STAT 542: Homework 3

Arka Mitra (netid : arkam2)

Spring 2021, by Ruoqing Zhu (rqzhu)

Date: 'Due: Tue, Feb 23, 11:59 PM CT'

About HW3

In the first question, we will use a simulation study to confirm the theoretical analysis we developed during the lecture. In the second question, we will practice several linear model selection techniques such as AIC, BIC, and best subset selection. However, some difficulties are at the data processing part, in which we use the Bitcoin data from Kaggle. This is essentially a time-series data, and we use the information in previous days to predict the price in a future day. Make sure that you process the data correctly to fit this task.

Question 1 [40 Points] A Simulation Study

Let's use a simulation study to confirm the bias-variance trade-off of linear regressions. Consider the following model.

$Y = \sum_j^p 0.9^j \times X_j + \epsilon$

All the covariates and the error term follow i.i.d. standard Gaussian distribution. The true model involves all the variables; however, variables with larger indexes do not contribute significantly to the variation. Hence, there could be a benefit using a smaller subset for prediction purposes. Let's confirm that with a simulation study. The study essentially repeats the following steps 200 times and obtain the averaged results:

- Generate 300 training data (both covariates and outcomes) with $p=100$, and generate another 300 outcomes as the testing data $Y$ using the same covariate value.
- Consider using only the first $j$ variables to fit the linear regression. Let $j$ ranges from 1 to 100. Calculate and record the corresponding prediction error.
- For each $j$ value, we also have the theoretical analysis of the testing error based on the lecture. In that analysis, we have the formula of both the Bias$^2$ and variance. Plug-in the simulated data to calculate the Bias$^2$ and use the theoretical value for the variance.

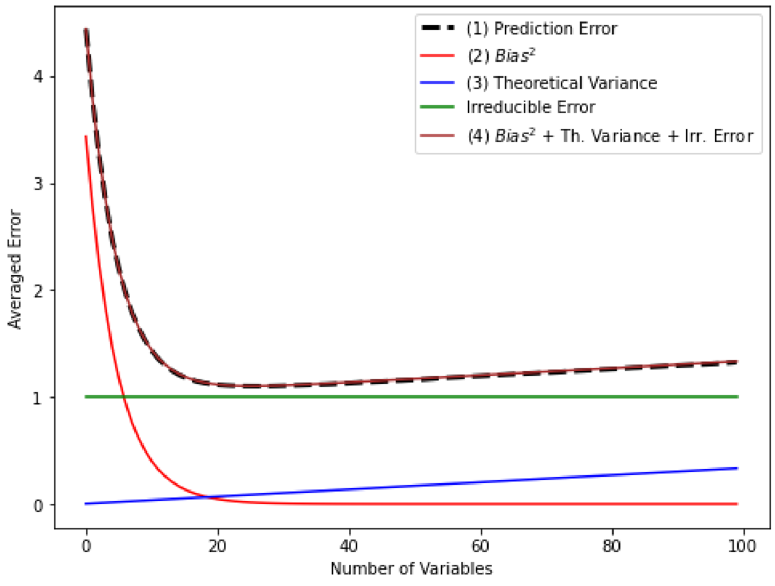
After finishing all simulation runs, plot your results using the `number of variables` as the x-axis, and the following 4 lines:

- The averaged prediction error based on your 200 simulation runs
- The averaged Bias$^2$ based on your 200 simulation runs
- The theoretical variance
- The sum of Bias$^2$ + variance + Irreducible Error

Does your simulation result match our theoretical analysis? Comment on your findings.

