A PROJECT
ON
“Cryptocurrency Data Analysis, Bitcoin Price Prediction System and Chat Bot”

SUBMITTED IN

PARTIAL FULFILLMENT OF THE REQUIREMENT

FOR THE COURSE OF DIPLOMA IN BIG DATA ANALYTICS FROM C-DAC

SUNBEAM INSTITUTE OF INFORMATION TECHNOLOGY

Sunbeam IT Park, Rajiv Gandhi Info tech Park,

Phase 2,Hinjawadi, Pune - 411057, MH-INDIA

SUBMITTED BY:

Utkarsh Grover

UNDER THE GUIDANCE OF:

Mr. Girish Gaikwad

Faculty Member

Sunbeam Institute of Information Technology, Pune

ACKNOWLEDGMENT

A project usually falls short of its expectation unless aided and guided by the right persons at the right time. We avail this opportunity to express our deep sense of gratitude towards Mr. Nitin Kudale (Center Coordinator, SIIT, Pune) and Mrs. Pradnya Dindorkar (Course Coordinator, SIIT ,Pune) and Project Guide Mr. Girish Gaikwad

We are deeply indebted and grateful to them for their guidance, encouragement and deep concern for our project. Without their critical evaluation and suggestions at every stage of the project, this project could never have reached its present form.

Last but not the least we thank the entire faculty and the staff members of Sunbeam Institute of Information Technology, Pune for their support.

Utkarsh Grover

DBDA, August 2019 Batch,

SIIT Pune

TABLE OF CONTENTS

1. Introduction

What Is Cryptocurrency?

- Cryptocurrency is an internet-based medium of exchange which uses cryptographical functions to conduct financial transactions. Cryptocurrencies leverage blockchain technology to gain decentralization, transparency, and immutability.
- The most important feature of a Cryptocurrency is that it is not controlled by any central authority: the decentralized nature of the block-chain makes cryptocurrencies theoretically immune to the old ways of government control and interference.
- Cryptocurrencies can be sent directly between two parties via the use of private and public keys. These transfers can be done with minimal processing fees, allowing users to avoid the steep fees charged by traditional financial institutions.

What is a Block Chain?

- A blockchain is a decentralized and digitized ledger that records all complete transactions in chronological order.
- It is a mathematical structure that stores data in a way that is impossible to counterfeit or hack.
- Blockchains allow market participants to track all cryptocurrency transactions without the need for central record keeping. The computers connected to the network also known as a node, receive a downloadable copy of the blockchain once a trade is complete.
- A block is the part of the blockchain that records the transactions. Once complete, the block is stored in a chain through cryptography
- Once a block completes its job a new one is generated. The number of blocks in block chains are many causing issues related to storage and synchronization. However, each block is traceable as it contains a hash of the former block. These blocks once recorded can't be deleted, copied or altered; they can only be distributed

Blockchain Vs Cryptocurrency

◆ ***Similarities***

1. Intangible

Both are intangible and virtual.

2. Technology

Blockchains and cryptocurrencies form part of recent technology innovations. The first block chain was invented recently after the breakthrough of gurus under the name Satoshi Nakamoto in the late 2000s.

3. **Interdependence**

Both cryptocurrencies and block chains depend on each other. Blockchains provide the path for transaction records while cryptocurrencies are the actual tools being transferred.

◆ ***Differences***

1.Nature of Block Chain Vs. Cryptocurrency

A blockchain is a decentralized technology which records cryptocurrency transactions. A cryptocurrency is a virtual tool used in the transactions within a block.

2. Use

Cryptocurrencies can be used to make payments, investments and storage of wealth. A blockchain is a vehicle that drives the cryptocurrency transactions.

3. Value

Cryptocurrencies have monetary value and can be used as a measure of wealth. Blockchains have no monetary value and can't be used as a measure of wealth.

4. Mobility

Cryptocurrencies like bitcoins can be transferred from one account to another. Blockchains are not mobile.

How CryptoCurrency Works?

Bitcoin Mining

Lets Consider below example to know how Cryptocurrency mining works using the Bitcoin network.

1. George owes Michael 10 BTC. George announces that he is sending Michael 10 BTC to the Bitcoin network
2. Miners take the information and encrypt it. This is called ***hashing***. To this information, they add other transaction information and hash that too. More and more information is added and hashed until there is enough to form a block.
3. The miners now race against each other to guess the encrypted code or ***block hash*** that will be given to the new block before it's added to the blockchain. The lucky miner that guesses the right code gets to add the new block to the blockchain.
4. Now, all the other nodes on the network verify the transaction information in the new block. They check the whole blockchain to make sure that the new information matches. If it does, then the new block is valid, and the winning miner can add the new block to the blockchain. This is called ***confirmation***.
5. Michael receives 10 BTC from George.

1.1 Statement Of the Problem

The project is divided into 3 parts

a) Cryptocurrency Data Analysis

b) Bitcoin Price Prediction System

c) Basic Chatbot with Crypto-Functionalities

a) Cryptocurrency Data Analysis

A cryptocurrency, broadly defined, is virtual or digital money which takes the form of tokens or "coins." Cryptocurrency available in the multiple forms like Bitcoin(BTC),Dentacoin(DTC),Altcoin,Ethereum(ETH),Ripple(XRP),Litecoin (LTC),Libra(LIBRA) and many more.

