FinalQ2

December 21, 2021

0.1 Part 2

0.2 Model Specification and Fitting:

In this part, various regression will be used. To simplified the implementation and reduce possible bugs, models implementation from sklearn will be used.

```
[1]: import warnings
     import numpy as np
     from scipy.stats.distributions import chi2
     from genetic_algorithm import GeneticAlgorithm as ga
     import _pickle as pickle
     import scipy.integrate as integrate
     from kmeans import KMeans
     import matplotlib.pyplot as plt
     from sklearn import linear_model
```

get_data is a simple function for opening and reshape the raw data so that can be used in further analyzation.

```
[2]: def get_data(path):
         data=pickle.load(open(path, "rb"))
         VIX=np.ndarray(shape=(382,1))
         category=np.ndarray(shape=(382,44))
         for i in range(382):
             for j in range(1,45):
                 category[i][j-1]=data.iloc[i][j]
             VIX[i]=data.iloc[i]["VIX"]
         return VIX, category
```

cal_err is a function mianly used for calculating the training error of our models and original data, while also ploting the predictions result and original data y.

```
[3]: def cal_err(result,y):
         x=np.arange(382)
         plt.scatter(x,y)
         plt.scatter(x,result)
         err=0
         for i in range(y.size):
             err+=np.power(result[i]-y[i],2)
```

```
    return np.sqrt(err)
```

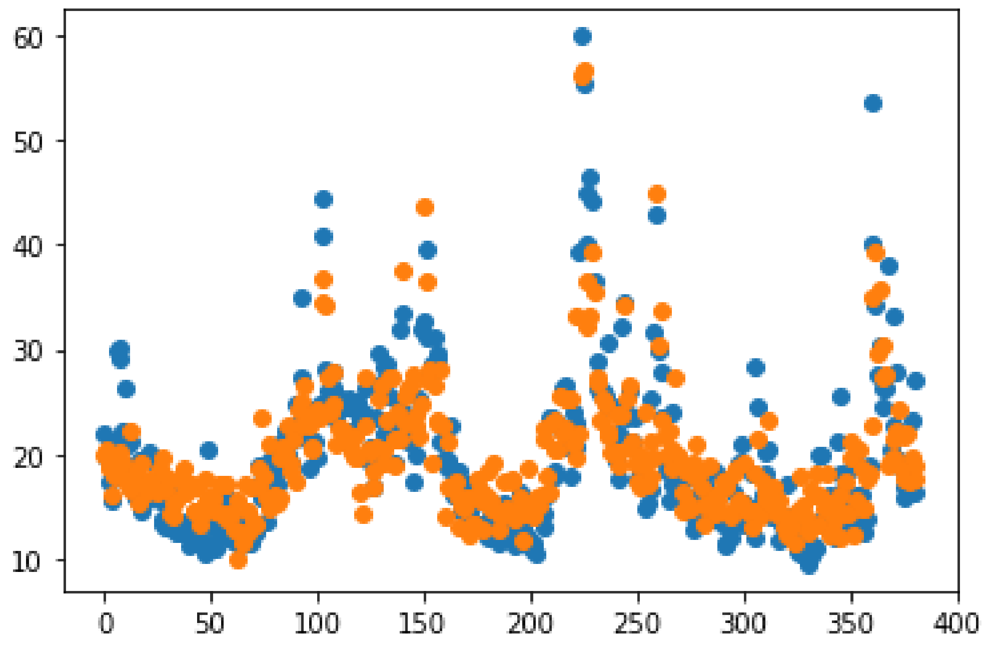
The first regression model going to be used is OLS, ordinary least square. The function OLSlearn returns the matrix b including weights of each factor that minimize the least square difference of our predicted function and original data.

```
[4]: def OLSlearn(matrix,y):
         b=np.matmul(np.matmul(np.linalg.inv(np.matmul(matrix.
      ↪transpose(),matrix)),matrix.transpose()),y)
         return b
```

The OLS is the most basic and simple way of implementing a regression model. But since it minimizes the squared distance of each data point to the prediction, the prediction will be largely affected by outliers. Let's see how it works on our data.

```
[5]: (VIX,category)=get_data("MEMV.pkl")
     ols=linear_model.LinearRegression()
     ols.fit(category,VIX)
     result=ols.predict(category)
     print("Weights given by OLS are:\n",ols.coef_)
     print("Error of OLS is:",cal_err(result,VIX))
```

