Multivariate Financial Time Series Analysis

Applying multiple machine learning methods

UCLA Master of Applied Economics

Capstone Research

Minxuan Wang

Instructed by Prof. Rashed Iqbal

Abstract

Prediction of financial time series is one of the most challenging tasks of time series prediction, due to its characteristics and dynamic nature. In linear time series analysis, one of the most popular approaches is ARIMA model, however, limited to its structural features and modeling methods, it is difficult to capture the characteristics other than the secondary moment of time series, especially for the nonlinear features. Then the method of time series prediction using machine learning is derived. For the more-complicated financial time series analysis, it could be influenced by many factors such as market participants' sentiment, related stock prices / market trend and attention rate. In this paper, I chose Google stock prices as target dependent variable and combine the independent features of other stock prices, stock 3market indices, market sentiment score and Google Trends indices, upgrading the use of ARIMA model with exogenous variables model (ARIMAX), k-NN regression, regression tree, random forest and LSTMs neural network to predict the Google stock prices.

Keywords: Google, stock price, time series, machine learning, predicting

I. Introduction

Stock prices prediction is a challenging application of time series analysis, because they are influenced by different numbers of factors and the nonlinear relationships between factors existed in different periods such that predicting the value of stock prices or trends is way more difficult. The most common methods and approaches in time series are ARMA/ARIMA, ARCH/GARCH models, which can capture the dependence of observed data in time series, as well as the disturbance of random fluctuation. With the development of machine learning and deep learning, more advanced and complicated methods are introduced into this field. To apply these algorithms to time series data, more work is needed. For example, univariate time-series data consists a series of observations, which have to be transformed into input and output features before they can be used in supervised machine learning algorithms. For complex time series with nonlinear and high dimensional features in finance market, the prediction of these time series is the basis of understanding and decision-making of the market with important realistic meaning. In recent years, with the infiltration and influence of various fields in society, the dimension of multifactor time series in social and economic fields has increased sharply. However, because of the inherent characteristics of the financial time series, such as nonlinear, non-stationary, noise, lag influence and hidden relationships, those traditional statistical analysis methods have many defects in capturing all the features and accurately describing the time series. The efficiency and effect of traditional ARIMA and machine learning methods in processing high dimensional time series in stock market are not ideal. The reason is that a large number of unrelated and redundant information are hidden in the time series with high dimension, which reduces the precision and efficiency of analysis. Therefore, feature selection for high

dimensional feature time series is also very necessary. In this paper, we are aimed to apply different models to analyze Google's stock prices, come up with multiple possible features which could influence Google stock prices, in order to cover the infiltration and influence of various fields in society as much as possible, combine market indices, sentiment indices and Google Trends indices; apply the optimization methods for linear regression, for example, Ridge, LASSO and ElasticNet models to high dimensional multivariate time series' feature selection engineering to reduce the dimension; apply ARIMAX, k-NN, regression tree, random forest and LSTMs models with the selected features, finally compare the predictive performance of those models using RMSE values.

II. Methodology

2.1 Feature Selection Model

This is often the case when the machine learning method is used to predict: the performance of the trained model on the training set is good, but the effect on the test set is very poor, which is called overfitting; if the model itself has a poor performance on the training set, it is called less fitting. In extreme cases, if the data set has more features than the sample points $(\mathrm{X}_{\mathrm{N}\times\mathrm{d}}, \mathrm{d} > \mathrm{N})$ or there is collinearity between variables. The coefficients obtained by OLS are unstable and the variance is very large because we have problem when calculating $(\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1}$. In order to prevent from over fitting, some new methodologies have optimized linear regression, resulting in Ridge, LASSO and ElasticNet regression, and following the three kinds of regression are introduced.

2.1.1 Ridge Regression

In linear regression, we have:

$$\mathrm{h}_\theta \mathrm{x} = \theta_0 + \theta_1 \mathrm{x}_1 + \theta_2 \mathrm{x}_2 + \cdots + \theta_n x_n$$

Convert to array expression:

$$h_\theta x = \sum_{i=0}^{n} \theta_i x_i = \theta^T x$$

Loss function (squared error):

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right)^2$$

The goal is to find the optimal θ which minimized the loss function $\mathrm{J}(\theta)$ and there are two ways to get there: Gradient Decent and Regularization Equation. In regularization, the optimal θ satisfies $\frac{\partial}{\partial \theta_j} \mathrm{J}(\theta) = 0$ can minimize the loss $\mathrm{J}(\theta)$. We get:

$$\theta = (\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1} X^T Y$$

Where X is $\mathrm{m} \times \mathrm{n}$ matrix. In order to prevent the occurrence of over fitting, we add

a regular term $\lambda \sum_i^n \theta_i^2$ to the original loss function. Hence:

$$J(\theta) = \frac{1}{2m}\left(\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda \sum_{i}^{n} \theta_i^2\right)$$

In matrix expression:

$$\mathrm{J}(\theta) = \frac{1}{2m}((X\theta - Y)^T(X\theta - Y) + \lambda\theta^T\theta)$$

$$= \frac{1}{2m}\left((\theta^T X^T X\theta - \theta^T X^T Y - Y^T X\theta + Y^T Y) + \lambda\theta^T\right)$$

Compute partial derivative of θ:

$$\frac{\partial J(\theta)}{\partial \theta} = X^T\mathrm{X}\theta - X^T\mathrm{Y} + \lambda\theta$$

Let $\frac{\partial J(\theta)}{\partial \theta} = 0$, $\theta = (\mathrm{X^T X} + \lambda \mathrm{I})^{-1}\mathrm{X^T Y}$, which is the solution of the Ridge regression coefficient. In other words, when $\mathrm{X^T X}$ is positive semidefinite matrix (Hessian matrix is positive) but not inversible, we can add $\lambda \mathrm{I}$ to make sure $\mathrm{X^T X} + \lambda \mathrm{I X^T X} + \lambda \mathrm{I}$ is inversible.

The reason we add a coefficient square term when constructing the loss equation of ridge is from the overfitting situation, that is to say, the model has the characteristics of low bias and high variance, and the model may be very volatile in the smaller interval which means the derivative of the model is very large. The order of the model and the value of the independent variable have little effect on the derivative. According to the Goss Markoff theorem, multicollinearity does not affect the unbiased and minimum variance of the least squares estimator. However, although the least square estimator is the least variance in all linear estimators, the variance is not necessarily small. So we can use a biased estimator. Although it's biased, its accuracy is much higher than the unbiased estimator. In fact, when there is an over fitting situation, the coefficients of the model will be very large when features have multicollinearity, which can cause this problem. To reduce the variance, we use λ to restrict the sum of all parameters θ. By introducing this penalty, the unimportant parameters can be reduced (which is also known as shrinkage). In Ridge regression, this term is also called as L2 norm, representing the square sum of elements in vector X:

$$\left|\left|\theta\right|\right|^2 = \sum_{i=1}^{n} \theta_i^2 \leq \lambda$$

This term can prevent the model from being too complex to match the training model and improve the generalization ability.

2.1.2 LASSO Regression

We can use another penalty term:

$$\left|\left|\theta\right|\right|_1 = \sum_{i=1}^{n} |\theta_i| \leq \lambda$$

The loss function will be:

$$J(\theta) = \frac{1}{2m}\left(\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda\sum_{i=1}^{n}|\theta_i|\right)$$

This constraint uses the absolute value instead of the sum of squares. Although the constraint form is only slightly changed, the result is quite different: when λ is small enough, some of the coefficients will be reduced to 0. In LASSO, this term is also called L1 norm. Because it can shrink the coefficients to exactly 0, LASSO has the function of feature selection and dimension reduction, remove some of the features that have no explanatory information.

2.1.3 ElasticNet Regression

It combines Ridge and LASSO, the loss function is:

$$J(\theta) = \frac{1}{2m}\left(\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda_1\sum_{i}^{n}\theta_i^2 + \lambda_2\sum_{i=1}^{n}|\theta_i|\right)$$

2.2 ARIMAX Model

Let's say we have a stationary time series represented as $y_1, y_2 \ldots, y_n$. First, we can define an $\mathrm{ARMA}(p, q)$ model with no covariates:

$$y_t = \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} - \theta_1 z_{t-1} - \cdots - \theta_q z_{t-q} + z_t$$

where z_t is a white noise process (i.e., zero mean and i.i.d).