Since Blockchain evolved to be one of the most feasible technology, we have countless options in the cryptocurrency list to invest. Some are proving beyond expectation while some are majorly disappointing. Most surprisingly, the top 10 cryptocurrencies to invest might not be very popular or common to all the traders.

Before thinking of investing, it's very crucial to understand their features and potential in the long turn. For that, following their move is equally important. Just like real estate investment and significant investments, you can't just rush into any decision.

For Solving this Problem we are doing Analysis of top 10 CryptoCoin so that it will be helpful for investor to invest on right coin to gain more profit as time moves on

b) Bitcoin Price Prediction System

The purpose of this study is to find out with what accuracy the direction of the price of Bitcoin can be predicted using machine learning methods. This is fundamentally a time series prediction problem. While much research exists surrounding the use of different machine learning techniques .In terms of a machine learning task, predicting the price of Bitcoin can be considered analogous to other financial time series prediction tasks such as forex and stock prediction.

c) Basic Chatbot with Crypto-functionalities

The purpose of this basic chatbot is to provide functionalities to the system in question-answers format. The chatbot to build should keep limited to the cryptocurrency only. Chatbot will assist you about the Cryptocurrency information like what is cryptocurrency, how it works and many more

1.2 Technical analysis

In contrast to fundamental analysis, technical analysis does not try to gain deep insight into a company's business. It assumes the available public information does not offer a competitive trading advantage. Instead, it focuses on studying a company's historical bitcoin price and on identifying patterns in the chart. The intention is to recognize trends in advance and to capitalize on them.

1.3 Goal

The goal was to build a system capable of the following tasks:

1. **Collecting fundamental and technical data from the internet**

   The system uses specific websites to extract fundamental data like CoinmarketCap and analyst recommendations. Furthermore, it should be able to collect technical data in the form of historical bitcoin prices.

2. **Data Analysis of Top CryptoCoins**
3. **Bitcoin Price prediction System**

   The system should predict the price of bitcoin as user give input through a web interface in the form of date

4. **Making Simple chatbot for Crypto-functionality**

   Chatbot should answer to common question related to cryptocurrencies

1.4 Overview of proposed solution approach

1. Collect Cryptocurrency data and do analysis with the help of Tableau

2. Collect the bitcoin OHLCV information information for some previous years and then accordingly predict the -results for the predicting what would happen next. So for we are going to use of two well-known techniques Machine Learning

3. build basic Chatbot with the help of Chatter-bot library available in pythons

2. <u>**Product Overview and Summary**</u>

2.1 Purpose

a) Cryptocurrency Data Analysis
b) Bitcoin Price Prediction System
c) Basic Chatbot with Crypto-Functionalities

2.2 Scope

Bitcoin market includes daily activities like Open, High, Low, Close, BTC_volume

The exchange provides an efficient and transparent market for trading in equity, debt instruments and derivatives.

The Price of Bitcoin depend on many factors, Contrarily, bitcoin prices are influenced by the following factors:

1. The supply of bitcoin and market demand for it
2. The cost of producing a bitcoin through the mining process
3. The rewards issued to bitcoin miners for verifying transactions to the blockchain
4. The number of competing cryptocurrencies
5. The exchanges it trades on4
6. Regulations governing its sale
7. Its internal governance

3. **<u>Data Analysis and Study</u>**

3.1 Data Contents and Format

a) CSV files for select bitcoin exchanges for the time period of Jan 2012 to August 2019, with minute to minute updates of OHLC (Open, High, Low, Close), Volume in BTC and indicated currency, and weighted bitcoin price.

b) Timestamps are in Unix time. Timestamps without any trades or activity have their data fields filled with NaNs.

c) If a timestamp is missing, or if there are jumps, this may be because the exchange (or its API) was down, the exchange (or its API) did not exist, or some other unforseen technical error in data reporting or gathering.

d) All effort has been made to deduplicate entries and verify the contents are correct and complete to the best of my ability, but obviously trust at your own risk.

3.2 Data Columns Study

a) Timestamp

Start time of time window (60s window), in Unix time

b) Open

Open price at start time window

c) High

High price within time window

d) Low

Low price within time window

e) Close

Close price at end of time window

f) Volume_(BTC)

Amount of BTC transacted in time window

g) Volume_(Currency)

