the
# pair ("cost", "gamma") among the 25 values you have listed


# Set the parameters by cross-validation with C= best_param of linear and gamma as below

C_range = [svm_linear.best_params['C']]
gamma_range = np.logspace(-2, 2, 5)

tuned_parameters = {'kernel': ['rbf'], 'C': C_range, 'gamma': gamma_range}
nfolds = StratifiedShuffleSplit(n_splits=10, test_size=0.2, random_state=42)
```

```
svm_radial.tune(tuned_parameters,nfolds,False)


# In[172]:



#finetuning for gamma between 0.1 to 10
C_range = [svm_linear.best_params['C']]
#gamma_range = [0.05,0.1,0.5,1,5,8]
gamma_range = [0.05,0.03,0.1,0.2,0.3]

tuned_parameters = {'kernel': ['rbf'], 'C': C_range, 'gamma': gamma_range}
nfolds = StratifiedShuffleSplit(n_splits=10, test_size=0.2, random_state=42)
svm_radial.tune(tuned_parameters,nfolds,False)


# In[174]:


# gamma fixed to 0.1
#broad tuning  for cost
C_range = np.logspace(0, 4, 5)
gamma_range = [0.1]
tuned_parameters = {'kernel': ['rbf'], 'C': C_range, 'gamma': gamma_range}
nfolds = StratifiedShuffleSplit(n_splits=10, test_size=0.2, random_state=42)
svm_radial.tune(tuned_parameters,nfolds,False)


# In[175]:


# gamma fixed to 0.1

#fine tuning  for cost in range
C_range = [20,50,80,110,140]
gamma_range = [0.1]
#gamma_range = [0.1,0.05,1,5,8]
tuned_parameters = {'kernel': ['rbf'], 'C': C_range, 'gamma': gamma_range}
nfolds = StratifiedShuffleSplit(n_splits=10, test_size=0.2, random_state=42)
svm_radial.tune(tuned_parameters,nfolds,False)


# In[176]:


# Evaluate the performance characteristics of the "best" radial svm as in question 2

svm_radial.best_params = {'C': 20, 'gamma': 0.1, 'kernel': 'rbf'}
parameters = svm_radial.best_params
print(parameters)
# svm_radial.best_params_ = grid.best_params_
svm_radial.svm_call("rbf",parameters,False)
svm_radial.calc_correct_pred()
print('Predictions Train:\n',svm_radial.pred_train)
print('Predictions Test:\n',svm_radial.pred_test)
```

```
svm_radial.calc_confusionmatrix()
print('Confusion matrix for Train:\n',svm_radial.confmat_train)
print('Confusion matrix for Test:\n',svm_radial.confmat_test)

svm_radial.error_estimate()
print('prediction errors train:\n',svm_radial.pred_train)
print('prediction errors test:\n',svm_radial.pred_test)

# print('confmat_train_error:\n',svm_radial.confmat_train_error)
print('confmat_train_confint:\n',svm_radial.confmat_train_confint)
# print('confmat_test_error:\n',svm_radial.confmat_test_error)
print('confmat_test_confint:\n',svm_radial.confmat_test_confint)

#Interpret your results


# In[178]:


# # Question 6 : SVM classification using a polynomial kernel
# Implement the steps of question 4 and
# K(x,y) = (a + <x,y>)^4
# You will have to optimize the choice of the two parameters "a" >0 and ''cost"

svm_poly = SvmClassification(df_train_std.copy(),df_test_std.copy())

svm_radial.best_params


# In[179]:


parameters = {'kernel': ['poly'], 'C': svm_radial.best_params['C'],
        'degree' : 4,'coef0' : 1,'gamma':1}
svm_poly.svm_call('poly',parameters,False)



# In[180]:


svm_poly.calc_correct_pred()
print('Predictions Train:\n',svm_poly.pred_train)
print('Predictions Test:\n',svm_poly.pred_test)

svm_poly.calc_confusionmatrix()
print('Confusion matrix for Train:\n',svm_poly.confmat_train)
print('Confusion matrix for Test:\n',svm_poly.confmat_test)


svm_poly.error_estimate()
print('prediction errors train:\n',svm_poly.pred_train)
print('prediction errors test:\n',svm_poly.pred_test)