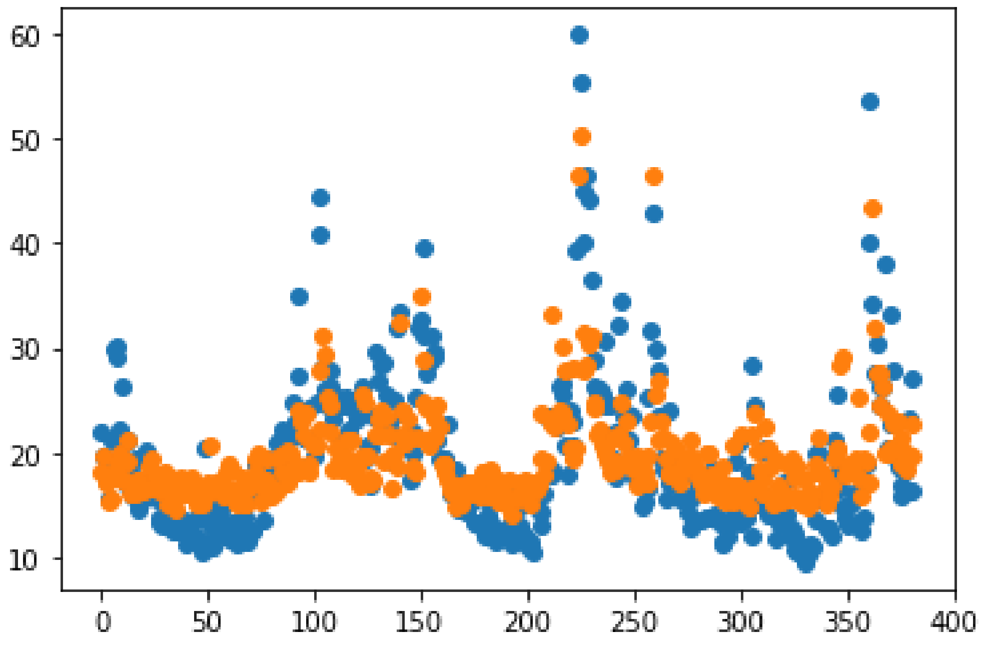
```
Weights given by OLS are:
 [[ 0.81470959  0.12316069  0.71763188  0.22068412  0.12180897 -1.23103337
  -0.83568732  0.06581011 -0.19124516 -0.89196529  1.24789239  0.05807143
   0.6131868   0.17164026 -0.03802225 -2.71561627 -1.29671913 -0.00888025
   3.60557379  3.11482644  0.58524551 -0.69005896 -1.1736371  -0.04528518
  -0.59594294 -1.23787363  1.60789727  0.74400331 -1.30846874  0.72071475
  -0.36038058  3.7005371   0.58234562  1.00188005 -1.69892618 -0.3386382
   0.50650244 -0.64114433  1.64767491  3.04738913 -0.37117688  0.01509176
  -8.87128398 -0.37799556]]
Error of OLS is: [88.79396646]
```



The next one is least absolute shrinkage and selection operator, as known as LASSO. Comparing to OLS, LASSO does selection on variables while reducing the squared distances as in OLS. Instead just reducing squared distances, LASSO reduce the Lagrangian form which add an additional term of L1 norm of the weights multiplied by lagrangue multiplier, the alpha in this implementation. As the lagrange multiplier increases, the norm of weights will be decrease, so if the lagrange multiplier is set to a large value, the LASSO will select only a subset of variables for predicting. The following code runs LASSO with lagrange multiplier set to 2. The weights returned are mostly 0, which means the variables corresponding to these weights are not selected by LASSO. That is, the selected variables with non-zero multipliers are the variables affecting the prediction most. The drawbacks of LASSO is obivious that its training error might be high, since it only selects some of the variables. The error of LASSO grows as the lagrange multiplier increases, meaning that the selection part of LASSO is more important.

[6]:
```
lasso=linear_model.Lasso(alpha=2)
lasso.fit(category,VIX)
result=lasso.predict(category)
print("Weights given by LASSO are:\n",lasso.coef_)
print("Error of LASSO is:",cal_err(result,VIX))
```