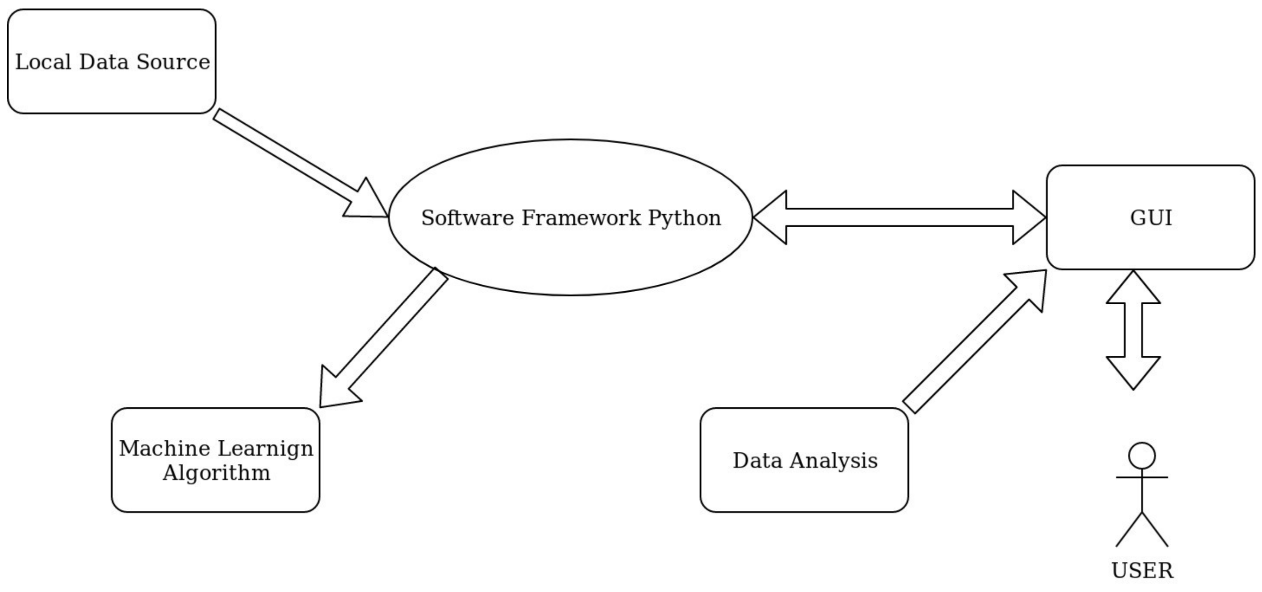
Amount of Currency transacted in time window

f) Weighted_Price

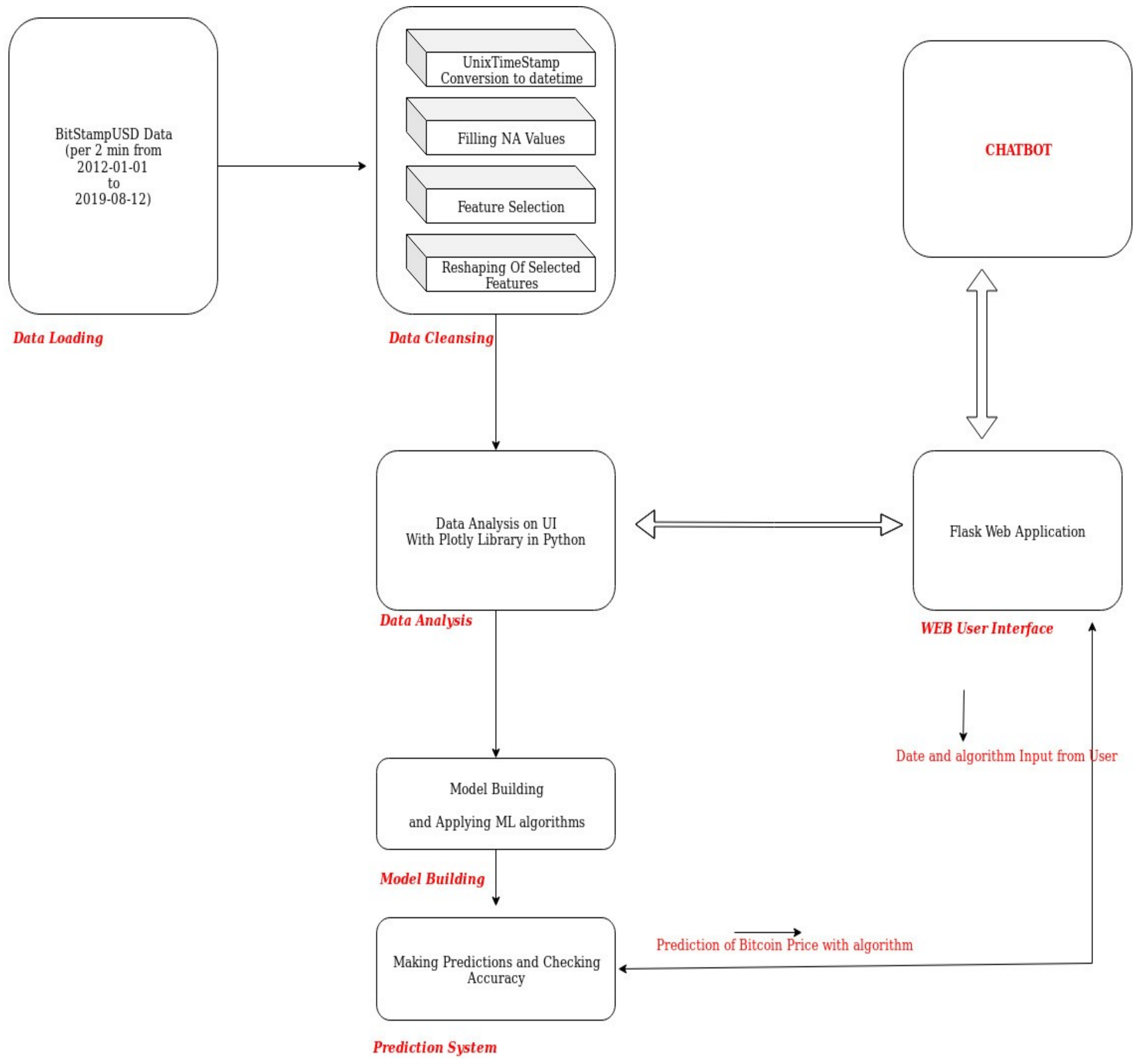
volume-weighted average price (VWAP)

