TIME SERIES PRICE PREDICTION BY SENTIMENT ANALYSIS

PROJECT REPORT

Submitted by

SANJAY B (715514104039)

KAUSHIK KUMAR TRS (715514104021)

VASANTHAN S (715514104055)

VIGNESH D (715514104306)

in partial fulfillment for the award of the degree Of

BACHELOR OF ENGINEERING

In

COMPUTER SCIENCE AND ENGINEERING

PSG INSTITUTE OF TECHNOLOGY AND APPLIED RESEARCH, COIMBATORE

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2018

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"Time series data prediction using sentimental analysis"** is the bonafide work of "**Sanjay B(715514104039) , Kaushikkumar TRS (715514104021) and Vignesh D(715514104018) and Vasanthan S (715514104055)"** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr.T.Hamsapriya	**Dr.T.Hamsapriya**
Head of the Department	**SUPERVISOR**
Department of Computer Science and Engineering, PSG Institute of Technology and Applied Research, Neelambur, Coimbatore-641062	Department of Computer Science and Engineering, PSG Institute of Technology and Applied Research, Neelambur, Coimbatore-641062

Submitted for the University Project Viva Voce held on _____________

--------------------------	-------------------------
EXTERNAL EXAMINAR	**INTERNAL EXAMINAR**

ACKNOWLEDGEMENT

We are personally indebted to a number of people who gave us their useful insights to aid in our overall progress for this project. First of all, we would like to give our deepest gratitude to **our parents** for all their support.

Our sincere thanks and heartfelt sense of gratitude goes to our respected **Principal**, **Dr. P V Mohanram, PSG Institute of Technology and Applied Research** for all his efforts and administration in educating us in his premiere institution.

We are deeply indebted to our beloved **Head of the Department** ,who also acted as our **Project Coordinator** and **Guide**, **Dr. T. Hamsapriya, Department of Computer Science and Engineering** who molded us both technically and morally for achieving greater success in life

We also convey our sincere thanks to all the staff members for their invaluable suggestions throughout the project and for their technical support rendered during the course of our project.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	**ii**
	LIST OF FIGURES	
1.	**INTRODUCTION TO BITCOIN**	
	1.1 INTRODUCTION	
	1.2 OBJECTIVE	
	1.3 PROBLEM STATEMENT	
	1.4 SCOPE	
	1.5 OVERVIEW	
	1.5.1 Machine Learning	
	1.5.1.1 Applications	
2.	**BACKGROUND**	
	2.1 BITCOIN	
	2.1.1History of Bitcoin	
	2.1.2 What is Bitcoin?	
	2.1.3 Blockchain	
	2.1.4 Proof-of-work	
	2.2 OPINION MINING	
	2.3 SENTIMENT ANALYSIS	
	2.3.1 Polarity classification	
	2.3.2 Lexicon-based approach	
	2.3.3 TextBlob	

3. SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

3.1.1 Drawbacks

3.3 PROPOSED METHODOLOGY

3.3.1 Advantages

4. LITERATURE SURVEY

4.2 LITERATURE SURVEY

5. SOFTWARE DESCRIPTION

5.1 INTRODUCTION TO NEURAL NETWORK

5.2 TENSORFLOW

5.3 KERAS

5.4 PANDAS

5.5 SKLEARN

5.6 MATPLOTLIB

6. PROPOSED METHODOLOGY AND IMPLEMENTATION

6.1 DATA COLLECTION AND PREPROCESSING

6.1.1 Bitcoin Price Data

6.2.2 Tweet Collection

6.2 SENTIMENT ANALYSIS

6.3 TRAINING NEURAL NETWORK

6.3.1 Implementation

6.3.2 Data Preparation

6.3.3 Training Model

7. CONCLUSION AND FUTURE WORK

APPENDIX

REFERENCES

ABSTRACT

A time series is a sequence of observation taken sequentially in time. The purpose of time series prediction is to predict or forecast the future values of the series based on the history of that series. Since bitcoin price fluctuation depends on speculation this paper uses time series analysis to study the relation between bitcoin price and sentiments derived from twitter feed using Textblob to find the price change in the near future.. Historical bitcoin price data and tweets about bitcoin are collected, pre-processed and recurrent neural network (Long Short Term Memory) is used to develop a model. LSTM are special kind of recurrent neural network capable of remembering long term dependencies. LSTM with different epochs and lookbacks are tried to find out the least RMSE value. The live twitter sentiment values and live bitcoin price are fed into the trained model to predict the future value of bitcoin price. A model with RMSE value less than 10 was achieved using LSTM.

LIST OF ABBREVIATIONS

BTC	Bitcoin
RNN	Recurrent neural network
LSTM	Long short term memory
RMSE	Root mean squared error
GPOMS	Google profile of mood states

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION:

It’s 2017 the people of the world generate 2.5 million terabytes of information a day. 500 million tweets get generated each and every day,. Twitter specifically has become a location where news is quickly disseminated in a concise format.

For a financial commodity like bitcoin the public confidence in a particular commodity , supply and demand of that financial commodity is a core base which determines its value. Social media has served as a platform to express opinions about everything, and making use of the open APIs provided by the Facebook and Twitter the raw data can be processed to obtain meaningful information and these pieces of information become available with a sea of meta-data.

Bitcoin (BTC), the decentralized cryptographic currency, is similar to most commonly known currencies.These crypto currencies doesn’t have a physical form rather they exist digitally . Its price is affected by socially constructed opinions. Since the Bitcoin was revealed to the world, in 2009 , it quickly gained interest as an alternative to regular currencies. As such, like most things, opinions and information about Bitcoin are prevalent throughout the Social Media sphere.

1.2 OBJECTIVE

To create a high accuracy model for predicting the bitcoin price and to find the right time for buying or selling a bitcoin by analyzing the predicted time series data.

1.3 PROBLEM STATEMENT

By analyzing the sentiments of tweets regarding bitcoin and comparing with Bitcoin’s price

- Is there a correlation between Twitter sentiment and BTC price fluctuation?
- Can a deep learning prediction model be used to predict the future bitcoin price with high accuracy.

1.4 SCOPE

During the last 25 years Time Series Analysis has become one of the most important and widely used branches of Mathematical Statistics. Its fields of application range from neurophysiology to astrophysics and it covers such well-known areas as economic forecasting, study of biological data, control systems, signal processing and communications and vibrations engineering.

Sentiments are only collected from one micro-blogging source; Twitter. Due to Twitters establishment in the micro-blogging sphere, as well as its accessible programmatic interface for data collection. Similarly, the decision to limit the cryptocurrency to Bitcoin came down to Bitcoin being the most established cryptocurrency both in age and cryptocurrency market share, reflecting its acceptance \by public

In the simplest terms, Bitcoin is a digital asset and a payment system which uses peer-to-peer technology to operate with no central authority or banks. Geographical boundaries don't limit it, and unlike paper currency, managing transactions and the issuing of Bitcoins is carried out collectively by the network, using complex and cryptographic code used in its design.

The presented prediction model can be used to any other cryptocurrency by adopting the underlying data collection ,preprocessing and training a machine learning model.. The accuracy estimations would have to be recomputed and would likely vary vastly based on the data volume ,model selection and its parameter . The sentiments classification is limited to the most naive binary form of positive or negative, not attempting to capture sentiment on a more complex emotional level. On the BTC side, the key value will be limited to an increase and decrease in price over specific time intervals, disregarding volume and other key metrics. Further, BTC transactions are collected for the BTC/USD currency pair, and only collected from the Coindesk API.

1.5 OVERVIEW:

1.5.1 MACHINE LEARNING:

Machine learning is a core sub-area of artificial intelligence. It enables computers to get into a mode of self-learning without being explicitly programmed. When exposed to new data, these computer programs are enabled to learn, grow, change, and develop by themselves. The iterative aspect of machine learning is the ability to adapt to new data independently. This is possible as programs learn from previous computations and use “pattern recognition” to produce reliable results.

1.5.1.1 APPLICATIONS:

1. Predictions while Commuting
2. Videos Surveillance
3. Email Spam and Malware Filterin
4. Product Recommendations

CHAPTER 2

BACKGROUND

2.1 BITCOIN

2.1.1 History of Bitcoin

On 18 August 2008, the domain name "bitcoin.org" was registered. In November that year, a link to a paper authored by Satoshi Nakamoto titled Bitcoin: A Peer-to-Peer Electronic Cash System was posted to a cryptography mailing list. Nakamoto implemented the bitcoin software as a open source code and released it in January 2009 on SourceForge. The identity of Nakamoto remains unknown.