Assuming that the response sequence y and the input variable sequence X (i.e. the sequence of independent variables) are all stationary, a regression model of response sequence and input variable sequence can be constructed to analyze the influence of other time series and the lag terms on a time series, I.e. an ARMAX model, by simply adding in the covariate terms:

$$y_t = \beta x_t + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} - \theta_1 z_{t-1} - \cdots - \theta_q z_{t-q} + z_t$$

Where x_t is a covariate at time t and β is its coefficient. More generally, if we have multiple time series as independent variables $x_t^{(i)}, i = 1, \ldots, k$, i.e. the features we'll use to explain the change in the response time series y_t, combine them together if we write the model using backshift operators:

$$y_t = \sum_{j=0}^{\infty} v_j^{(1)} B^j x_t^{(1)} + \sum_{j=0}^{\infty} v_j^{(2)} B^j x_t^{(2)} + \cdots + \sum_{j=0}^{\infty} v_j^{(k)} B^j x_t^{(k)} + \frac{\theta(B)}{\phi(B)} z_t$$

Where $\theta(B) = 1 - \theta_1 B - \cdots - \theta_q B_q$, $\phi(B) = 1 - \phi_1 B - \cdots - \phi_p B^p$ are called transfer function model, $x_t^{(i)}$ is input factor and y_t is output factor. Therefore, we can see that the ARIMAX model represents the corresponding sequence as the combination of autoregressive lag term, random fluctuation lag term and other sequences' lag term.

2.3 k-NN Regression

2.3.1 Description

k-NN (k Nearest Neighbors) is a classic algorithm in machine learning. It computes the distance between a specific data point and the eigenvalues of the training data, and then selects k nearest neighbors ($k \geq 1$) based on the distance to classify or regression. If $k = 1$, then the new data point will be assigned to the class of its nearest neighbor.

k-NN algorithm is a supervised learning. When the k-NN algorithm is used for classification, each training data has a clear label, and it can clearly determine the label of the new data point. When it is used for regression, a clear value will be predicted according to the neighbor's value.

2.3.2 The process of k-NN algorithm

Input: Train set $\mathbb{T} = (\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \dots, (\mathbf{x_n}, y_n)$, samples to be classified (regressed) $\mathbf{x}'$ and the determined number of neighbors k.

Output: Category label of $\mathbf{x}'$ for classification, expected value of $\mathbf{x}'$ for regression.

Algorithm: Search the training set $\mathbb{T}$, according to a given distance measure like Euclidean distance $d = \sqrt{\sum_i (x_i - y_i)^2}$, find the nearest k neighbors to $\mathbf{x}'$ and mark the fields covered with these neighbors as $N_k(\mathbf{x}')$. For regression, return the weighted values of these neighbors as predicted values. In general, Euclidean distance is used as distance measure for continuous variables.

The k-NN regression algorithm is when the category label of the data point are continuous values. The difference between regression and classification lies in the processing of the k neighbors. The k-NN regression uses the weighted values of the nearest neighbors' attribute values as the prediction value of the new data point. The weighting method can be: the mean value of the attribute values of the k nearest neighbors; $\frac{1}{d}$ as the weight (more effective by giving the nearest neighbors a high weight and far neighbors lower); the Gauss function.

2.4 Regression Tree

2.4.1 Description

A regression tree corresponds to a division of input space (i.e. feature space) and the output value on the partition unit. In classification tree, we use the method of information theory to select the best partition point by calculation. In regression tree, the heuristic method is adopted. If we have n features, each feature has $s_i(i \in (1, n))$ values, then we traverse all the features, try all the values of the feature and divide the space until the value s of feature j minimize the loss function, we obtain a division point:

$$\min_{j,s}\left[\min_{c_1} Loss(y_i, c_1) + \min_{c_2} Loss(\mathrm{y}_2, \mathrm{c}_2)\right]$$

Assuming that the input space is divided into M units $\mathrm{R}_1, \mathrm{R}_2, \ldots, \mathrm{R}_\mathrm{m}$, the output value of each region will be $\mathrm{c}_\mathrm{m} = \mathrm{average}(\mathrm{y}_\mathrm{i} | \mathrm{x}_\mathrm{i} \in \mathrm{R}_\mathrm{m})$, which is the average of y values of all points in the region.

2.4.2 Algorithm (OLS regression tree)

Input: Training set $\mathbb{T}$.

Output: Regression tree $\mathrm{f(x)}$.

In the input space where the training set in, divide each region into two sub-region and determine the output value of each sub-region recursively and construct binary decision tree:

(1) choose the optimal segmentation variable j and segmentation point s that minimize

$$\min_{j,s}\left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2\right]$$

Recursively, find the optimal (j, s).

(2) use the selected (j, s) divide the space and determine the output value

$$\mathrm{R}_1(\mathrm{j}, \mathrm{s}) = \{\mathrm{x} | \mathrm{x}^{(\mathrm{j})} \leq \mathrm{s}\}, \qquad \mathrm{R}_2(\mathrm{j}, \mathrm{s}) = \{\mathrm{x} | \mathrm{x}^{(\mathrm{j})} > \mathrm{s}\}$$

$$\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m(j,s)} y_i\,, \qquad x \in R_m, \qquad m = 1,2$$

(3) repeat (1) and (2) for each sub-region until satisfies the condition.

(4) divide the input space into M units $\mathrm{R}_1, \mathrm{R}_2, \ldots, \mathrm{R}_\mathrm{m}$, construct the decision tree

$$\mathrm{f(x)} = \sum_{m=1}^{M} \hat{c}_m I(x \in R_m)$$

2.5 Random Forest

A random forest is made up of multiple CART (Classification and Regression Tree). For each tree, the training sets they use are sampled from the total training set, which means that some samples of the entire training set may appear in the training set of a tree many times or never appears. When training each tree node, the features used are randomly chosen from all features.

Therefore, the training process of random forests can be summarized as follows:

(1) given the training set $\mathbb{T}$, test set $\mathbb{S}$ and the characteristic dimension $\mathbb{F}$. Determine the parameters: the number of CART used t, the depth d, the number of features used in each node f, and the terminating condition: the minimum number of samples on the node s, and the minimum information gain on the node m.

For tree $1 - \mathrm{t}$, $\mathrm{i} = 1 - \mathrm{t}$:

(2) starting from the root node, extract training sample $\mathrm{T(i)}$ from $\mathbb{T}$, which has the

same length with $\mathbb{T}$, as a sample of root nodes.
(3) if the current node satisfies the terminating condition, the current node is set as the leaf node. In regression problem, the prediction output is the mean values of the current node samples. Then continue to train the other nodes. If the current node does not satisfy the termination condition, continue randomly selecting f dimension features from $\mathbb{F}$. Use this f dimension features to find the optimal one-dimension feature k and its threshold th. Then we can allocate the current node by comparing the feature dimension and the threshold. For regression problem, the output is the mean value of the output of all trees.

2.6 Long Short-Term Memory networks

LSTM's (Long short-term memory) is a specific form of RNN (Recurrent neural network), which can process sequence data like time series. In general, RNN can generate one output at each time node, the connection between hidden layers is cyclic and the output of a time node is only circularly connected to the hidden layer of the next time node. So, when dealing with sequence data it can output a single forecast in each single time node. Since the RNN model needs long-term memory, it is necessary to link the computation of the hidden state with the preceding n times:

$$S_t = f(U * X_t + W_1 * S_{t-1} + W_2 * S_{t-2} + \cdots + W_n * S_{t-n})$$

In that case, the amount of computation will increase exponentially, resulting in a significant increase in the training time of the model. Therefore, the RNN model is usually used for long-term memory computation.
The characteristic of LSTM's is that the gate nodes of each layer are added outside the RNN structure. There are 3 types of gates: forget gate, input input gate and output gate. These gates can be opened or closed to determine whether the memory state of the previous network is added to the computation in the current layer. Because of these features, LSTM's are pretty good at extracting patterns in input feature space, where the input data spans over long sequences. Given the gated architecture of LSTM's that has this ability to manipulate its memory state.
Another great advantage of LSTM's is it can contain multiple input features as independent variables, just like ARIMAX model. If we choose to use 1-time-period lag memorize, it looks exactly like an ARIMAX model plus another AR model of input features! We will take a look later to see if LSTM's is also good at predicting auto regressive model like ARIMAX model.

III. Data Acquisition

3.1 Data collection

3.1.1 Topic modeling

How to select possible features is a major problem, a intuitive idea is that we can find the factors directly related to Google stock prices. However, we can not do it only by imagination, in deep learning field, LDA (Latent Dirichlet Allocation) is a well-known

method which can do Natural Language Processing. So how can LDA help us find related features? The general idea is to find the “topic” from the articles related to Google. From statistical view, we use a specific word frequency distribution to describe the topic, and think that an article, a paragraph, and a sentence are generated from a probability model. In python where are some existed function can help us do that, so I will not go deep into the theory of topic modeling, we only have to know LDA is an unsupervised machine learning technique that can be used to identify the hidden topic information in a large document collection or a corpus. It uses the bag of words method, which treats each document as a word frequency vector, which converts text information into easy-to-model digital information. Each document represents a probability distribution of some topics, and each topic represents a probability distribution of some specific words. Other than, I also used NMF (Nonnegative Matrix Factor), while LDA is good in identifying coherent topics where as NMF usually gives incoherent topics. And these functions are in “sklearn” package in Python which we can use directly, what we need to do is extracting Google related article, determine the number of features and number of topics, following are examples of what I get using NMF and LDA by setting #features is 100 and #topics is 10:

NMF	LDA
Topic 0:	Topic 0:
said company people	uber driving car
Topic 1:	Topic 1:
company billion market	apple app phone
Topic 2:	Topic 2:
facebook ads ad	search like use
Topic 3:	Topic 3:
mr ms chief	facebook media social
Topic 4:	Topic 4:
york new times	company year amazon
Topic 5:	Topic 5:
uber driving self	china says internet
Topic 6:	Topic 6:
data privacy facebook	mr said company
Topic 7:	Topic 7:
trump president government	new trump like
Topic 8:	Topic 8:
app apple phone	ads youtube ad
Topic 9:	Topic 9:
like people just	data companies privacy

By topic modeling, we can have a look at what topic is highly correlated to Google, and select what we think as good predictors for further utilization.