# print('confmat_train_error:\n',svm_poly.confmat_train_error)
print('confmat_train_confint:\n',svm_poly.confmat_train_confint)
# print('confmat_test_error:\n',svm_poly.confmat_test_error)
print('confmat_test_confint:\n',svm_poly.confmat_test_confint)
```

```
# In[187]:


# Q6a: optimize the parameters 'cost' and 'a' based on the polynomial kernel

#broad tuning for coef0(a) between 0.01 to 100
C_range = [svm_radial.best_params['C']]
gamma_range = [1]
# coef0_range = np.logspace(-2, 2,5)
coef0_range = [15,20,25,30,50]
tuned_parameters = {'kernel': ['poly'], 'C': C_range, 'coef0' : coef0_range,'gamma': [1]}
nfolds = StratifiedShuffleSplit(n_splits=10, test_size=0.2, random_state=42)
#tuning for Train set
svm_poly.tune(tuned_parameters,nfolds,False)


# In[189]:


#broad tuning for C between 0.01 to 100
C_range = np.logspace(-2, 2,5)
gamma_range = [1]
#best value from broad tuning
coef0_range = [20]
tuned_parameters = {'kernel': ['poly'], 'C': C_range, 'coef0' : coef0_range,'gamma': gamma_range}
nfolds = StratifiedShuffleSplit(n_splits=10, test_size=0.2, random_state=42)
#tuning for Train set
svm_poly.tune(tuned_parameters,nfolds,False)


# In[193]:


#fine tuning for C and coef0
gamma_range = [1]
C_range = [0.01,1,10,30,50,70,100]
coef0_range = [20]

tuned_parameters = {'kernel': ['poly'], 'C': C_range, 'coef0' : coef0_range,'gamma': gamma_range}
nfolds = StratifiedShuffleSplit(n_splits=10, test_size=0.2, random_state=42)
#tuning for Train set
svm_poly.tune(tuned_parameters,nfolds,False)


# In[201]:


#fine tuning for C and coef0
gamma_range = [0.05]
C_range = [0.01]
#coef0_range = [10,30,70,90,100]
coef0_range = [20]
tuned_parameters = {'kernel': ['poly'], 'C': C_range, 'coef0' : coef0_range,'gamma': gamma_range}
nfolds = StratifiedShuffleSplit(n_splits=10, test_size=0.2, random_state=42)
#tuning for Train set
svm_poly.tune(tuned_parameters,nfolds,False)
```

```
# In[202]:


# svm_poly.best_params = {'C': 10, 'coef0': 30, 'gamma': 1, 'kernel': 'poly'}
svm_poly.best_params = {'C': 0.01, 'coef0': 20, 'gamma': 0.05, 'kernel': 'poly'}
parameters = svm_poly.best_params
print(parameters)
svm_poly.svm_call("poly",parameters,False)

svm_poly.calc_correct_pred()
print('Predictions Train:\n',svm_radial.pred_train)
print('Predictions Test:\n',svm_radial.pred_test)

svm_poly.calc_confusionmatrix()
print('Confusion matrix for Train:\n',svm_radial.confmat_train)
print('Confusion matrix for Test:\n',svm_radial.confmat_test)

svm_poly.error_estimate()
print('prediction errors train:\n',svm_radial.pred_train)
print('prediction errors test:\n',svm_radial.pred_test)

# print('confmat_train_error:\n',svm_radial.confmat_train_error)
print('confmat_train_confint:\n',svm_radial.confmat_train_confint)
# print('confmat_test_error:\n',svm_radial.confmat_test_error)
print('confmat_test_confint:\n',svm_radial.confmat_test_confint)


# # KRR regression

# In[243]:


df_temp = df_totaldata.copy()
df_temp.drop(columns = 'Date',inplace=True)

