**Stock Price Prediction Using LSTM and GRU**

|  |  |
| --- | --- |
| **CONTENT** | **PAGE NO** |
| **Abstract** |  |
| **1.INTRODUCTION** |  |
| 1.1 Motivation |  |
| 1.2 Problem Statement |  |
| 1.3 Objective of the Project |  |
| 1.4 Scope |  |
| 1.5 Project Introduction |  |
| **2.** **2. LITERATURE SURVEY** |  |
| 2.1 Related Work |  |
| **3. SYSTEM ANALYSIS** |  |
| 3.1 Existing System |  |
| 3.2 Disadvantages |  |
| 3.3 Proposed System |  |
| 3.4 Advantages |  |
| 3.5 work Flow of Proposed system |  |
| **4. REQUIREMENT ANALYSIS** |  |
| 4.1Function and non-functional requirements |  |
| 4.2 Hardware Requirements |  |
| 4.3 Software Requirements |  |
| 4.4 Architecture |  |
| **5. SYSTEM DESIGN** |  |
| 5.1 Introduction of Input design |  |
| 5.2 UML Diagram(class, use case, sequence, collaborative, deployment, activity, ER diagram and Component diagram) |  |
| 5.3 Data Flow Diagram |  |
| **6. IMPLEMENTATION** |  |
| 6.1 Modules |  |
| 6.2 Algorithms |  |
| 6.3 Output Screens |  |
| 1. **SYSTEM STUDY AND TESTING** |  |
| 7.1 Feasibility study |  |
| 7.2 Types of test |  |
| **8. CONCLUSION** |  |
| **9. FUTURE ENHANCEMENT** |  |
| **10. REFERENCES** |  |

**ABSTRACT:**

The stock market is a highly complex nonlinear movement system, and its fluctuation law is affected by many factors, so the prediction of the stock price index is a very challenging task. There are many examples showing that Neural Network algorithms are well suited for such time series predictions and often achieve satisfactory results. In this paper, based on the existing models, we proposed a Regularized GRULSTM neural network model and applied it to the short-term forecast of closing price of the two stocks. The experimental results show that our proposed model is superior to the existing GRU and LSTM network models in stock time series prediction.

**Keywords -** LSTM; GRU; Time series prediction, Svm, rf.

1. **INTRODUCTION**

**1.1 Motivation:**

In today's dynamic stock market, predicting price movements is pivotal. With its intricate nonlinear patterns, accurate forecasts remain elusive. Leveraging Neural Network models like LSTM and GRU offers promise. Our study introduces a refined Regularized GRULSTM approach, demonstrating its efficacy in surpassing existing models for short-term stock price predictions**.**

**1.2 Problem Statement:**

The intricate and multifaceted nature of stock market fluctuations presents a formidable challenge in accurately predicting stock price indices. Neural Network algorithms have showcased promise in deciphering such complex time series data, often yielding satisfactory results. However, optimizing existing models remains crucial. This study addresses this by introducing a novel Regularized GRULSTM neural network model for short-term forecasting of stock closing prices. The objective is to enhance predictive accuracy compared to conventional GRU and LSTM models. Through empirical analysis, this research aims to demonstrate the superior performance of the proposed model in stock time series prediction, thereby contributing to more reliable market forecasting methodologies.

**1.3 Objective of the Project:**

The objective of this study is to enhance stock price prediction by proposing a novel Regularized GRULSTM neural network model. Acknowledging the complexity of stock market fluctuations and the challenge of accurate predictions, this research aims to improve upon existing models by leveraging the strengths of both GRU and LSTM networks. Focusing on short-term forecasts for the closing prices of two stocks, the study evaluates the efficacy of the proposed model against conventional GRU and LSTM models. Through empirical experimentation, the aim is to demonstrate the superiority of the Regularized GRULSTM network in stock time series prediction, contributing advancements to this critical field.

**1.4 Scope:**

The scope of the project involves enhancing stock price prediction by introducing a Regularized GRULSTM neural network model, surpassing existing models like GRU and LSTM. This study explores time series prediction, leveraging SVM and RF, aiming for superior short-term forecast accuracy in stock closing prices.

**1.5 Project Introduction:**

The stock market is more popular in recent years due to its high return rates. In spite of high risk, some investors and institutions still choose the stock market to invest. Therefore, the stock price index prediction has attracted the attention of both private and institution investors. In addition to its inherent complexity, there has been an unchanging argument on the predictability of stock returns and various of methods for predicting and modeling stock price index have been object of study of many different subjects, such as physics, economics, computer science and statistics. In 1970, Fama introduced the Efficient-Market hypothesis, which defines that the current price of an asset always reflects all of the previous available information. It is worth mentioning that in 2012, it was estimated that about 85% of the transactions in the US stock market could be carried out by algorithms. Many methods have been used to forecast the stock price index, including traditional models and the recently popular neural network models. The traditional models include Autoregressive Integrated Moving Average(ARIMA), and Autoregressive Conditional Heteroskedasticity(GARCH) volatility. These models are based on the assumption that a linear correlation structure exists among time series values. Therefore, non-linear patterns cannot be captured by these models. To overcome this limitation, neural network models have been widely used in the prediction of nonlinear time series such as stock price index.

Recurrent Neural Network(RNN) have been proved to be one of the most forceful models for processing sequential data, it can recognize complex nonlinear relationships which are difficult to capture using traditional forecasting models, Long Short-Term Memory(LSTM) and Gated Recurrent Unit(GRU) are the two most satisfactory RNN structures. LSTM adopts the memory cell, a unit of calculation, which displaces traditional artificial neuron in the hidden layers of the network. With these memory cells, networks are able to effectively link the memories and the new input, and seize the architecture of data dynamically, which make the prediction more accurate. GRU is very similar to LSTM, the main difference between them is that GRU does not have the output gate as in LSTM. On the basis of the two RNN structures, there are many improved models in recent years, such as the Bi-directional LSTM structure, which has also been used widely.

Note that, LSTM network has a longer memory capacity for preserving and processing the previous information, then, for large data, the LSTM network may derive better results. However, GRU is much faster than LSTM since it has fewer parameters. In this paper, we combined LSTM and GRU, and proposed a new Regularized GRU-LSTM network model with better performance. With this model, we predicted the closing prices of two stocks.

**2. LITERATURE SURVEY**

**[1] E. Fama, “Efficient Capital Markets: A Review of Theory and Empirical Work,” The Journal of Finance. vol. 25, pp. 383-417, February 1970.**

We study a class of permutation tests of the randomness of a collection of Bernoulli sequences and their application to analyses of the human tendency to perceive streaks of consecutive successes as overly representative of positive dependence - the hot hand fallacy. In particular, we study permutation tests of the null hypothesis of randomness (i.e., that trials are i.i.d.) based on test statistics that compare the proportion of successes that directly follow k consecutive successes with either the overall proportion of successes or the proportion of successes that directly follow k consecutive failures. We characterize the asymptotic distributions of these test statistics and their permutation distributions under randomness, under a set of general stationary processes, and under a class of Markov chain alternatives, which allow us to derive their local asymptotic power. The results are applied to evaluate the empirical support for the hot hand fallacy provided by four controlled basketball shooting experiments. We establish that substantially larger data sets are required to derive an informative measurement of the deviation from randomness in basketball shooting. In one experiment, for which we were able to obtain data, multiple testing procedures reveal that one shooter exhibits a shooting pattern significantly inconsistent with randomness - supplying strong evidence that basketball shooting is not random for all shooters all of the time. However, we find that the evidence against randomness in this experiment is limited to this shooter. Our results provide a mathematical and statistical foundation for the design and validation of experiments that directly compare deviations from randomness with human beliefs about deviations from randomness, and thereby constitute a direct test of the hot hand fallacy.