3.1.2 Sentiment Score

Behavioral economics tells us that emotions can profoundly affect individual behavior and decision-making. Societies experience mood states can also affect their collective decision making, by extension the public mood could be correlated or even predictive of economic indicators. As more and more market participants use network to obtain information and share communication, market related data and emotional information have become an important factor affecting investors' psychology and behavior. Behavioral finance puts forward that the psychology and behavior of irrational stock market investors will affect the stock market situation, make the stock out of its own

value, and there are a lot of information about stock investor behavior and emotion in the news report, which is of great significance to the research of stock market. In 2010, some scholars pointed out that the sentiment analysis of Twitter can be used to predict the fluctuation of the stock market. By applying Neural Networks, the prediction accuracy rate of sentiment on Dow Jones Index is as high as 87.6%[1].

In this paper, I will also consider the influence of market sentiment on Google stock prices. Generally speaking, the purpose of sentiment analysis is to find out the author's attitude on some topics, which in our case is Google. One of the basic steps of sentiment analysis is to classify the two polarity of a given text. The basic role of classification is to determine whether the views expressed in this text are positive, negative or neutral. Also, "TextBlob" package in Python can help us deal with sentiment analysis, the results are returned in range of $[-1,1]$, positive number is positive sentiment, negative number is negative sentiment. In this part I extracted all the articles related to Google on US major newspapers and media from January 2016 to June 2018, and used sentiment analysis get every day's sentiment score of Google. Following it shows the head of sentiment score data.

Sentiment score

	sentiment
time	
2016-01-04	0.097372
2016-01-05	0.119200
2016-01-06	0.100297
2016-01-07	0.062459
2016-01-08	0.153506

3.1.3 Market indices

Based on the related topic, we can choose some related companies' stock prices and the market indices. In this paper I chose Stock prices of **Amazon (AMZN)**, **Apple (AAPL)** and **Facebook (FB)**, Index of **SP500**, **NASDAQ** and **DJI**, **ETF** prices like **SPDR S&P 500 ETF** (**SPY**), **First Trust Dow Jones Internet ETF** (**FDN**), **iShares S&P 500 Growth** (**IVW**), **iShares US Technology (IYW)**, **Vanguard Growth ETF** (**VUG**) as potential predictors and all of them are directly related to Google stock prices. All these data can be queried from Yahoo! Financial using "fix_yahoo_finance" package in Python. Consistent with sentiment score, stock market data also range from January 2016 to June 2018, and for each day, I removed "Open", "Low", "Close", "High", "Volume" and only keep "Adjusted Close". Here is the data I got:

[1] Bollen, J., & Mao, H. (2011). Twitter Mood as a Stock Market Predictor.

Stock Market Data

	googl	amzn	aapl	fb	dji	sp500	nasdaq	spy	fdn	ivw	...
time											
2016-01-04	759.440002	636.989990	100.626175	102.220001	17148.939453	2012.660034	4903.089844	192.270966	72.330002	110.118095	...
2016-01-05	761.530029	633.789978	98.104546	102.730003	17158.660156	2016.709961	4891.430176	192.596176	72.080002	110.331047	...
2016-01-06	759.330017	632.650024	96.184654	102.970001	16906.509766	1990.260010	4835.759766	190.166733	71.669998	109.120979	...
2016-01-07	741.000000	607.940002	92.125244	97.919998	16514.099609	1943.089966	4689.430176	185.604340	69.010002	106.352303	...
2016-01-08	730.909973	607.049988	92.612358	97.330002	16346.450195	1922.030029	4643.629883	183.567032	68.220001	105.326141	...

3.1.4 Google Trends Index

Google Trends (https://trends.google.com) provides weekly reports on the volume of queries people are searching related to various industries. What if the query data may be correlated with the current level of economic activity in given industries and thus may be helpful in predicting the subsequent data state. In existed studies, people investigated the intriguing possibility of analyzing search query data from Google Trends to provide new insights into the information gathering process that precedes the trading decisions recorded in the stock market data[2]. So, this paper assumes that the Google trend is indeed effective in explaining the trend of the stock market based on the existing research.

Furthermore, to determine which terms' Google Trends indices should be used, I combine the result I got from topic modeling and some other related news reports, selected 18 Google Trends features: **AI**, **android**, **VR**, **gmail**, **google glass**, **pixel**, **self-driving car**, **youtube**, **uber, etc.** The "pytrends" package in Python can help us extract whatever indices we want from Google Trends. Here I got the 18 related Google Trends indices range from January 2016 to June 2018. Here is part of the Google Trends data we will use later.

Google Trends Data

	AI	android	android vr	apple	enterprise software	facebook	gmail	google assistant	google glass	google news	google play	google
time												
2016-01-04	89.857143	85.428571	12.142857	51.285714	62.285714	99.428571	91.142857	4.857143	88.142857	68.857143	84.428571	99.857143
2016-01-05	89.714286	84.857143	12.285714	50.571429	62.571429	98.857143	91.285714	4.714286	88.285714	68.714286	83.857143	99.714286
2016-01-06	89.571429	84.285714	12.428571	49.857143	62.857143	98.285714	91.428571	4.571429	88.428571	68.571429	83.285714	99.571429
2016-01-07	89.428571	83.714286	12.571429	49.142857	63.142857	97.714286	91.571429	4.428571	88.571429	68.428571	82.714286	99.428571
2016-01-08	89.285714	83.142857	12.714286	48.428571	63.428571	97.142857	91.714286	4.285714	88.714286	68.285714	82.142857	99.285714

[2] Preis, T., Moat, H. S., & Stanley, H. E. (2013). Quantifying Trading Behavior in Financial Markets Using Google Trends.

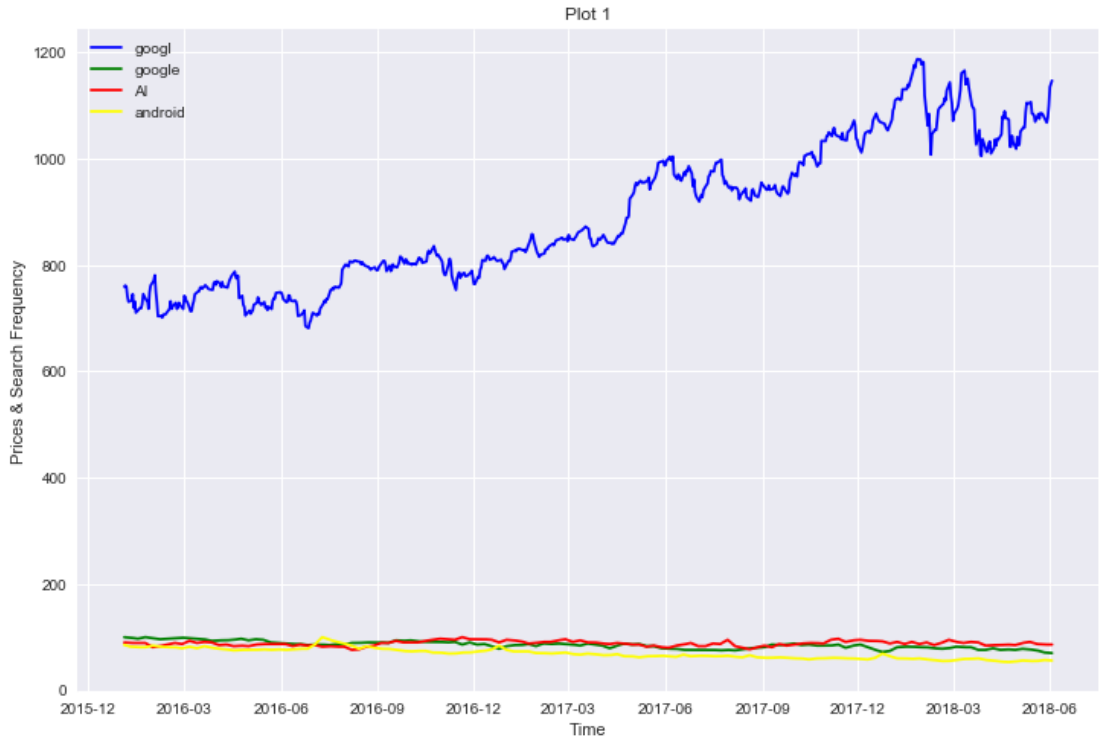
3.2 Data Pre-processing

3.2.1 Data Resampling

In the previous step, I got different types of data from different sources, even if they have the same time period, the frequency of the data is not always the same. What's more, Google Trends data is weekly released, however other data we are using are all daily. In order to make all data in same length and frequency, I have to up-sample the data into a higher frequency, and fill the blank values using a linear method in Python. In this way, weekly data has been converted into daily and the missing values are filled in a linear way. And we can combine all the data together now.

