Machine Learning Engineer Nanodegree

Capstone Project

Shreyas Matade
January 13st, 2019

I. Definition

Deep Learning Stock Price Predictor

Project intuition

Like most of the tech engineers, I was impressed and surprised by the bitcoin-cryptocurrency boom. Like any other human, I was drawn to FOMO and started investing in crypto. This is where I started my curiosity toward trading bots.
After taking Machine learning nanodegree course, I learned a lot about stats, data analysis and various machine learning models, methodologies.
I really wanted to apply my newly learned skills to work. However, cryptocurrency market is highly volatile and as of now it is becoming unreliable. Hence, just to get the taste of analyzing time series data I thought of exploring "comparatively" more reliable `Stock Market`.

Domain background

Stocks Market Trading

stock market, it's an equity market or share market is the aggregation of buyers and sellers of stocks (also called shares), which represent ownership claims on businesses [Wiki]. Anyone can make profits in stock market by two ways, Trading and Investing. We will concentrate on trading part of it.
Stock markets are getting more and more influenced by the **Algorithms in Trading - Computational Trading**. When algorithmic trading strategies were first introduced, they were wildly profitable and swiftly gained market share. In May 2017, capital market research firm Tabb Group said that high-frequency trading (HFT) accounted for 52% of average daily trading volume. But as competition has increased, profits have declined. In this increasingly difficult environment, traders need a new tool to give them a competitive advantage and increase profits. Algorithms like simple moving average, Exponential moving average are fair to predict stock prices for shorter period in future but these techniques has limitations to predict prices at far time in future. These limitations can be avoided by momentum based algorithms. But more appropriate solution to this problem would be **Machine Learning.**

Machine Learning

Machine learning techniques are much more capable of learning hidden patterns in the historical data and come up with better and profitable strategy. We can use Supervised Machine learning algorithms linear regressions, neural networks, support vector machines, and naive Bayes, to name a few. However, time series prediction problems are a difficult type of predictive modeling problem. This Time series nature of stocks adds complexity of a sequence dependence among the input variables. Such a complexity can be handled by better by applying **Recurrent Neural networks** a.k.a **RNN**. A powerful type of neural network designed to handle sequence dependence is called recurrent neural networks. The Long Short-Term Memory network or LSTM network is a type of recurrent neural network used in deep learning because very large architectures can be successfully trained.

Project outline

In this project, I am planning to apply and explore different versions of LSTM RNN models for time series prediction of target Adjusted Close Price.
While working on this project, I found that a lot of academic work has been done applying RNNs for time series data.
Data and Inputs
For any machine learning problem finding appropriate data is vital, fortunately python library 'fix-yahoo-finance' was able to provide required dataset. I have used data from year 2000 till current date.
This data is already **sorted**, meaning 1st record is of date 2000-01-01 and last record is of current date.

1. Apple(AAPL)
2. Microsoft(MSFT)
3. Amazon(AMZN)
4. IBM(IBM)

This data is logged on daily basis. We will discuss inputs in this data in later part in detail.

Dataset characteristics 1. Each Dataset would have about 4770 records.
2. Our target variable is **"Adj Close Price"**.
3. Our features (input variables) are derived from basic input variables in initial data We will elaborate on that in later part.

Data Prepossessing Since the variables such as 'Open', 'Close', 'Volume' hardly gives any indication on Adj Close price, it is necessary to come up with features which can be good technical indicators in forecasting Adj close price. These feature may or may not help in making model better, this will be explored by running multivariate as well as univariate models.
Data Preparation
Since, we are using LSTM RNN for our forcasting target and the fact that we are dealing with Time Series data I felt data preparation for modeling and forecasting was challenging.Also, one feature of LSTM models, Unlike other machine learning algorithms, long short-term memory recurrent neural networks are capable of automatically learning features from sequence data, support multiple-variate data, and can output a variable length sequences that can be used for multi-step forecasting. That comes in handy.
Predictive Modeling
As discussed, we are trying to predict "Adj Close" price for given stock in future. As we are interested in predicting not only next day data but multiple day data, this makes it more complicated. There are several models we can apply, we are concentrating on RNN model LSTM. Considering the nature of inputs and nature of target we are planning to predict, that makes this problem Multistep Time Series Forecasting problem.

We will apply and use multiple models and see comes best on our metric, RMSE (root mean square error)

We also compare between Univariate Models (where single feature is used) versus Multivariate Models (where multiple features are used) and see added complexity is worth to achieve any better metric.

Problem Statement

```
            For a given a stock ticker, train a predictive model, to predict the close value for next 7 open market da
```

The model shall be trained on historical time series data of a given stock and the model shall be backtested to compare its accuracy over the span of time period. This model then shall be applied different stocks to see its performance.
Considering problem statement we will be interested in training models in **Univariate Multistep** and **Multivariate Multistep Step**.

RMSE Metric

Our forecast will be comprised of 7 days "Adj Close" values, since we are predicting for 7 days in future.It is common with multi-step forecasting problems to evaluate each forecasted time step separately.

This is helpful for a few reasons:
1. To comment on the skill at a specific lead time (e.g. +1 day vs +3 days).
2. To contrast models based on their skills at different lead times (e.g. models good at +1 day vs models good at days +5).

Since our target variable is in USD, our forecasted value should be in USD as well Mean Absolute Error (MSE) and Root Mean Square Error (RMSE) both are applicable here as metric. I choose RMSE since it is more punishing of forecast errors.

The performance metric for this problem will be the RMSE for each lead time from day 1 to day 7.

So, applying this forecast evaluation we can choose model which performs best and given best overall RMSE value over the 7 day prediction.

II. Analysis

Data Exploration

Datasets

I will be using fix-yahoo-finance python package to get the stock data. Below are the variables of the time series data of a single stock. Solution shall be able to get the historical data given a Ticker symbol of the desired stock. I will be using data for following tickers from `START_DATE = '2000-01-01'` till current date.

1. Apple(AAPL)

2. Microsoft(MSFT)
3. Amazon(AMZN)
4. IBM(IBM)

Dataset variables

The datasets we get from fix-yahoo-finance has below columns. This is a time series daily data, **One record for one day** Date is the index for the dataset. Ex: 2000-01-01

1. Open - The opeing price of the stock for given day.
2. High - Maximum price of the stock for given day.
3. Low - Minimum price of the stock for given day.
4. Close - Price of the stock at which day was closed for market.
5. Adj Close - An adjusted closing price is a stock's closing price on any given day of trading that has been amended to include any distributions and corporate actions that occurred at any time before the next day's open. **This will be our Target variable in prediction**
6. Volume - This value is the indicator of total transactions of the stock market buys/sells for given ticker.

I have used 'Adj Close' as my target variable. And for the features i will using techinical indicators which are derived from above mentioned inputs.

Storing Data

We are using **fix-yahoo-finance** library. To avoid frequent query to this library database and downloads I am saving data locally in a csv file. I am using 'ALLOWED_DELTA' constant to see if there is any need to get latest data. if the time delta is greater than this constant then only I pull data from fix-yahoo-finance else I use pre-stored data from csv file.

```
# Max allowed time delta
ALLOWED_DELTA = datetime.timedelta(days=7)
START_DATE = '2000-01-01'
```

All csv file are stored in `data` folder

Impute Data

Since we are dealing with the time series data, we cannot just replace 'nan' or missing values by over all mean, since data is sequence dependent and overall mean will not appropriate value to replace any missing values in data. There are two methods provided by `pandas` to resolve this problem. We need to be careful in using these techniques though. First we have to apply 'Fill Forward' and then apply 'Fill Backwards' in order to avoid any `Look ahead` issues.

Hence, I am using `fill forward` and then `fill backward` method to replace the `nan` in the data.

Need for Feature Exploration

Finding a better feature for Stock Market prediction is a topic for more exploration and research. I found more I explore more different indicators I find. It is only when I apply them in model, I would get better idea about whether they are helping the model learn or making it worst.