4. System Design and Architecture

4.1 Use Case Diagram



4.2 System / Control Flow Diagram


BitStampUSD Data (per 2 min from 2012-01-01 to 2019-08-12)
Data Loading
UnixTimeStamp Conversion to datetime
Filling NA Values
Feature Selection
Reshaping Of Selected Features
Data Cleansing
CHATBOT
Data Analysis on UI With Plotly Library in Python
Data Analysis
Flask Web Application
WEB User Interface
Date and algorithm Input from User
Model Building and Applying ML algorithms
Model Building
Making Predictions and Checking Accuracy
Prediction System
Prediction of Bitcoin Price with algorithm

4.3 Proposed Algorithm

4.4.1 Linear Regression

-Linear regression has been around for so long (more than 200 years).
It has been studied from every possible angle and often each angle has a new and different name.

-Linear regression is a linear model, e.g. a model that assumes a linear relationship between the input variables (x) and the single output variable (y). More specifically, that y can be calculated from a linear combination of the input variables (x).

-When there is a single input variable (x), the method is referred to as simple linear regression. When there are multiple input variables, literature from statistics often refers to the method as multiple linear regression

1. **Simple Linear Regression**

With simple linear regression when we have a single input, we can use statistics to estimate the coefficients.

This requires that you calculate statistical properties from the data such as means, standard deviations, correlations and covariance. All of the data must be available to traverse and calculate statistics.

This is fun as an exercise in excel, but not really useful in practice.

2. Ordinary Least Squares

When we have more than one input we can use Ordinary Least Squares to estimate the values of the coefficients.

The Ordinary Least Squares procedure seeks to minimize the sum of the squared residuals. This means that given a regression line through the data we calculate the distance from each data point to the regression line, square it, and sum all of the squared errors together. This is the quantity that ordinary least squares seeks to minimize.

This approach treats the data as a matrix and uses linear algebra operations to estimate the optimal values for the coefficients. It means that all of the data must be available and you must have enough memory to fit the data and perform matrix operations.

It is unusual to implement the Ordinary Least Squares procedure yourself unless as an exercise in linear algebra. It is more likely that you will call a procedure in a linear algebra library. This procedure is very fast to calculate.

3. **Regularization**

There are extensions of the training of the linear model called regularization methods. These seek to both minimize the sum of the squared error of the model on the training data (using ordinary least squares) but also to reduce the complexity of the model (like the number or absolute size of the sum of all coefficients in the model).

Two popular examples of regularization procedures for linear regression are:

- Lasso Regression: where Ordinary Least Squares is modified to also minimize the absolute sum of the coefficients (called L1 regularization).
- Ridge Regression: where Ordinary Least Squares is modified to also minimize the squared absolute sum of the coefficients (called L2 regularization).

These methods are effective to use when there is col linearity in your input values and ordinary least squares would over fit the training data.

Making Predictions with Linear Regression

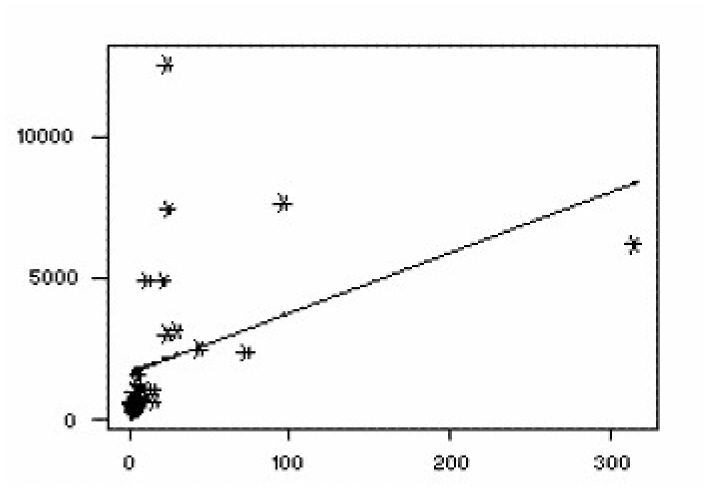
Given the representation is a linear equation, making predictions is as simple as solving the equation for a specific set of inputs.

Let's make this concrete with an example. Imagine we are predicting weight (y) from height (x). Our linear regression model representation for this problem would be:

$$y = B0 + B1 * x1$$

Outliers and Influential Observations

-After a regression line has been computed for a group of data, a point which lies far from the line (and thus has a large residual value) is known as an ***outlier***. Such points may represent erroneous data, or may indicate a poorly fitting regression line.
-If a point lies far from the other data in the horizontal direction, it is known as an ***influential observation***. The reason for this distinction is that these points have may have a significant impact on the slope of the regression line.



Cost function

The prediction function is nice, but for our purposes we don't really need it. What we need is a cost function so we can start optimizing our weights.