3.2.2 Data Scaling

Note that different data has different units, and the some of the absolute value of them might be vary large while some are small, if we use the absolute values directly, it's hard to compare different data. For example, here is the plot of GOOGL stock prices and Google Trends search frequency data:



Because the influence of different units, it's hard to distinguish each variable in absolute value. Also, with different dimensions and units, machine learning methods are not sufficient. In order to eliminate the dimensionless impact in indicators, we need to standardize the data. After the data scaling processing, the original data are in the same order of magnitude, which is suitable for comprehensive comparative evaluation. After scaling they are comparable now:

3.2.3 Data Stationary

The basic idea of stationarity is that the probability laws that govern the behavior of the process do not change over time. Fundamentally speaking, the stationary of data depends largely on its consistent variance. In most algorithm we will use in this paper, this assumption must be held, especially in time series analysis. In machine learning algorithm, Independent Identically Distributed and Generalization Ability are also needed, the generalization ability of a machine learning model refers to its fitting ability on new samples. The data guarantee that the model can acquire strong generalization is that its training data are sampled from the parent distribution independently and identically.

To test stationarity, we can use ADF (Augmented Dickey Fuller) test, which is unit root test for stationarity. Unit roots can cause unpredictable results in time series analysis. The null hypothesis for this test is that there is a unit root. The alternate hypothesis differs slightly according to which equation you're using. The basic alternate is that the time series is stationary (or trend-stationary). Following is part of the ADF test result:

ADF Test

googl
ADF Statistic: -0.409411938727
p-value: 0.908515420466
amzn
ADF Statistic: 1.17640396697
p-value: 0.995818732823
aapl
ADF Statistic: -0.12517103951
p-value: 0.946872967978
fb
ADF Statistic: -0.843612593959
p-value: 0.805974444524
dji
ADF Statistic: -0.921193608685
p-value: 0.780912964972
sp500
ADF Statistic: -0.978666433886
p-value: 0.76096549382
nasdaq
ADF Statistic: 0.00399572000617
p-value: 0.958861284521
spy
ADF Statistic: -0.848973811658
p-value: 0.804311605874
fdn
ADF Statistic: 1.39870689817
p-value: 0.997108259633
ivw
ADF Statistic: 0.0157473295302
p-value: 0.959815165359
iyw
ADF Statistic: 0.272449698911
p-value: 0.976044581349
vug
ADF Statistic: -0.17954254234
p-value: 0.940916565815
AI
ADF Statistic: -2.46186299976
p-value: 0.125031294617
android
ADF Statistic: -1.56796250181
p-value: 0.499664086379
android vr
ADF Statistic: -2.20409232362
p-value: 0.204763533681
apple
ADF Statistic: -2.43840310435
p-value: 0.131189405211
enterprise software
ADF Statistic: -2.62125948296
p-value: 0.0886824851346
facebook
ADF Statistic: -0.251697262406
p-value: 0.932072308659

Due to all of the p-values are greater than 0.05, all these features series are non-stationary. In order to get stationary series, we need to difference the dataset. By taking the first difference $y_t - y_{y-1}$, all the series are stationary, and our target is converted from value to return.

ADF Test

googl
ADF Statistic: -10.3869261306
p-value: 2.0684230675e-18
amzn
ADF Statistic: -8.13231022751
p-value: 1.08200322925e-12
aapl
ADF Statistic: -9.18324190972
p-value: 2.22262160706e-15
fb
ADF Statistic: -12.5343110741
p-value: 2.3903864425e-23
dji
ADF Statistic: -9.63036241699
p-value: 1.6183612129e-16
sp500
ADF Statistic: -9.86970100086
p-value: 4.02536715649e-17
nasdaq
ADF Statistic: -10.346252619
p-value: 2.60637959802e-18
spy
ADF Statistic: -9.86712841086
p-value: 4.08580998881e-17
fdn
ADF Statistic: -10.3000285097
p-value: 3.39123981828e-18
ivw
ADF Statistic: -10.2299941379
p-value: 5.05826861448e-18
iyw
ADF Statistic: -10.2192104582
p-value: 5.38002610283e-18
vug
ADF Statistic: -10.1421164359
p-value: 8.36753762865e-18
AI
ADF Statistic: -6.02663627236
p-value: 1.4506752698e-07
android
ADF Statistic: -5.66568375846
p-value: 9.16342057907e-07
android vr
ADF Statistic: -6.45450972119
p-value: 1.49495739602e-08
apple
ADF Statistic: -6.57399123778
p-value: 7.80816320194e-09
enterprise software
ADF Statistic: -6.01282155917
p-value: 1.55876148028e-07
facebook
ADF Statistic: -6.01339901674
p-value: 1.55408932106e-07

IV. Linear Model and Feature Selection

4.1 Data splitting

Because we want to know the predictive performance of each model, also prepare for cross-validation later on, we should split our dataset into a training set and a test set. Here is an example of 10-fold cross validation, which will generate 10 training sets and 10 test sets:

For now, because our data is sequence time series, we can't randomly split it, we will simply use first 75% as training set and the rest 25% as test set.

4.2 Determine the statistical indicator to test accuracy

There are some common used statistical indicators to compute the model performance: MAE (Mean Absolute Error) is a basic assessment method, MSE (Mean Square Error) can enlarge the value with high prediction deviation and compare the stability of different prediction models. As for RMSE (Root Mean Square Error), because the mean error is used, which is more sensitive to the outlier. If a predicted value of a model is not reasonable, its error will be larger, which will have a greater impact on the value of RMSE, that is, it is not robust. So we will use RMSE as the benchmark of accuracy measurement.

Error metrics equation to measure prediction accuracy

Abbreviations	Formulas
MSE	$= \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2$
RMSE	$= \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}$
MAE	$= \frac{1}{N}\sum_{i=t}^{N}\frac{\lvert x_i - \hat{x}_i\rvert}{\lvert x_i\rvert} =$
R	$= \frac{\sum_{i=1}^{N}(x_i - \bar{x_i})\left(\hat{x_i} - \hat{\bar{x}}_i\right)}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x_i})^2 \sum_{i=1}^{N}\left(\hat{x_i} - \hat{\bar{x}}_i\right)^2}}$
SD	$= \sqrt{\frac{\sum(x_i - \hat{x}_i)^2}{N-1}}$

4.3 Linear Model

First use training set to build the model. We'll use all 30 features' training set X_train and Google stock prices' training set y_train to fit a linear model with all variables which is called linear regression full model, and use X_test and y_test to test the out-of-sample prediction accuracy.

```
# Training a linear regression model on train set
lreg.fit(X_train, y_train)
# Predicting on test set
pred_test = lreg.predict(X_test)
```

The prediction accuracy:

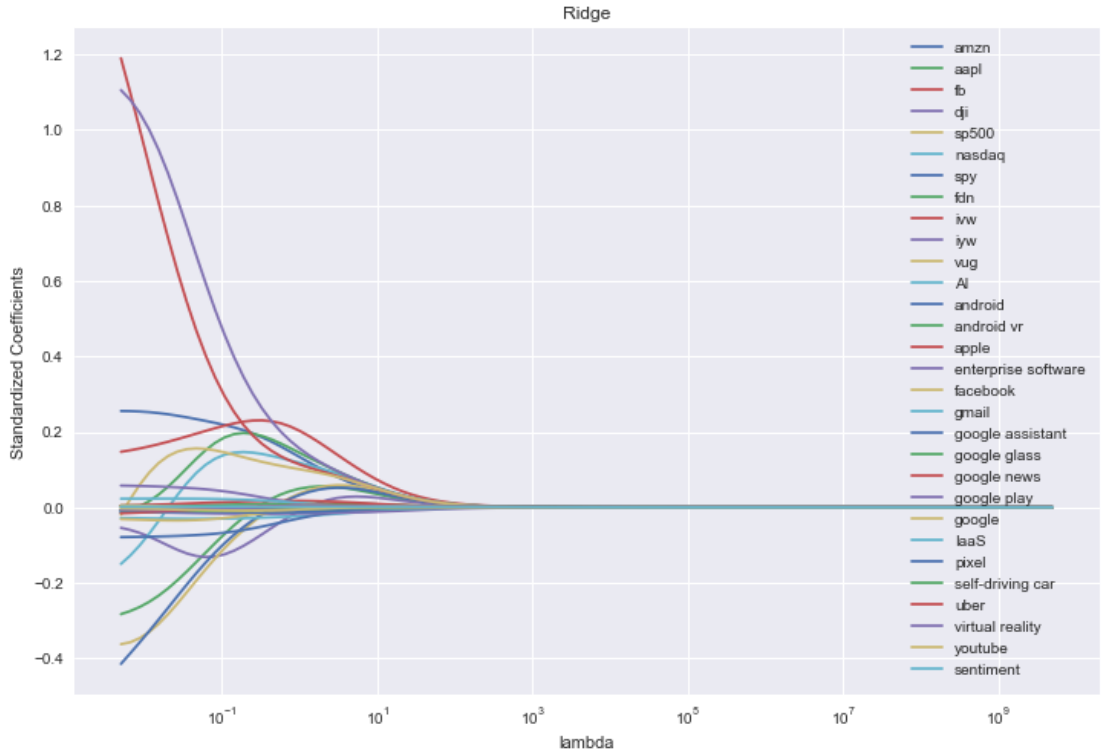
```
RMSE of full linear model is: 0.04826528132866753
R square of full linear model is: 0.770041909638
```

4.4 Feature Selection

4.4.1 Ridge Regression

As mentioned before, the coefficients are shrinking with recursive λ, so we can simply generate an array of alpha values ranging from very big to very small, which could essentially covering the full range of scenarios from the null model containing only the intercept, to the least squares fit. The "sklearn" package in Python can help us.

```
from sklearn.linear_model import Ridge, RidgeCV, Lasso, LassoCV
alphas = 10 ** np.linspace(10, -2, 100) *0.5
```



Refit our ridge regression model on the full data set, using the value of alpha chosen by cross-validation, and examine the coefficient estimates.