I am considering to use 5 features, shifted version **Adj Close** (Shift depends on the `lookback` factor) and 3 technical indicators and one functional indicator.

- MACD (Trend)
- MOMETM (Momentum)
- Average True Range (Volume)
- Days since market open (Functional Indicator)

Details on Feature Calculations

Exponential Moving Average: An exponential moving average (EMA) is a type of moving average (MA) that places a greater weight and significance on the most recent data points.

```
Initial SMA: 10-period sum / 10
Multiplier: (2 / (Time periods + 1) ) = (2 / (10 + 1) ) = 0.1818 (18.18%)
EMA: {Close - EMA(previous day)} x multiplier + EMA(previous day).
```

https://stockcharts.com/school/doku.php?id=chart*school:technical*indicators:moving_averages

MACD: The Moving Average Convergence Divergence (MACD) is a trend-following momentum indicator that shows the relationship between two moving averages of a security's price. The MACD is calculated by subtracting the 26-period Exponential Moving Average (EMA) from the 12-period EMA.

Stochastics oscillator: The stochastic oscillator is a momentum indicator comparing the closing price of a security to the range of its prices over a certain period of time. The sensitivity of the oscillator to market movements is reducible by adjusting that time period or by taking a moving average of the result.

Average True Range: Is an indicator to measure the volatility (NOT price direction). The largest of: - Method A: Current High less the current Low - Method B: Current High less the previous Close (absolute value) - Method C: Current Low less the previous Close (absolute value)

Calculation:

```
Current ATR = [(Prior ATR x 13) + Current TR] / 14

  - Multiply the previous 14-day ATR by 13.
  - Add the most recent day's TR value.
  - Divide the total by 14
```

DSLO : It is the days since market is open. This might affect the prices if market was closed for multiple days. My gut feeling tells me this might come handy in predicting target.

```
We calculate this indictor by subtracting consucutive timestamp and take the day difference.
```

```
def DaySinceLastOpen(df):
"""This function assumes that timeseries df indexed by date is passed """
DSLO = [] DSLO.append(1.0)
for i in range(df.shape[0]-1):
diff = pd.to_datetime(df.index[i+1]) - pd.to_datetime(df.index[i])
DSLO.append(float(diff.days))
return pd.Series(DSLO, index=df.index)
```

Since `dslo` is a categorical variable we need to apply one hot encoding on this variable before we use this as a feature. We use one-hot encoding to convert this categorical variable.

Sample Data and Stats

Since we have 4 different datasets for each ticker but the basic info about the data would be pretty simillar across these four dataset. So just to get sense of data let's see IBM sample data.

IBM - Sample Dataset with derived features
Below image shows the snapshot dataset we made after feature exploration.

```
In [140]: final_data.head(10)
```

Out[140]:

Date	Adj Close	MACD	MOMETM	ATR	dslo_0	dslo_1	dslo_2	dslo_3	dslo_4	dslo_5	dslo_6	dslo_7
2000-01-21	62.0625	-0.247378	0.000000	4.7500	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2000-01-24	70.1250	0.124545	40.566038	11.3125	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
2000-01-25	69.2500	0.332524	89.147287	5.2500	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2000-01-26	64.8125	0.229188	34.108527	5.2500	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2000-01-27	66.9375	0.279880	60.465116	3.1250	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2000-01-28	61.6875	0.016640	0.000000	6.9375	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2000-01-31	64.5625	0.012401	34.074074	6.3125	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
2000-02-01	67.4375	0.187916	68.148148	6.2500	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2000-02-02	69.4375	0.431235	91.851852	4.8125	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2000-02-03	84.1875	1.485749	100.000000	16.5000	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0

Below image shows datatype and basic information about our dataset.

```
: final_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 4775 entries, 2000-01-21 to 2019-01-14
Data columns (total 12 columns):
Adj Close    4775 non-null float64
MACD         4775 non-null float64
MOMETM       4775 non-null float64
ATR          4775 non-null float64
dslo_0       4775 non-null float32
dslo_1       4775 non-null float32
dslo_2       4775 non-null float32
dslo_3       4775 non-null float32
dslo_4       4775 non-null float32
dslo_5       4775 non-null float32
dslo_6       4775 non-null float32
dslo_7       4775 non-null float32
dtypes: float32(8), float64(4)
memory usage: 335.7 KB
```

Basic Statistics

From below stats, it is very clear that feature 'dslo*6' and 'dslo*0' are not really required in the analysis as its value is 0 for all records. So, we drop these two columns from our final data and save the data in **"[ticker]_processed.csv"** . This is our final prepossessed time series data. This data is further prepared according to model impementation and forecast evaluation's requirement. We will discuss **Data Preparation** in later sections.

Basic Statistics

```
In [22]: ibm_data.describe()
```

Out[22]:

	Adj Close	MACD	MOMETM	ATR	dslo_0	dslo_1	dslo_2	dslo_3	dslo_4	dslo_5	dslo_6	dslo_7
count	4774.000000	4774.000000	4774.000000	4774.000000	4774.0	4774.000000	4774.000000	4774.000000	4774.000000	4774.000000	4774.0	4774.000000
mean	103.327803	-0.003463	53.607749	2.441992	0.0	0.782782	0.009845	0.181190	0.025555	0.000419	0.0	0.000209
std	38.571346	0.798667	36.804881	1.582888	0.0	0.412396	0.098743	0.385216	0.157820	0.020466	0.0	0.014473
min	38.591488	-3.882847	0.000000	0.360001	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000
25%	66.731832	-0.480945	17.709823	1.429995	0.0	1.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000
50%	92.518040	0.019750	57.506852	2.040001	0.0	1.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000
75%	142.759674	0.467855	90.105145	2.970001	0.0	1.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000
max	175.264389	2.698121	100.000000	22.750000	0.0	1.000000	1.000000	1.000000	1.000000	1.000000	0.0	1.000000

```
In [23]: aapl_data.describe()
```

Out[23]:

	Adj Close	MACD	MOMETM	ATR	dslo_0	dslo_1	dslo_2	dslo_3	dslo_4	dslo_5	dslo_6	dslo_7
count	4774.000000	4774.000000	4774.000000	4774.000000	4774.0	4774.000000	4774.000000	4774.000000	4774.000000	4774.000000	4774.0	4774.000000
mean	45.514683	-0.015970	58.120048	1.125256	0.0	0.782782	0.009845	0.181190	0.025555	0.000419	0.0	0.000209
std	55.174150	0.604943	36.825707	1.435248	0.0	0.412396	0.098743	0.385216	0.157820	0.020466	0.0	0.014473
min	0.627570	-4.903227	0.000000	0.013571	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000
25%	2.989566	-0.107592	22.756815	0.165714	0.0	1.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000
50%	17.662357	0.002236	65.717552	0.692143	0.0	1.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000
75%	69.123907	0.114137	95.694468	1.521427	0.0	1.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000
max	231.263092	3.484961	100.000000	16.800003	0.0	1.000000	1.000000	1.000000	1.000000	1.000000	0.0	1.000000

```
In [24]: msft_data.describe()
```

Out[24]:

	Adj Close	MACD	MOMETM	ATR	dslo_0	dslo_1	dslo_2	dslo_3	dslo_4	dslo_5	dslo_6	dslo_7
count	4774.000000	4774.000000	4774.000000	4774.000000	4774.0	4774.000000	4774.000000	4774.000000	4774.000000	4774.000000	4774.0	4774.000000
mean	32.247452	-0.002092	54.442580	0.853218	0.0	0.782782	0.009845	0.181190	0.025555	0.000419	0.0	0.000209
std	21.344378	0.283997	36.603577	0.719238	0.0	0.412396	0.098743	0.385216	0.157820	0.020466	0.0	0.014473
min	11.904654	-1.680255	0.000000	0.129999	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000
25%	20.036353	-0.147886	19.752792	0.430001	0.0	1.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000
50%	22.885535	0.005152	58.511028	0.649999	0.0	1.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000
75%	36.259312	0.139960	90.904893	1.000000	0.0	1.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000
max	115.112709	1.187403	100.000000	8.125000	0.0	1.000000	1.000000	1.000000	1.000000	1.000000	0.0	1.000000

```
In [25]: amzn_data.describe()
```