**[2] Kwon, Yung-Keun & Moon, Byung-Ro. (2007). A Hybrid Neurogenetic Approach for Stock Forecasting. IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council. 18. 851-64. 10.1109/TNN.2007.891629.**

In this paper, we propose a hybrid neurogenetic system for stock trading. A recurrent neural network (NN) having one hidden layer is used for the prediction model. The input features are generated from a number of technical indicators being used by financial experts. The genetic algorithm (GA) optimizes the NN's weights under a 2-D encoding and crossover. We devised a context-based ensemble method of NNs which dynamically changes on the basis of the test day's context. To reduce the time in processing mass data, we parallelized the GA on a Linux cluster system using message passing interface. We tested the proposed method with 36 companies in NYSE and NASDAQ for 13 years from 1992 to 2004. The neurogenetic hybrid showed notable improvement on the average over the buy-and-hold strategy and the context-based ensemble further improved the results. We also observed that some companies were more predictable than others, which implies that the proposed neurogenetic hybrid can be used for financial portfolio construction

**[3] P. Franses and H. Ghijsels, “Additive outliers, GARCH and forecasting volatility,” Internatioal Journal of Forecasting. vol. 15, pp. 1-9, January 1999**

The Realized generalized autoregressive conditional heteroskedasticity (GARCH) model proposed by Hansen is often applied to forecast volatility in high-frequency financial data. It is frequently found, however, that the distribution of the estimated residuals from Realized GARCH models has peak fat-tail characteristics. Considering this feature may be a result of neglected additive outliers (AOs) and innovative outliers (IOs), this paper proposes the Realized GARCH model with additive outlier and innovative outlier (Realized GARCH-AI model) for forecasting volatility. This model can detect and correct abnormal returns and realized volatility by estimating the coefficients of volatility models and calculating the outlier test statistics. In the process of simulation, this paper considers different outlier cases in the GARCH model and the Realized GARCH model, and evaluates the performance of the proposed procedure through the accuracy of parameter estimation under different critical values. We find that the critical values will affect the results of outlier detection and correction. When the value is in a suitable range, the proposed procedure based on high-frequency data can obtain unbiased parameter estimation, and the estimation result is close to those of the intervention model containing outlier information. Finally, we use the MCS test proposed by Hansen et al. to study the volatility prediction accuracy, value at risk, and expected shortfall prediction ability of the new model. The empirical analysis demonstrates that the proposed model produces better prediction effects than the traditional Realized GARCH model.

**[4] Ferreira, A. E. Tiago, G. C. Vasconcelos, and P. J. L. Adeodato, “A New Intelligent System Methodology for Time Series Forecasting with Artificial Neural Networks,” Neural Processing Letters. vol. 28, pp. 113-129, February 2008**

The Time-delay Added Evolutionary Forecasting (TAEF) approach is a new method for time series prediction that performs an evolutionary search for the minimum number of dimensions necessary to represent the underlying information that generates the time series. The methodology proposed is inspired in Takens theorem and consists of an intelligent hybrid model composed of an artificial neural network combined with a modified genetic algorithm. Initially, the TAEF method finds the best fitted model to forecast the series and then performs a behavioral statistical test in order to adjust time phase distortions that may appear in the representation of some series. An experimental investigation conducted with relevant time series show the robustness of the method through a comparison, according to several performance measures, to previous results found in the literature and those obtained with more traditional methods.

**[5] Jigar Patel, Sahil Shah, Priyank Thakkar, K Kotecha, Predicting stock and stock price index movement using Trend Deterministic Data Preparation and machine learning techniques, Expert Systems with Applications, Volume 42, Issue 1, 2015, Pages 259-268.**

This paper addresses problem of predicting direction of movement of stock and stock price index for Indian stock markets. The study compares four prediction models, Artificial Neural Network (ANN), Support Vector Machine (SVM), random forest and naive-Bayes with two approaches for input to these models. The first approach for input data involves computation of ten technical parameters using stock trading data (open, high, low & close prices) while the second approach focuses on representing these technical parameters as trend deterministic data. Accuracy of each of the prediction models for each of the two input approaches is evaluated. Evaluation is carried out on 10 years of historical data from 2003 to 2012 of two stocks namely Reliance Industries and Infosys Ltd. and two stock price indices CNX Nifty and S&P Bombay Stock Exchange (BSE) Sensex. The experimental results suggest that for the first approach of input data where ten technical parameters are represented as continuous values, random forest outperforms other three prediction models on overall performance. Experimental results also show that the performance of all the prediction models improve when these technical parameters are represented as trend deterministic data.

**3. SYSTEM ANALYSIS**

**3.1 Existing System**

In existing system, we tend to propose that a company’s performance, in terms of its stock worth movement, is foreseen by internal communication patterns. to get early warning signals, we tend to believe that it’s vital for patterns in company communication networks to be detected earlier for the pre- diction of serious stock worth movement to avoid attainable adversities that an organization could face within the securities market in order that stakeholders’ interests is protected the maximum amount as attainable. Despite the potential importance of such data regarding corporate communication, very little work has been tired this vital direction. We attempt to bridge these research gaps by employing a data-mining method to examine the linkage between a firm’s communication data and its share price

**3.2 Disadvantages:**

**1. Overfitting Concerns:** Neural networks, including LSTM and GRU models, can be prone to overfitting due to their high complexity, especially when dealing with limited data, leading to inaccurate predictions in real-world scenarios.

**2. Difficulty in Interpreting Results:** These models often operate as black boxes, making it challenging to interpret how and why certain predictions are made, which could limit their practical application in financial decision-making.

**3. Sensitivity to Hyperparameters:** Fine-tuning the hyperparameters of LSTM and GRU networks is crucial. However, finding the optimal configuration can be a time-consuming and iterative process, impacting the model's performance and efficiency.

**3.3 Proposed System:**

Accuracy plays an important role in stock market prediction. Although many algorithms are available for this purpose, selecting the most accurate one continues to be the fundamental task in getting the best results. In order to achieve this, in this paper we have compared and analysed the performance of various available algorithms such as LSTM ,GRU ,Svm, rf. This involves training the algorithms, executing them, getting the results, comparing various performance parameters of these algorithms and finally obtaining the most accurate one.

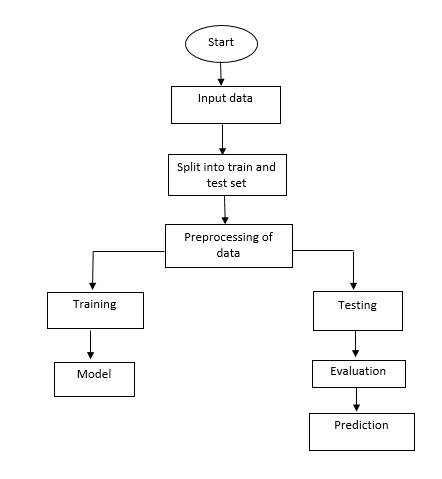
**3.4** **Advantages**:

**1. Enhanced Accuracy:** The Regularized GRU-LSTM model offers heightened precision in stock price forecasting by effectively capturing intricate nonlinear patterns in market movements, outperforming conventional LSTM and GRU models.

**2. Improved Generalization:** Its regularization techniques enable better generalization, reducing overfitting and enhancing the model's adaptability to diverse market conditions, leading to more reliable predictions.

**3. Robustness and Adaptability:** The proposed model demonstrates robustness against fluctuations and adapts adeptly to varying market dynamics, showcasing its resilience and efficacy in short-term stock price prediction.

**3.5 Block Diagram:**



**4. REQUIREMENT ANALYSIS**

**4.1 Functional and non-functional requirements**

Requirement’s analysis is very critical process that enables the success of a system or software project to be assessed. Requirements are generally split into two types: Functional and non-functional requirements.

**Functional Requirements**: These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

Examples of functional requirements:

1. Authentication of user whenever he/she logs into the system
2. System shutdown in case of a cyber-attack
3. A verification email is sent to user whenever he/she register for the first time on some software system.