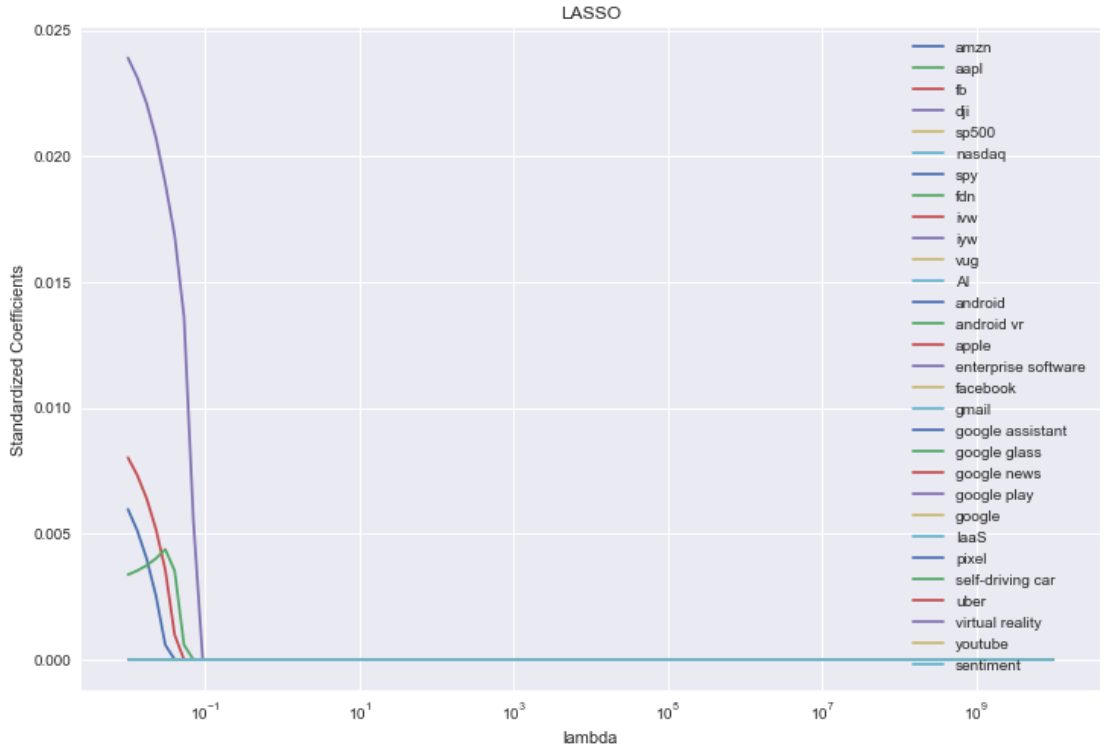
Ridge Coefficients

amzn	0.040769	enterprise software	-0.001223
aapl	-0.234416	facebook	0.010013
fb	0.127627	gmail	0.005918
dji	-0.001071	google assistant	-0.000615
sp500	-0.425293	google glass	-0.012624
nasdaq	-0.307477	google news	0.011073
spy	-0.294720	google play	0.057580
fdn	0.180217	google	0.003562
ivw	1.337393	IaaS	0.022524
iyw	1.065540	pixel	-0.006667
vug	-0.129209	self-driving car	-0.005514
AI	-0.016324	uber	-0.014667
android	-0.090713	virtual reality	-0.003415
android vr	0.015395	youtube	0.004649
apple	-0.001387	sentiment	0.001454

None of the coefficients are exactly zero, ridge regression does not perform variable selection.

4.4.2 LASSO Regression

Similarly, we can look at the coefficients of LASSO model with λ changing and the coefficients as well.



LASSO Coefficients

amzn	0.039479	enterprise software	-0.001274
aapl	-0.231655	facebook	0.000000
fb	0.134779	gmail	0.000190
dji	-0.000000	google assistant	-0.000000
sp500	-0.655656	google glass	-0.009163
nasdaq	-0.093437	google news	0.012755
spy	-0.000000	google play	0.045407
fdn	0.103292	google	0.000000
ivw	1.057171	IaaS	0.021518
iyw	1.014879	pixel	-0.005772
vug	-0.000000	self-driving car	-0.004308
AI	-0.015751	uber	-0.011111
android	-0.072308	virtual reality	0.000000
android vr	0.009756	youtube	0.002932
apple	-0.000820	sentiment	0.001432

The lasso has a substantial advantage over ridge regression in that the resulting coefficient estimates are sparse. Here we see that 7 of the 29 coefficient estimates are exactly zero, so we can remove these features: `'dji'`, `'spy'`, `'vug'`, `'facebook'`, `'google assistant'`, `'google'`, `'virtual reality'`.

V. ARIMAX Model

Here all the series are stationary, so firstly we use ACF and PACF plot do determine the optimal p and q order of ARIMA model.

Because the ACF plot tails off and PACF plot cuts off at lag 1, we say it is an AR(1) model. Now we can add exogenous variables into our AR(1) model and use training set to build the ARIMAX model.

ARMA Model Results

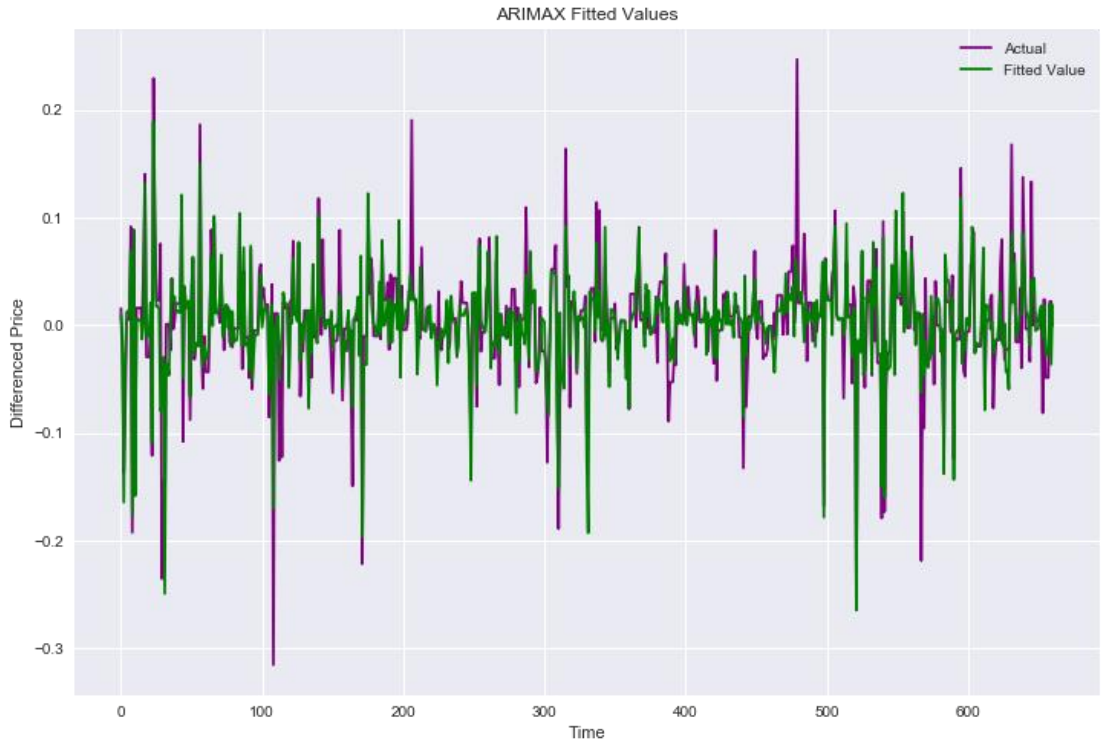
Dep. Variable:	googl	No. Observations:	661
Model:	ARMA(1, 0)	Log Likelihood	1402.686
Method:	css-mle	S.D. of innovations	0.029
Date:	Thu, 07 Jun 2018	AIC	-2753.372
Time:	20:00:58	BIC	-2636.534
Sample:	01-05-2016 - 10-26-2017	HQIC	-2708.088