#create dataset with predictor variables as v(t) to v(t-9) with X(t+1) as response variable
list_columns = list(range(10*df_temp.shape[1]))
#list_columns = list(map(lambda x : 'X'+str(x + 1), list_numbers))
df_krrdata = pd.DataFrame(columns = list_columns)
LIMIT_LOWER = 9
limit_upper = df_temp.shape[0]-1

#create new dataset
for i in range(LIMIT_LOWER,limit_upper):
    row = df_temp.loc[i-9:i][::-1].T.melt()['value']
    row.index = range(0,len(list_columns))
    df_krrdata = df_krrdata.append(row,ignore_index = True)


list_columns = list(map(lambda x : 'X'+str(x + 1), list_columns))
df_krrdata.columns = list_columns
#adding class column with name X_tplus1
X_tplus1 = df_totaldata.loc[LIMIT_LOWER+1:limit_upper,'CVS']
X_tplus1.index = range(0,limit_upper-LIMIT_LOWER)

#adding Date of X_tplus1
```

```
Date_col = df_totaldata.loc[LIMIT_LOWER+1:limit_upper,'Date']
Date_col.index = range(0,limit_upper-LIMIT_LOWER)
df_krrdata.insert(0,'Date',Date_col)

df_krrdata['Y'] = X_tplus1


# In[244]:


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

with ExcelWriter(filepath) as writer:
    df_totaldata.to_excel(writer,sheet_name = 'total_data' )
    df_krrdata.to_excel(writer,sheet_name = 'df_krrdata' )
    writer.save()


# In[350]:


df_krrdata.head()


# In[367]:


class KernelRidgePrediction:
    #attributes


    #methods
    def __init__(self,df_totaldata):

        self.df_totaldata = df_totaldata.copy()
        self.df_totaldata_orig = df_totaldata.copy()
        #add Xp(j)=1 column
        self.df_totaldata.insert(101,'X101',np.repeat(1.,self.df_totaldata.shape[0]))
        self.f_size = (8,8)

    #missing value count
    def missingvalue_count(self):

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

        self.df_missingrec = pd.DataFrame()
        #iterating through columns to find null values count and percentage
        for column in self.df_totaldata:

            if self.df_totaldata[column].isnull().values.any():

                null_rows = self.df_totaldata[column].isnull()
                null_count = sum(null_rows)
                nullcount_percent = round(null_count/self.df_totaldata[column].size*100,2)
                self.df_missingrec = self.df_missingrec.append(self.df_totaldata[null_rows])
```

```
        else:

            null_count = 0
            nullcount_percent = 0
        #saving missing value counts in dataframe
        self.df_severity.loc[column,'Missing_Values'] = null_count
        self.df_severity.loc[column,'%_of_MV'] = nullcount_percent

    print('Data Severity\n ', self.df_severity)


def create_traintestdata(self,prop):

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

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


def write_traintest_files(self,filepath):
    with ExcelWriter(filepath) as writer:
        self.df_train.to_excel(writer,sheet_name = 'train_data' )
        self.df_test.to_excel(writer,sheet_name = 'test_data' )
        self.df_train.loc[:,'X1':'X101'].to_excel(writer,sheet_name = 'train_data_nolabel' )
        self.df_train['Y'].to_excel(writer,sheet_name = 'train_labels_all' )
        self.df_test.loc[:,'X1':'X101'].to_excel(writer,sheet_name = 'test_data_nolabel' )
        self.df_test['Y'].to_excel(writer,sheet_name = 'test_labels_all' )
        writer.save()


def calc_kl_radial(self,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(self,val_gamma,val_lambda,df_train):
    #gramian matrix
    mat_train = np.matrix(df_train.loc[:,'X1':'X101'])
    val_m =mat_train.shape[0]

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

        mat_vx[i,0] = self.calc_kl_radial(val_gamma,mat_input_X,mat_train_X[i])

    pred_x = linevector_A*mat_vx
    return np.around(pred_x[0,0],4)

def cal_rmse(self,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(self,val_gamma,linevector_A,df_test,df_train):

    #matrix formulation
    mat_train = np.matrix(df_train.loc[:,'X1':'X101'])
    val_m =mat_train.shape[0]
    mat_test = np.matrix(df_test.loc[:,'X1':'X101'])
    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] = self.cal_predx(val_gamma,linevector_A,mat_test[i],mat_train)