Out[25]:

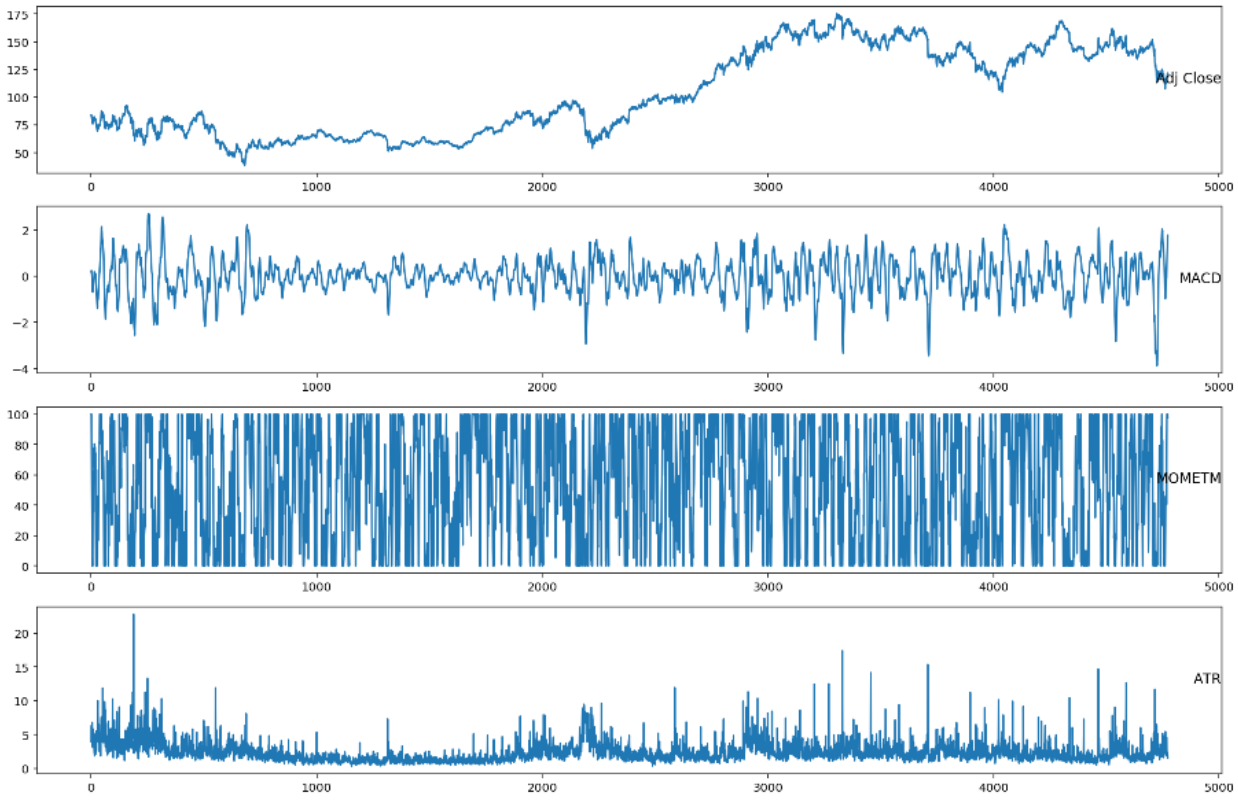
	Adj Close	MACD	MOMETM	ATR	dslo_0	dslo_1	dslo_2	dslo_3	dslo_4	dslo_5	dslo_6	dslo_7
count	4774.000000	4774.000000	4774.000000	4774.000000	4774.0	4774.000000	4774.000000	4774.000000	4774.000000	4774.000000	4774.0	4774.000000
mean	288.527210	-0.044906	55.564040	7.488032	0.0	0.782782	0.009845	0.181190	0.025555	0.000419	0.0	0.000209
std	414.141961	4.580064	36.928089	13.332007	0.0	0.412396	0.098743	0.385216	0.157820	0.020466	0.0	0.014473
min	5.970000	-41.803910	0.000000	0.220000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000
25%	39.475001	-0.894270	19.414347	1.390003	0.0	1.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000
50%	89.924999	0.031365	60.908475	3.640000	0.0	1.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000
75%	323.615005	0.868185	92.539284	7.579998	0.0	1.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000
max	2039.510010	34.860972	100.000000	179.170044	0.0	1.000000	1.000000	1.000000	1.000000	1.000000	0.0	1.000000

Exploratory Visualization

We plot features 'ATR', 'MACD' and 'MOMETM' along with Target variable 'Adj Close'. We avoid functional indicator 'DSLO' since it will be difficult to see its affect on target variable by visualization, we rely on predictive models for that.

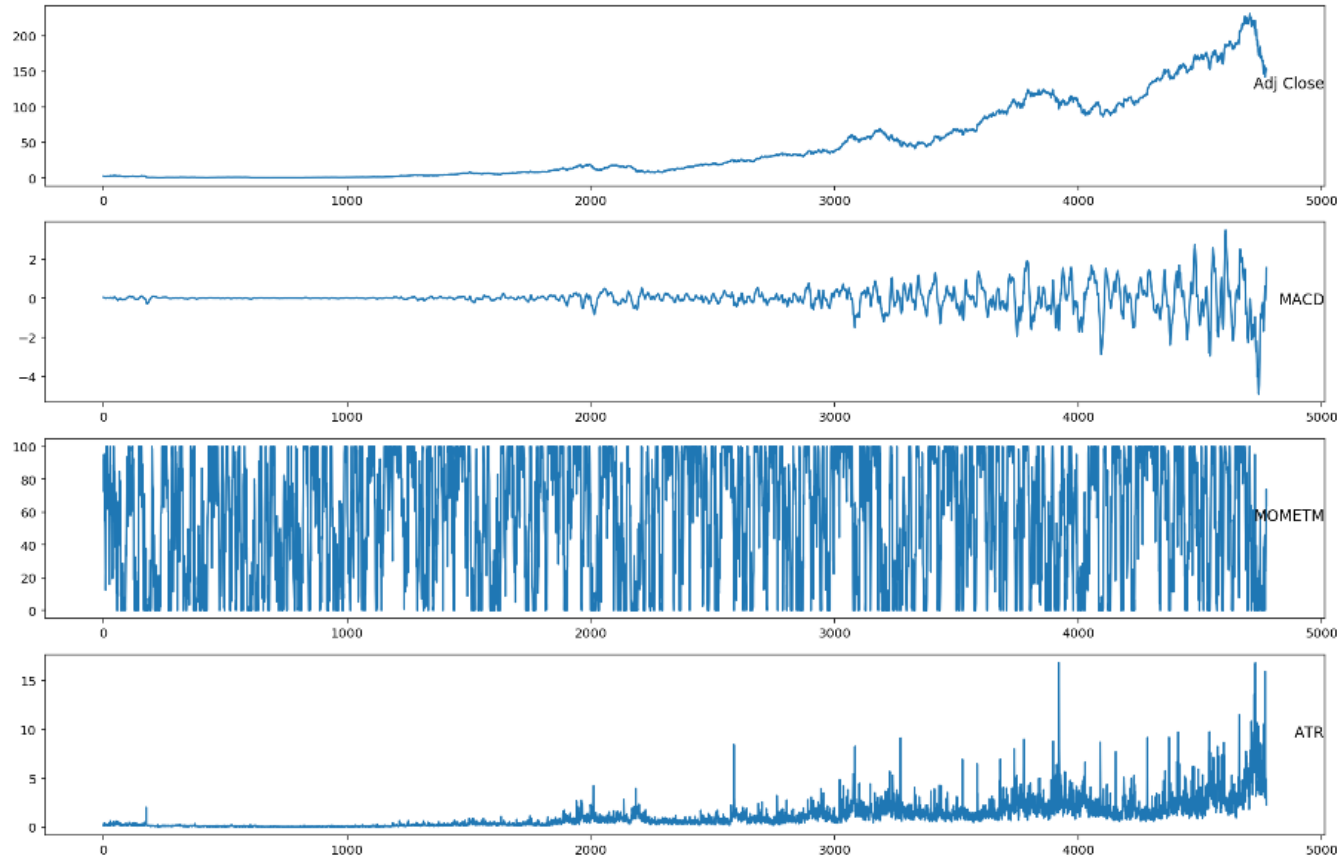
For IBM

In [49]:
```
explore_features(ibm_data.drop(['dslo_0','dslo_1','dslo_2','dslo_3','dslo_1','dslo_5','dslo_6','dslo_7'],axis=1),tickers[0])
```