**Non-functional requirements**: These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements.  
They basically deal with issues like:

* Portability
* Security
* Maintainability
* Reliability
* Scalability
* Performance
* Reusability
* Flexibility

Examples of non-functional requirements:

1. Emails should be sent with a latency of no greater than 12 hours from such an activity.
2. The processing of each request should be done within 10 seconds
3. The site should load in 3 seconds whenever of simultaneous users are > 10000

**SOFTWARE AND HARDWARE REQUIREMENTS:**

**4.2 Hardware:**

Operating system : Windows 7 or 7+

RAM : 8 GB

Hard disc or SSD : More than 500 GB

Processor : Intel 3rd generation or high or Ryzen with 8 GB Ram

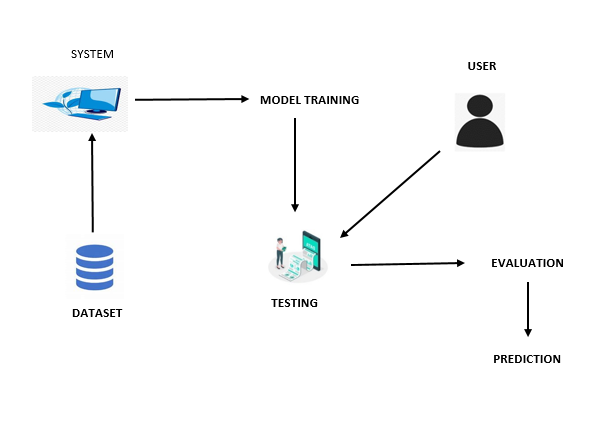
**4.3 Software:**

Software’s : Python 3.6 or high version

IDE : PyCharm, Google collab

Framework : Flask

**4.4 Architectural Diagram:**

****

**5. SYSTEM DESIGN**

**5.1 Introduction of Input Design:**

In an information system, input is the raw data that is processed to produce output. During the input design, the developers must consider the input devices such as PC, MICR, OMR, etc.

Therefore, the quality of system input determines the quality of system output. Well-designed input forms and screens have following properties −

* It should serve specific purpose effectively such as storing, recording, and retrieving the information.
* It ensures proper completion with accuracy.
* It should be easy to fill and straightforward.
* It should focus on user’s attention, consistency, and simplicity.
* All these objectives are obtained using the knowledge of basic design principles regarding −
  + What are the inputs needed for the system?
  + How end users respond to different elements of forms and screens.

### **Objectives for Input Design:**

The objectives of input design are −

* To design data entry and input procedures
* To reduce input volume
* To design source documents for data capture or devise other data capture methods
* To design input data records, data entry screens, user interface screens, etc.
* To use validation checks and develop effective input controls.

**Output Design:**

The design of output is the most important task of any system. During output design, developers identify the type of outputs needed, and consider the necessary output controls and prototype report layouts.

### **Objectives of Output Design:**

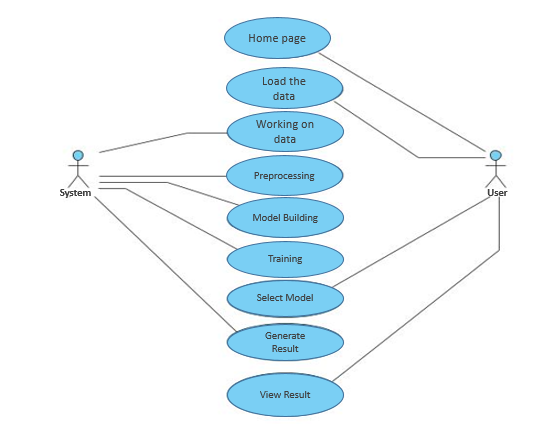
The objectives of input design are:

* To develop output design that serves the intended purpose and eliminates the production of unwanted output.
* To develop the output design that meets the end user’s requirements.
* To deliver the appropriate quantity of output.
* To form the output in appropriate format and direct it to the right person.
* To make the output available on time for making good decisions.

**5.2 UML Diagrams:**

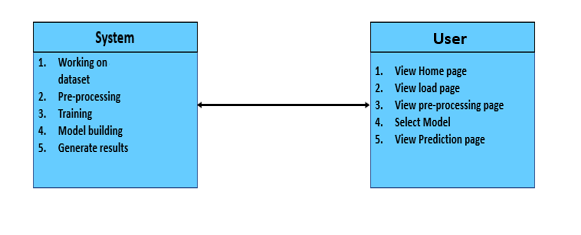
**USECASE DIAGRAM:**

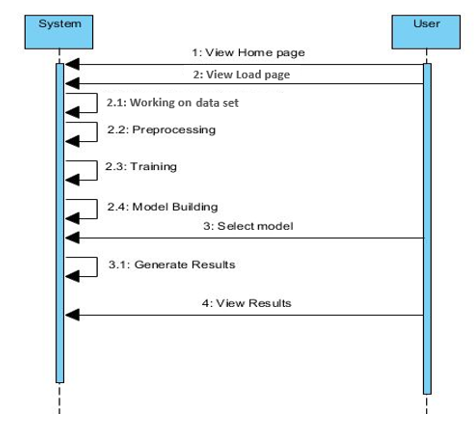
A use case diagram in the Unified Modelling Language (UML) is a type of behavioural diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted



**CLASS DIAGRAM:**

In software engineering, a class diagram in the Unified Modelling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

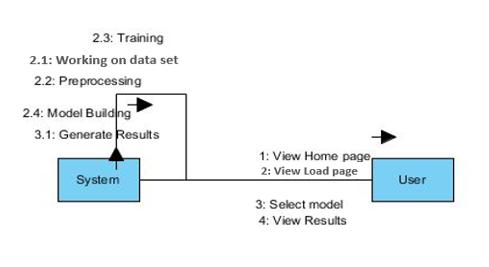


**SEQUENCE DIAGRAM:**

A sequence diagram in Unified Modelling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams

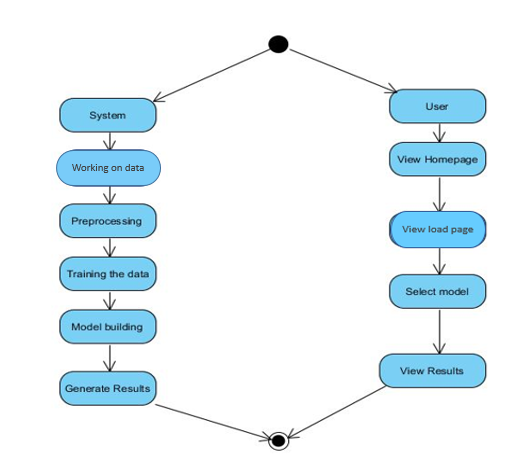
**COLLABORATION DIAGRAM:**

In collaboration diagram the method call sequence is indicated by some numbering technique as shown below. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram. The method calls are similar to that of a sequence diagram. But the difference is that the sequence diagram does not describe the object organization whereas the collaboration diagram shows the object organization.

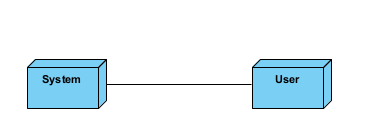


**ACTIVITY DIAGRAM:**

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modelling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control

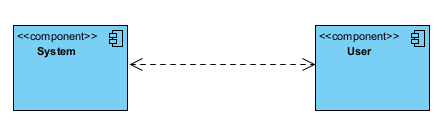


**DEPLOYMENT DIAGRAM**

Deployment diagram represents the deployment view of a system. It is related to the component diagram. Because the components are deployed using the deployment diagrams. A deployment diagram consists of nodes. Nodes are nothing but physical hard wares used to deploy the application

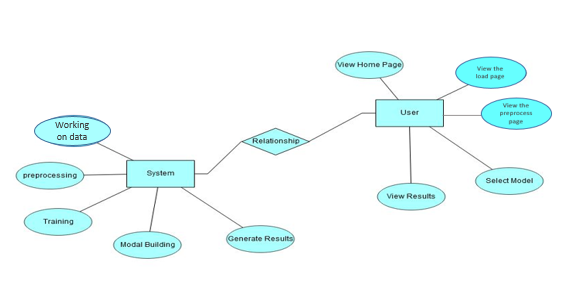
**COMPONENT DIAGRAM**

A component diagram, also known as a UML component diagram, describes the organization and wiring of the physical components in a system. Component diagrams are often drawn to help model implementation details and double-check that every aspect of the system's required functions is covered by planned development.