Let's use MSE (L2) as our cost function. MSE measures the average squared difference between an observation's actual and predicted values. The output is a single number representing the cost, or score, associated with our current set of weights. Our goal is to minimize MSE to improve the accuracy of our model.

Gradient descent

To minimize MSE we use Gradient Descent to calculate the gradient of our cost function.

4.4.2 Random Forest

Random forest algorithm is a supervised classification algorithm. As the name suggest, this algorithm creates the forest with a number of trees.

In general,the more trees in the forest the more robust the forest looks like. In the same way in the random forest classifier, the higher the number o trees in the forest gives the high accuracy results. To work on Random forest we need to have knowledge of how decision tree works

Working Of Decision Tree

Decision Tree Algorithm Pseudocode

1. Place the best attribute of the dataset at the **root** of the tree.
2. Split the training set into **subsets**. Subsets should be made in such a way that each subset contains data with the same value for an attribute.
3. Repeat step 1 and step 2 on each subset until you find **leaf nodes** in all the branches of the tree.

In decision trees, for predicting a class label for a record we start from the root of the tree. We compare the values of the root attribute with record's attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.

We continue comparing our record's attribute values with other internal nodes of the tree until we reach a leaf node with predicted class value. As we know how the modeled decision tree can be used to predict the target class or the value. Now let's understanding how we can create the decision tree model.

Assumptions while creating Decision Tree

The below are the some of the assumptions we make while using Decision tree:

- At the beginning, the whole training set is considered as the root.
- Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
- Records are distributed recursively on the basis of attribute values.
- Order to placing attributes as root or internal node of the tree is done by using some statistical approach.

The primary challenge in the decision tree implementation is to identify which attributes do we need to consider as the root node and each level. Handling this is know the attributes selection

The popular attribute selection measures:

- Information gain
- Gini index

Attributes Selection

If data set consists of "n" attributes then deciding which attribute to place at the root or at different levels of the tree as internal nodes is a complicated step. By just randomly selecting any node to be the root can't solve the issue. If we follow a random approach, it may give us bad results with low accuracy.

For solving this attribute selection problem, researchers worked and devised some solutions. They suggested using some *criterion* like information gain, gini index, etc. These criterion will calculate values for every attribute. The values are sorted, and attributes are placed in the tree by following the order i.e., the attribute with a high value(in case of information gain) is placed at the root.

While using information Gain as a criterion, we assume attributes to be categorical, and for gini index, attributes are assumed to be continuous.

By using information gain as a criterion, we try to estimate the information contained by each attribute. We are going to use some points deducted from Information Theory To measure the randomness or uncertainty of a random variable X is defined by Entropy.

For a binary classification problem with only two classes, positive and negative class.

- If all examples are positive or all are negative then entropy will be zero i.e, low.
- If half of the records are of positive class and half are of negative class then entropy is one i.e, high.

$$H(X) = \mathbb{E}_X[I(x)] = -\sum_{x \in \mathbb{X}} p(x) \log p(x).$$

•

By calculating entropy measure of each attribute we can calculate their information gain. Information Gain calculates the expected reduction in entropy due to sorting on the attribute. Information gain can be calculated.

Entropy= E(Target)-E(Target,A)

Gini Index

Gini Index is a metric to measure how often a randomly chosen element would be incorrectly identified. It means an attribute with lower gini index should be preferred

Why Random forest algorithm

To address why random forest algorithm. Consider the below advantages.

- The same random forest algorithm or the random forest classifier can use for both classification and the regression task.
- Random forest classifier will handle the missing values.
- When we have more trees in the forest, random forest classifier won't overfit the model.
- Can model the random forest classifier for categorical values also.

The pseudo code for random forest algorithm can split into two stages.

- Random forest creation pseudo code.
- Pseudocode to perform prediction from the created random forest classifier.

First, let's begin with random forest creation pseudocode

Random Forest pseudocode:

```
1. Randomly select "k" features from total "m" features.
    1. Where k << m
2. Among the "k" features, calculate the node "d" using the best split point.
3. Split the node into daughter nodes using the best split.
4. Repeat 1 to 3 steps until "l" number of nodes has been reached.
5. Build forest by repeating steps 1 to 4 for "n" number times to create "n"
   number of trees.
```

The beginning of random forest algorithm starts with randomly selecting “k” features out of total “m” features. In the image, you can observe that we are randomly taking features and observations.

In the next stage, we are using the randomly selected “k” features to find the root node by using the best split approach.

The next stage, We will be calculating the daughter nodes using the same best split approach. Will the first 3 stages until we form the tree with a root node and having the target as the leaf node.

Finally, we repeat 1 to 4 stages to create “n” randomly created trees. This randomly created trees forms the random forest.

Random forest prediction pseudocode:

To perform prediction using the trained random forest algorithm uses the below pseudocode.

1. Takes the test features and use the rules of each randomly created decision tree to predict the outcome and stores the predicted outcome (target)

2. Calculate the votes for each predicted target.

3. Consider the high voted predicted target as the final prediction from the random forest algorithm.

To perform the prediction using the trained random forest algorithm we need to pass the test features through the rules of each randomly created trees.