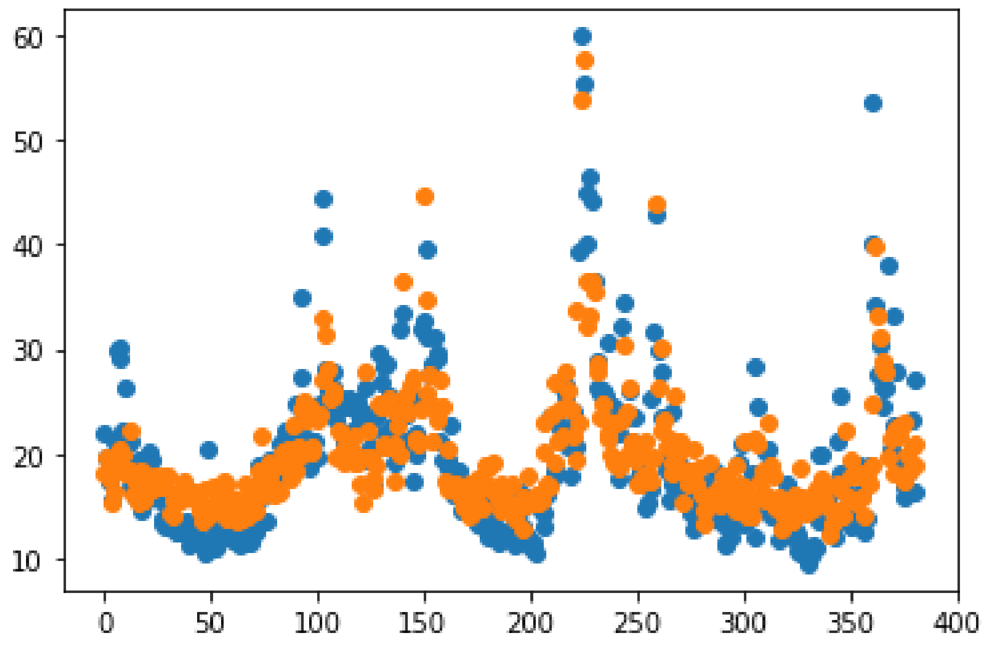
```
Weights given by LASSO are:
 [ 0.24548928  0.          0.73260428  0.         -0.         -0.52045238
  0.          0.          0.         -0.         -0.          0.
  0.          0.          0.         -0.          0.          0.
  0.          0.          0.          0.         -0.          0.
 -0.          0.          0.16220328  0.          0.          0.
 -0.          0.         -0.          0.          0.          0.
  0.         -0.         -0.          0.          0.         -0.
 -0.          0.         ]
Error of LASSO is: [108.73242097]
```



Ridge Regression is another way of reducing our weights assigned to each variable, besides LASSO. The difference between LASSO and Ridge Regression is the lagrangean term. Instead of L1 norm of weights, Ridge Regression uses L2 norm. Recall that in OLS, the weights depends on the inverse of X's transpose multipled by X, so if the variables in X are highly correlated, we are more likely to have a singular or nearly singular matrix to compute inverse. Ridge Regression is designed for this, in other words, it reduces the effect of multicollinearity. Comparing to LASSO, the selection is not that obivious. So we are more likely to see non-zero weights when using Ridge Regression, while still reducing the norm of weights. In the following code, the lagrange multiplier of Ridge Regression is set to 100, but all the weights returned are non-zero. The problem of Ridge Regression is also similar to LASSO: as the lagrange multiplier increase, the error is likely to increase.

[7]:
```
ridge=linear_model.Ridge(alpha=100)
ridge.fit(category,VIX)
result=ridge.predict(category)
print("Weights given by Ridge regression are:\n",ridge.coef_)
print("Error of Ridge regression is:",cal_err(result,VIX))
```

```
Weights given by Ridge regression are:
 [[ 0.65223934 -0.0140736   0.65912545  0.20010382  0.05375259 -1.03893133
  -0.08952908  0.1277474  -0.18512419 -0.13572547  0.08365098  0.29966804
   0.51269289  0.0078555   0.05392433 -0.47623363 -0.28889317  0.20491801
   1.13465792  0.51044221  0.03794635 -0.06844842 -0.58273855  0.09371907
  -0.48804652 -0.2221446   0.77010625  0.34573961  0.01270324  0.07552712
  -0.04086007  0.34969556 -0.26243759  0.06155845 -0.04568516 -0.21909689
   0.33794719 -0.48619262 -0.21575075  0.20935325  0.06935854 -0.19512172
  -0.11889471 -0.23266661]]
Error of Ridge regression is: [94.24104756]
```