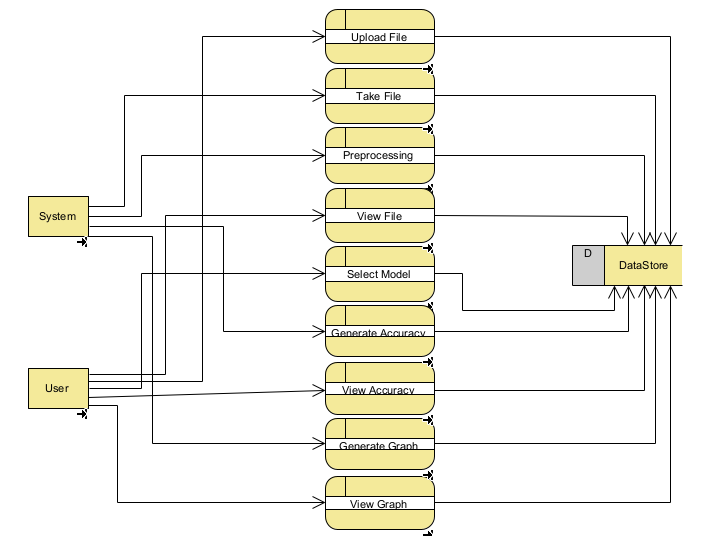
**ER DIAGRAM:**

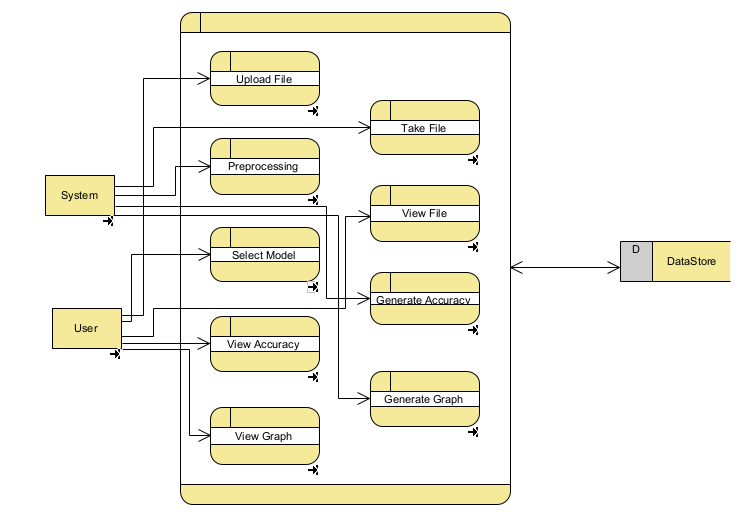
An Entity–relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (ER Diagram). An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are: entity set and relationship set. An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database. Let’s have a look at a simple ER diagram to understand this concept.

5.3

**5.3 DFD DIAGRAM:**

A Data Flow Diagram (DFD) is a traditional way to visualize the information flows within a system. A neat and clear DFD can depict a good amount of the system requirements graphically. It can be manual, automated, or a combination of both. It shows how information enters and leaves the system, what changes the information and where information is stored. The purpose of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communications tool between a systems analyst and any person who plays a part in the system that acts as the starting point for redesigning a system.





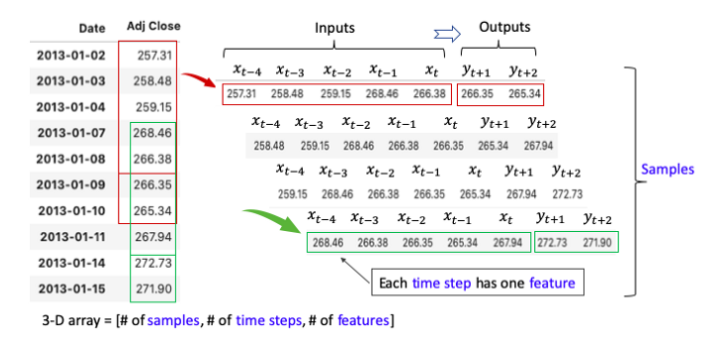
**6. IMPLEMENTATION AND RESULTS**

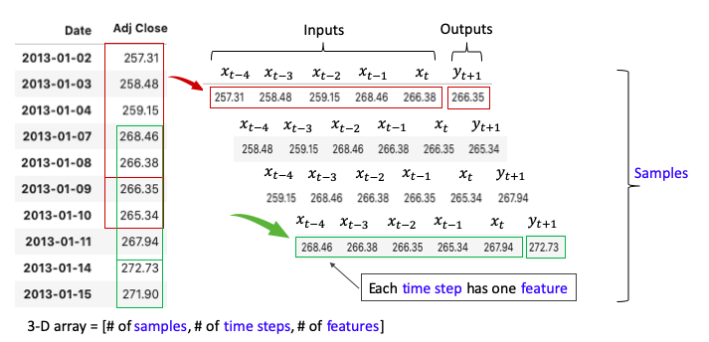
**6.1 MODULES**

**Importing Data:** Here we have collected the amazon stock price data form 2013 to 2022 August month. And then we checked the noisiness in our data and we found that the data is noise free and then we performed some Exploratory Data Analysis on our data.

**Create the Training Data for GRU/LSTM:** LSTM/GRU are supervised ML techniques. We need to create inputs and outputs for model training. Let me explain the data structure by creating the training data from a uni-variate stock market price time series. There are two popular data structures: many-to-many and many-to-one. The many-to-many is more interesting. It means we can forecast many periods in the future.

**Many-to-many:** We are interested in using the prices of the past X days to forecast those of the future Y days. For the sake of illustration, let me use the prices of only 5 days to forecast the prices for the next 2 days. There are multiple inputs (5 data points) and multiple outputs (2 data points). This data structure is called many-to-many. From the below figure we can see that it creates samples from the univariate time series as the red window moves along the series. Each sample has 5 inputs and 2 outputs. Each input of a sample is called the time step, and each time step has one number, called a feature. The number of features can be multiple. For example, if we model both the “Adj. Close” and “Open” prices together in a time step, there are two features. Here we just model “Adj. Close” so the number of features is one.

**Many-to-one:** The below figure shows that the case when there is only one output. This is called many-to-one.

****

**LSTM/GRU Requires a 3-D Array as the Inputs:**

The three dimensions are:

Tensor: One tensor is a vector that enters the model

Time Step: One-time step is one observation in the tensor.

Feature: One feature is one observation at a time step.

The 1-D array above should be converted to a 3-D array =

[# of samples, # of time steps, # of features].

**What Are the “Units” in the LSTM/GRU Layer of Keras?**

The official Keras’ description for “units” is “dimensionality of the output space”. I still think it is not easy to understand. (Some online sources confuse it with the number of time steps in a sample.) I feel it may be better to name the “units” as “latent dimension” or “latent\_dim”. This renaming at least implies that the dimensionality is internal and has nothing to do with the outside parameters. The dimension of the hidden layer is the “units”. Since there are hidden layers, we need to specify the number of neurons for the hidden layers. The hidden dimension can be any number. The hidden dimensionality determines the capability of RNN to retain the memory for all the past information. It is usually not a small number and is conventionally the multiple of 32, such as 32, 64, or 128.