2.1.2 What is Bitcoin

Bitcoin is a cryptocurrency, or a digital currency, that uses rules of cryptography for regulation and generation of units of currency. It's decentralized -- there's no government, institution or other authority that controls it. Owners are anonymous; instead of using names, tax IDs, or social security numbers, bitcoin connects buyers and sellers through encryption keys. The network is peer-to-peer and transactions take place between users directly, without an intermediary. These transactions are verified by network nodes through the use of cryptography and recorded in a public distributed ledger called a blockchain.

2.1.3 Blockchain

A blockchain is a digitized, decentralized, public ledger of all cryptocurrency transactions. Constantly growing as ‘completed’ blocks (the most recent transactions) are recorded and added to it in chronological order, it allows market participants to keep track of digital currency transactions without central recordkeeping. Each node (a computer connected to the network) gets a copy of the blockchain, which is downloaded automatically.

Originally developed as the accounting method for the virtual currency Bitcoin, blockchains – which use what's known as distributed ledger technology (DLT) – are appearing in a variety of commercial applications today. Currently, the technology is primarily used to verify transactions, within digital currencies though it is possible to digitize code and insert practically any document into the blockchain.

2.1.4 Proof –of- work

A proof-of-work (PoW) system is an economic measure to deter denial of service attacks and other service abuses such as spam on a network .It is a piece of data which was difficult (costly, time-consuming) to produce so as to satisfy certain requirements. Producing a proof of work can be a random process with low probability, so that a lot of trial and error is required on average before a valid proof of work is generated. In bitcoin usually “The proof of work” refers to a particular number that produces certain number of leading zeros in the hash value of the block.

2.2 OPINION MINING

Social Networks have grown rapidly since their inception early this millennium. Global total users surpassed 2 billion in 2015, with continuous steady growth to come according to Statista.com estimation. Social networks provide users with a platform to express their thoughts, views, and opinions.

Twitter, the micro-blogging platform and public company, was launched in 2006. 10 years on, in 2016, the platform has over 300 million monthly active users. As is characteristic for micro-blogging services, Twitter provides its users with the possibility to publish short messages. These messages are commonly called tweets and are limited in length to 140 characters. User can also include metadata inline in the text of a tweet [15, 12], by either # (’hash tag’) or @ (’at’). The two operators have different intentions, with the former (#) used as a symbol to signal a specific context while the latter (@) references to another Twitter user. Hash tag contextually link together tweets, creating a web of contextually similar data points.

Twitter also provides facilities for both searching and consuming a real-time stream of tweets based on specific hash tags. Twitter is a centralized location to publish (as well as consume) internally and externally generated content. For some companies, it has become an additional channel to communicate with the marketplace, and for others to use as a resource. Twitter has over the years become a platform for media: news, company announcements, government communication, to individual user's personal opinions, world views, or daily life. Together, Twitter users are generating millions of short messages, all public and some already labeled with contextual data. Due to the message length restriction and the classifying nature of tweets hash tags, Twitter has become a gold mine for opinionated data, in its semi-structured form. Researchers and other entities are regularly mining Twitter for tweets, attempting to gain value, information or understanding in varying subjects and disciplines. As such Twitter is widely used as a source when looking for sentiment data sets.

2.3 SENTIMENT ANALYSIS

Sentiment analysis is about finding the underlining opinions, sentiment, and subjectivity in texts, which all are important factors in influencing behavior. The advent of machine learning tools, wider availability of digital data sets, and curiosity, has greatly reduced the cost of performing sentiment analysis allowing for more research. This type of data is highly sought after and has attracted the attention from researchers and companies alike.

2.3.1 Polarity classification

Since the rise of social media, a large part of the current research has been focused on classifying natural language as either positive or negative sentiment. Polarity classification has been found to achieve high accuracy in predicting change or trends in public sentiment, for a myriad of domains (e.g. stock price prediction).

2.3.2 Lexicon-based approach

A lexicon is a collection of features (e.g. words and their sentiment classification). The lexicon-based approach is a common method used in sentiment analysis where a piece of text is compared to a lexicon and attributed sentiment classifications. Lexicons can be complex to create, but once created require little resources to use. Well designed lexicons can achieve a high accuracy and are held in high regard in the scientific community.

2.3.4 Text blob

Text Blob is a Python (2 and 3) library for processing textual data. It provides a consistent API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation and more. Text blob is based on naive bayes analyzer. Naive Bayes is a kind of classifier which uses the Bayes Theorem. It predicts membership probabilities for each class such as the probability that given record or data point belongs to a particular class. The class with the highest probability is considered as the most likely class. This is also known as **Maximum A Posterior (MAP)**.

Text blob uses an xml file called sentiment.xml which enlists polarity, subjectivity, intensity for each word. The polarity of the text can be calculated by parsing the text. The analysis returns two values polarity and subjectivity and the polarity score is a range between [-1, 0, 1] where 0 indicates neutral, +1 indicates a positive attitude,-1 indicated a negative attitude. Subjectivity is a float value within the range [0, 1] where 0.0 is very objective and 1.0 is very subjective.

CHAPTER 3

SYSTEM ANALYSIS

Application development can be generally being thought of having two major components: analysis and design. In analysis more emphasis is given to understanding the details of an existing system or a proposed one and then deciding whether the proposed system is desirable or not and whether the existing system needs improvement. Thus, analysis is the process of investigating a system, identifying problems, and using the information to recommended improvements to the system.

3.1 Existing system:

In an attempt to substance the possible identification of correlation between Bitcoin price change and Twitter sentiment change, two temporal aspects are taken into account; frequency length and shift. Short term predictions are made on discrete time intervals, i.e. time series. Short term is regarded as a time series ranging from 5 minutes, to 4 hours in length. Each time series is evaluated over four different shifts forward: 1, 2, 3, and 4. Where a shift indicates that an event predicts for that period in time in the future, e.g. if a positive event occurs in a test with freq. 30 minutes at 14:30:00, and the shift is 2, the prediction is set for the BTC price change at 15:30:00.

3.1.1 Drawback:

No machine learning model was used. Only a naive prediction model was presented, based on the intensity of sentiment fluctuations from one time interval to the next. The model showed that the most accurate aggregated time to make predictions over was 1 hour. The number of predictions that were made was very few so the prediction model conclusions were unfounded and the model yielded only 83% accuracy.

3.2 Proposed methodology:

Since time series prediction also adds the complexity of sequence dependence among the input variables a powerful type of neural network designed to handle sequence dependence called recurrent neural network is used. 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.

A recurrent neural network was trained over input time series data(BTC price and sentiment) to develop a model to predict the future bitcoin price.

3.2.1 Advantages:

An accuracy of 90% was able to be achieved by using recurrent neural network (LSTM).The model is helpful in predicting the value of bitcoin one day in advance.

CHAPTER 4
LITERATURE SURVEY

1) Twitter mood predicts the stock market

by Johan Bollena, Huina Maoa, Xiaojun Zeng

METHOD:

Investigation of whether measurements of the collective mood states derived from large-scale Twitter feeds are correlated to the value of the Dow Jones Industrial Average (DJIA) over time is done. The daily Twitter feeds are analyzed by two mood tracking tools, namely OpinionFinder that measures positive vs. negative mood and Google-Profile of Mood States (GPOMS) that measures mood in terms of 6 dimensions (Calm, Alert, Sure, Vital, Kind, and Happy). The resulting mood time series is cross validated by comparing their ability to detect the public's response to the presidential election and Thanksgiving Day in 2008.A Granger causality analysis and a Self-Organizing Fuzzy Neural Network are then used to investigate the hypothesis that public mood states, as measured by the OpinionFinder and GPOMS mood time series, are predictive of changes in DJIA closing values. The results indicate that the accuracy of DJIA predictions can be significantly improved by the inclusion of specific public mood dimensions but not others. Accuracy of 86.7% was found by predicting the daily up and down changes in the closing values of the DJIA and a reduction of the Mean Average Percentage Error (MAPE) by more than 6%.

2) **Sentiment Analysis of Twitter Data: A Survey of Techniques**

by Vishal A. Kharde, S.S. Sonawane

METHOD:

Sentiment analysis of twitter data is made which is helpful to analyze the information in the tweets where opinions are highly unstructured, heterogeneous and are either positive or negative, or neutral in some cases. The comparative analyses of existing techniques for opinion mining like machine learning and lexicon-based approaches, together with evaluation metrics. Using various machine learning algorithms like Naive Bayes, Max Entropy, and Support Vector Machine, we provide research on twitter data streams.