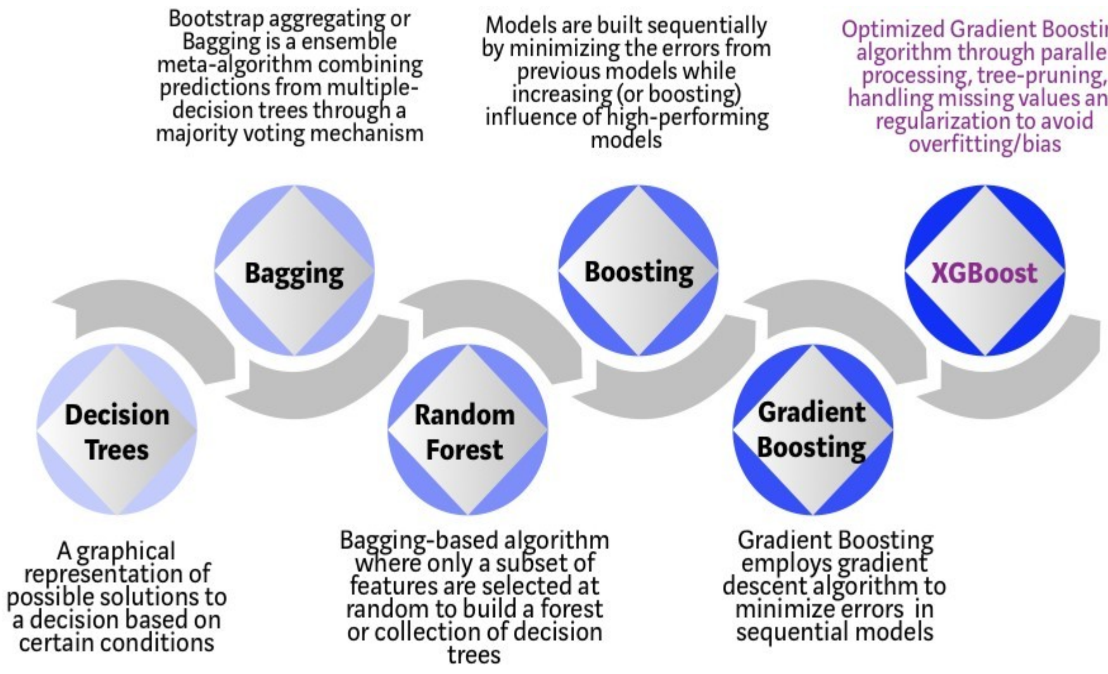
This concept of voting is known as majority voting.

Applications of random forest algorithm

1. **Banking**
   These are for finding the loyal customer and finding the fraud customers.
2. **Medicine**
   random forest algorithm is used identify the correct combination of the components to validate the medicine. Random forest algorithm also helpful for identifying the disease by analyzing the patient’s medical records.
3. **Stock Market**
   In the stock market, random forest algorithm used to identify the stock behavior as well as the expected loss or profit by purchasing the particular stock.

4. **E-commerce**
   In e-commerce, the random forest used only in the small segment of the recommendation engine for identifying the likely hood of customer liking the recommend products base on the similar kinds of customers

4.4.3 XGBoost



Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism
Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models
Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias
Bagging
Boosting
XGBoost
Decision Trees
Random Forest
Gradient Boosting
A graphical representation of possible solutions to a decision based on certain conditions
Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees
Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

XGBoost (**Ex**treme **G**radient **Boost**ing) is an optimized distributed gradient boosting library. Yes, it uses gradient boosting (GBM) framework at core. Yet, does better than GBM framework alone.

what makes it so good:

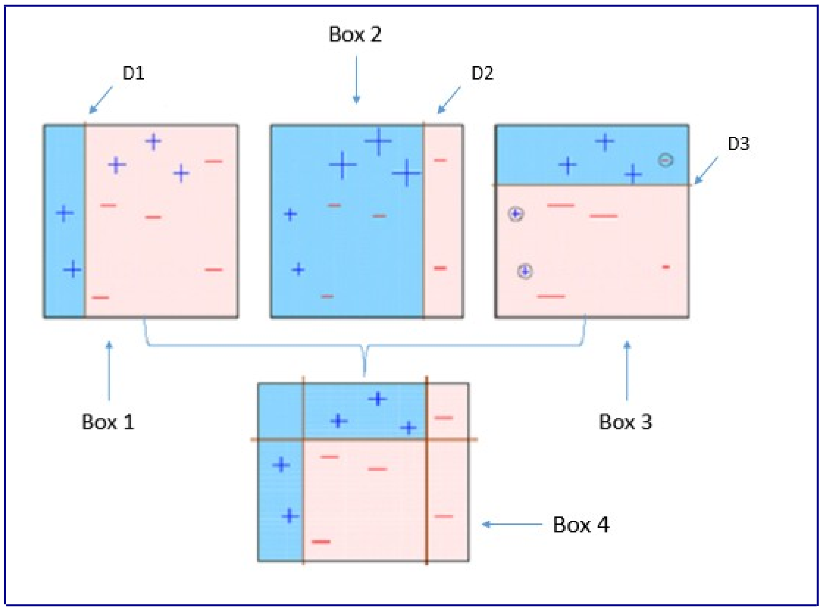
1. **Parallel Computing:** It is enabled with parallel processing (using OpenMP); i.e., when you run xgboost, by default, it would use all the cores of your laptop/machine.
2. **Regularization:** I believe this is the biggest advantage of xgboost. GBM has no provision for regularization. Regularization is a technique used to avoid overfitting in linear and tree-based models.
3. **Enabled Cross Validation:** In R, we usually use external packages such as caret and mlr to obtain CV results. But, xgboost is enabled with internal CV function (we'll see below).