**Why Do We Need LSTM/GRU?**

The optimizer of RNN gets the first-order derivative of the loss function to search for the optimal values. Because RNN is recursive, the first-order derivation process will make a number smaller and smaller, then eventually vanish. This is called gradient vanishing. This certain mathematical process makes RNN not a good choice to retain memories. **We need a recursive structure so that the information does not vanish quickly. This is the motive for LSTM and GRU.** (For readers who may not be familiar with the optimization process: A loss function is a metric that measures the errors between the actual and the predicted values. An optimizer is an algorithm that changes the weights of the neurons to pursue the minimum error. A popular optimizer is the Stochastic Gradient Descent (SGD).

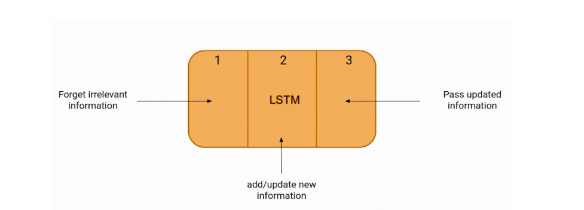
**6.2 ALGORITHMS**

**1. Long Short Term Memory (LSTM)**

**L**ong Short Term Memory Network is an advanced RNN, a sequential network, that allows information to persist. It is capable of handling the vanishing gradient problem faced by RNN. A recurrent neural network is also known as RNN is used for persistent memory.

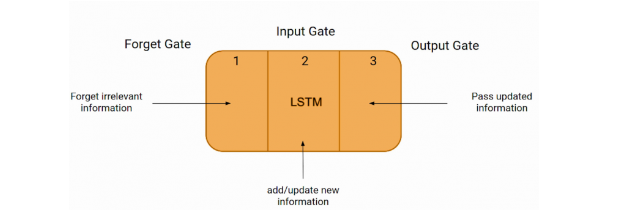
Let’s say while watching a video you remember the previous scene or while reading a book you know what happened in the earlier chapter. Similarly RNNs work, they remember the previous information and use it for processing the current input. The shortcoming of RNN is, they can not remember Long term dependencies due to vanishing gradient. LSTMs are explicitly designed to avoid long-term dependency problems.

At a high-level LSTM works very much like an RNN cell. Here is the internal functioning of the LSTM network. The LSTM  consists of three parts, as shown in the image below and each part performs an individual function.

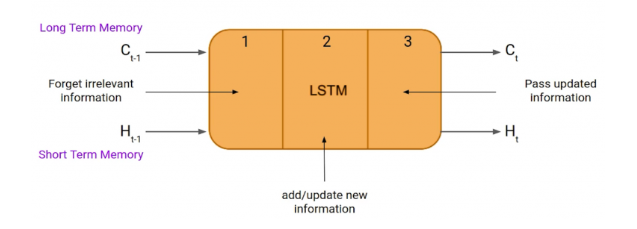


The first part chooses whether the information coming from the previous timestamp is to be remembered or is irrelevant and can be forgotten. In the second part, the cell tries to learn new information from the input to this cell. At last, in the third part, the cell passes the updated information from the current timestamp to the next timestamp.

These three parts of an LSTM cell are known as gates. The first part is called **Forget gate, the**second part is known as **the Input gate** and the last one is **the Output gate**.



Just like a simple RNN, an LSTM also has a hidden state where H(t-1) represents the hidden state of the previous timestamp and Ht is the hidden state of the current timestamp. In addition to that LSTM also have a cell state represented by C(t-1) and C(t) for previous and current timestamp respectively. Here the hidden state is known as Short term memory and the cell state is known as Long term memory. Refer to the following image.



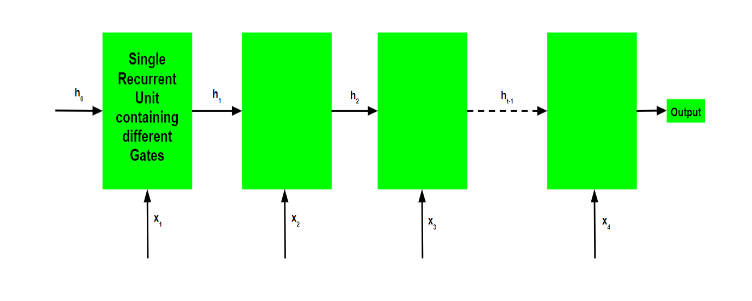
**2. Gated Recurrent Unit (GRU)**

To solve the Vanishing-Exploding gradients problem often encountered during the operation of a basic Recurrent Neural Network, many variations were developed. One of the most famous variations is the **Long Short Term Memory Network (LSTM)**. One of the lesser-known but equally effective variations is the **Gated Recurrent Unit Network (GRU)**.

Unlike LSTM, it consists of only three gates and does not maintain an Internal Cell State. The information which is stored in the Internal Cell State in an LSTM recurrent unit is incorporated into the hidden state of the Gated Recurrent Unit. This collective information is passed onto the next Gated Recurrent Unit. The different gates of a GRU are as described below:-

1. **Update Gate (z):** It determines how much of the past knowledge needs to be passed along into the future. It is analogous to the Output Gate in an LSTM recurrent unit.
2. **Reset Gate(r):** It determines how much of the past knowledge to forget. It is analogous to the combination of the Input Gate and the Forget Gate in an LSTM recurrent unit.
3. **Current Memory Gate ():** It is often overlooked during a typical discussion on Gated Recurrent Unit Network. It is incorporated into the Reset Gate just like the Input Modulation Gate is a sub-part of the Input Gate and is used to introduce some non-linearity into the input and to also make the input Zero-mean. Another reason to make it a sub-part of the Reset gate is to reduce the effect that previous information has on the current information that is being passed into the future.

The basic work-flow of a Gated Recurrent Unit Network is similar to that of a basic Recurrent Neural Network when illustrated, the main difference between the two is in the internal working within each recurrent unit as Gated Recurrent Unit networks consist of gates which modulate the current input and the previous hidden state.

****

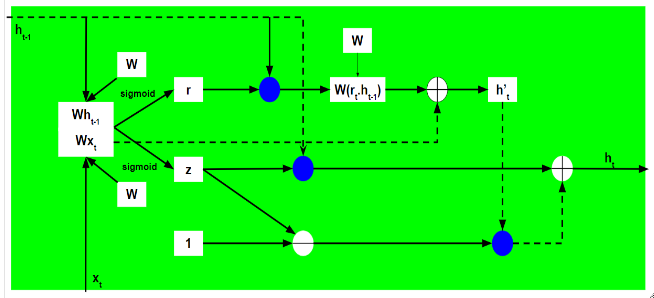
**Working of a Gated Recurrent Unit:**

* Take input the current input and the previous hidden state as vectors.
* Calculate the values of the three different gates by following the steps given below:-
  1. For each gate, calculate the parameterized current input and previously hidden state vectors by performing element-wise multiplication (Hadamard Product) between the concerned vector and the respective weights for each gate.
  2. Apply the respective activation function for each gate element-wise on the parameterized vectors. Below given is the list of the gates with the activation function to be applied for the gate.

**Update Gate : Sigmoid Function**

**Reset Gate : Sigmoid Function**

* The process of calculating the Current Memory Gate is a little different. First, the Hadamard product of the Reset Gate and the previously hidden state vector is calculated. Then this vector is parameterized and then added to the parameterized current input vector.
* To calculate the current hidden state, first, a vector of ones and the same dimensions as that of the input is defined. This vector will be called ones and mathematically be denoted by 1. First, calculate the Hadamard Product of the update gate and the previously hidden state vector. Then generate a new vector by subtracting the update gate from ones and then calculate the Hadamard Product of the newly generated vector with the current memory gate. Finally, add the two vectors to get the currently hidden state vector.



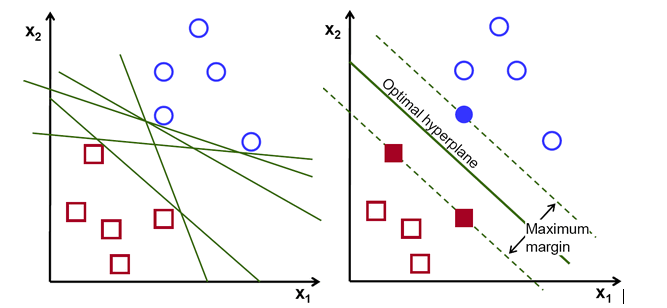
Note that the blue circles denote element-wise multiplication. The positive sign in the circle denotes vector addition while the negative sign denotes vector subtraction(vector addition with negative value). The weight matrix W contains different weights for the current input vector and the previous hidden state for each gate.