    #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 = self.cal_rmse(mat_y_true,mat_y_pred)

    #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)

    with ExcelWriter(filepath) as writer:
        self.df_train.to_excel(writer,sheet_name = 'train_data' )
        self.df_test.to_excel(writer,sheet_name = 'test_data' )
        self.df_train.loc[:,'X1':'X101'].to_excel(writer,sheet_name = 'train_data_nolabel' )
        self.df_train['Y'].to_excel(writer,sheet_name = 'train_labels_all' )
        self.df_test.loc[:,'X1':'X101'].to_excel(writer,sheet_name = 'test_data_nolabel' )
        self.df_test['Y'].to_excel(writer,sheet_name = 'test_labels_all' )
        writer.save()

    #rplotting true y vs yhat

    fig, ax = plt.subplots(figsize=self.f_size)
    scale = np.where(np.max(mat_y_true)>np.max(mat_y_pred),np.max(mat_y_true),np.max(mat_y_pred))
```

```
        #ax.plot( [0,np.max(mat_y_true)],[0,np.max(mat_y_true)] )
        ax.plot( [0,scale],[0,scale] )
        ax.scatter(np.asarray(mat_y_true),np.asarray(mat_y_pred),c = 'g')
        ax.set(title = 'y vs y hat ' , xlabel = 'y', ylabel = 'yhat')
        plt.show()

        #RMSE ratio and confidence intervals
        print('val_rmse: ',np.around(val_rmse_test,4))
        print('ratio rmse/avy : ',ratio_rmse_avy_test)
        confint_ratio = self.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))


        return val_rmse_test,ratio_rmse_avy_test,mat_y_pred

    ####---------------------------------

    def tune_krr(self,tuned_parameters):
        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 = self.cal_linevector_A(val_gamma,val_lambda,self.df_train)
                #train performance
                print('#train performance')
                val_rmse_train,ratio_rmse_avy_train,mat_train_pred                                    =
self.cal_performance_params(val_gamma,linevector_A,self.df_train,self.df_train)

                #test performance
                print('#test performance')
                val_rmse_test,ratio_rmse_avy_test,mat_test_pred                                       =
self.cal_performance_params(val_gamma,linevector_A,self.df_test,self.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(self,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:
            limit_lower = 0
```

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

    if limit_upper > 100:
        limit_upper = 100

    conf_int = [limit_lower,limit_upper]

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

def plot_results(self,data,title):

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

#calculate prediction coefficients list for input vector input_X
def cal_predx_list(self,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] = self.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(self,val_gamma,linevector_A,df_train,df_test):

    #matrix formulation
    mat_train = np.matrix(df_train.loc[:,'X1':'X101'])
    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':'X101'])
    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] = self.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
```

```
# In[368]:


# # 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 ;
krr_radial = KernelRidgePrediction(df_krrdata)

krr_radial.df_totaldata.describe()


# In[369]:


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

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

# 1.5. compute and display its  mean and standard deviation for Y
krr_radial.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
krr_radial.missingvalue_count()


# In[370]:


#create train and test datasets
PROP = 80
krr_radial.create_traintestdata(PROP)
#describe train and test data
print('############# Train data #######')

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

krr_radial.write_traintest_files(filepath)
krr_radial.df_train.describe().to_excel('df_train.describe.xlsx')
krr_radial.df_test.describe().to_excel('df_test.describe.xlsx')


# In[371]:


# 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(krr_radial.df_totaldata.corrwith(krr_radial.df_totaldata['Y'])))
sr_corr_XY.to_excel('sr_corr_XY.xlsx')


# In[372]:


# 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]


# In[373]:


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

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

krr_radial.df_totaldata.plot(x='X11',y='Y',kind = 'scatter',title = 'Scatter plot of Xv Vs Y')

krr_radial.df_totaldata.plot(x='X21',y='Y',kind = 'scatter',title = 'Scatter plot of Xw Vs Y')


# In[374]:


# 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
#saving train data into matrix form
mat_train = np.matrix(krr_radial.df_train.loc[:,'X1':'X101'])
val_m =mat_train.shape[0]
mat_gramian = np.zeros((val_m,val_m))
#sample gamma
VAL_SAMPLE_GAMMA = 0.01

#calculate gramian matrix for train data
for i in np.arange(val_m):
    for j in np.arange(val_m):
        mat_gramian[i,j] = krr_radial.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))


# In[375]:


#calculate and display eigen values statistics
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

pd.DataFrame(gramian_eig_val).describe()


# In[376]:


#sorting eigen values in descending order
gramian_eig_val[::-1].sort()
gramian_eig_val


# In[377]:
```

```
# 2.2.Plot Lj versus j

#plotting eigen values in descending order
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()


# In[378]:



# 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


# In[379]:


# 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


# In[380]:


#plotting increasing ratio of eigen values to identify 95% eigen value
fig, ax = plt.subplots(figsize=krr_radial.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()
```

```
# In[381]:


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

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)

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


# In[382]:


# 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=krr_radial.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()


# In[383]:


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


# In[384]:


# 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
linevector_A_sample = krr_radial.cal_linevector_A(val_gamma,val_lambda,krr_radial.df_train)
linevector_A_sample


# In[385]:
```