3) **Sentiment Analysis of Twitter Data for Predicting Stock Market Movements**

by Venkata Sasank Pagolu, Kamal Nayan Reddy Challa, Ganapati Panda

METHOD:

The changes in stock prices of a company, the rises and falls, are correlated with the public opinions being expressed in tweets about that company is observed. Understanding opinion from a piece of text is the objective of sentiment analysis. Two different textual representations, Word2vec and Ngram, for analyzing the public sentiments in tweets was presented. Sentiment analysis and supervised machine learning principles to the tweets extracted from twitter and analyze the correlation between stock market movements of a company and sentiments in tweets was applied. In an elaborate way, positive news and tweets in social media about a company would definitely encourage people to invest in the stocks of that company and as a result the stock price of that company would increase. A strong correlation exists between the rise and falls in stock prices with the public sentiments in tweets was shown.

4) Predicting the price of Bitcoin using Machine Learning

by Sean McNally

METHOD:

Predicting the price of Bitcoin using machine learning was concerned. The goal was to ascertain with what accuracy can the direction of Bitcoin price in USD can be predicted. The price data is sourced from the Bitcoin Price Index . The task is achieved with varying degrees of success through the implementation of a Bayesian optimised recurrent neural network (RNN) and Long Short Term Memory (LSTM) network. The LSTM achieves the highest classification accuracy of 52% and a RMSE of 8%. The popular ARIMA model for time series forecasting is implemented as a comparison to the deep learning models. The non-linear deep learning methods outperform the ARIMA forecast which performs poorly. Finally, both deep learning models are benchmarked on both a GPU and a CPU with the training time on the GPU outperforming the CPU implementation by 67.7%.

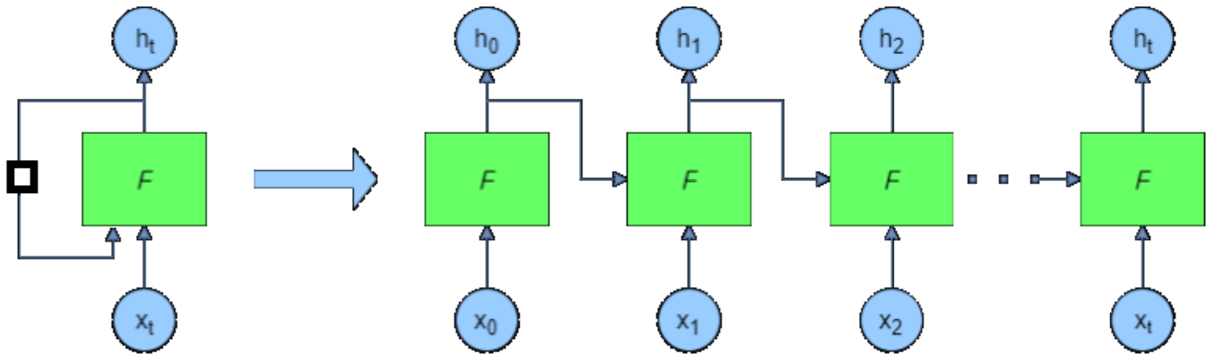
CHAPTER 5

SOFTWARE DESCRIPTION

5.1 INTRODUCTION TO LSTM NETWORKS:

5.1.1 RECURRENT NEURAL NETWORK:

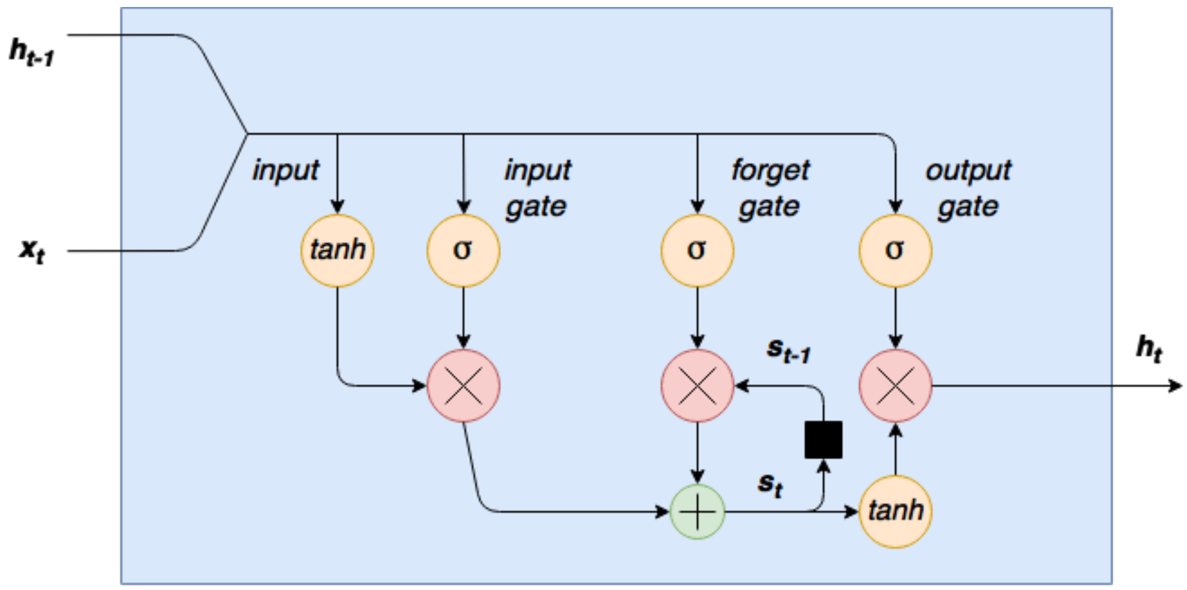
A LSTM network is a kind of recurrent neural network. A recurrent neural network is a neural network that attempts to model time or sequence dependent behaviour – such as language, stock prices, electricity demand and so on. This is performed by feeding back the output of a neural network layer at time *t* to the input of the same network layer at time t+1.



At each time step, a new word is being supplied – the output of the previous *F* (i.e. ht−1ht−1) is supplied to the network at each time step also. The problem with recurrent neural networks, constructed from regular neural network nodes, the dependencies between words or sequence values that are separated by a significant number of other words is modeled, vanishing gradient problem is observed. This is because small gradients or weights (values less than 1) are multiplied many times over through the multiple time steps, and the gradients shrink asymptotically to zero. This means the weights of those earlier layers won't be changed significantly and therefore the network won't learn long-term dependencies.

LSTM:

An LSTM network is a recurrent neural network that has LSTM cell blocks in place of our standard neural network layers. These cells have various components called the input gate, the forget gate and the output gate. Here is a graphical representation of the LSTM cell:



The mathematics of the LSTM cell looks like this:

Input

First, the input is squashed between -1 and 1 using a *tanh* activation function. This can be expressed by:

g=tanh(bg+xtUg+ht−1Vg)g=tanh(bg+xtUg+ht−1Vg)

Where UgUg and VgVg are the weights for the input and previous cell output, respectively, and bgbg is the input bias. The exponents *g* are not a raised power, but rather signify that these are the input weights and bias values (as opposed to the input gate, forget gate, output gate etc.).

This squashed input is then multiplied element-wise by the output of the *input gate,* which, as discussed above, is a series of sigmoid activated nodes:

i=σ(bi+xtUi+ht−1Vi)i=σ(bi+xtUi+ht−1Vi)

The output of the input section of the LSTM cell is then given by:

g∘ig∘i

Where the ∘∘ operator expresses element-wise multiplication.

Forget gate and state loop

The forget gate output is expressed as:

f=σ(bf+xtUf+ht−1Vf)f=σ(bf+xtUf+ht−1Vf)

The output of the element-wise product of the previous state and the forget gate is expressed as st−1∘fst−1∘f. The output from the forget gate / state loop stage is:

st=st−1∘f+g∘ist=st−1∘f+g∘i

Output gate

The output gate is expressed as:

o=σ(bo+xtUo+ht−1Vo)o=σ(bo+xtUo+ht−1Vo)

So the final output of the cell , with the *tanh* squashing, can be shown as:

ht=tanh(st)∘o

5.2 TENSORFLOW

TensorFlow is a tool for machine learning. While it contains a wide range of functionality, TensorFlow is mainly designed for deep neural network models.The virtualization layer (hypervisor) is installed on host operating system**.**

The TensorFlow use the model to make *predictions*on unknown data. A model is the relationship between features and the label. For the Data classification problem, the model defines the relationship between the past collected dataand predict the pattern. Some simple models can be described with a few lines of algebra, but complex machine learning models have a large number of parameters that are difficult to summarize.

5.3 KERAS

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Keras is an API designed for human beings, not machines. It puts user experience front and center. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.

A model in Keras is understood as a sequence or a graph of standalone, fully-configurable modules that can be plugged together with as little restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions, regularization schemes are all standalone modules that you can combine to create new models.