Just like Recurrent Neural Networks, a GRU network also generates an output at each time step and this output is used to train the network using gradient descent.

The optimizer of RNN gets the first-order derivative of the loss function to search for the optimal values. Because RNN is recursive, the first-order derivation process will make a number smaller and smaller, then eventually vanish. This is called gradient vanishing. This certain mathematical process makes RNN not a good choice to retain memories. We need a recursive structure so that the information does not vanish quickly. This is the motive for LSTM and GRU. Overfitting is a serious sin in machine learning. When you train a model on your training data and apply it to the test data, the accuracy of the test data usually is less than that of the training data. We know this is because the model has fitted the training data too well, including the noises in the training data. Deep learning uses the dropout technique to control over fitting. The dropout technique randomly drops or deactivates some neurons for a layer during each iteration. It is like some weights are set to zero. So in each iteration, the model looks at a slightly different structure of itself to optimize the model We have collected the Amazon stock price data from the yahoo financial.as we have mentioned once the data collection part is done then we will go for data pre-processing, here we are going to make our data noise free. When we will collect the sometimes the data will have more noise like outliers, missing values, scale difference and textual data, so here in pre-processing we will make our data noise free. As we know that the time series data is more sensitive we have to scale the data before model training and we have used standard scalar to scale the data and also we have to reshape the data into three dimensional data because of LSTM model. After that we will split our data into training and testing at some proportion. Here we have used the simple LSTM and GRU and to improve the accuracy we then used the stacked LSTM and GRU with regularization.

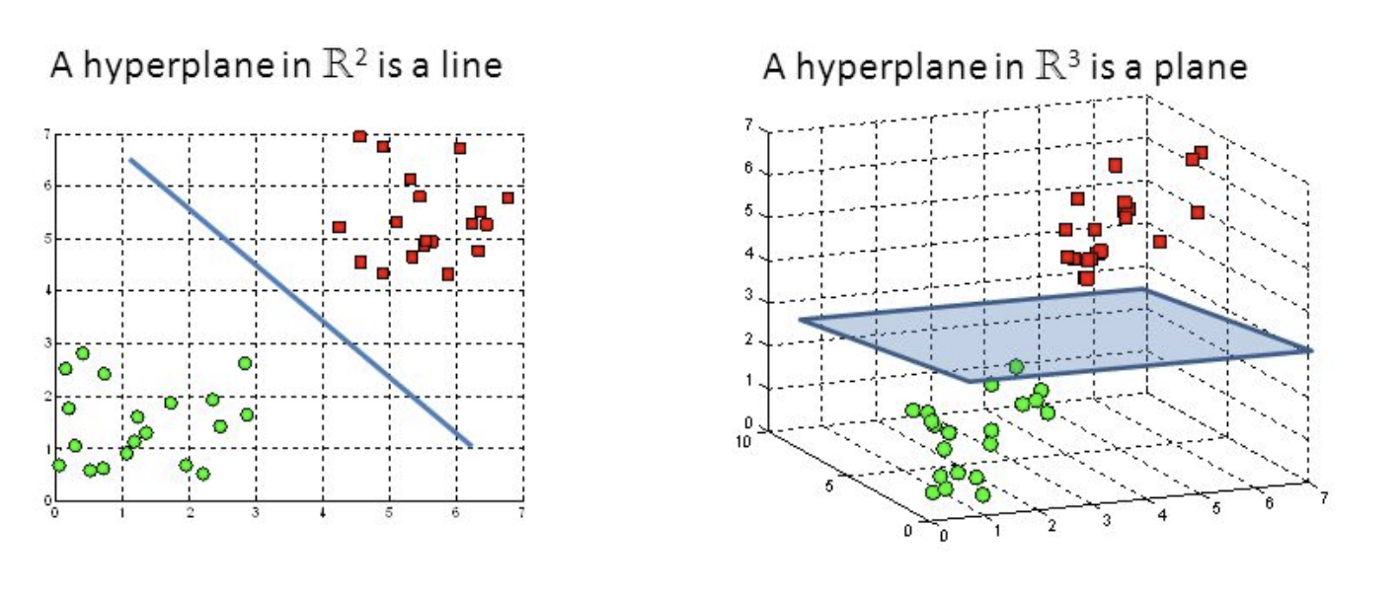
**SUPPORT VECTOR MACHINES:**

The objective of the support vector machine algorithm is to find a hyper plane in an N-dimensional space (N — the number of features) that distinctly classifies the data points.

**Possible hyper planes :**

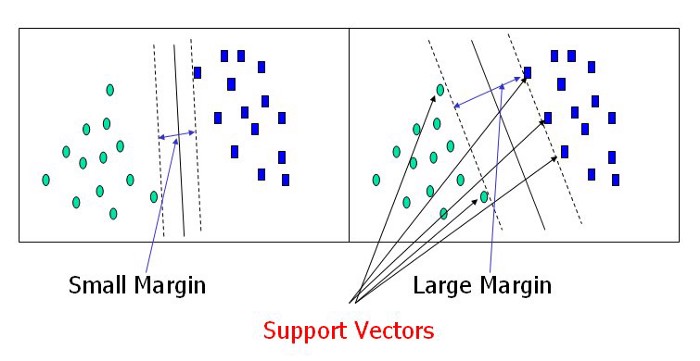
To separate the two classes of data points, there are many possible Hyper planes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e. the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

## Hyper planes and Support Vectors



**Hyper planes in 2D and 3D feature space**

Hyper planes are decision boundaries that help classify the data points. Data points falling on either side of the hyper plane can be attributed to different classes. Also, the dimension of the hyper plane depends upon the number of features. If the number of input features is 2, then the hyper plane is just a line. If the number of input features is 3, then the hyper plane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds



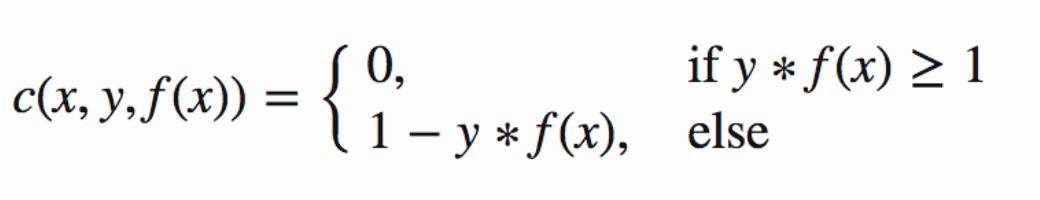
**Support Vectors**

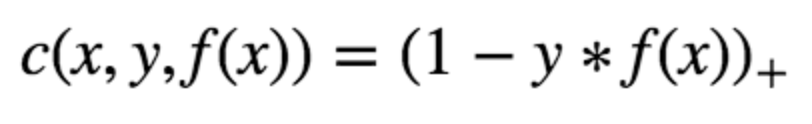
Support vectors are data points that are closer to the hyper plane and influence the position and orientation of the hyper plane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyper plane. These are the points that help us build our SVM.

**Large Margin Intuition**

In logistic regression, we take the output of the linear function and squash the value within the range of [0,1] using the sigmoid function. If the squashed value is greater than a threshold value (0.5) we assign it a label 1, else we assign it a label 0. In SVM, we take the output of the linear function and if that output is greater than 1, we identify it with one class and if the output is -1, we identify is with another class. Since the threshold values are changed to 1 and -1 in SVM, we obtain this reinforcement range of values ([-1, 1]) which acts as margin.