In [14]:

```
import scipy
import random
import numpy as np
from sklearn import metrics
from scipy.stats import norm
from scipy.stats import uniform
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import StandardScaler

avg_pr_err = np.zeros((200,100))
avg_bias = np.zeros((200,100))
variance = np.zeros((200,100))
irr_err = np.zeros((200,100))
beta = [0.9**j for j in range(1,101)]

#Loop over 200 trials
for trial in range(200):
    #Generate covariates and test data for each trial
    X = scipy.stats.norm.rvs(size = (300 , 100))
    mu = X.dot(beta)
    Y = mu + scipy.stats.norm.rvs(size = 300)
    Y_test  = mu + scipy.stats.norm.rvs(size = 300)

    mse = []; bias = []; var = []; irr = []

    #Loop with different number of predictive variables
    for p in range(1,101):

        #projection matrix and predicted values
        H = X[:,:p].dot(np.linalg.inv(X[:,:p].T.dot(X[:,:p])).dot(X[:,:p].T))
        y_pred = H.dot(Y)

        #Prediction Error = Mean Squared Error
        mse.append(metrics.mean_squared_error(Y_test,y_pred) )
        #Bias^2 = ||mu - H.mu||^2
        bias.append(np.linalg.norm(mu-H.dot(mu)) ** 2 )
        #Theoretical variance = p*sigma^2 = p
        var.append(p)
        #Irreducible error = n*sigma^2 = n
        irr.append(300)

    avg_pr_err[trial] = mse
    avg_bias[trial] = bias
    variance[trial] = var
    irr_err[trial] = irr

pred = np.mean(avg_pr_err, axis = 0)
bias = np.mean(avg_bias, axis = 0)
variance = np.mean(variance, axis = 0)
irred = np.mean(irr_err, axis = 0)


#Now we plot mean errors in each case
plt.figure(figsize = (8,6))
plt.plot(pred, label = '(1) Prediction Error', ls = '--', color = 'black', lw=3)
plt.plot(bias/300 , label = '(2) $Bias^2$', color='red')
plt.plot(variance/300, label = '(3) Theoretical Variance', color='blue')
plt.plot(irred/300, label = 'Irreducible Error', color='green')
plt.plot((variance + bias + irred)/300, label = '(4) $Bias^2$ + Th. Variance + Irr. Error', color=
'brown')
plt.xlabel('Number of Variables')
plt.ylabel('Averaged Error')
plt.legend()
plt.show()
```



Yes, the results agree with our theoretical findings.

(1) Bias decreases as variable count increases.

(2) The theoretical variance ($p\sigma^2$) increases monotonically with variable count.

(3) The mean squared prediction error overlaps with the sum of the $Bias^2$, Variance and the Irreducible Error. This verifies our theoretical expression:
$$ E[Test Error] = Bias^2 + n\sigma^2 + p\sigma^2$$

(4) From bias-variance tradeoff, best prediction error = ~ 22 covariates.

Question 2 [60 Points] Bitcoin price prediction

For this question, we will use the Bitcoin data provided on the course website. The data were posted originally on Kaggle (link). Make sure that you read relevant information from the Kaggle website. Our data is the `bitcoin_dataset.csv` file. You should use a training/testing split such that your training data is constructed using only information up to 12/31/2016, and your testing data is constructed using only information starting from 01/01/2017. The goal of our analysis is to predict the `btc_market_price`. Since this is longitudinal data, we will use the information from previous days to predict the market price at a future day. In particular, on each calendar day (say, day 1), we use the information from three days onward (days 1, 2, and 3) to predict the market price on the 7th day.

Hence you need to first reconstruct the data properly to fit this purpose. This is mainly to put the outcome (of day 7) and the covariates (of the previous days) into the same row. Note that for this question, you may face issues such as missing data, categorical predictors, outliers, scaling issue, computational issue, and maybe others. Use your best judgment to deal with them. There is no general ``best answer''. Hence the grading will be based on whether you provided reasoning for your decision and whether you carried out the analysis correctly.

[15 Points] Data Construction

Data pre-processing is usually the most time-consuming and difficult part of an analysis. We will use this example as a practice. Construct your data appropriately such that further analysis can be performed. Make sure that you consider the following:

- The data is appropriate for our analysis goal: each row contains the outcome on the seventh day and the covariates of the first three days
- Missing data is addressed (you can remove variable, remove observation, impute values or propose your own method)
- Process each single covariate/outcome by considering centering/scaling/transformation and/or removing outliers

For each of the above tasks, make sure that you **clearly document your choice**. In the end, provide a summary table/figure of your data. You can consider using boxplots, quantiles, histogram, or any method that is easy for readers to understand. You are required to pick one at least one method to present.