New modules are simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.

No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

5.4 PANDAS

Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
Pandas is a NumFOCUS sponsored project. This will help ensure the success of development of *pandas* as a world-class open-source project, and makes it possible to donate to the project.

Python has long been great for data munging and preparation, but less so for data analysis and modelling. pandas helps fill this gap, enabling you to carry out your entire data analysis workflow in Python without having to switch to a more domain specific language like R.

Combined with the excellent IPython toolkit and other libraries, the environment for doing data analysis in Python excels in performance, productivity, and the ability to collaborate.
Pandas does not implement significant modelling functionality outside of linear and panel regression; for this, look to states models and scikit-learn. More work is still needed to make Python a first class statistical modelling environment, but we are well on our way toward that goal.

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language**.** It is already well on its way toward this goal.

Pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
- Ordered and unordered (not necessarily fixed-frequency) time series data.
- Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
- Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure

The two primary data structures of pandas, Series (1-dimensional) and DataFrame (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For R users, DataFrame provides everything that R's data.frame provides and much more. pandas is built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

Here are just a few of the things that pandas does well:

- Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data
- Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects
- Automatic and explicit data alignment: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let *Series*, *DataFrame*, etc. automatically align the data for you in computations
- Make it easy to convert ragged, differently-indexed data in other Python and NumPy data structures into DataFrame objects
- Intelligent label-based slicing, fancy indexing, and subsetting of large data sets
- Intuitive merging and joining data sets
- Flexible reshaping and pivoting of data sets
- Hierarchical labeling of axes (possible to have multiple labels per tick)

- Robust IO tools for loading data from flat files (CSV and delimited), Excel files, databases, and saving / loading data from the ultrafast HDF5 format.
- Time series-specific functionality: date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging, etc.
- Pandas is fast. Many of the low-level algorithmic bits have been extensively tweaked in Cythoncode. However, as with anything else generalization usually sacrifices performance. So if you focus on one feature for your application you may be able to create a faster specialized tool.

- Pandas is a dependency of stats models, making it an important part of the statistical computing ecosystem in Python.
- Pandas has been used extensively in production in financial applications.

5.5 SKLEARN

Scikit-learn (formerly **scikits.learn**) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

5.6 MATPLOTLIB

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.

Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

CHAPTER 6

PROPOSED METHODOLOGY AND IMPLEMENTATION

The chapter is structured in the chronological order of events: starting with data gathering, followed by dataset pre-processing, analyzing for sentiment, to finally describe the prediction model and it's evaluation.

6.1 DATA COLLECTION AND PREPROCESING:

Historical and live bitcoin price [BTC/USD], tweets are collected from coinbase and twitter respectively over the same time period. The collected data is then preprocessed into desired format to input into the neural network.

6.1.1 BITCOIN PRICE DATA

Historical price of Bitcoin were gathered daily from **Coinbase API** depending on the time interval length of the requested data. The coinbase API provides different types of data such as high, low, ask price, open price and closed price. Close price is collected since it is the price at the end of the day. The bitcoin price is collected over a period from Feb 2014 to FEB 16. Live bitcoin price data is collected from bitstamp in json format and then it's loaded into a text file.

6.1.2 TWEET COLLECTION

To collect data for the sentiment analysis Twitter's streaming API was used in combination with Tweepy. Tweepy, an open source framework written in Python, facilitates tweet collection from Twitter's API Tweepy allows for filtering based on hashtags or words, and as such was considered as an efficient way of collecting relevant data.

STEPS FOR COLLECTING TWEETS USING TWEEPY:

Step 1: Getting Twitter API keys

In order to access Twitter Streaming API, 4 pieces of information from Twitter is used: API key, API secret, Access token and Access token secret.

Step 2: Connecting to Twitter streaming data and downloading data.

A Python library called Tweepy is used to connect to Twitter Streaming API and downloading the data.

```
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)

auth.set_access_token(access_token, access_token_secret)

twitter_api = tweepy.API(auth)
```

6.2 SENTIMENT ANALYSIS

The tweets collected from twitter are then analysed to calculate the sentiment associated with them. A python library called Textblob was used to perform the sentiment analysis process.Textblob internally uses naive bayes analyzer to classify the tweets. The Naive Bayes Analyzer available in TextBlob is already trained on a data set of movie reviews.and it uses the underlying NLTK package for preprocessing the tweets.

Pre-processing refers to the transformations applied to the data before feeding it to the algorithm**.** Data Preprocessing improves the training efficacy of the neural network and helps to build a better model for prediction. The performance of neural network is highly influenced by the size of the dataset and the data-preprocessing techniques used.

PREPROCESSING TECHNIQUES:

Stopword removal:

A majority of the words in a given text are connecting parts of a sentence rather than showing subjects, objects or intent. Word like “the” or “and” can be removed by comparing text to a list of stopword.

Parts of Speech Tagging:

Understand parts of speech can make difference in determining the meaning of a sentence. Part of Speech (POS) often requires look at the proceeding and following words and combined with either a rule-based or stochastic method.

Tokenization:

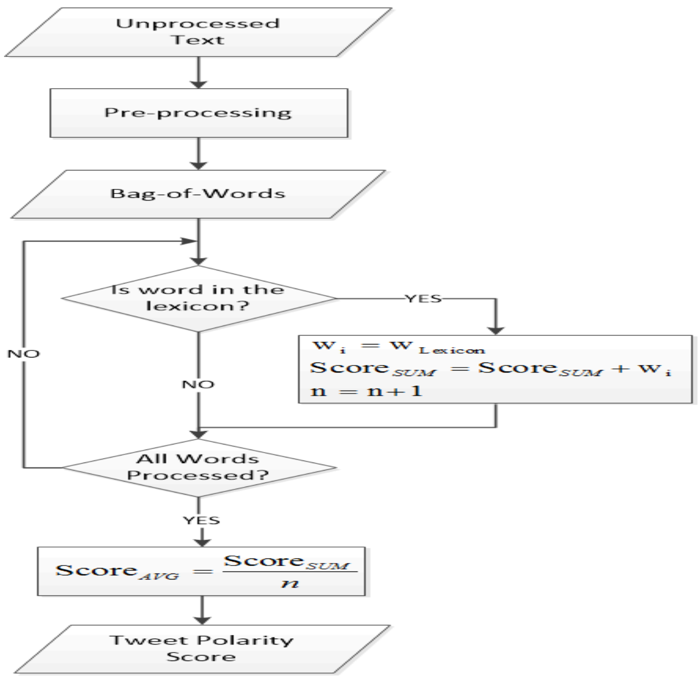
Tokenization describes splitting paragraphs into sentences, or sentences into individual words. Sentences can be split into individual words and punctuation through a similar process

Stemming:

Stemming is a process where words are reduced to a root by removing inflection through dropping unnecessary characters, usually a suffix.

Once the tweets are preprocessed we can perform sentiment analysis process to calculate the polarity of the tweets by inputting them to Textblob.

SENTIMENT ANALYSIS PROCESS:



6.3 TRAINING NEURAL NETWORK:

The increase/decrease in Bitcoin's price with large percentages over short periods of time is an interesting phenomenon which cannot be predicted at all. However, it would be nice to see the effect of any powerful machine learning model over this price. Recurrent neural network and a collection of Bitcoin's prices were used to predict the future of the crypto currency.

6.3.1 IMPLEMENTATION:

Keras is used for training the model. Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlOW, CNTK, or Theano. Numpy (matrix manipulations), panda (defines data structures), matplotlib (visualization) and sklearn (normalizing our data) is used.

6.3.2 DATA PREPARATION

Our next step will be to prepare the data. This includes fetching from a 3rd party source, clearing up and splitting into training and testing.

- Load data and remove the unused fields (in this case 'Date')

We are using pandas to read the *.csv* file and remove the useless columns.

- Split into training and testing

The decision made here is just for the purpose of this tutorial. In real projects, **you should always split your data into training, testing** (usually 80%, 20%).

- Normalize and prepare for training

We use fit_transform to center the data in a way that it has 0 mean and 1 standard error. Then, we divide the data into x_train and y_train. Essentially, that is made because we are using Recurrent neural network. Our model will get the 0-th element from x_train and try to predict the 0-th element from y_train (which is the first element of x_train). This way we are creating a sequence—exactly what RNNs need for training. Finally, we reshape the x_train data to match the requirements for training using keras .

6.3.3 TRAINING MODEL:

Now we need to train our model using the above data. With keras, this process is extremely easy and understandable. You don't need to know the math behind any of the operations in order to produce decent results.