4. **Missing Values:** XGBoost is designed to handle missing values internally. The missing values are treated in such a manner that if there exists any trend in missing values, it is captured by the model.
5. **Flexibility:** In addition to regression, classification, and ranking problems, it supports user-defined objective functions also. An objective function is used to measure the performance of the model given a certain set of parameters. Furthermore, it supports user defined evaluation metrics as well.
6. **Availability:**Currently, it is available for programming languages such as R, Python, Java, Julia, and Scala.
7. **Save andReload:** XGBoost gives us a feature to save our data matrix and model and reload it later. Suppose, we have a large data set, we can simply save the model and use it in future instead of wasting time redoing the computation.
8. **Tree Pruning:** Unlike GBM, where tree pruning stops once a negative loss is encountered, XGBoost grows the tree upto max_depth and then prune backward until the improvement in loss function is below a threshold.

How does XGBoost work ?

XGBoost belongs to a family of boosting algorithms that convert weak learners into strong learners. A weak learner is one which is slightly better than random guessing. Let's understand **boosting first** (in general).

Boosting is a sequential process; i.e., trees are grown using the information from a previously grown tree one after the other. This process slowly learns from data and tries to improve its prediction in subsequent iterations. Let's look at a classic classification example:



Four classifiers (in 4 boxes), shown above, are trying hard to classify + and - classes as homogeneously as possible. Let's understand this picture well.

1. **Box 1:** The first classifier creates a vertical line (split) at D1. It says anything to the left of D1 is + and anything to the right of D1 is -. However, this classifier misclassifies +points.
2. **Box 2:** The next classifier says don't worry I will correct your mistakes. Therefore, it gives more weight to the three + misclassified points (see bigger size of +) and creates a vertical line at D2. Again it says, anything to right of D2 is - and left is +. Still, it makes mistakes by incorrectly classifying three - points.
3. **Box 3:** The next classifier continues to bestow support. Again, it gives more weight to the three - misclassified points and creates a horizontal line at D3. Still, this classifier fails to classify the points (in circle) correctly.
4. Remember that each of these classifiers has a misclassification error associated with them.
5. Boxes 1,2, and 3 are weak classifiers. These classifiers will now be used to create a strong classifier Box 4.
6. **Box 4:** It is a weighted combination of the weak classifiers. As you can see, it does good job at classifying all the points correctly.

XGBoost can used to solve both regression and classification problems. It is enabled with separate methods to solve respective problems.

Classification Problems: To solve such problems, it uses booster = gbtree parameter; i.e., a tree is grown one after other and attempts to reduce misclassification rate in subsequent iterations. In this, the next tree is built by giving a higher weight to misclassified points by the previous tree (as explained above).

Regression Problems: To solve such problems, we have two methods : booster = gbtree and booster = gblinear. n gblinear, it builds generalized linear model and optimizes it using regularization (L1,L2) and gradient descent. In this, the subsequent models are built on residuals (actual - predicted) generated by previous iteration

Understanding XGBoost Tuning Parameters

XGBoost parameters can be divided into three categories (as suggested by its authors):

- **General Parameters:**Controls the booster type in the model which eventually drives overall functioning
- **Booster Parameters:** Controls the performance of the selected booster
- **Learning Task Parameters:** Sets and evaluates the learning process of the booster from the given data

1. General Parameters

1. **Booster[default=gbtree]**
   - Sets the booster type (gbtree, gblinear or dart) to use. For classification problems, you can use gbtree, dart. For regression, you can use any.
2. **nthread[default=maximum cores available]**

- Activates parallel computation. Generally, people don't change it as using maximum cores leads to the fastest computation.

3. **silent[default=0]**
    - If you set it to 1, your R console will get flooded with running messages. Better not to change it.

2. Booster Parameters

As mentioned above, parameters for tree and linear boosters are different. Let's understand each one of them:

Parameters for Tree Booster

1. **nrounds[default=100]**
    - It controls the maximum number of iterations. For classification, it is similar to the number of trees to grow.
    - Should be tuned using CV
2. **eta[default=0.3][range: (0,1)]**
    - It controls the learning rate, i.e., the rate at which our model learns patterns in data. After every round, it shrinks the feature weights to reach the best optimum.
    - Lower eta leads to slower computation. It must be supported by increase in nrounds.
    - Typically, it lies between 0.01 - 0.3
3. **gamma[default=0][range: (0,Inf)]**
    - It controls regularization (or prevents overfitting). The optimal value of gamma depends on the data set and other parameter values.
    - Higher the value, higher the regularization. Regularization means penalizing large coefficients which don't improve the model's performance. default = 0 means no regularization.
    - *Tune trick:* Start with 0 and check CV error rate. If you see train error >>> test error, bring gamma into action. Higher the gamma, lower the difference in train and test CV. If you have no clue what value to use, use gamma=5 and see the performance. Remember that gamma brings improvement when you want to use shallow (low max_depth) trees.
4. **max_depth[default=6][range: (0,Inf)]**
    - It controls the depth of the tree.
    - Larger the depth, more complex the model; higher chances of overfitting. There is no standard value for max_depth. Larger data sets require deep trees to learn the rules from data.
    - Should be tuned using CV

3. Learning Task Parameters

These parameters specify methods for the loss function and model evaluation. In addition to the parameters listed below, you are free to use a customized objective / evaluation function.