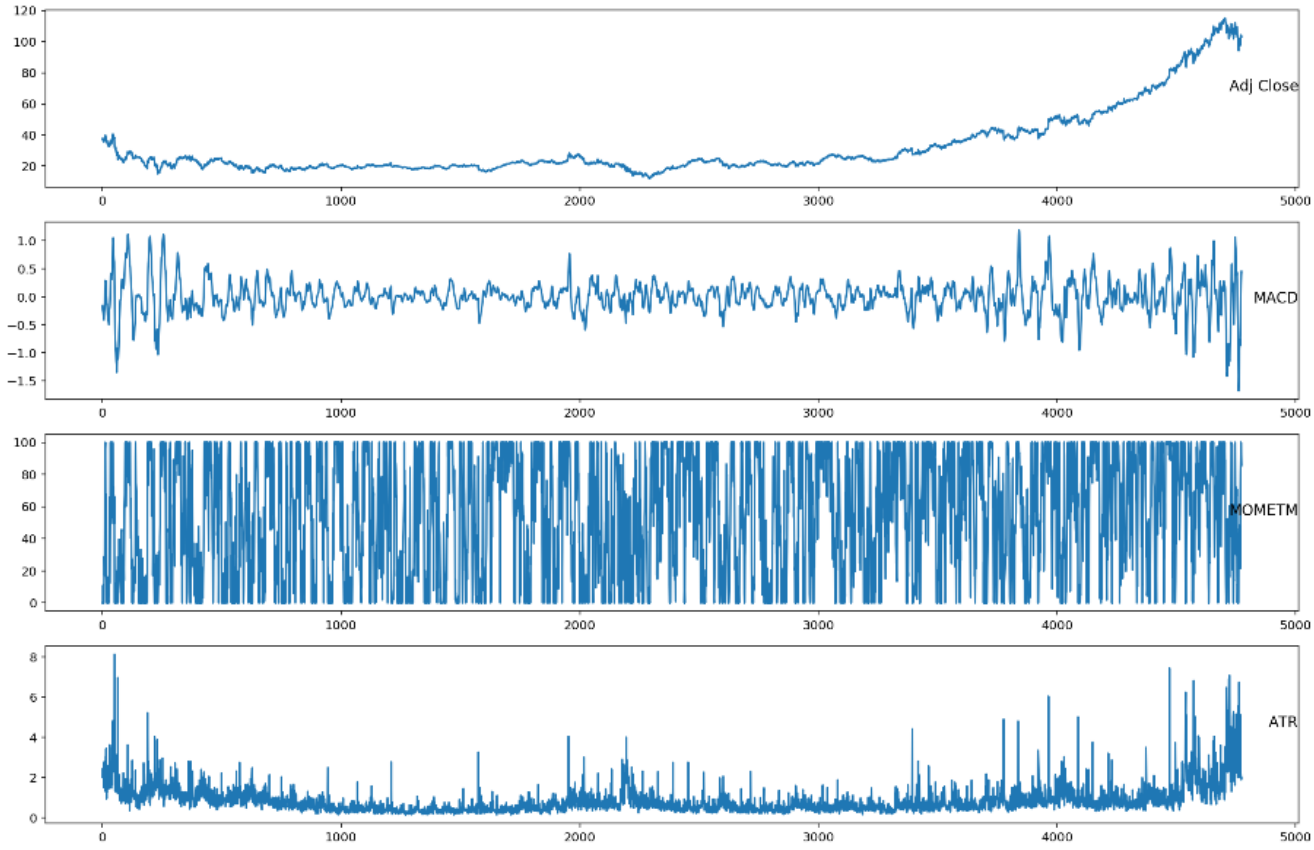
For AAPL

```
explore_features(aapl_data.drop(['dslo_0','dslo_1','dslo_2','dslo_3','dslo_1','dslo_5','dslo_6','dslo_7'],axis=1),tickers[])
```



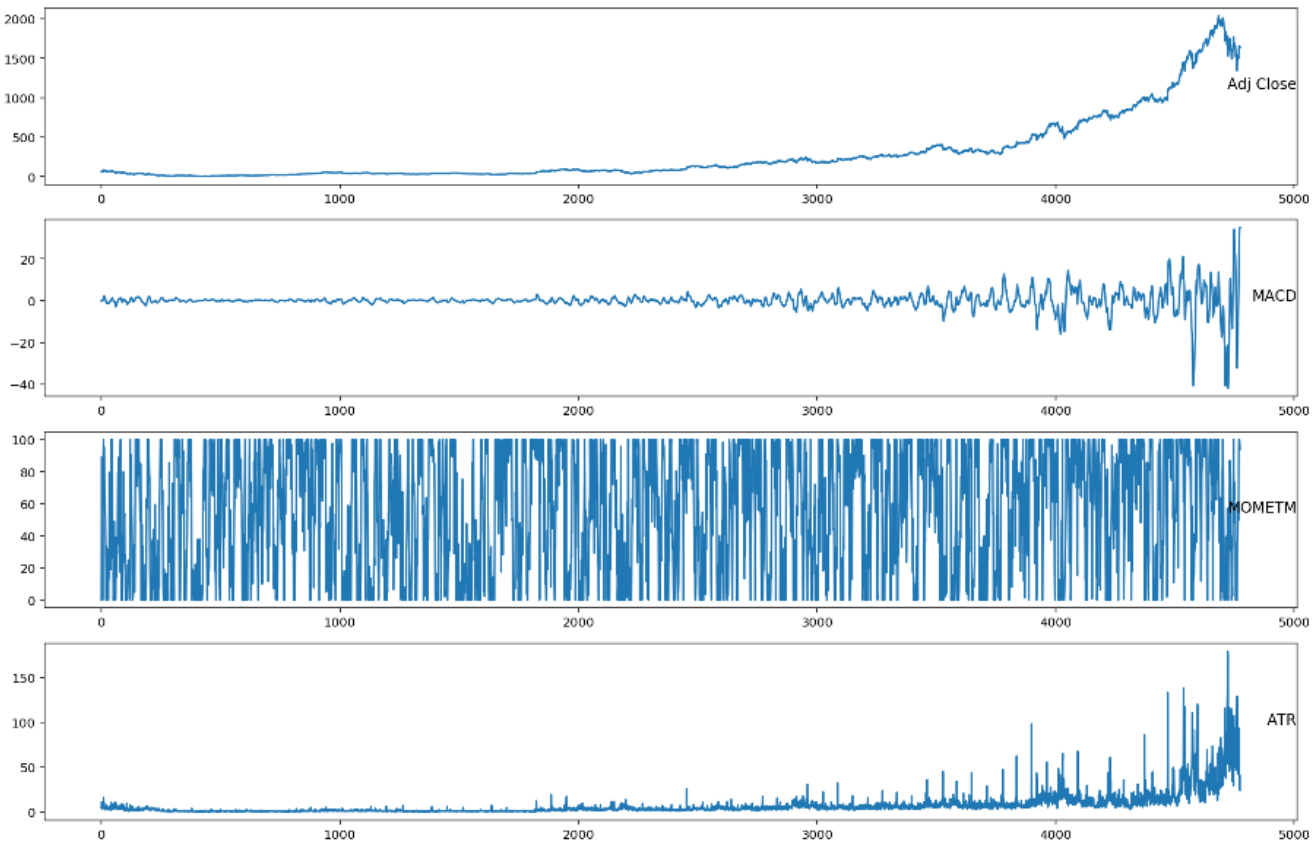
For MSFT

```
]: explore_features(msft_data.drop(['dslo_0','dslo_1','dslo_2','dslo_3','dslo_1','dslo_5','dslo_6','dslo_7'],axis=1),tickers[2])
```