In [2]:

```
import seaborn as sns
import pandas as pd
import datetime as dt

#We use 'temp_df' dataframe to store bitcoin data, forward fill takes care of missing data
temp_df = pd.read_csv('bitcoin_dataset.csv', usecols = range(1,24)).fillna(method ='ffill', axis =
0)
#Interval and prediction day
n_days, pred_day = 3 , 7
#Let's check the quantiles of the original data
temp_df.quantile([0.25,0.5,0.75]).T.round(2)
```

Out[2]:

	0.25	0.50	0.75
btc_market_price	6.77	2.363100e+02	6.039100e+02
btc_total_bitcoins	8405100.00	1.242910e+07	1.523764e+07
btc_market_cap	55605880.64	3.364730e+09	8.210042e+09
btc_trade_volume	301851.43	1.024287e+07	2.913615e+07
btc_blocks_size	777.25	1.513200e+04	5.930175e+04
btc_avg_block_size	0.02	2.000000e-01	6.900000e-01
btc_n_orphaned_blocks	0.00	0.000000e+00	0.000000e+00
btc_n_transactions_per_block	55.00	3.785000e+02	1.238570e+03
btc_median_confirmation_time	6.13	7.930000e+00	1.024000e+01

	0.25	0.50	0.75
btc_median_confirmation_time	[illegible]	[illegible]	[illegible]
btc_hash_rate	11.70	2.640642e+04	1.097174e+06
btc_difficulty	1617683.85	3.129573e+09	1.584272e+11
btc_miners_revenue	47693.74	8.895575e+05	1.865030e+06
btc_transaction_fees	9.72	2.112000e+01	5.078000e+01
btc_cost_per_transaction_percent	1.18	2.460000e+00	5.840000e+00
btc_cost_per_transaction	4.22	7.870000e+00	1.529000e+01
btc_n_unique_addresses	17190.00	1.325365e+05	3.666370e+05
btc_n_transactions	8157.00	6.300600e+04	1.914652e+05
btc_n_transactions_total	2484692.75	3.340692e+07	1.116330e+08
btc_n_transactions_excluding_popular	6995.50	5.501550e+04	1.871828e+05
btc_n_transactions_excluding_chains_longer_than_100	6888.75	3.567000e+04	1.149200e+05
btc_output_volume	497532.28	1.118162e+06	2.028524e+06
btc_estimated_transaction_volume	96609.00	1.787410e+05	2.587235e+05
btc_estimated_transaction_volume_usd	988958.25	3.899174e+07	1.369944e+08

We normalize the data to be in $[-1,1]$ using the following method, $\hat{X} = \frac{X - \mu_X}{\sigma_X}$

In [5]:

```
ss = StandardScaler()
norm_df = pd.DataFrame(ss.fit_transform(temp_df.values) , columns = temp_df.columns)
#The quantiles after normalizing
scaled_df.quantile([0.25,0.5,0.75]).T.round(2)
```

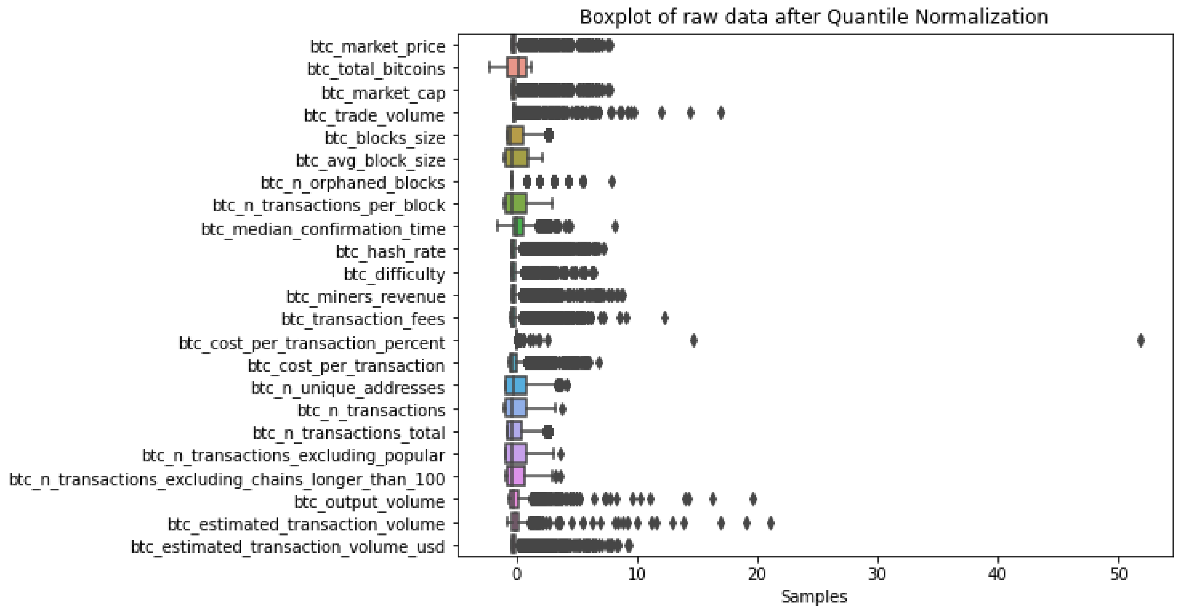
Out[5]:

	0.25	0.50	0.75
btc_market_price	-0.37	-0.28	-0.12
btc_total_bitcoins	-0.74	0.22	0.88
btc_market_cap	-0.36	-0.27	-0.15
btc_trade_volume	-0.26	-0.23	-0.17
btc_blocks_size	-0.79	-0.47	0.52
btc_avg_block_size	-0.93	-0.44	0.93
btc_n_orphaned_blocks	-0.43	-0.43	-0.43
btc_n_transactions_per_block	-0.90	-0.43	0.82
btc_median_confirmation_time	-0.29	0.08	0.54
btc_hash_rate	-0.41	-0.40	-0.09
btc_difficulty	-0.42	-0.41	-0.05
btc_miners_revenue	-0.39	-0.24	-0.07
btc_transaction_fees	-0.44	-0.34	-0.09
btc_cost_per_transaction_percent	-0.03	-0.03	-0.03
btc_cost_per_transaction	-0.51	-0.34	0.00
btc_n_unique_addresses	-0.85	-0.30	0.82
btc_n_transactions	-0.91	-0.38	0.85
btc_n_transactions_total	-0.80	-0.43	0.49
btc_n_transactions_excluding_popular	-0.85	-0.39	0.89
btc_n_transactions_excluding_chains_longer_than_100	-0.82	-0.40	0.73
btc_output_volume	-0.47	-0.20	0.20
btc_estimated_transaction_volume	-0.40	-0.09	0.21
btc_estimated_transaction_volume_usd	-0.36	-0.29	-0.13

For the most part, the data is quite well-behaved, but there are a few outliers from the following boxplot figure.

In [7]:

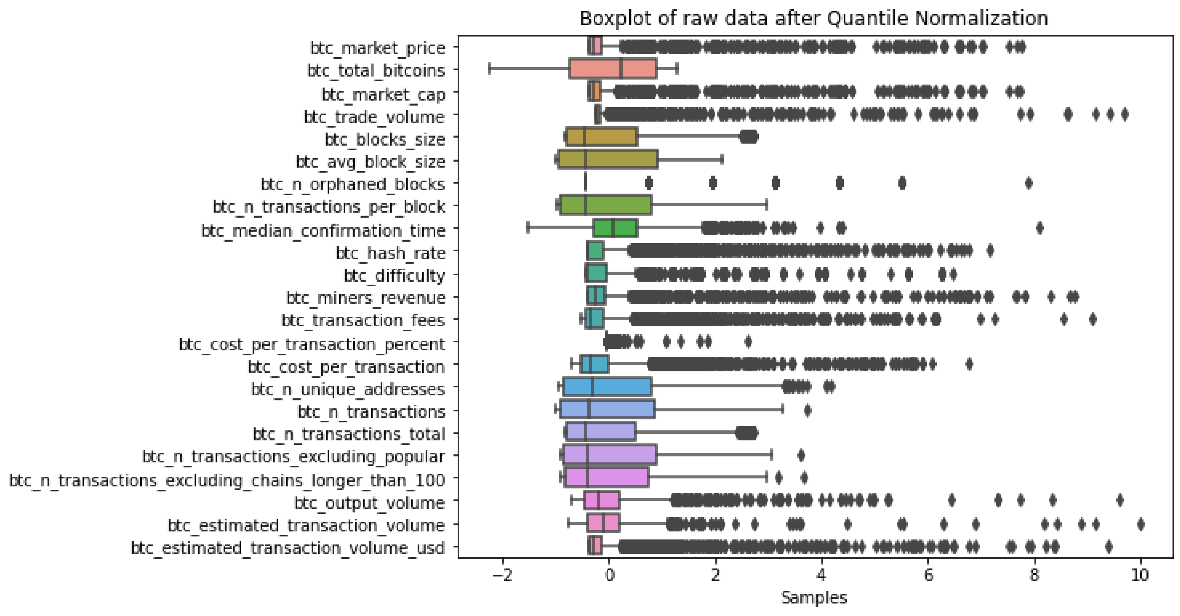
```
plt.figure(figsize=(8,6))
sns.boxplot(data=norm_df,orient='h')
# set x-axis label
plt.xlabel("Samples")
plt.title("Boxplot of raw data after Quantile Normalization")
plt.show()
```



We get rid of the outliers in our data by removing points which are $10 \sigma$ away from the corresponding mean, and remove these points from both datasets. The next figure shows the boxplot of the scaled dataset after outlier removal.