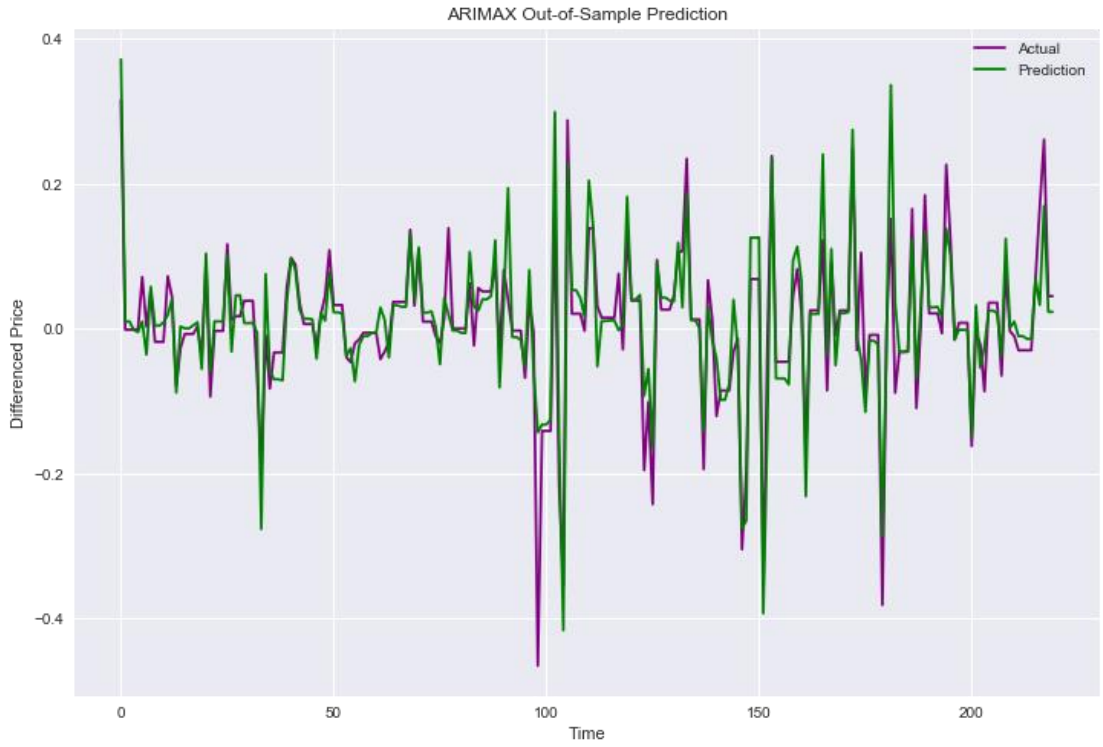
	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.0018	0.001	-1.478	0.140	-0.004	0.001
('amzn',)	0.2659	0.058	4.564	0.000	0.152	0.380
('aapl',)	-0.3204	0.051	-6.279	0.000	-0.420	-0.220
('fb',)	0.1312	0.034	3.830	0.000	0.064	0.198
('sp500',)	-0.8884	0.117	-7.574	0.000	-1.118	-0.659
('nasdaq',)	-0.2366	0.179	-1.324	0.186	-0.587	0.114
('fdn',)	-0.0631	0.108	-0.582	0.561	-0.276	0.150
('ivw',)	1.4111	0.199	7.093	0.000	1.021	1.801
('iyw',)	1.1967	0.156	7.660	0.000	0.891	1.503
('AI',)	-0.0272	0.012	-2.180	0.030	-0.052	-0.003
('android',)	-0.0839	0.037	-2.249	0.025	-0.157	-0.011
('android vr',)	-0.0008	0.013	-0.059	0.953	-0.027	0.026
('apple',)	-0.0062	0.008	-0.746	0.456	-0.023	0.010
('enterprise software',)	-0.0114	0.008	-1.360	0.174	-0.028	0.005
('gmail',)	-0.0127	0.025	-0.501	0.617	-0.062	0.037
('google glass',)	0.0017	0.013	0.132	0.895	-0.024	0.028
('google news',)	0.0031	0.009	0.361	0.718	-0.014	0.020
('google play',)	0.0604	0.026	2.322	0.021	0.009	0.111
('IaaS',)	0.0241	0.013	1.866	0.062	-0.001	0.049
('pixel',)	-0.0063	0.009	-0.698	0.486	-0.024	0.011
('self-driving car',)	-0.0051	0.011	-0.469	0.639	-0.026	0.016
('uber',)	-0.0171	0.016	-1.087	0.277	-0.048	0.014
('youtube',)	-0.0036	0.017	-0.217	0.828	-0.036	0.029
('sentiment',)	0.0017	0.001	2.112	0.035	0.000	0.003
ar.L1.googl	0.0303	0.040	0.757	0.450	-0.048	0.109

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	33.0155	+0.0000j	33.0155	0.0000

To test the ARIMAX model, we apply it to our test set and have a look at the plot of in-sample and out-of-sample prediction:





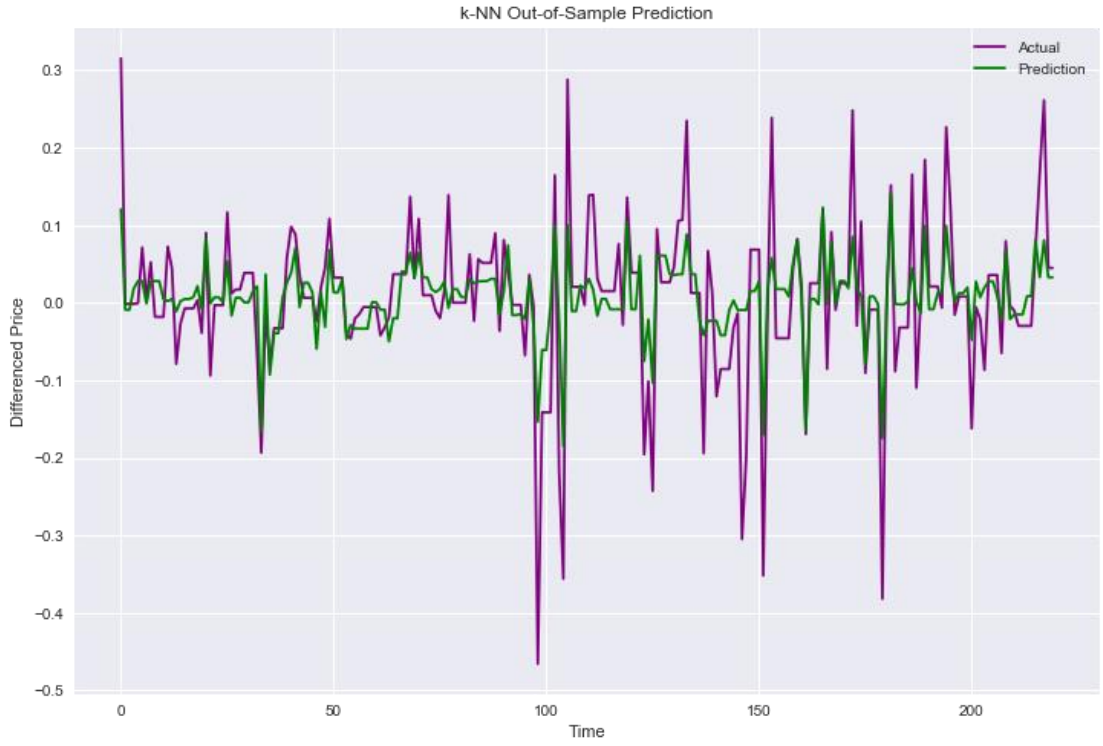
The prediction accuracy is:

RMSE of ARIMAX model (out-of-sample) is: 0.04841033456039497

VI. Machine Learning models

6.1 k-NN Regression

Using “sklearn” package in Python, with $\mathrm{n\_neighbors} = 3$, we get the prediction plot of test set. Here I chose the most common $k = 3$ by experience, in practice k could change and different k can result in different accuracy and different complexity.