For AMZN

```
]: explore_features(amzn_data.drop(['dslo_0','dslo_1','dslo_2','dslo_3','dslo_1','dslo_5','dslo_6','dslo_7'],axis=1),tickers[3])
```



By observing above visualization, we can see there is obvious relation between ATR, MACD and Target variable, Adj Close. When there

is movement in target variable there is corresponding fluctuation in ATR and MACD. It will be interesting to see how it helps the model for better prediction. MOMETM seems highly fluctuating for all values of target. We will later explore if really need this variable.

Algorithms and Techniques

We have already discussed feature exploration and how we calculated different technical indicators and prepared our dataset. These techniques were in detail discussed above.

Persistance Model Algorithm
As name suggest persistance alorithm is easy and simple, to predict value at timestamp `t` it simple repeats value at timestamp `t-1`. This is a forecasting model where the last observation is persisted forward. Because of its simplicity, it is often called the naive forecast.
Used in Project because
I have Persistence Model as my benchmark model, since it is easy,steady and reliable.

Long Short Temporal Memory Algorithm
The Long Short-Term Memory network, or LSTM network, is a recurrent neural network that is trained using Backpropagation Through Time and overcomes the vanishing gradient problem.

As such, it can be used to create large recurrent networks that in turn can be used to address difficult sequence problems in machine learning and achieve state-of-the-art results. Instead of neurons, LSTM networks have memory blocks that are connected through layers.

A block has components that make it smarter than a classical neuron and a memory for recent sequences. A block contains gates that manage the block's state and output. A block operates upon an input sequence and each gate within a block uses the sigmoid activation units to control whether they are triggered or not, making the change of state and addition of information flowing through the block conditional.

There are three types of gates within a unit:

```
Forget Gate: conditionally decides what information to throw away from the block.
Input Gate: conditionally decides which values from the input to update the memory state.
Output Gate: conditionally decides what to output based on input and the memory of the block.
```

Each unit is like a mini-state machine where the gates of the units have weights that are learned during the training procedure.

Since, we are dealing with time series data where each data point has a temporal characteristic attached to it, LSTM network is one of the best model for forecasting temporal data.

Used in project because
1. We are forecasting time series data, which is temporal in nature. LSTM network is better fit since it has memory element which is missing in other state-less supervised models and Neural nets.
2. Unlike other machine learning algorithms, long short-term memory recurrent neural networks are capable of automatically learning features from sequence data, support multiple-variate data, and can output a variable length sequences that can be used for multi-step forecasting.
3. Multivariate Inputs: LSTMs directly support multiple parallel input sequences for multivariate inputs, unlike other models where multivariate inputs are presented in a flat structure.
4. Vector Output. Like other neural networks, LSTMs are able to map input data directly to an output vector that may represent multiple output time steps.

Benchmark - Persistance Model

A baseline in forecast performance provides a point of comparison. It is a point of reference for all other modeling techniques on your problem. If a model achieves performance at or below the baseline, the technique should be fixed or abandoned. I am using Persistance model as my baseline model. I have chosen this as my baseline model as it has following three characteristics
Simple: A method that requires little or no training or intelligence.
Fast: A method that is fast to implement and computationally trivial to make a prediction.
Repeatable: A method that is deterministic, meaning that it produces an expected output given the same input.

Data Preparation

Need for Re-framing Time Series data into Supervised Leanring data
A time series is a sequence of numbers that are ordered by a time index. This can be thought of as a list or column of ordered values. On the other hand, A supervised learning problem is comprised of input patterns (X) and output patterns (y), such that an algorithm can learn how to predict the output patterns from the input patterns. So for any time series forecasting problem it is necessary to convert time series

data into supervised learning style data where our Target Variable(y), future Adj close price at 't' can be predicted from past value at 't-1', in this case we are using lookback = 1.

Converting Time Series Data to Supervised Learning data
We are making use of pandas 'shift()' function to achieve supervised dataset. Default parameters will construct a DataFrame with t-1 as X and t as y.

The function, `SeriesToSupervised` in our analysis takes four arguments:

1. **data:** Sequence of observations as a list or 2D NumPy array. Required.
2. **n_in:** Number of lag observations as input (X). Values may be between [1..len(data)] Optional. Defaults to 1.
3. **n_out:** Number of observations as output (y). Values may be between [0..len(data)-1]. Optional. Defaults to 1.
4. **dropnan:** Boolean whether or not to drop rows with NaN values. Optional. Defaults to True.

The function returns a single value:

1. **return:** Pandas DataFrame of series framed for supervised learning.

Model Prediction

Function named `persistence`, that takes the last observation and the number of forecast steps to persist. This function returns an array containing the forecast. We can then call this function for each time step in the `test` dataset. function make_forecasts() that does this and takes the train, test, and configuration for the dataset as arguments and returns a list of forecasts.

Evaluate Forecasts

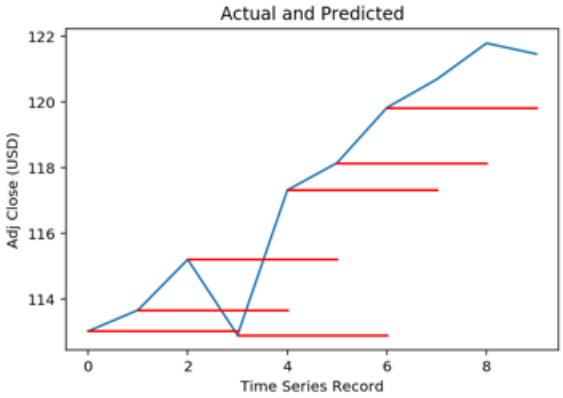
Finally, we evaluate our forecast

We can do that by calculating the RMSE for each time step of the multi-step forecast. This is done in evaluate*forecasts() function, where we calculate RMSE by means of predicted and actual value. Predictions are also plotted using plot*forecasts() function.

Following results were obtained when we ran persistence model, to predict target variable for next 7 days.