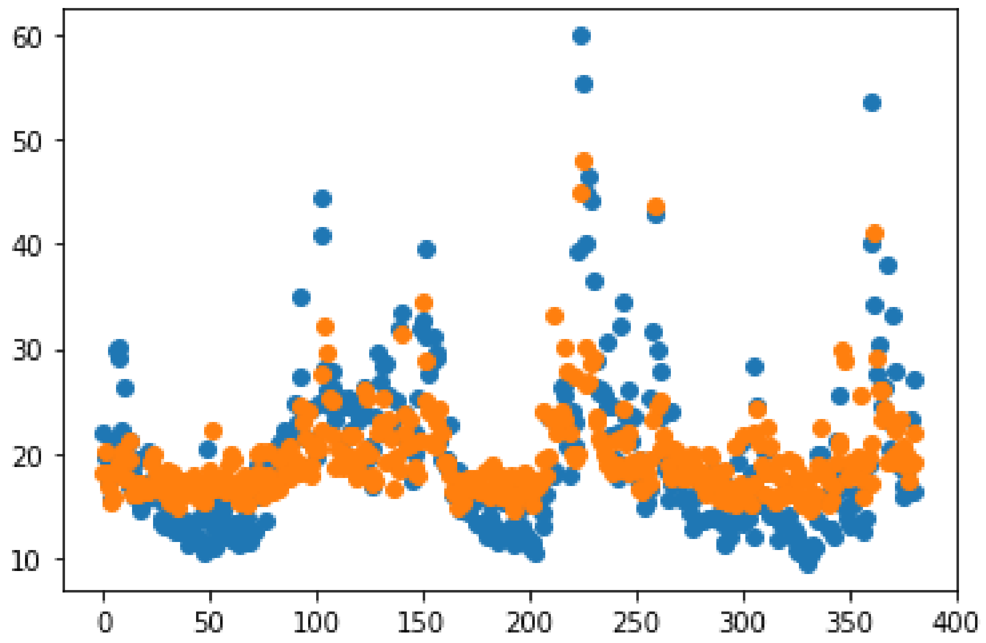


The Elastic Net is a combination of LASSO and Ridge Regression. That is, it has both L1 norm and L2 norm. In hyperparameters, Elastic Net has 2 lagrange multipliers, corresponding to two addition lagrangean term comparing to OLS. In the code implementation, we are using two parameters two set these lagrange multipliers, alpha is the sum of them and l1_ratio is the ratio od L1 term's multiplier in alpha. If the l1_ratio is set to 1, this implementation of Elastic Net performs exactly the same as LASSO. If it is set to 0, then simplely just another implementation of Ridge Reegression. By setting a ratio between 0 and 1, we can balance the L1 and L2 norm to obtain the advantages of both LASSO and Ridge Regression. So, the Elastic Net can reduce the effects of multicollinearity while also performing the selection. As a result, when selecting variables, the Elastic Net is likely to include a group of highly correlated variables if some of them are selected in LASSO. In the code below, the ElasticNet runs with the lagrange multipliers set to 2 and 3, the same multiplier for L1 term as in LASSO, so that we can compare the weights given by LASSO and Elastic Net to see the difference.

[8]:
```
elastic=linear_model.ElasticNet(alpha=5,l1_ratio=0.4)
elastic.fit(category,VIX)
result=elastic.predict(category)
print("Weights given by Elastic Net are:\n",elastic.coef_)
print("Error of Elastic Net is:",cal_err(result,VIX))
```

```
Weights given by Elastic Net are:
 [ 0.2788007   0.          0.44422241  0.         -0.         -0.16599753
```

```
  0.          0.          0.0929177  -0.          -0.          0.
  0.09336748  0.          0.          0.          0.          0.
  0.          0.          0.          0.          -0.          0.
 -0.          0.099537    0.13366051  0.          0.          0.
 -0.          0.          -0.          0.          0.          0.
  0.          -0.          0.          0.          0.          -0.
 -0.          0.        ]
Error of Elastic Net is: [112.16125621]
```