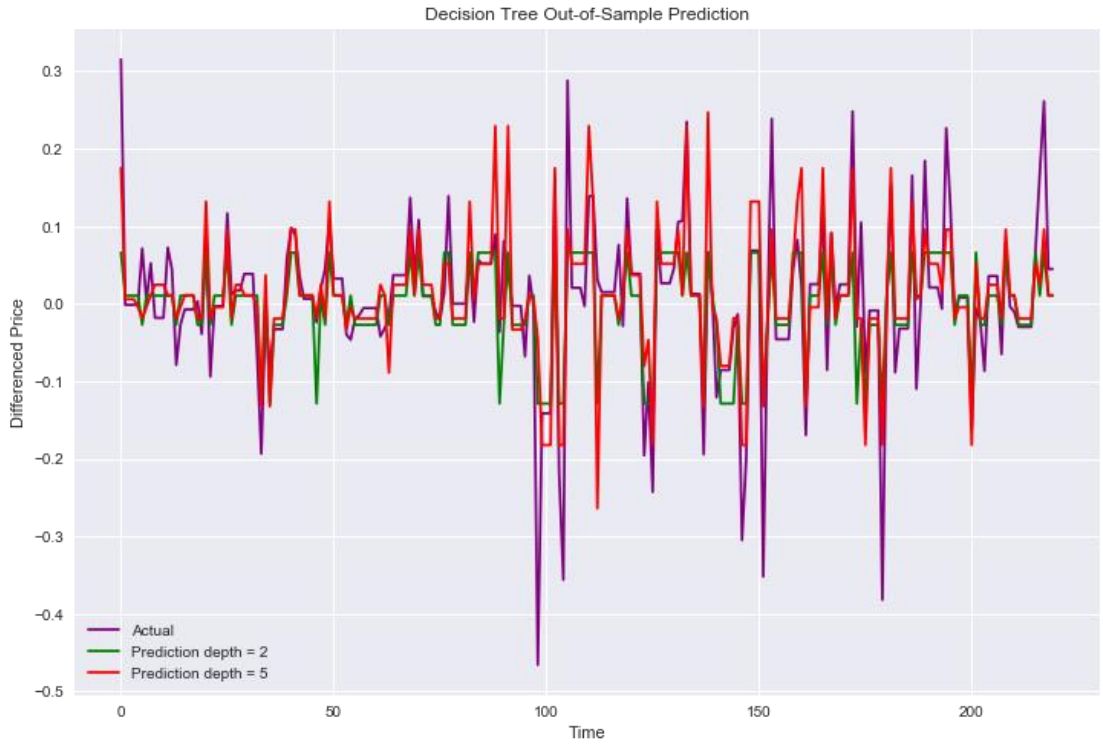


The prediction accuracy is:

RMSE of k-NN model (out-of-sample) is: 0.06870840396735291

6.2 Regression Tree

Here I tried different with different depth and will use depth 2 and 5 to explain how regression tree works and compare their predictive performance. Following is the plot of regression tree with depth is 2 and 5:

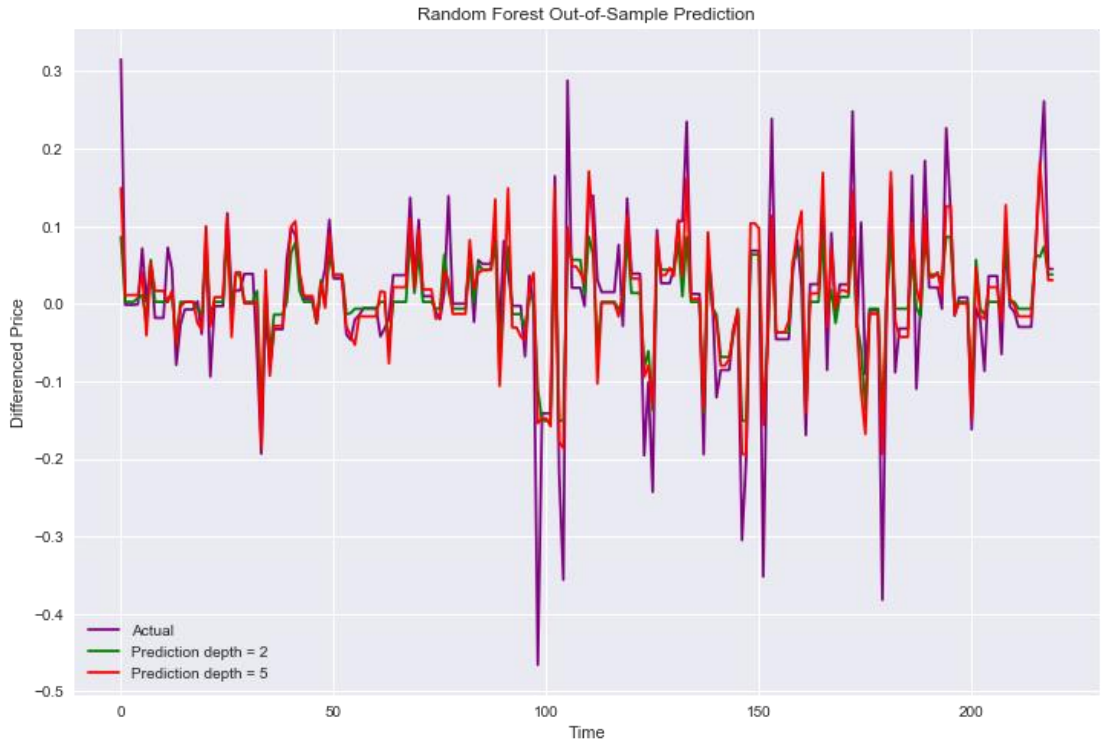


The prediction accuracy is:

```
RMSE of Decision Tree (depth = 2) model (out-of-sample) is: 0.06783243080989874
RMSE of Decision Tree (depth = 5) model (out-of-sample) is: 0.0654532514120387
```

6.3 Random Forest

Consistent with the regression tree, I also used depth is 2 and 5 for random forest model. Here is the plot:



The prediction accuracy is:

RMSE of Random Forest (depth = 2) model (out-of-sample) is: 0.06136990801364945

RMSE of Random Forest (depth = 5) model (out-of-sample) is: 0.05397764805970328

6.4 LSTM's Model

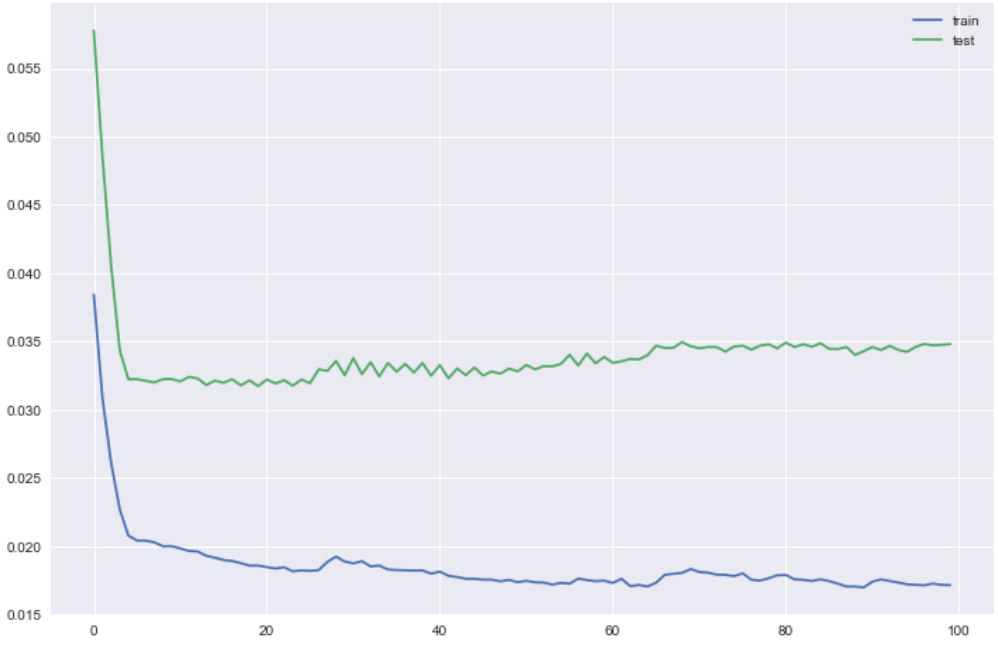
There are a lot of packages in Python can help us deal with LSTM neural networks such as "Keras" and "Tensorflow", they are also the most commonly used packages in deep learning. Here I used "Keras". However, how to prepare data and fit an LSTM for a multivariate time series forecasting is a problem, involving framing the dataset as a supervised learning problem and convert the input variables into 3 dimensions to satisfy the input constraint of LSTM's model. Finally, the inputs features (X) are reshaped into the 3D format expected by LSTMs, namely [samples, timesteps, features]. As we already used lag = 1 in ARIMAX model, we still use timesteps = 1, i.e. one-time-period memory in LSTM's model.

We will frame the supervised learning problem as predicting the Google stock prices at the current day (t) given the features at the prior time step (t-1).