We are building the model as follows:

- Line 1: the number of units (dimension of the output state) used for the LSTM cell.
- Line 2: the activation function used for the LSTM cell (sigmoid in our case).
- Line 3: the optimizer used to minimize the loss function (Adam in our case).
- Line 4: the loss function which we need to minimize while adjusting the weights and biases of the network (Mean squared error in our case).
- Line 5: neural networks are typically trained on batches which mean that on every iteration we pick 5 examples from our training set and use them for training.
- Line 6: the number of epochs determines how many iterations we need to make.

After explaining what each hyper parameter means, we start training the model:

- Line 9: define the Sequential model which stacks all layers (input, hidden and output)
- Line 12: add the LSTM cell to our model.
- Line 15: add the output layer—is a simple Dense with default linear activation function and dimension 1.
- Line 18: this is the last step before the actual training. We need to set the optimizer and the loss function specified above.
- Line 21: train the model using inputs x_train and labels y_train.

6.3.4 PREDICTING PRICE:

We are g   oing to predict the price and plot it to compare with the real results.

- Predict the price for the Test dataset.

Lines 1–6: we do exactly the same as for the training set. Using fit_transform to scale the data and then reshape it for the prediction.

Line 7: predict.

Line 8: rescale the predicted data to match its real values (prices in USD)

- Visualize results

  we are going to plot test and predicted prices using iplot function in matplot library.

CHAPTER 7
CONCLUSION AND FUTURE ENCHANCEMENT

FUTURE ENHANCEMENT

Lexicon provided by textblob is a general lexicon containing sentiment for most commonly used words in social media. Since tweets related to bitcoin mostly has financial and cryptocurrency trade terms Sentiments are not captured effectively and hence most of the tweets has a neutral sentiments. Using a Domain Specific Lexicon would improve the sentiment analysis process and thus could improve the quality of the model developed. The sentiment analysis process yielded sentiment score is between 0 to 1. Google profile of mood states tries to analyse the sentiment in six different dimensions, so by analysing the sentiment in higher dimension we can identify which mood correlates maximum to the bitcoin price fluctuation which helps in increasing the accuracy of the model.

CONSCLUSON

Here we studied if sentiments derived from twitter data related to bitoin can be used to predict the time series data such as bitcoin price. We were able to use the sentiments on the bitcoin price to train a Recurrent Neural Network and develop a model that predict a future bitcoin price with a RMSE value below 10. We were able to make decision on whether buy or sell on the next day.

APPENDIX

LSTM_VERSION_1.py

```
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
import pandas as pd
from datetime import datetime
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
import plotly.offline as py
import plotly.graph_objs as go
import numpy as np
import seaborn as sns



import quandl
data = quandl.get('BCHARTS/KRAKENUSD', returns='pandas',
authtoken="VHG6EKysZ5sWhyyfocsZ")


data1 = pd.read_csv(filepath_or_buffer="bitprice.csv")
#data2 = pd.read_csv(open('sentiment6.txt', 'r'))
data2 = pd.read_csv(filepath_or_buffer="sentiment6.txt")
```

```
data3 = open("bit_new.txt","w")

data1.info()


data2.info()


#floatlst = float(data2.sentiment)
data2.dropna(axis=1,how='all')
data2['sentiment'] = data2.sentiment.astype('float64', errors = 'ignore')

data1.head()
data2.head()

data = pd.merge(data1,data2, on='stamp', how='inner')
data.describe()

data['stamp'] = pd.to_datetime(data['stamp'].apply(str),format='%Y%m%d')
#data = data.sort_values(by='stamp')
data.head()

btc_trace = go.Scatter(x=data['stamp'], y=data['price'], name= 'Price')
py.iplot([btc_trace])

data['price'].replace(0, np.nan, inplace=True)
```

```
data['price'].fillna(method='ffill', inplace=True)

btc_trace = go.Scatter(x=data['stamp'], y=data['price'], name= 'Price')
py.iplot([btc_trace])

from sklearn.preprocessing import MinMaxScaler
values = data['price'].values.reshape(-1,1)
sentiment = data['sentiment'].values.reshape(-1,1)
values = values.astype('float32')
sentiment = values.astype('float32')
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)

train_size = int(len(scaled) * 0.7)
test_size = len(scaled) - train_size
train, test = scaled[0:train_size,:], scaled[train_size:len(scaled),:]
print(len(train), len(test))
split = train_size

def create_dataset(dataset, look_back, sentiment):
    dataX, dataY = [], []
    for i in range(len(dataset) - look_back):
        a = dataset[i:(i + look_back), 0]
        np.append(a,sentiment[i])
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    print(len(dataY))

    return np.array(dataX), np.array(dataY)


look_back = 1

trainX, trainY = create_dataset(train, look_back, sentiment[0:train_size])

testX, testY = create_dataset(test, look_back,
sentiment[train_size:len(scaled)])


trainX.shape


trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))

testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))


model = Sequential()

model.add(LSTM(100, input_shape=(trainX.shape[1], trainX.shape[2])))

model.add(Dense(1))

model.compile(loss='mae', optimizer='adam')

history = model.fit(trainX, trainY, epochs=300, batch_size=100,
validation_data=(testX, testY), verbose=0, shuffle=False)


pyplot.plot(history.history['loss'], label='train')

pyplot.plot(history.history['val_loss'], label='test')

pyplot.legend()

pyplot.show()


yhat = model.predict(testX)

pyplot.plot(yhat, label='predict')
pyplot.plot(testY, label='true')

pyplot.legend()

pyplot.show()


yhat_inverse = scaler.inverse_transform(yhat.reshape(-1, 1))

testY_inverse = scaler.inverse_transform(testY.reshape(-1, 1))


rmse = sqrt(mean_squared_error(testY_inverse, yhat_inverse))

print('Test RMSE: %.3f' % rmse)


pyplot.plot(yhat_inverse, label='predict')

pyplot.plot(testY_inverse, label='actual', alpha=0.5)

pyplot.legend()

pyplot.show()


predictDates = data.tail(len(testX)).stamp


testY_reshape = testY_inverse.reshape(len(testY_inverse))

yhat_reshape = yhat_inverse.reshape(len(yhat_inverse))


actual_chart = go.Scatter(x=predictDates, y=testY_reshape, name= 'Actual
Price')

predict_chart = go.Scatter(x=predictDates, y=yhat_reshape, name= 'Predict
Price')

py.iplot([predict_chart, actual_chart])

#show graph
```

```
sns.heatmap(data.corr(), annot=True, cmap='RdYlGn', linewidths=0.1,
vmin=0)


def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = pd.DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = pd.concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg


values = data[['price']].values
values = values.astype('float32')

scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)

reframed = series_to_supervised(scaled, 1, 1)
reframed.head()

#reframed.drop(reframed.columns[[1,3]], axis=1, inplace=True)
#print(reframed.head())

values = reframed.values
n_train_hours = int(len(values) * 0.7)
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]
# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

multi_model = Sequential()
multi_model.add(LSTM(100, input_shape=(train_X.shape[1],
train_X.shape[2])))
multi_model.add(Dense(1))
multi_model.compile(loss='mae', optimizer='adam')

multi_history = multi_model.fit(train_X, train_y, epochs=300,
batch_size=100, validation_data=(test_X, test_y), verbose=0,
shuffle=False)


pyplot.plot(multi_history.history['loss'], label='multi_train')

pyplot.plot(multi_history.history['val_loss'], label='multi_test')

pyplot.legend()

pyplot.show()


yhat = multi_model.predict(test_X)

pyplot.plot(yhat, label='predict')

pyplot.plot(test_y, label='true')

pyplot.legend()

pyplot.show()


test_X = test_X.reshape((test_X.shape[0], test_X.shape[1]))

# invert scaling for forecast

inv_yhat = concatenate((yhat, test_X[:, 1:]), axis=1)

inv_yhat = scaler.inverse_transform(inv_yhat)

inv_yhat = inv_yhat[:,0]

# invert scaling for actual

test_y = test_y.reshape((len(test_y), 1))

inv_y = concatenate((test_y, test_X[:, 1:]), axis=1)

inv_y = scaler.inverse_transform(inv_y)

inv_y = inv_y[:,0]
```

```
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)

actual_chart = go.Scatter(x=predictDates, y=inv_y, name= 'Actual Price')
multi_predict_chart = go.Scatter(x=predictDates, y=inv_yhat, name= 'Multi
Predict Price')
predict_chart = go.Scatter(x=predictDates, y=yhat_reshape, name= 'Predict
Price')
py.iplot([predict_chart, multi_predict_chart, actual_chart])
```