BenchMark Results

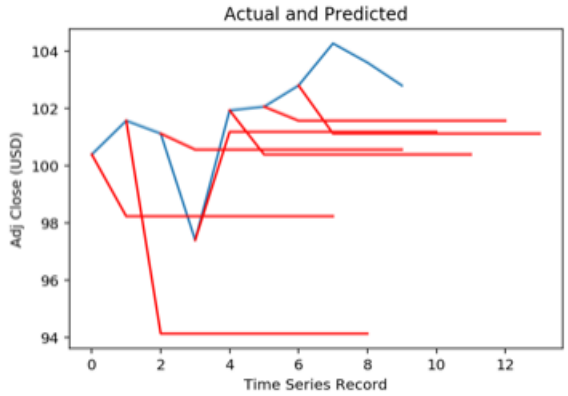
For IBM Overall RMSE: 4.307973




```
t+1 RMSE: 2.399258
t+2 RMSE: 2.705824
t+3 RMSE: 3.096104
t+4 RMSE: 4.083876
t+5 RMSE: 4.997928
t+6 RMSE: 5.853293
t+7 RMSE: 7.019526
Overall RMSE: 4.307973
```

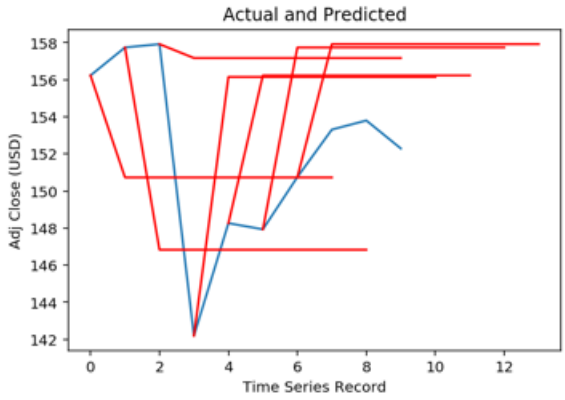
For Microsoft Overall RMSE: 3.178238

```
t+1 RMSE: 3.264457
t+2 RMSE: 3.248963
t+3 RMSE: 2.899928
t+4 RMSE: 3.381786
t+5 RMSE: 3.480314
t+6 RMSE: 2.418524
t+7 RMSE: 3.553691
Overall RMSE: 3.178238
```



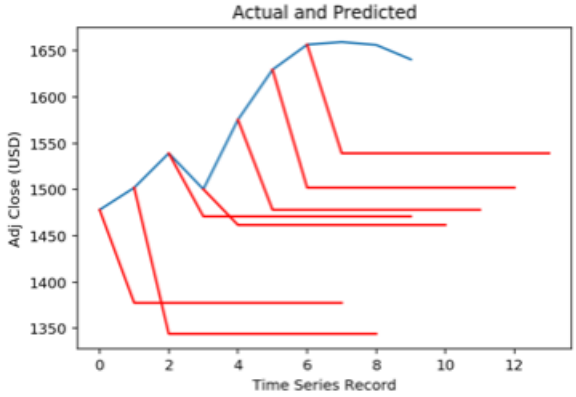
For Apple Overall RMSE: 7.783148

```
t+1 RMSE: 7.298900
t+2 RMSE: 8.191651
t+3 RMSE: 8.518772
t+4 RMSE: 8.911374
t+5 RMSE: 9.241080
t+6 RMSE: 6.401026
t+7 RMSE: 5.919230
Overall RMSE: 7.783148
```



For Amazon Overall RMSE: 109.06394

```
t+1 RMSE: 54.841590
t+2 RMSE: 64.709003
t+3 RMSE: 81.138913
t+4 RMSE: 107.418907
t+5 RMSE: 135.605513
t+6 RMSE: 151.213724
t+7 RMSE: 168.519983
Overall RMSE: 109.063948
```



Above results we have gotten in our benchmark and shall be compared with our other models to decide on final model.

III. Methodology

To explain the methodology that I followed for various LSTM model, we start from base LSTM model. Let's see **Univariate-Multistep-Base-LSTM Model** and go through all the pipeline from data preparation to forecast evaluation. This is basic methodology I followed I improved models on top of this model. Initial Steps, such as Data preparation and evaluation metric are by far the same for all other improved models with minor tweaks.

Data Preparation

This was most challenging step when using LSTM network for time series prediction. Especially, deciding the shape of data for training and then using it to evaluate the forecast was complicated. Fortunalty, explaination provided in Jason Brownlee's Blog helped.

Splitting Data in Train/Test

As explained in Data Preprocessing section, we are using time series data of stock market which is logged on daily basis. We are using data from year 2000 to current date. So, that gives us around **4700 records for single ticker dataset**. I have decided to split data in **70/30 ~ Train/Test** percent, i.e. Training Data on 70% records of total records and test on remaing 30% However, since it is a time series data we cannot split randomly. so we train on initial 70% of data and we test on later 30% data, preserving temporal nature of the data.

Since we are predicting 7 days in future, it make sence to group data in 7 open market days. These days are not necessary any specific day of the week we are just interested in grouping records in chunks of 7 days so that it will be easer more modeling and forcasting.

In order to achieve that we need records divisible by 7, so we get latest number of records divisible by 7. Then we move forward to split data for train and test

Train Data : This is obtained by starting from beginning of the data to the record which is divisible by 7 and nearest to 70% of the total data Test Data : This is obtained by getting last number of records which are divisible by 7 and nearest to 30% percent.

This is all implemented in split_dataset() function.

For Example: Suppose we have dataset, in shape, (4777,10)
Total Data : 4777 / 7 = 682.4 ==> 7* 682 ==> 4774, we take latest 4774 records, now shape is (4774,10)
Train Data : 4774 * 70 / 100 ==>3341 / 7==>477.28 ==> 477 * 7 ==> 3339, so we take 0 to 3339 records for training,(3339,10)
Test Data : 4774-3339 ==> 1435 , so we take last 1435 records for testing, (1435,10)

And we group records by 7 open days so , **Train Data shape (477,7,10) and Test data shape becomes (205,7,10) **

Converting Time Series to Supervised Learning Data

As per LSTM model requirement to have data in the shape: [samples, timesteps, features]

Our split_dataset() function will provide us data in that shape.

In order to create a lot more training data, we are to change the problem during training to predict the next seven days given the prior seven days, regardless of the standard week.

This only impacts the training data, and the test problem remains the same: predict the daily power consumption for the next standard week given the prior standard week.

This will require a little preparation of the training data.

The training data is provided in 7 day window with 10 features, specifically in the shape [477, 7, 10]. The first step is to flatten the data so that we have 10 time series sequences.

We then need to iterate over the time steps and divide the data into overlapping windows; each iteration moves along one time step and predicts the subsequent next window of 7 days.

For example:
Input, Output
[d01, d02, d03, d04, d05, d06, d07], [d08, d09, d10, d11, d12, d13, d14]
[d02, d03, d04, d05, d06, d07, d08], [d09, d10, d11, d12, d13, d14, d15]
and so on...

We can do this by keeping track of start and end indexes for the inputs and outputs as we iterate across the length of the flattened data in terms of time steps.

We can also do this in a way where the number of inputs and outputs are parameterized (e.g. n *input, n*out) so that you can experiment with different values or adapt it for your own problem.

function named to_supervised() implements discussed technique.

Base LSTM Model Implementation

This multi-step time series forecasting problem is an autoregression. That means it is likely best modeled where that the next prediction window (7 days) is some function of observations at prior time steps.

LSTM Network :

```
1. Hidden LSTM layer            : Single layer with 200 units. The number of units in the hidden layer is unrelated to
2. Fully connected Dense layer  : 200 nodes that will interpret the features learned by the LSTM layer.
3. Output layer                 : This will directly predict a vector with seven elements
```

Hyper Parameters : 1. Loss function : Mean Square Error (MSE), as it goes well with our metric RMSE 2. Optimiser : adam, stochastic gradient descent make sense 3. Epochs : 70 4. Batch size : 16

Refinement

To improve on forecasting we ran different variants of LSTM,

1. Univariate-Multistep-EncoderDecoder-LSTM
2. Multivariate-Multistep-EncoderDecoder-LSTM
3. Univariate-Multistep-CNN-EncoderDecoder-LSTM

Let's see them one by one and see how and what changes we made to our base LSTM model.

Univariate Multistep Encoder Decoder LSTM

Data Preparation

Data Preparation steps Spliting Data in Train/Test Split and Converting Time Series to Supervised Learning Data remains the same as in previous model

Base Encoder Decoder LSTM Model Implementation

This is slight improvement over base LSTM model, here model will not output a vector sequence directly. Instead, the model will be comprised of two sub models, the encoder to read and encode the input sequence, and the decoder that will read the encoded input sequence and make a one-step prediction for each element in the output sequence. The Major difference comes in the decoder, allowing it to both know what was predicted for the prior day in the sequence and accumulate internal state while outputting the sequence.

Here we are using encoder-decoder arch.