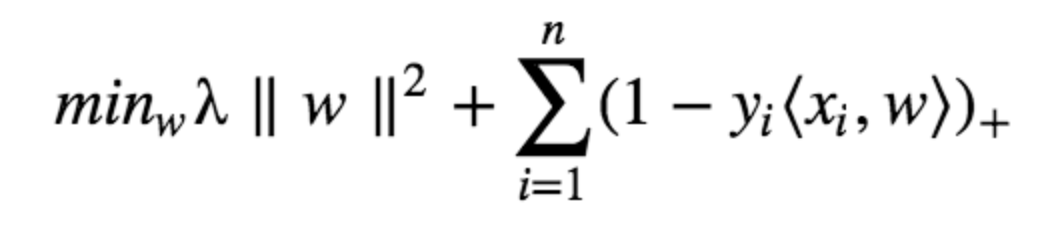
**Cost Function and Gradient Updates**

In the SVM algorithm, we are looking to maximize the margin between the data points and the hyper plane. The loss function that helps maximize the margin is hinge loss.



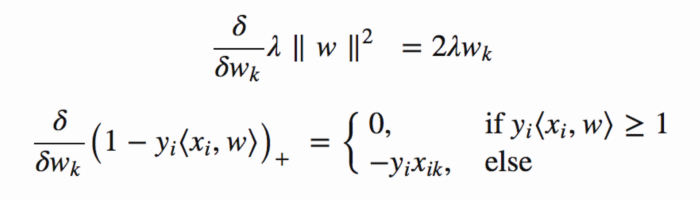
Hinge loss function (function on left can be represented as a function on the right)

The cost is 0 if the predicted value and the actual value are of the same sign. If they are not, we then calculate the loss value. We also add a regularization parameter the cost function. The objective of the regularization parameter is to balance the margin maximization and loss. After adding the regularization parameter, the cost functions looks as below.



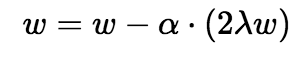
**Loss function for SVM**

Now that we have the loss function, we take partial derivatives with respect to the weights to find the gradients. Using the gradients, we can update our weights.



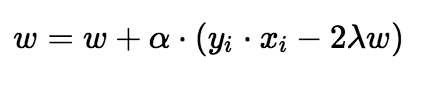
Gradients

When there is no misclassification, i.e. our model correctly predicts the class of our data point, we only have to update the gradient from the regularization parameter.



Gradient Update — No misclassification

When there is a misclassification, i.e. our model make a mistake on the prediction of the class of our data point, we include the loss along with the regularization parameter to perform gradient update.



**Random Forest:**

A random forest is a machine learning technique that’s used to solve regression and classification problems. It utilizes ensemble learning, which is a technique that combines many classifiers to provide solutions to complex problems.

A random forest algorithm consists of many decision trees. The ‘forest’ generated by the random forest algorithm is trained through bagging or bootstrap aggregating. Bagging is an ensemble meta-algorithm that improves the accuracy of machine learning algorithms.

The (random forest) algorithm establishes the outcome based on the predictions of the decision trees. It predicts by taking the average or mean of the output from various trees. Increasing the number of trees increases the precision of the outcome.

A random forest eradicates the limitations of a decision tree algorithm. It reduces the over fitting of datasets and increases precision. It generates predictions without requiring many configurations in packages (like [Scikit-learn](https://en.wikipedia.org/wiki/Scikit-learn)).

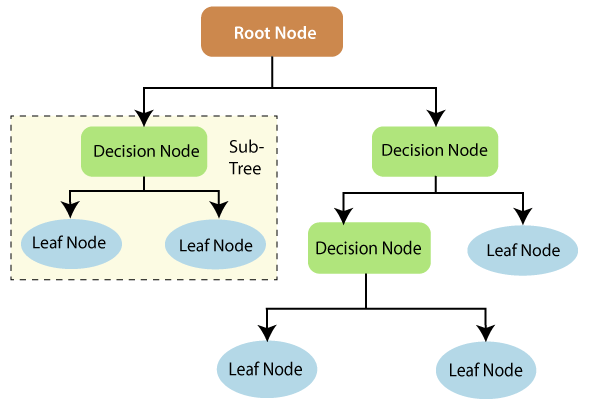
Features of a Random Forest Algorithm:

* It’s more accurate than the decision tree algorithm.
* It provides an effective way of handling missing data.
* It can produce a reasonable prediction without hyper-parameter tuning.
* It solves the issue of over fitting in decision trees.
* In every random forest tree, a subset of features is selected randomly at the node’s splitting point.

Decision trees are the building blocks of a random forest algorithm. A decision tree is a decision support technique that forms a tree-like structure. An overview of decision trees will help us understand how random forest algorithms work.

A decision tree consists of three components: decision nodes, leaf nodes, and a root node. A decision tree algorithm divides a training dataset into branches, which further segregate into other branches. This sequence continues until a leaf node is attained. The leaf node cannot be segregated further.

The nodes in the decision tree represent attributes that are used for predicting the outcome. Decision nodes provide a link to the leaves. The following diagram shows the three types of nodes in a decision tree.



The information theory can provide more information on how decision trees work. Entropy and information gain are the building blocks of decision trees. An overview of these fundamental concepts will improve our understanding of how decision trees are built.

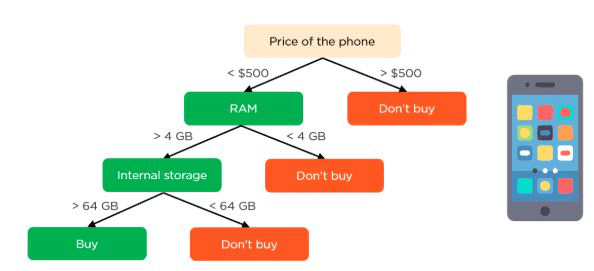
Entropy is a metric for calculating uncertainty. Information gain is a measure of how uncertainty in the target variable is reduced, given a set of independent variables.

The information gain concept involves using independent variables (features) to gain information about a target variable (class). The entropy of the target variable (Y) and the [conditional entropy](https://en.wikipedia.org/wiki/Conditional_entropy) of Y (given X) are used to estimate the information gain. In this case, the conditional entropy is subtracted from the entropy of Y.

Information gain is used in the training of decision trees. It helps in reducing uncertainty in these trees. A high information gain means that a high degree of uncertainty (information entropy) has been removed. Entropy and information gain are important in splitting branches, which is an important activity in the construction of decision trees.

Let’s take a simple example of how a decision tree works. Suppose we want to predict if a customer will purchase a mobile phone or not. The features of the phone form the basis of his decision. This analysis can be presented in a decision tree diagram.

The root node and decision nodes of the decision represent the features of the phone mentioned above. The leaf node represents the final output, either *buying* or *not buying*. The main features that determine the choice include the price, internal storage, and Random Access Memory (RAM). The decision tree will appear as follows.



Applying decision trees in random forest

The main difference between the decision tree algorithm and the random forest algorithm is that establishing root nodes and segregating nodes is done randomly in the latter. The random forest employs the bagging method to generate the required prediction.

Bagging involves using different samples of data (training data) rather than just one sample. A training dataset comprises observations and features that are used for making predictions. The decision trees produce different outputs, depending on the training data fed to the random forest algorithm. These outputs will be ranked, and the highest will be selected as the final output.