LSTM_MAIN.py

```
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
import pandas as pd
from datetime import datetime
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
import plotly.offline as py
import plotly.graph_objs as go
import plotly
import numpy as np
import seaborn as sns
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import csv as csv


import talib


print(talib.__version__)


def dateParser(x):
	return datetime.strptime(x,'%Y%m%d')



btc = pd.read_csv('bitprice-1.csv', names=['Time', 'Price'])
btc.columns = ["Time","Price"]
sent = pd.read_csv('sentiment6-1.txt', names=['Time', 'Sentiment'])
sent.columns = ["Time","Sentiment"]
merged = sent.merge(btc, left_index=False, right_index=False,
how="inner")
merged.to_csv('merged_data.csv')


data = pd.read_csv("merged_data.csv", index_col=False)


with open('merged_data.csv', 'r') as f_input, open('output.csv', 'w') as
f_output:
    csv_output = csv.writer(f_output, delimiter = ',')
    for row in csv.reader(f_input, delimiter = ','):
        if row[2] != 'None' and row[3] != 'None':
            csv_output.writerow(row)



data = pd.read_csv("output.csv", index_col=False, parse_dates=['Time'],
date_parser=dateParser)


data[0:3]


plotly.__version__


#talib.SMA(numpy.asarray(f['Close']), 20)
data['Sma20'] = talib.SMA(np.asarray(data['Price']), 20)
data['Sma5'] = talib.SMA(np.asarray(data['Price']), 5)


# Make plotly work with Jupyter notebook using the plotly.js CDN
py.init_notebook_mode(connected=True)


#btc_trace = go.Scatter(x=data['Time'], y=data['Price'], name= 'Price')
btc_trace = go.Scatter(x=data['Time'], y=data['Price'], name= 'Price')
btc_trace_sma20 = go.Scatter(x=data['Time'], y=data['Sma20'], name=
'Price_sma20')
btc_trace_sma5 = go.Scatter(x=data['Time'], y=data['Sma5'], name=
'Price_sma5')
py.iplot([btc_trace, btc_trace_sma20, btc_trace_sma5])


sent_trace = go.Scatter(x=data['Time'], y=data['Sentiment'], name=
'Sentiment')
py.iplot([sent_trace])

data[0:3]

datag = data[['Price','Sentiment']].groupby(data['Time']).mean()
datag[0:3]

from sklearn.preprocessing import MinMaxScaler
values = datag['Price'].values.reshape(-1,1)
sentiment = datag['Sentiment'].values.reshape(-1,1)
values = values.astype('float32')
sentiment = sentiment.astype('float32')
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)

sentiment[0:3]

sentiment = sentiment.astype('float32')
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)

train_size = int(len(scaled) * 0.7)
test_size = len(scaled) - train_size
train, test = scaled[0:train_size,:], scaled[train_size:len(scaled),:]
#print(len(train), len(test))
```

```
split = train_size


def create_dataset(dataset, look_back, sentiment, sent=False):
    def Tracer():{}
    dataX, dataY = [], []
    for i in range(len(dataset) - look_back):
        if i >= look_back:
            a = dataset[i-look_back:i+1, 0]
            a = a.tolist()
            if(sent==True):
                a.append(sentiment[i].tolist()[0])
            dataX.append(a)
            dataY.append(dataset[i + look_back, 0])
    #print(len(dataY))
    return np.array(dataX), np.array(dataY)


look_back = 1
trainX, trainY = create_dataset(train, look_back, sentiment[0:train_size])
testX, testY = create_dataset(test, look_back,
sentiment[train_size:len(scaled)])


trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))


model = Sequential()
model.add(LSTM(100, input_shape=(trainX.shape[1], trainX.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')

history = model.fit(trainX, trainY, epochs=300, batch_size=100,
validation_data=(testX, testY), verbose=0, shuffle=False)


yhat = model.predict(testX)

pyplot.plot(yhat, label='predict')

pyplot.plot(testY, label='true')

pyplot.legend()

pyplot.show()


yhat_inverse_1 = scaler.inverse_transform(yhat.reshape(-1, 1))

testY_inverse_1 = scaler.inverse_transform(testY.reshape(-1, 1))


rmse_1 = sqrt(mean_squared_error(testY_inverse_1, yhat_inverse_1))

#print('Test RMSE: %.3f' % rmse_1)


model_1 = model


look_back = 2

trainX, trainY = create_dataset(train, look_back, sentiment[0:train_size])

testX, testY = create_dataset(test, look_back,
sentiment[train_size:len(scaled)])


trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))

testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))


model = Sequential()
```

```
model.add(LSTM(100, input_shape=(trainX.shape[1], trainX.shape[2])))

model.add(Dense(1))

model.compile(loss='mae', optimizer='adam')

history = model.fit(trainX, trainY, epochs=300, batch_size=100,
validation_data=(testX, testY), verbose=0, shuffle=False)


yhat = model.predict(testX)

pyplot.plot(yhat, label='predict')

pyplot.plot(testY, label='true')

pyplot.legend()

pyplot.show()


yhat_inverse_2 = scaler.inverse_transform(yhat.reshape(-1, 1))

testY_inverse_2 = scaler.inverse_transform(testY.reshape(-1, 1))


rmse_2 = sqrt(mean_squared_error(testY_inverse_2, yhat_inverse_2))

print('Test RMSE: %.3f' % rmse_2)


model2 = model


look_back = 3

trainX, trainY = create_dataset(train, look_back, sentiment[0:train_size])

testX, testY = create_dataset(test, look_back,
sentiment[train_size:len(scaled)])


trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))

testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

```
model = Sequential()

model.add(LSTM(100, input_shape=(trainX.shape[1], trainX.shape[2])))

model.add(Dense(1))

model.compile(loss='mae', optimizer='adam')

history = model.fit(trainX, trainY, epochs=300, batch_size=100,
validation_data=(testX, testY), verbose=0, shuffle=False)


yhat = model.predict(testX)

pyplot.plot(yhat, label='predict')

pyplot.plot(testY, label='true')

pyplot.legend()

pyplot.show()


yhat_inverse_3 = scaler.inverse_transform(yhat.reshape(-1, 1))

testY_inverse_3 = scaler.inverse_transform(testY.reshape(-1, 1))


rmse_3 = sqrt(mean_squared_error(testY_inverse_3, yhat_inverse_3))

print('Test RMSE: %.3f' % rmse_3)


model3 = model


pyplot.plot(yhat_inverse_1, label='predict_lookup_1')

pyplot.plot(yhat_inverse_2, label='predict_lookup_2')

pyplot.plot(yhat_inverse_3, label='predict_lookup_3')

pyplot.plot(testY_inverse_3, label='true')

pyplot.legend()
pyplot.show()


len(datag.index.values)


#print(len(train), len(test))


btc_1_trace = go.Scatter(x=datag.index.values[len(train)-
len(yhat_inverse_1):], y=yhat_inverse_1.reshape(len(yhat_inverse_1)),
name= 'predict_lookup_1')

btc_2_trace = go.Scatter(x=datag.index.values[len(train)-
len(yhat_inverse_2):], y=yhat_inverse_2.reshape(len(yhat_inverse_2)),
name= 'predict_lookup_2')

btc_3_trace = go.Scatter(x=datag.index.values[len(train)-
len(yhat_inverse_3):], y=yhat_inverse_3.reshape(len(yhat_inverse_3)),
name= 'predict_lookup_3')

btc_t_trace = go.Scatter(x=datag.index.values[len(train)-
len(yhat_inverse_1):], y=testY_inverse_1.reshape(len(yhat_inverse_1)),
name= 'True')

py.iplot([btc_1_trace,btc_2_trace, btc_3_trace, btc_t_trace])


look_back = 2

trainX, trainY = create_dataset(train, look_back,
sentiment[0:train_size],sent=True)

testX, testY = create_dataset(test, look_back,
sentiment[train_size:len(scaled)], sent=True)


trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))

testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))