In [10]:

```
#Removing outliers
outlier_id = np.where(norm_df > 10)[0]
temp_df.drop(outlier_id , inplace = True)
norm_df.drop(outlier_id , inplace = True)
#Plotting the normalized data
plt.figure(figsize=(8,6))
sns.boxplot(data=norm_df,orient='h')
# set x-axis label
plt.xlabel("Samples")
plt.title("Boxplot of raw data after Quantile Normalization")
plt.show()
```



Now, we create the final dataframe for analysis, and as instructed, we put the outcome (of day 7) and the covariates (of the previous days) into the same row.

In [15]:

```
#Define a new dataframe df for the preprocessed data
df = pd.DataFrame()

#Rearrage the data from each column into 3 columns, each
#with data from a day later
for i in range(len(temp_df.columns)):
    for j in range(n_days):
        df[i*n_days + j] = np.array(norm_df.iloc[j:-(pred_day-1)+j , i])

#Rename the columns of the new dataframe as:
#btc_market_price --> btc_market_price1, btc_market_price2, btc_market_price3
#and so on
df.columns = [col+str(i) for col in temp_df.columns for i in range(1,n_days+1)]

#Insert the prediction price column
df.insert(0, 'output', np.array(norm_df['btc_market_price'][(pred_day - 1):]) )

dates = np.array(pd.to_datetime(pd.read_csv('bitcoin_dataset.csv')['Date'].drop(outlier_id))[(pred_
day - 1):])
df.insert(0, 'output date', dates[:len(df)])

# Defining the training and testing datasets and saving them in text files
train_df, test_df = df[df['output date'] < '2017-01-01'] , df[df['output date'] >= '2017-01-01']
train_df.to_csv('bit_train.csv')
test_df.to_csv('bit_test.csv')
test_df
```

Out[15]:

	output date	output	btc_market_price1	btc_market_price2	btc_market_price3	btc_total_bitcoins1	btc_total_bitcoins2	btc_total_bitco
2498	2017-01-01	-0.027911	0.003572	-0.046781	-0.039218	1.089005	1.089461	1.08
2499	2017-01-02	0.002650	-0.046781	-0.039218	-0.029666	1.089461	1.089973	1.09
2500	2017-01-03	-0.009372	-0.039218	-0.029666	-0.033159	1.089973	1.090419	1.09
2501	2017-01-04	-0.000703	-0.029666	-0.033159	-0.031369	1.090419	1.090931	1.09
2502	2017-01-05	-0.001610	-0.033159	-0.031369	-0.027911	1.090931	1.091428	1.09
...	...	...	...	...	...	...	...	
2891	2018-01-29	3.846101	3.092987	3.102812	3.297786	1.270597	1.271046	1.27
2892	2018-01-30	4.143978	3.102812	3.297786	3.208787	1.271046	1.271541	1.27
2893	2018-01-31	4.002841	3.297786	3.208787	3.515846	1.271541	1.272029	1.27
2894	2018-02-01	4.256062	3.208787	3.515846	3.783591	1.272029	1.272561	1.27
2895	2018-02-02	4.372502	3.515846	3.783591	3.846101	1.272561	1.272951	1.27

398 rows × 71 columns

[15 Points] Model Selection Criterion

Use AIC and BIC criteria to select the best model and report the result from each of them. Use the forward selection for AIC and backward selection for BIC. Report the following two error quantities from **both training and testing data**.

- The mean squared error: $n^{-1} \sum_{i}(Y_i - \widehat{Y}_i)^2$

- The proportion of explained variation ($R^2$): $1 - \frac{n^{-1} \sum_{i}(Y_i - \widehat{Y}_i)^2}{ n^{-1} \sum_{i}(Y_i - \overline{Y}_i)^2}$

Since these quantities can be affected by scaling and transformation, make sure that you **state any modifications applied to the outcome variable**. Compare the training data errors and testing data errors, which model works better? Provide a summary of your results.