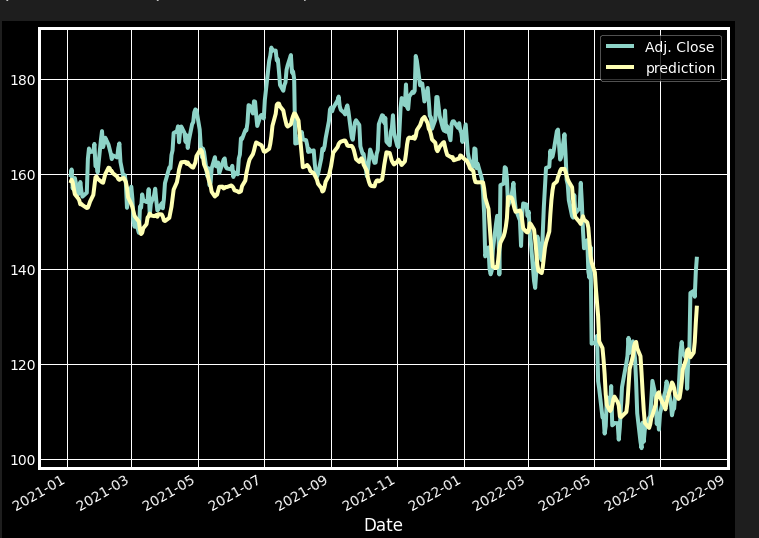
Our first example can still be used to explain how random forests work. Instead of having a single decision tree, the random forest will have many decision trees. Let’s assume we have only four decision trees. In this case, the training data comprising the phone’s observations and features will be divided into four root nodes.

The root nodes could represent four features that could influence the customer’s choice (price, internal storage, camera, and RAM). The random forest will split the nodes by selecting features randomly. The final prediction will be selected based on the outcome of the four trees.

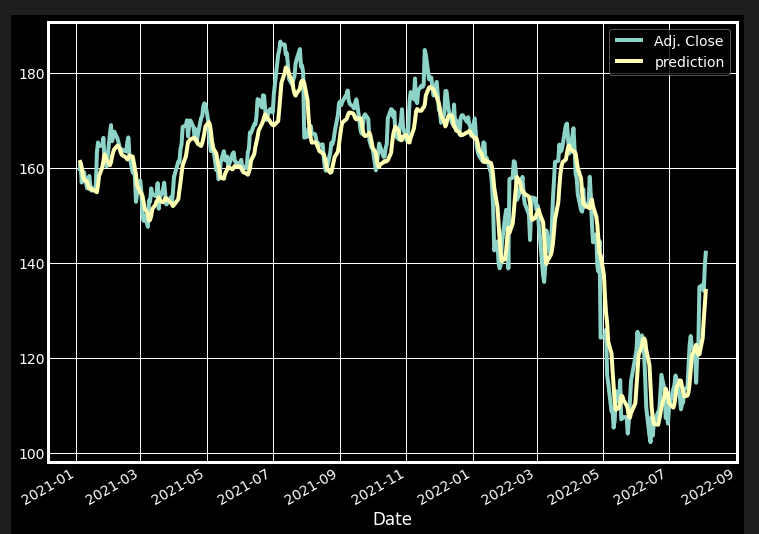
The outcome chosen by most decision trees will be the final choice. If three trees predict *buying*, and one tree predicts *not buying*, then the final prediction will be *buying*. In this case, it’s predicted that the customer will buy the phone.

**6.3 OUTPUT SCREENS:**

**LSTM**

****

**GRU**

****

**7. SYSTEM STUDY AND TESTING**

**7.1 Feasibility Study**

The feasibility of the project is analysed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

* Economical feasibility
* Technical feasibility
* Social feasibility

**Economical Feasibility**

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

### **Technical Feasibility**

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

**Social Feasibility**

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

**System Testing**

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

**7.2 Types of Tests**

**7.2.1 Unit testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

**7.2.2 Integration testing**

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

**Acceptance Testing**

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

**7.2.3 Functional testing**

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

**7.2.4 White Box Testing**

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

**7.2.5 Black Box Testing**

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

1. **CONCLUSION**

In conclusion, our study introduced a Regularized GRULSTM neural network model for short-term stock price prediction. Through empirical experimentation, our proposed model exhibited superior performance compared to existing LSTM and GRU models. By leveraging regularization techniques, we achieved enhanced accuracy in forecasting the closing prices of two stocks. The results underscore the efficacy of our approach in handling the intricate nonlinear dynamics of stock market fluctuations. This advancement holds promise for refining predictive models in financial markets, showcasing the potential of neural networks, specifically the Regularized GRULSTM, as a robust tool for effective stock price prediction.

1. **FUTURE ENHANCEMENT**

Future enhancements in stock price prediction using LSTM and GRU could focus on integrating ensemble learning techniques, such as combining multiple models like SVM and Random Forest, to create a more robust forecasting system. Incorporating attention mechanisms within the LSTM and GRU networks could improve their ability to capture significant features in the stock market data. Additionally, exploring reinforcement learning methodologies to adaptively adjust model parameters based on changing market dynamics could further enhance prediction accuracy. Embracing advanced optimization techniques and incorporating external data sources, like news sentiment analysis or macroeconomic indicators, might offer a more comprehensive understanding of market behavior, thus refining the predictive capabilities of the models.

**10. REFERENCES**

[1] Kwon, Yung-Keun & Moon, Byung-Ro. (2007). A Hybrid Neurogenetic Approach for Stock Forecasting. IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council. 18. 851-64. 10.1109/TNN.2007.891629.

[2]Singh, R. & Srivastava, S. Multimed Tools Appl (2017) 76: 18569 9

[3]Jigar Patel, Sahil Shah, Priyank Thakkar, K Kotecha, Predicting stock and stock price index movement using Trend Deterministic Data Preparation and machine learning techniques, Expert Systems with Applications, Volume 42, Issue 1, 2015, Pages 259-268

[4]George S. Atsalakis, Kimon P. Valavanis, Surveying stock market forecasting techniques – Part II: Soft computing methods, Expert Systems with Applications, Volume 36, Issue 3, Part 2, 2009, Pages 5932-5941

[5]Jigar Patel, Sahil Shah, Priyank Thakkar, K Kotecha, Predicting stock market index using fusion of machine learning techniques, Expert Systems with Applications, Volume 42, Issue 4, 2015, Pages 2162-2172

[6]Lamartine Almeida Teixeira, Adriano Lorena Inácio de Oliveira, A method for automatic stock trading combining technical analysis and nearest neighbor classification, Expert Systems with Applications, Volume 37, Issue 10, 2010, Pages 6885-6890

[7]Chen, Kai & Zhou, Yi & Dai, Fangyan. (2015). A LSTM-based method for stock returns prediction: A case study of China stock market. 2823-2824. 10.1109/BigData.2015.7364089.

[8]Erkam Guresen, Gulgun Kayakutlu, Tugrul U. Daim, Using artificial neural network models in stock market index prediction, Expert Systems with Applications, Volume 38, Issue 8, 2011, Pages 10389-10397

[9]Wei Shen, Xiaopen Guo, Chao Wu, Desheng Wu, Forecasting stock indices using radial basis function neural networks optimized by artificial fish swarm algorithm, Knowledge-Based Systems, Volume 24, Issue 3, 2011, Pages 378-385

[10]Emad W. Saad Danil V. Prokhorov Donald C. Wunsch "Comparative study of stock trend prediction using time delay recurrent and probabilistic neural networks" Neural Networks IEEE Transactions on vol. 9 no. 6 pp. 1456-1470 1998.

[11]Yudong Zhang, Lenan Wu, Stock market prediction of S&P 500 via combination of improved BCO approach and BP neural network, Expert Systems with Applications, Volume 36, Issue 5, 2009, Pages 8849-8854,

[12]Apostolos Nicholas Refenes, Achileas Zapranis, Gavin Francis, Stock performance modeling using neural networks: A comparative study with regression models, Neural Networks, Volume 7, Issue 2, 1994, Pages 375-388,

[13]-Yahoo Finance, https://in.finance.yahoo.com/quote/SPY?p=SPY, last accessed 2017/10/30. [14]Yiu-ming Cheung, Lei Xu,Independent component ordering in ICA time series analysis, Neurocomputing, Volume 41, Issues 1–4, 2001, Pages 145-152

[15]Chung, Junyoung & Gulcehre, Caglar & Cho, KyungHyun & Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.

[16]Sak, H & Senior, Andrew & Beaufays, F. (2014). Long short-term memory recurrent neural network architectures for large scale acoustic modeling. Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH. 338-342