0.3 Forecasting:

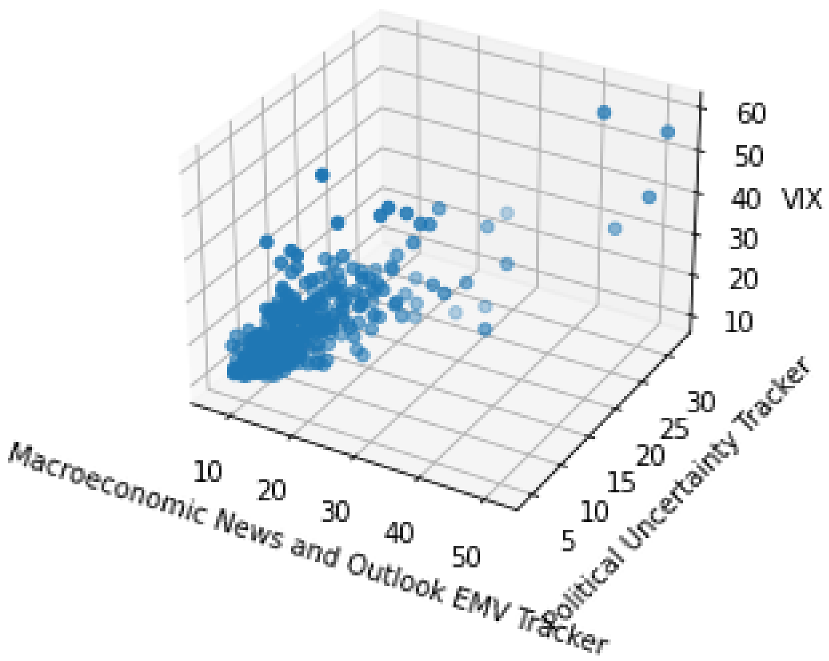
For forecasting, we first need to identify the important variables and how they relates to each other and the VIX. To identify them, we will use LASSO and Elastic Net model mentioned aboved. Let's run the LASSO first to find the most significant variable. Then run Elastic Net to find the correlated categories.

[9]:
```
lasso=linear_model.Lasso(alpha=10)
lasso.fit(category,VIX)
max_label=np.where(lasso.coef_==np.max(lasso.coef_))
elastic=linear_model.ElasticNet(alpha=20,l1_ratio=0.5)
elastic.fit(category,VIX)
corr_label=np.where(elastic.coef_!=0)
labels=pickle.load(open("MEMV.pkl", "rb")).columns
print("The most significant variable is:", labels[max_label[0][0]+1])
```

```
group=[]
for i in range(corr_label[0].size):
    group.append(labels[corr_label[0][i]+1])
print("The correlated group is:",group)
macroEMV=np.zeros(shape=(382,1))
macroVIX=np.zeros(shape=(382,1))
politicEMV=np.zeros(shape=(382,1))
macroData=pickle.load(open("MEMV.pkl", "rb"))
for i in range(382):
    macroEMV[i]=macroData.iloc[i]["Macroeconomic News and Outlook EMV Tracker"]
    politicEMV[i]=macroData.iloc[i]["Political Uncertainty Tracker"]
    macroVIX[i]=macroData.iloc[i]["VIX"]
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.scatter(macroEMV,politicEMV,macroVIX)
ax.set_xlabel('Macroeconomic News and Outlook EMV Tracker')
ax.set_ylabel('Political Uncertainty Tracker')
ax.set_zlabel('VIX')
```

```
The most significant variable is: Macroeconomic News and Outlook EMV Tracker
The correlated group is: ['Political Uncertainty Tracker', 'Macroeconomic News
and Outlook EMV Tracker']
```

```
[9]: Text(0.5, 0, 'VIX')
```



It is obivious from the plot that as ‘Political Uncertainty Tracker’, ‘Macroeconomic News and Outlook EMV Tracker’ increases, the VIX increases. So in the future, when there are some changes in political uncertainty or macroeconomic news, the VIX is likely to change.

Discussion: –

By doing the regression, mainly using LASSOS and Elastic Net, we identify the most impactful factor to the stock market: Macroeconomic News and Outlook, and it correlated factor: Political Uncertainty. Since they are correlated, so when there are some shocks in either one, it is likely to affects the others and making VIX in the stock market to change.

However, this conclusion maybe not be totally trust-worthy due to some limitations on the data. The EMV is only about the frequency of words of each category mentioned on newspaper. So if a word mentioned frequently in both positive and negative way, it is likely to have large EMV but relatively small effects on the stock market, since the positive and negative effects of that word may offset the other one’s. Also, VIX is an index of the overall stock market. If some words have effects on only a part of market, it is unlikely be realized in analyzation like this one. But this kind of effects actually exists, like Elon Musk saying something that causes the stock price of Tesla to change. Then what he said is likely to be mentioned in newspaper, while Tesla is relatively small comparing to the whole stock market.

0.4 Bibliography:

Genetic Algorithm Solver: https://q.utoronto.ca/courses/226292/files/17294209?wrap=1

Background of VIX and EMV: https://q.utoronto.ca/courses/226292/files/18132962?wrap=1

sklearn Regression Model Implementation: https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model