Solution:

For the model selection, the 'bit_train.csv' and 'bit_test.csv' were taken to R, where the step() function was used to select the model through forward (AIC) and backward (BIC) selection.

Here is the R code snippet used for the forward model selection calculation (suitable changes were made for BIC):

```
{r}
rsq <- function (x, y) cor(x, y) ^ 2
lmfit=lm(output~., data=train_df[-c(1:2)])
step.model <- step(lmfit, direction = "forward", k=2)
y_train = predict(lmfit, train_df)
y_test = predict(lmfit, test_df)
train_mse = mean((train_df[,3] - y_train)^2)
test_mse = mean((test_df[,3] - y_test)^2)
test_mse
train_mse
rsq(train_df[,3],y_train)
rsq(test_df[,3],y_test)
```

For Forward Selection (AIC), Training MSE: 0.0001505368 and Testing MSE: 0.3434714 Training $R^2 =$ 0.9873273 and Testing MSE : 0.9592604 For Backward Selection (AIC), Training MSE: 0.0001725394 and Testing MSE: 0.2689831 Training $R^2 =$ 0.9871564 and Testing MSE : 0.9619352

The training MSE is lower for AIC but the BIC testing MSE is lower. It looks like AIC is overfitting. The BIC model works better on the testing data. However, the differences in accuracy are only slightly different and might not be that significant in terms of model selection.

[15 Points] Best Subset Selection

Fit the best subset selection to the dataset and report the best model of each model size (up to 7 variables, excluding the intercept) and their prediction errors. Make sure that you simplify your output so that it only presents the essential information. If the algorithm cannot handle this many variables, then consider using just day 1 and day 2 information.

Again, I could not find methods in Python to do this, so regsubsets() from the leaps() library was used to get the best variable subset up to 7 parameters.

Here are the various subsets of model paramaters for given nvar (no. of variables):

(nvar = 1) btc_market_price_day1

Testing MSE: 0.3235

(nvar = 2) btc_market_price_day1 + btc_estimated_transaction_volume_usd_day2

Testing MSE: 0.3159

(nvar = 3): btc_market_price_day1 + btc_estimated_transaction_volume_usd_day2 + btc_cost_per_transaction_day1

Testing MSE: 0.2983

(nvar = 4): btc_market_price_day1 + btc_estimated_transaction_volume_usd_day2 + btc_n_transactions_total_day1 + btc_blocks_size_day2

Testing MSE: 0.2882

(nvar = 5): btc_market_price_day1 + btc_estimated_transaction_volume_usd_day2 + btc_n_transactions_total_day1 + btc_blocks_size_day2 + btc_avg_block_size_day2

Testing MSE: 0.2526

(nvar = 6): btc_market_price_day1 , btc_estimated_transaction_volume_usd_day2, btc_n_transactions_total_day1, btc_blocks_size_day2, btc_total_bitcoins_day2, btc_cost_per_transaction_day2

Testing MSE: 0.2737

(nvar = 7): btc_market_price_day2 + btc_estimated_transaction_volume_usd_day2 + btc_n_transactions_total_day1 + btc_blocks_size_day2 + btc_total_bitcoins_day2 + btc_cost_per_transaction_day2 + btc_estimated_transaction_volume_usd1