model = Sequential()
```

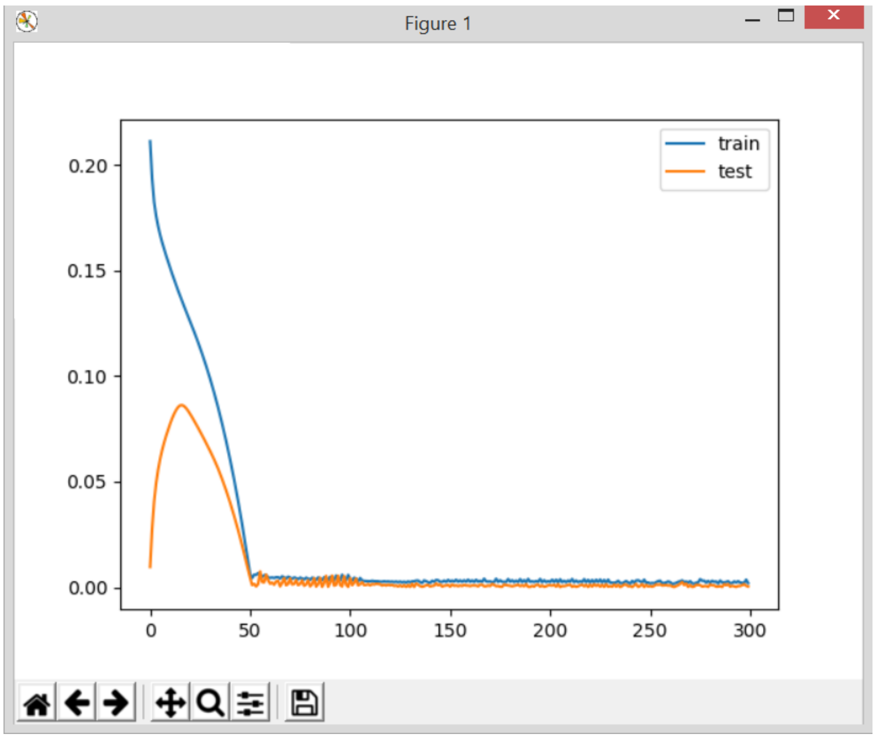
```
model.add(LSTM(100, input_shape=(trainX.shape[1], trainX.shape[2]), return_sequences=True))

model.add(LSTM(100))

model.add(Dense(1))

model.compile(loss='mae', optimizer='adam')

history = model.fit(trainX, trainY, epochs=300, batch_size=100, validation_data=(testX, testY), verbose=0, shuffle=False)


yhat = model.predict(testX)

pyplot.plot(yhat, label='predict')

pyplot.plot(testY, label='true')

pyplot.legend()

pyplot.show()


yhat_inverse_sent = scaler.inverse_transform(yhat.reshape(-1, 1))

testY_inverse_sent = scaler.inverse_transform(testY.reshape(-1, 1))


rmse_sent = sqrt(mean_squared_error(testY_inverse_sent, yhat_inverse_sent))

print('Test RMSE: %.3f' % rmse_sent)


len(yhat)


#print(len(trainX), len(testX))


btc_1_trace = go.Scatter(x=datag.index.values[len(trainX)-len(yhat_inverse_sent):][0:500], y=yhat_inverse_sent.reshape(len(yhat_inverse_sent))[0:500], name= 'With_Sentiment')
```

```
btc_2_trace = go.Scatter(x=datag.index.values[len(trainX)-
len(yhat_inverse_1):][0:500],
y=yhat_inverse_1.reshape(len(yhat_inverse_1))[0:500], name=
'No_Sentiment')

btc_3_trace = go.Scatter(x=datag.index.values[len(trainX)-
len(testY_inverse_sent):][0:500],
y=testY_inverse_sent.reshape(len(testY_inverse_sent))[0:500], name=
'True')

py.iplot([btc_1_trace,btc_2_trace,btc_3_trace])


model_sent = model


import MySQLdb

#Enter the values for you database connection

dsn_database = "bitcoin"         # e.g. "MySQLdbtest"

dsn_hostname = "173.194.231.244"      # e.g.: "mydbinstance.xyz.us-east-
1.rds.amazonaws.com"

dsn_hostname = "localhost"

dsn_port = 3306                 # e.g. 3306

dsn_uid = "root"            # e.g. "user1"

dsn_pwd = "98651sanjay"             # e.g. "Password123"


conn = MySQLdb.connect(host=dsn_hostname, port=dsn_port,
user=dsn_uid, passwd=dsn_pwd, db=dsn_database)


cursor=conn.cursor()

cursor.execute("""SELECT * FROM live_data""")

cursor.fetchone()

print("1st cursor")
```

```
print ("\nShow me the records:\n")
rows = cursor.fetchall()
import pprint
pprint.pprint(rows)

cursor.execute("""
    INSERT INTO live_data
    (time,
    price,
    sentiment)
     values
        (STR_TO_DATE(\'18.2.2018 00:00:00\',\'%d.%m.%Y %H:%i:%s\'),
        2342.23,
        234.34);
    """)

conn.commit()

cursor.close()

import queue
import time
import matplotlib

matplotlib.__version__
```

```
import queue
import matplotlib.pyplot as plt
true_q = queue.Queue()
pred_q = queue.Queue()
'''
fig = plt.figure()
ax = fig.add_subplot(111)
fig.show()
fig.canvas.draw()
plt.ion()
'''

def process_data(in_data):
    out_data = []
    for line in in_data:
        out_data.append(float(line.split(',')[0]))
    return np.array(out_data).reshape(-1,1)
prev = 15000
threshold = 0.5
while True:
    btc = open('live_bitcoin.csv','r')
    sent = open('live_tweet.csv','r')
    bit_data = btc.readlines()
    sent_data = sent.readlines()
    bit_data = process_data(bit_data[len(bit_data)-5:])
    sent_data = process_data(sent_data[len(sent_data)-5:])
    live = scaler.transform(bit_data)
    testX, testY = create_dataset(live, 2, sent_data, sent=True)
    testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
    yhat = model.predict(testX)
    yhat_inverse = scaler.inverse_transform(yhat.reshape(-1, 1))
    true_q.put(bit_data[4])
    pred_q.put(yhat_inverse[0])
    val = 100*((yhat_inverse[0][0] - prev)/prev)
    if val > threshold:
        decision = 'buy'
    elif val <-threshold:
        decision = 'sell'
    else:
        decision = ''
    prev = yhat_inverse[0][0]
    input_string = "INSERT INTO live_data values
({},{},{},'{}','{}');".format(yhat_inverse[0][0],bit_data[0][0],sent_data[4][0
],datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S '),decision)
    cursor.execute(input_string)
    conn.commit()
    time.sleep(60)
    '''
    if true_q.qsize() > 9:
        true_q.get()
        pred_q.get()
        ax.clear()
        ax.plot()
        ax.plot(list(true_q.queue),'-',label='True')
        ax.plot(list(pred_q.queue),'--',label='Predict')
        ax.legend()
        fig.canvas.draw()
        time.sleep(60)
    '''