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

#train performance
print('##Train Performance for sample parameters###')
val_rmse_train,ratio_rmse_avy_train,mat_y_pred_train = krr_radial.cal_performance_params(
    val_gamma,linevector_A_sample,krr_radial.df_train,krr_radial.df_train)


# In[391]:


# 2.12.Compute the RMSEtest   of the prediction function pred(x) by running it on all x in TEST set
# 2.13 Compare these two RMSE values , and compute their ratios RMSE/ avy
# where avy = mean of the m absolute values |Y1|,  ... , |Ym|
#test performance for sample parameters
print('####test performance for sample parameters####')
val_rmse_test,ratio_rmse_avy_test,mat_y_pred_test = krr_radial.cal_performance_params(
    val_gamma,linevector_A_sample,krr_radial.df_test,krr_radial.df_train)


# In[387]:


# # 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('tuning parameters: gamma = {0},lambda ={1}'.format(val_gamma,val_lambda))

#adding train prediction column to train dataset copy
df_train_orig_lv = krr_radial.df_train.copy()
df_train_orig_lv['yhat'] = np.asarray(mat_y_pred_train.T)


krr_radial.plot_results(df_train_orig_lv,title= 'y vs y hat for train data with original parameters')


# In[392]:


mat_y_pred_test.shape


# In[ ]:
```

```
# In[393]:


#adding test prediction column to test dataset copy
df_test_orig_lv = krr_radial.df_test.copy()
df_test_orig_lv['yhat'] = np.asarray(mat_y_pred_test.T)

krr_radial.plot_results(df_test_orig_lv,title= 'y vs y hat for test data with chosen parameters')


# In[394]:


#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)
krr_radial.tune_krr(tuned_parameters)


# In[395]:


#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)
krr_radial.tune_krr(tuned_parameters)


# In[396]:


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}
#krr_radial.tune_krr(tuned_parameters,df_train,df_test)
krr_radial.tune_krr(tuned_parameters)


# In[397]:


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)
krr_radial.tune_krr(tuned_parameters)
```

```
# In[398]:


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)
krr_radial.tune_krr(tuned_parameters)


# In[399]:


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)
krr_radial.tune_krr(tuned_parameters)


# In[400]:


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)
krr_radial.tune_krr(tuned_parameters)


# In[401]:


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)
krr_radial.tune_krr(tuned_parameters)


# In[402]:


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)
```

```
krr_radial.tune_krr(tuned_parameters)


# In[403]:


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)
krr_radial.tune_krr(tuned_parameters)


# In[404]:


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)
krr_radial.tune_krr(tuned_parameters)


# In[405]:


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)
krr_radial.tune_krr(tuned_parameters)


# In[439]:


#best parameters
best_params = {'lambda': [round(lambda_5,4)], 'gamma': [round(gamma_5/2,4)]}
print('best_params: ',best_params)