Testing MSE: 0.2861

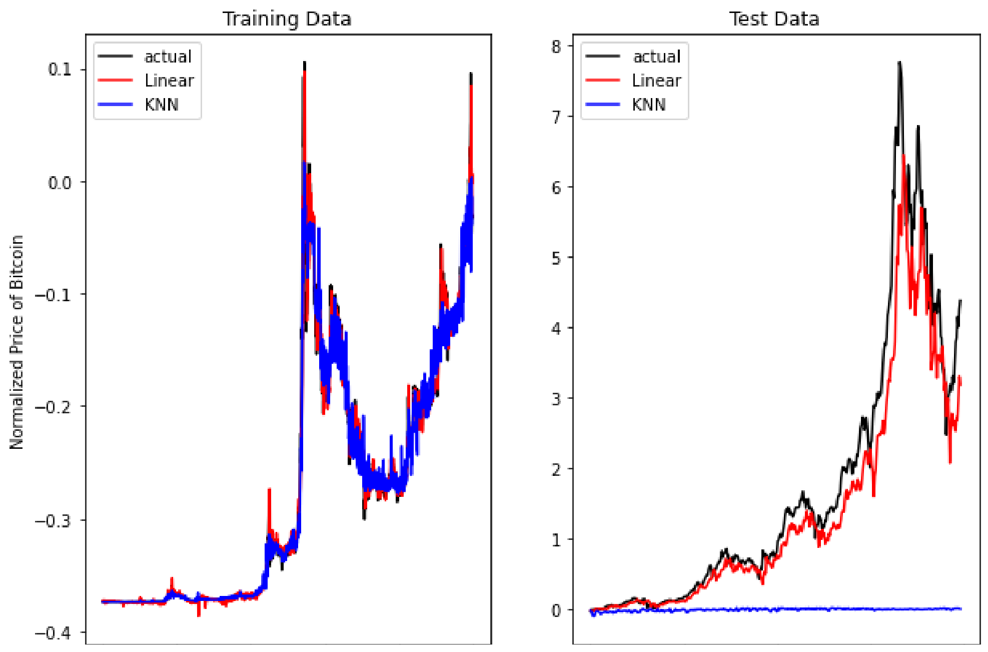
[15 Points] KNN

Use KNN to perform this prediction task. Do you expect KNN to perform better or worse than the linear model, and why? Does the analysis result match your intuition? Report your model fitting results.

We expect the KNN to have poorer prediction accuracy as compared to the linear regression model, as the covariates are clearly closely related to each other, but the same can not be likely said about the outputs. So we expect the test RMSE from KNN to be poor.

In [44]:

```
X_train, y_train = train_df.iloc[:,2:].values , train_df.iloc[:,1].values
X_test, y_test = test_df.iloc[:,2:].values , test_df.iloc[:,1].values
knnreg = KNeighborsRegressor(n_neighbors = 10); linreg = LinearRegression()
knnreg.fit(X_train, y_train); linreg.fit(X_train, y_train)
y_predl_train, y_predk_train = linreg.predict(X_train) , knnreg.predict(X_train)
y_predl_test, y_predk_test = linreg.predict(X_test) , knnreg.predict(X_test)

fig, axes = plt.subplots(1, 2, figsize=(10,7))
ax1 = axes.flat[0]

ax1.plot(y_train, color = 'black', label = 'actual')
ax1.plot(y_predl_train, color='red', label = 'Linear')
ax1.plot(y_predk_train, color='blue', label = 'KNN')
ax1.set_xlabel('Days')
ax1.set_ylabel('Normalized Price of Bitcoin')
ax1.set_title('Training Data')
ax1.legend()
ax1.set_aspect('auto')

ax2 = axes.flat[1]
ax2.plot(y_test, color = 'black', label = 'actual')
ax2.plot(y_predl_test, color='red', label = 'Linear')
ax2.plot(y_predk_test, color='blue', label = 'KNN')
ax2.set_xlabel('Days')
ax2.set_title('Test Data')
ax2.legend()
ax2.set_aspect('auto')

plt.show()
```



In [35]:

```
#Now we calculate fitting metrics for both linear and KNN regression models

testl, trainl = metrics.mean_squared_error(y_test,y_predl_test),
metrics.mean_squared_error(y_train,y_predl_train)
testk, traink = metrics.mean_squared_error(y_test,y_predk_test),
metrics.mean_squared_error(y_train,y_predk_train)
testlr2, trainlr2 = metrics.r2_score(y_test,y_predl_test), metrics.r2_score(y_train,y_predl_train)
testkr2, trainkr2 = metrics.r2_score(y_test,y_predk_test), metrics.r2_score(y_train,y_predk_train)
testl, trainl, testk, traink, testlr2, trainlr2, testkr2, trainkr2 #All names are of course represe
ntative
```

Out[35]:

```
(0.34347142254387314,
 0.00015053676140474093,
 6.462431621628105,
 0.000253528220466723,
 0.903906675423322,
 0.9873272528310638,
 -0.8080006038708343,
 0.9786570469021403)
```

Thus, we see from the above results that the testing MSE is much poorer in KNN than the linear, but training MSE is nearly comparable.

Similarly, testing R^2 for KNN is much poorer than testing R^2 the linear model. Training R^2 is close.

In [ ]:

```

```