import datetime
datetime.datetime.now()
```

OUTPUT:



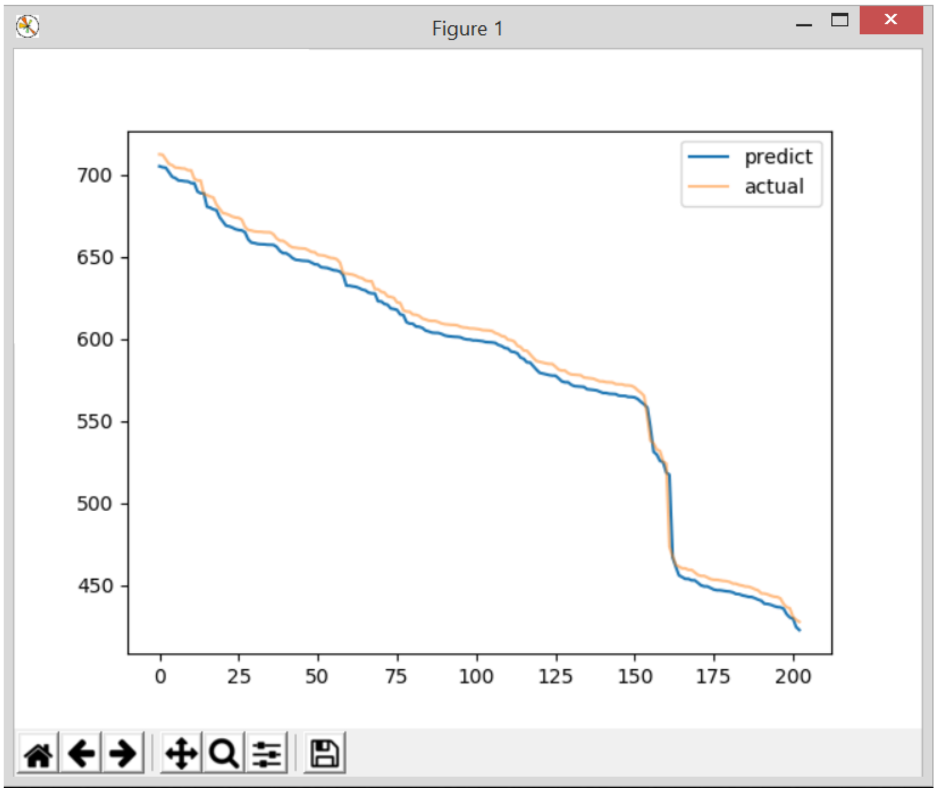


Figure 1
predict
actual
700
650
600
550
500
450
0
25
50
75
100
125
150
175
200

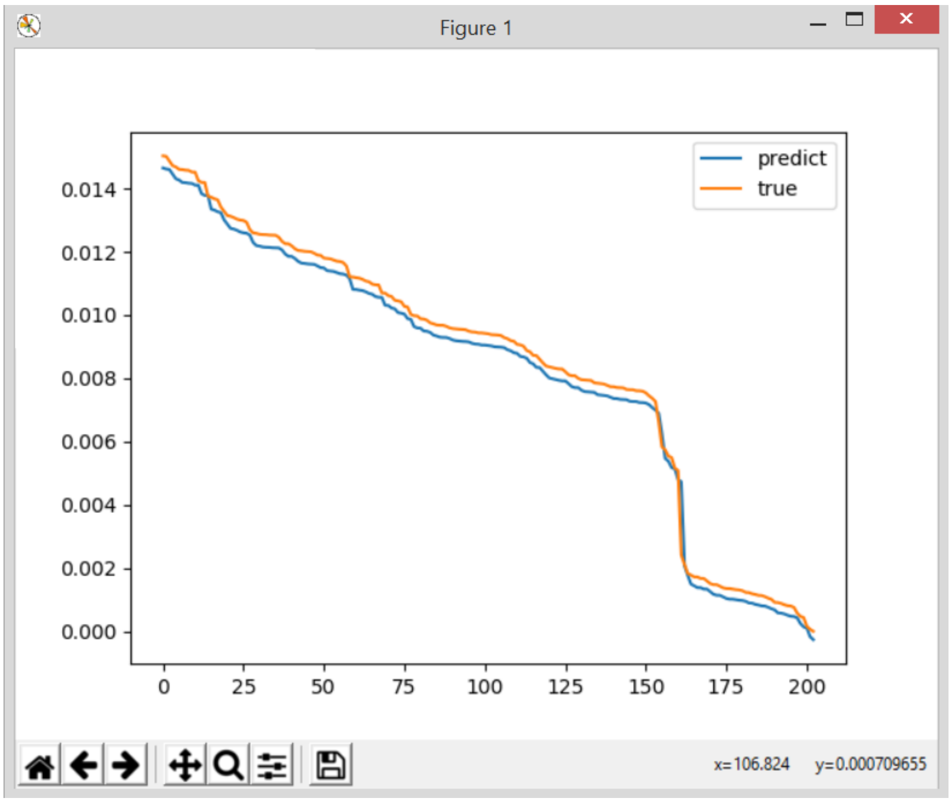


Figure 1
predict
true
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
0
25
50
75
100
125
150
175
200
x=106.824 y=0.000709655

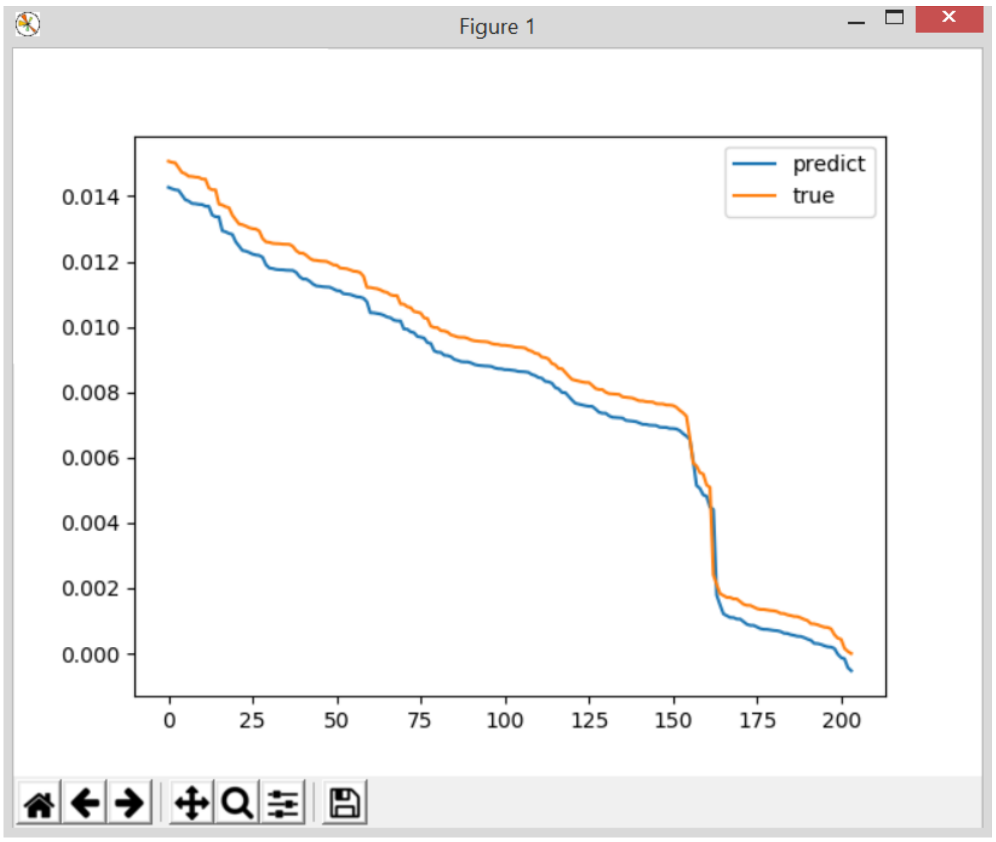


Figure 1
predict
true
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
0
25
50
75
100
125
150
175
200

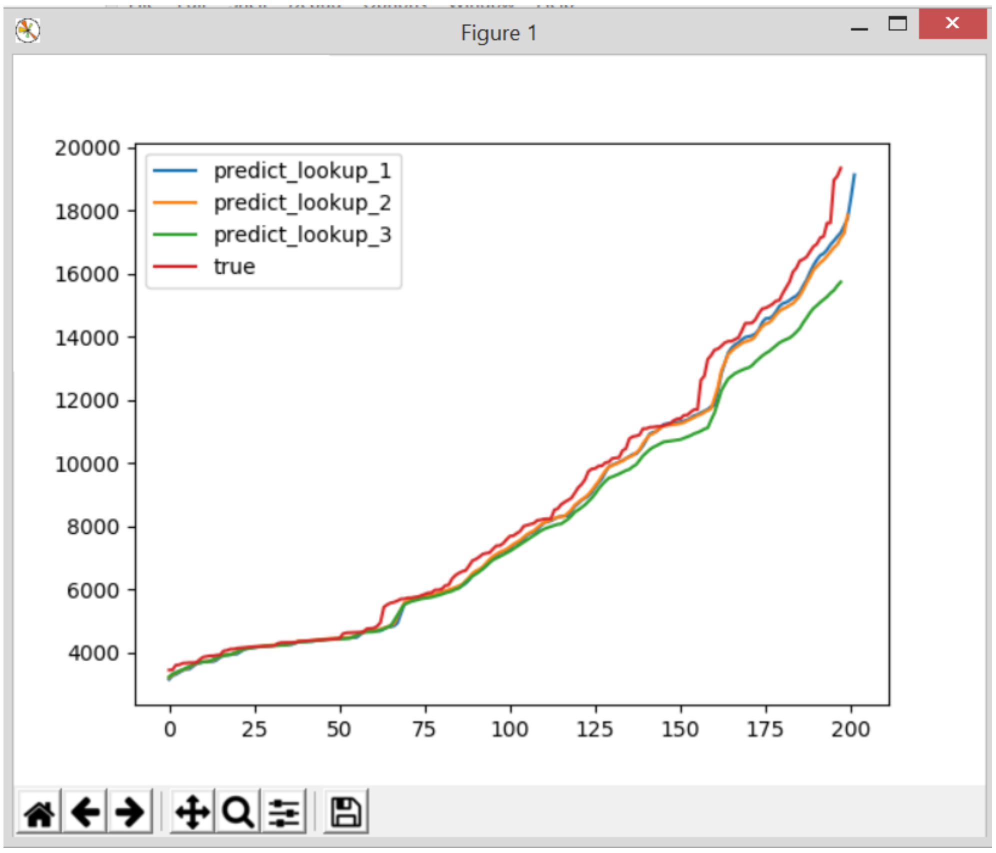


Figure 1
predict_lookup_1
predict_lookup_2
predict_lookup_3
true
20000
18000
16000
14000
12000
10000
8000
6000
4000
0
25
50
75
100
125
150
175
200