# In[440]:




# In[408]:


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

```
#train performance for best parameters
print('####train performance for best parameters####')
#calc linevector_A for  best params
linevector_A_best = krr_radial.cal_linevector_A(best_params['gamma'][0],best_params['lambda'][0],krr_radial.df_train)

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

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

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

#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 ')


#test performance for best parameters
print('####test performance for best parameters####')

#calc y_predictions for test data for best params
val_rmse_test_best,ratio_rmse_avy_test_best,mat_y_pred_test_best = krr_radial.cal_performance_params(
    best_params['gamma'][0],linevector_A_best,krr_radial.df_test,krr_radial.df_train)

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

krr_radial.plot_results(df_test_best_lv,title= 'y vs y hat for test data with best parameters')

#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(krr_radial.df_test['Y'])
df_test_error = krr_radial.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.to_excel('df_test_error.xlsx')

df_test_error


# In[441]:


# 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 test data--------")
df_test_error_reduced = pca.fit_transform(df_test_error.loc[:,'X1':'X101'])
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()


# In[442]:


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


# In[ ]:




# In[443]:
```

```
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()


# In[444]:


#calculate centroid of 10 worst cases
centroid_worst = np.sum(df_test_error.loc[:9,'X1':'X101' ])/10
centroid_worst

#distance between centroid and each case
df_test_error['Dn_centr_w'] = np.linalg.norm(df_test_error.loc[:,'X1':'X101']- centroid_worst, axis=1)
df_test_error

#scatterplot of distances
fig, ax = plt.subplots(figsize=krr_radial.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()


# In[445]:


# 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))
#
#

# In[419]:


best_params

#worst 10 cases indices in pred(x)
mat_pred_worst10 = krr_radial.cal_indices_pred(best_params['gamma'][0],linevector_A_best,
                        krr_radial.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 = krr_radial.cal_indices_pred(best_params['gamma'][0],linevector_A_best,
                        krr_radial.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')
```

```
# In[420]:


# 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


# In[421]:


#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=krr_radial.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()


# In[422]:



# 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


# In[424]:
```

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

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


# In[425]:


# 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


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')


# In[426]:


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()
```

```
# In[427]:


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])


# In[428]:


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

print('best_params: ',best_params)

#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 = krr_radial.cal_performance_params(
   best_params['gamma'][0],linevector_A_red,krr_radial.df_train,krr_radial.df_train)

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

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

#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')


print('#####performance of reduced linevector on Test data#####')
val_rmse_test,ratio_rmse_avy_test,mat_y_pred_test_red = krr_radial.cal_performance_params(
   best_params['gamma'][0],linevector_A_red,krr_radial.df_test,krr_radial.df_train)

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

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

#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')


# In[ ]:


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


# In[ ]:
```

```
# In[435]:


#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

clf = KernelRidge(alpha=best_params['lambda'][0],kernel = 'rbf',gamma = best_params['gamma'][0])
clf.fit(krr_radial.df_train.loc[:,'X1':'X101'], krr_radial.df_train.loc[:,'Y'])


#pre-existing model prediction for train data
print('###pre-existing model prediction for train data###')
pred_y = clf.predict(krr_radial.df_train.loc[:,'X1':'X101'])
rmse_model_train = krr_radial.cal_rmse(krr_radial.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(krr_radial.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 = krr_radial.err_est_element(ratio_rmse_avy_test,krr_radial.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 = krr_radial.df_train.copy()
df_train_model['yhat'] = np.asarray(pred_y)

krr_radial.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=krr_radial.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
print('###pre-existing model prediction for test data###')
pred_y = clf.predict(krr_radial.df_test.loc[:,'X1':'X101'])
rmse_model_test = krr_radial.cal_rmse(krr_radial.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(krr_radial.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 = krr_radial.err_est_element(ratio_rmse_avy_test,krr_radial.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 = krr_radial.df_test.copy()
df_test_model['yhat'] = np.asarray(pred_y)

krr_radial.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=krr_radial.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')


# In[ ]:
```