Network :

1.Hidden LSTM layer
Single layer with 200 units. with relu activation.
2. RepeatVector layer:
To repeat the internal representation of the input sequence once for each time step in the output sequence. This sequence of vectors will be presented to the LSTM decoder.
3. Define decoder:
Define the decoder as an LSTM hidden layer with 200 units.
4. TimeDistrubuted Wrapper:
Wrap the interpretation layer and the output layer in a TimeDistributed wrapper that allows the wrapped layers to be used for each time step from the decoder.

Hyper Parameters :
1. Loss function : Mean Square Error (MSE), as it goes well with our metric RMSE 2. Optimiser : adam, stochastic gradient descent make sense 3. Epochs : 70 4. Batch size : 16

The network therefore outputs a three-dimensional vector with the same structure as the input, with the dimensions [samples, timesteps, features].

Multistep Multistep Encoder Decoder LSTM

This model is similar to univariate encoder decoder LSTM. As name 'multivariate' suggests, we will be using all 10 features from our processed dataset to predict the 7 day prediction on Adj Close Price.

Data Preparation

Data Preparation step Spliting Data in Train/Test Split remains same as in base LSTM model.

Converting Time Series to Supervised Learning Data

Here we have to provide each one-dimensional time series to the model as a separate sequence of input. Then, The LSTM will in turn create an internal representation of each input sequence that will together be interpreted by the decoder. This is achived in to_supervised() function.

Multivariate Encoder Decoder LSTM Model Implementation

This is slight improvement over base LSTM model, here model will not output a vector sequence directly. Instead, the model will be comprised of two sub models, the encoder to read and encode the input sequence, and the decoder that will read the encoded input sequence and make a one-step prediction for each element in the output sequence. The Major difference comes in the decoder, allowing it to both know what was predicted for the prior day in the sequence and accumulate internal state while outputting the sequence.

Here we are using encoder-decoder arch.

Network :

1.Hidden LSTM layer
Single layer with 200 units. with relu activation.
2. RepeatVector layer:
To repeat the internal representation of the input sequence once for each time step in the output sequence. This sequence of vectors will be presented to the LSTM decoder.
3. Define decoder:
Define the decoder as an LSTM hidden layer with 200 units.
4. TimeDistrubuted Wrapper:
Wrap the interpretation layer and the output layer in a TimeDistributed wrapper that allows the wrapped layers to be used for each time step from the decoder.

Hyper Parameters :
1. Loss function : Mean Square Error (MSE), as it goes well with our metric RMSE 2. Optimiser : adam, stochastic gradient descent make sense 3. Epochs : 50 4. Batch size : 16

The network therefore outputs a three-dimensional vector with the same structure as the input, with the dimensions [samples, timesteps, features].

Univariate Multistep CNN Encoder Decoder LSTM

This is slight improvement over encoder-decoder model, we are using CNN, convolutional neural net to encoder-decoder architecture.

Data Preparation

Data Preparation steps Spliting Data in Train/Test Split and Converting Time Series to Supervised Learning Data remains the same as in Encoder-decoder LSTM model

CNN Encoder Decoder LSTM Model Implementation

As we know CNN does not directly support sequence input, But a 1D CNN can be handy in the same. They might be usefull in reading across sequence input and automatically learning the salient features. We apply this CNN-LSTM to our encoder-decoder architecture.

Here we are using encoder-decoder arch.

CNN :

1.1st Convulutional layer

Reads across the input sequence with 64 filters with relu activation unit and projects the results onto feature maps
2.2st Convulutional layer
With same number of filter as in 1st layer, it might be usefull in recognising salient features.
3. Max Pooling layer
The max pooling layer simplifies the feature maps by keeping 1/4 of the values with the largest (max) signal.
4.Flatten Now these feature maps after the pooling layer are then flattened into one long vector that can then be used as input to the decoding process.

After that we pipe the same network we constructed in Encoder Decoder LSTM model.

Hyper Parameters :
1. Loss function : Mean Square Error (MSE), as it goes well with our metric RMSE 2. Optimiser : adam, stochastic gradient descent make sense 3. Epochs : 20 4. Batch size : 16

The network therefore outputs a three-dimensional vector with the same structure as the input, with the dimensions [samples, timesteps, features].

IV. Results

Let's discuss the model evaluation for different models that I ran, We are using walk forward validation for model evaluation

Model Evaluation

Models is be evaluated using a scheme called walk-forward validation.

This is where a model is required to make prediction for fixed window, here as we grouped data in 7 days that is our window. then the actual data for that window is made available to the model so that it can be used as the basis for making a prediction on the next window. This is both realistic for how the model may be used in practice and beneficial to the models allowing them to make use of the best available data.

We can demonstrate this below with separation of input data and output/predicted data.

```
**Input,**                          **Predict**
[Window1]                      ===>   [Window2]
[Window1 + Window2]           ===>    [Window3]
[Window1 + Window2 + Window3] ===>   [Window4]
```

and so on...

The walk-forward validation approach to evaluating predictive models on this dataset is provided below named evaluate_model().

The train and test datasets in windowed format are provided to the function as arguments. An additional argument n_input is provided that is used to define the number of prior observations that the model will use as input in order to make a prediction.

We are working with neural networks, and as such, they are generally slow to train but fast to evaluate. This means that the preferred usage of the models is to build them once on historical data and to use them to forecast each step of the walk-forward validation.

Model Forecasting

In our case, the expected shape of an input pattern is one sample, 7 days of Adj Close Price: [1, 7, 1]

In order to predict the next standard week, we need to retrieve the last days of observations. As with the training data, we must first flatten the history data to remove the weekly structure so that we end up with 10 parallel time series. Next, we need to retrieve the last seven days of Adj Close Price (feature index 0). We will parameterize this as we did for the training data so that the number of prior days used as input by the model can be modified in the future. Next, we reshape the input into the expected three-dimensional structure. We then make a prediction using the fit model and the input data and retrieve the vector of seven days of output.

Below are the results we got for different models that we ran,

1. Univariate Multistep Base LSTM
   IBM : Overall RMSE: [3.175] 1.6, 2.1, 2.7, 3.1, 3.5, 3.9, 4.4
   Apple : Overall RMSE: [15.809] 15.1, 15.8, 15.0, 16.0, 16.9, 15.2, 16.5
   Amazon : Overall RMSE: [35.133] 17.6, 25.3, 27.6, 32.5, 34.9, 45.6, 50.7
   Microsoft: Overall RMSE: [3.292] 2.2, 2.4, 2.6, 3.1, 3.4, 4.0, 4.6

2. Univariate-Multistep-EncoderDecoder-LSTM
   IBM : Overall RMSE: [7.263] 6.3, 6.4, 6.6, 7.2, 7.8, 8.1, 8.2
   Apple : Overall RMSE: [13.290] 11.9, 12.1, 13.0, 13.6, 14.0, 14.3, 14.1
   Amazon : Overall RMSE: [115.855] 80.9, 102.2, 114.3, 118.9, 123.8, 128.5, 133.8
   Microsoft: Overall RMSE: [5.870] 2.8, 2.3, 5.8, 5.6, 6.9, 7.8, 7.4

3. Multivariate-Multistep-EncoderDecoder-LSTM
   IBM : Overall RMSE: [3.811] 2.6, 3.0, 3.3, 3.5, 4.1, 4.6, 5.0
   Apple : Overall RMSE: [9.111] 6.8, 7.9, 8.6, 9.4, 10.3, 10.1, 10.2
   Amazon : Overall RMSE: [40.295] 29.4, 34.1, 34.7, 38.3, 40.1, 48.3, 52.3
   Microsoft: Overall RMSE: [8.048] 6.6, 7.0, 7.2, 8.6, 8.7, 8.9, 8.9

4. Univariate-Multistep-CNN-EncoderDecoder-LSTM
   IBM : Overall RMSE: [4.918] 3.4, 4.2, 4.5, 4.9, 5.3, 5.8, 5.8
   Apple : Overall RMSE: [4.564] 2.8, 3.4, 3.9, 5.7, 4.2, 5.2, 5.8
   Amazon : Overall RMSE: [38.541] 23.2, 26.7, 31.5, 34.2, 44.3, 49.4, 50.8
   Microsoft: Overall RMSE: [3.085] 1.7, 2.0, 1.8, 3.2, 3.2, 5.0, 3.4

Justification

By looking at the results from different models and comparing it to our baseline model, we can see "Univariate-Multistep-CNN-EncoderDecoder-LSTM" is performing the best with minimum overall RMSE for all datasets in general. It also has less variance for each day RMSE. We observe that as we forecast more in the future our RMSE grows for all models, which is expected.