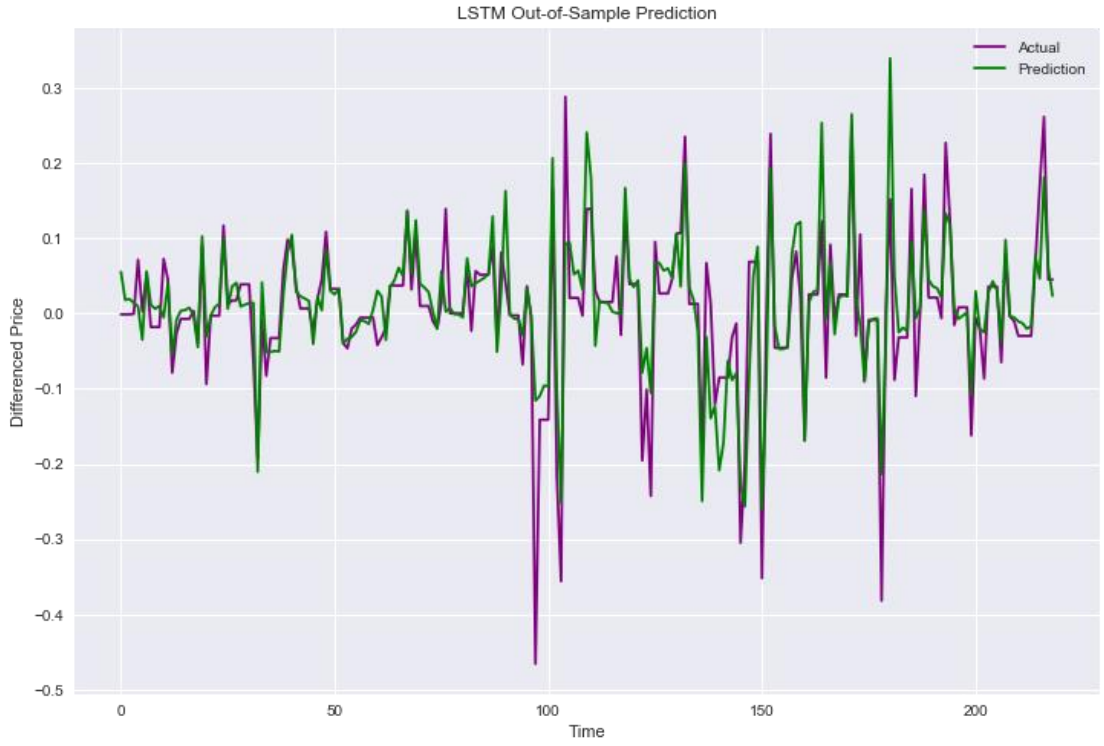
```
   var1(t-1)  var2(t-1)  var3(t-1)  var4(t-1)  var5(t-1)  var6(t-1)  \
1  -0.011005  -0.082061   0.018779   0.016032  -0.012931  -0.012997
2  -0.003920  -0.062479   0.008837  -0.104704  -0.061739  -0.021314
3  -0.084978  -0.132104  -0.185944  -0.186727  -0.162280  -0.138283
4  -0.003061   0.015852  -0.021724  -0.083368  -0.050793  -0.041069
5   0.012254   0.016267   0.002209   0.002164  -0.002085  -0.002079

   var7(t-1)  var8(t-1)  var9(t-1)  var10(t-1)    ...     var15(t)  var16(t)  \
1   0.012023  -0.020080  -0.031682   -0.059126    ...     0.013432 -0.028748
2  -0.068317  -0.050769  -0.031682   -0.059126    ...     0.013432 -0.028748
3  -0.156311  -0.128817  -0.031682   -0.059126    ...     0.013432 -0.028748
4  -0.057934  -0.029931  -0.031682   -0.059126    ...     0.013432 -0.028748
5   0.005465   0.006441  -0.031682   -0.059126    ...     0.013432 -0.028748

   var17(t)  var18(t)  var19(t)  var20(t)  var21(t)  var22(t)  var23(t)  \
1  0.059616  0.058239       0.0  0.071309 -0.031489  0.033933 -0.876708
2  0.059616  0.058239       0.0  0.071309 -0.031489  0.033933 -1.754915
3  0.059616  0.058239       0.0  0.071309 -0.031489  0.033933  4.222672
4  0.059616  0.058239       0.0  0.071309 -0.031489  0.033933 -2.111569
5  0.059616  0.058239       0.0  0.071309 -0.031489  0.033933 -0.462169

   var24(t)
1  0.016417
2  0.136782
3  0.075294
4  0.005373
5  0.005373
```

Then we design the network, I defined the LSTM with 50 neurons in the first hidden layer and 1 neuron in the output layer for predicting prices. The input shape will be 1 time step with 24 features. The model will be fit for 100 training epochs with a batch size of 36.

```
# design network
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1],train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
# fit network
history = model.fit(train_X, train_y, epochs=100, batch_size=36,
validation_data=(test_X, test_y), verbose=2, shuffle=False)
```



The plot of loss in LSTM's model shows both the train and test sample's loss are decreasing along with the iteration, which means the variance of our model is good. Following is the plot:



The prediction accuracy is:

RMSE of LSTM model (out-of-sample) is: 0.055606734492681414

VII. Conclusion and Future Work

The main framework of this paper is as follows: First we came up with some possible features which may help explain the Google stock prices by topic modeling, and extract the stock prices data and ETF prices data, Google Trends indices of these factors, get market sentiment score of Google, combine all these features together as our independent variables that might help predict Google stock prices. Then we process the data including resampling, normalizing and differencing. After this step we get clean relative daily return data in unified unit. Then used LASSO to select features with non-zero coefficients and get the significant features. In the end we applied ARIMAX, k-NN regression, regression tree, random forest and LSTM's neural network models to the time series data and compare their prediction accuracy using RMSE values.

Model	RMSE
ARIMAX Out-of-Sample Prediction	**0.048**
k-NN Out-of-Sample Prediction (k = 3)	**0.079**
Regression Tree Out-of-Sample Prediction (depth = 2)	**0.067**
Regression Tree Out-of-Sample Prediction (depth = 5)	**0.064**
Random Forest Out-of-Sample Prediction (depth = 2)	**0.061**
Random Forest Out-of-Sample Prediction (depth = 5)	**0.054**
LSTMs Out-of-Sample Prediction	**0.055**

As we can see, ARIMAX model has the lowest RMSE, in this case we think it is the best model to fit the Google stock prices. However, we are not claiming that these models help predict the future. Rather we are claiming that these models may help in predicting the present. In each model we are assuming that we already known the values of those features. One thing that I would like to investigate in future work is whether these models are helpful in predicting "turning points" in the data. I.e. predicting the "surprise". A possibly flexible idea is applying classification machine learning methods instead of regression and convert the response variables into binary and such that we can try Logistic Regression and SVM, if we apply more than one machine learning models, a majority voting system could also be built and we can use the voted label as our final prediction.

What's more, we can use some other dimension reduction approaches to do feature selection such as PCA and SVD, but the disadvantage of these methods is it's hard to explain our features after converting them into lower-dimension variables. Granger Causality Test is also worth trying to help select which are good predictors. In machine learning part, especially for k-NN, decision tree and random forest, I didn't try to fund the optimal threshold because I just wanted to compare the predictive ability, but, how to determine the optimal "k" and the depth of forest is also one of the most important works when build a model. Similarly, cross-validation should be introduced to optimize our model by using multiple training and test sets. We can also invert scaling data for actual prediction.

To see full codes of this paper please find my Github page:
https://github.com/WWH98932/Multivariate-Financial-Time-Series-Analysis

References

[1] Al-Hnaity, B., & Abbod, M. (2016). Predicting Financial Time Series Data Using Hybrid Model. *Studies in Computational Intelligence Intelligent Systems and Applications,* 19-41. doi:10.1007/978-3-319-33386-1_2

[2] Ball, P. (2013). Counting Google searches predicts market movements. *Nature*. doi:10.1038/nature.2013.12879

[3] Bulut, L. (2015). Google Trends and Forecasting Performance of Exchange Rate Models. *SSRN Electronic Journal*. doi:10.2139/ssrn.2641796

[4] Choi, H., & Varian, H. (2012). Predicting the Present with Google Trends. *Economic Record, 88*, 2-9. doi:10.1111/j.1475-4932.2012.00809.x

[5] Kordonis, J., Symeonidis, S., & Arampatzis, A. (2016). Stock Price Forecasting via Sentiment Analysis on Twitter. *Proceedings of the 20th Pan-Hellenic Conference on Informatics - PCI 16*. doi:10.1145/3003733.3003787

[6] Liew, J. K., & Mayster, B. (2017). Forecasting ETFs with Machine Learning Algorithms. *SSRN Electronic Journal*. doi:10.2139/ssrn.2899520

[7] Melody, Y., Huang, Randall, R., Rojas & Patrick, D., Convery. (2018). Forecasting Stock Market Movements using Google Trend Searches.

[8] Multivariate Time Series Forecasting with LSTMs in Keras. (2017, October 23). Retrieved from https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/

[9] Preis, T., Moat, H. S., & Stanley, H. E. (2013). Quantifying Trading Behavior in Financial Markets Using Google Trends. *Scientific Reports, 3*(1). doi:10.1038/srep01684

[10] Zhou, Z., Zhao, J., & Xu, K. (2016). Can Online Emotions Predict the Stock Market in China? *Web Information Systems Engineering – WISE 2016 Lecture Notes in Computer Science,* 328-342. doi:10.1007/978-3-319-48740-3_24