So,overall it is safe to say **"Univariate-Multistep-CNN-EncoderDecoder-LSTM"** is clear winner.

V. Conclusion

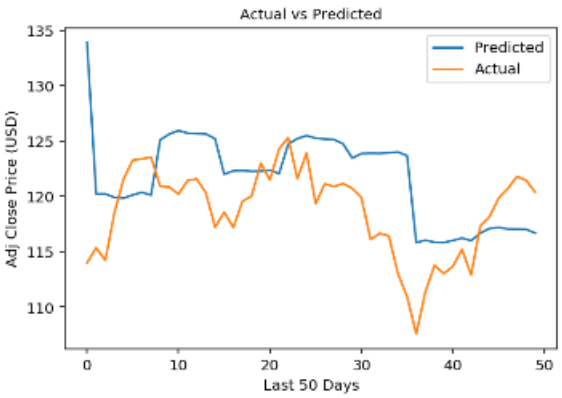
Free form visualization

Our final model Univariate-Multistep-CNN-EncoderDecoder-LSTM perform better among all other models we ran.

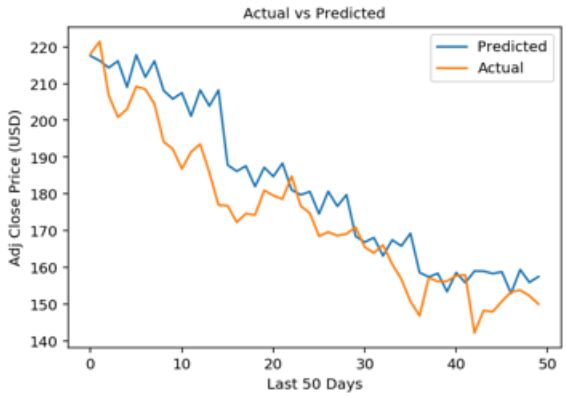
Final Model Prediction vs Actual

Below images show how our model is forecasting Adjusted Close Price for 4 tickers.

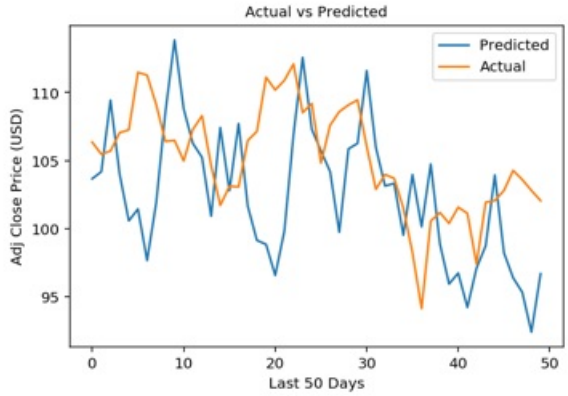
For IBM



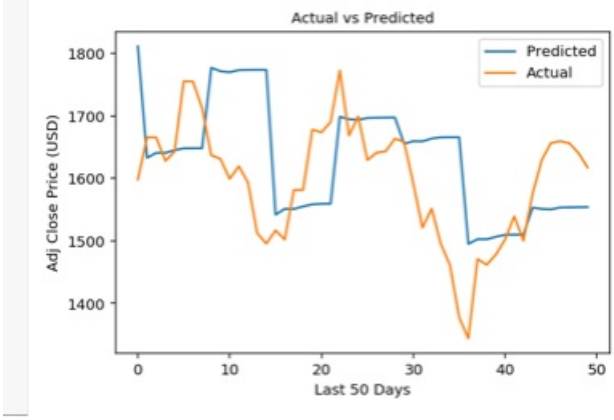
For Apple



For Microsoft



For Amazon



Reflection

When I started on the this problem, I spent a lot of time on EDA, feature extraction and exploration. I wanted to find really good technical indicators which can become vital in forecasting. We saw technical indicators like 'ATR', 'MACD' , 'Stochastic Oscillator' representing momentum, fluctuations in time series stock market data.

Then,we started with persistence model, which is naive way of forecasting Adjusted close price at time 't' by repeating Adjusted close price at time 't-1'. Simplistic nature of persistence model sets a good baseline for more sophisticated models. We explored LSTM networks to build multistep forecasting models, we found that data preparation for model fitting, model evaluation and forecasting can be challenging. We took extra care while splitting data for training and testing to avoid any 'Look Ahead' issues. Overall, we need to mindfull of the shape of the data while working with LSTM Time series forecasting models.

We ran univariate as well as multivariate LSTM models. However, we found that multivariate models are not out performing univariate models and are not worth the complexity they add in the forecasting. Univariate-CNN-Encoder-Decoder-LSTM model performed good among all other models with best overall RMSE, metric we used to compare results.

From our base line model, Persistance model to CNN-Encoder-Decoder-LSTM model, we have seen that Stock market forecasting is really hard problem. By applying various methods, tweaking the hyperparameters does not result in much chages to our metric, RMSE. Having said that, the final LSTM model was able to perform better than the benchmark.

Interseting findings :

1. There is one stock 'amazon' that gives very high RMSE for each and every model. 2. Multivariate models did not perform as expected, but univariate models are performed better with lesser complexity. 3. When tweaking hyperparameters, it was found that lesser complex models were performing better. We used higher batch size, more ecpochs and that actually reduced performance for univariate model that I ran.

Difficult aspects :
1. In Data prepossessing - feature exploration was one of the challange. After goinf through blogs and reasearch papers I believed in MACD, ATR would be vital in forecasting but I was not able to see the impact of multivariate nature of model.
2. Data Preparation for Multivariate LSTM models is tedious task. We have to take care about the shape of the data for training and testing and in model evaluation.
3. Data Preparation for modeling and model evaluation and forecasting Temporal nature of Time Series Analysis .

Comment on Final Model
Although I would have liked to see major differences between my final model and baseline model, I am satisfied with results. I understand that forecasting multistep time series data for stock market is really difficult and a moving target. So take any improvement over baseline model as win.

This model can be used to make educated guess on the stock prices in future.

Improvement

1. Finding better technical indicators is although a challenging task, however I believe that better derrived features can found. This needs more domain expertise which can be achieved with constant reasearch in the area and more experience.
2. Multivariate analysis with CNN-encoder-decoder LSTM model can be explored to see if that added complexity to model make any difference in our metric.
3. Re-enforcement learning models such Q-Learning, Deep-Q-learning can be applied.
4. This project can be extended to build a auto treading bot. Where output of the model can be three signals, Buy, Hold, Sell.
5. This further can be extended in Portfolio Management, in order buy and sell stocks depending on future predictions.

Links:

1. https://machinelearningmastery.com/how-to-develop-lstm-models-for-multi-step-time-series-forecasting-of-household-power-consumption/
2. https://machinelearningmastery.com/cnn-long-short-term-memory-networks/
3. https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/
4. https://stackoverflow.com/questions/24901637/what-is-a-recurrent-neural-network-what-is-a-long-short-term-memory-lstm-netw
5. https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/
6. https://www.investopedia.com/terms/m/macd.asp
7. https://www.investopedia.com/terms/s/stochasticoscillator.asp
8. https://stockcharts.com/school/doku.php?id=chart*school:technical*indicators:average_true*range*atr