1. **Objective[default=reg:linear]**
    - reg:linear - for linear regression
    - binary:logistic - logistic regression for binary classification. It returns class probabilities
    - multi: softmax - multi classification using softmax objective. It returns predicted class labels. It requires setting numclass parameter denoting number of unique prediction classes.
    - Multi: softprob - multi classification using softmax objective. It returns predicted class probabilities.
2. **eval_metric [no default, depends on objective selected]**
    - These metrics are used to evaluate a model's accuracy on validation data. For regression, default metric is RMSE. For classification, default metric is error.
    - Available error functions are as follows:
        - mae - Mean Absolute Error (used in regression)
        - Logloss - Negative log likelihood (used in classification)
        - AUC - Area under curve (used in classification)
        - RMSE - Root mean square error (used in regression)
        - error - Binary classification error rate [#wrong cases/#all cases]
        - mlogloss - multiclass logloss (used in classification)

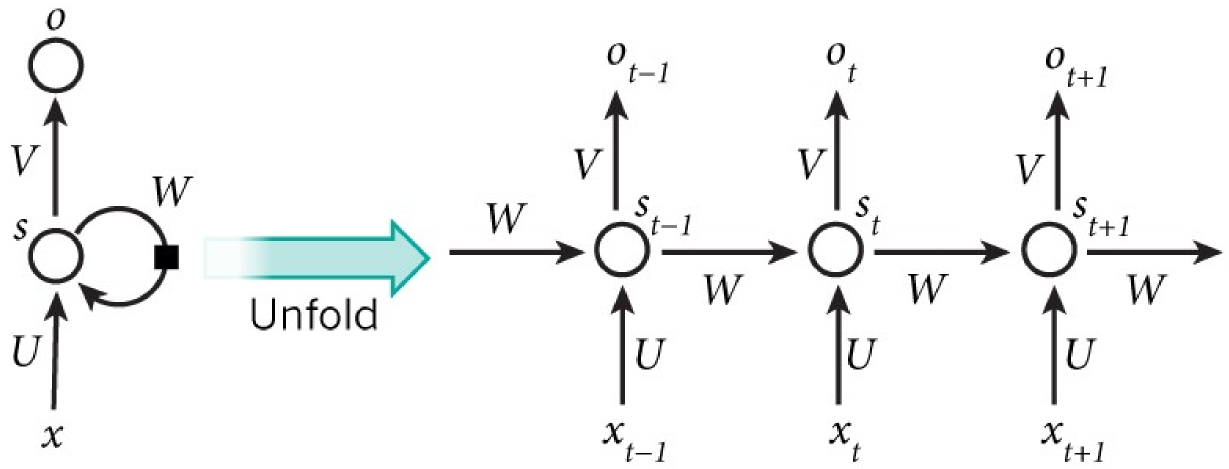
4.4.5 RNN

Feed forward Networks

In the case of feedforward networks, input examples are fed to the network and transformed into an output; with supervised learning, the output would be a label, a name applied to the input. That is, they map raw data to categories, recognizing patterns that may signal, for example, that an input image should be labeled “cat” or “elephant.”

What are RNNs?

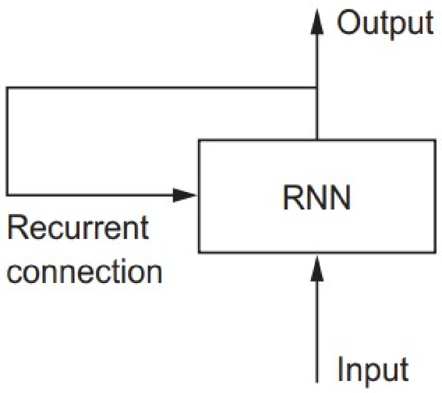
The idea behind RNNs is to make use of sequential information. In a traditional neural network we assume that all inputs (and outputs) are independent of each other. But for many tasks that’s a very bad idea. If you want to predict the next word in a sentence you better know which words came before it. RNNs are called *recurrent* because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a “memory” which captures information about what has been calculated so far. In theory RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps (more on this later). Here is what a typical RNN looks like:



The above diagram shows a RNN being *unrolled* (or unfolded) into a full network. By unrolling we simply mean that we write out the network for the complete sequence. For example, if the sequence we care about is a sentence of 5 words, the network would be unrolled into a 5-layer neural network, one layer for each word. The formulas that govern the computation happening in a RNN are as follows:

- x_t is the input at time step t. For example, $x_1$could be alone-hot vector corresponding to the second word of a sentence.
- s_t is the hidden state at time step t. It's the "memory" of the network. s_t is calculated based on the previous hidden state and the input at the current step: $s_t = f(Ux_t + Ws_{t-1})$. The function f usually is a nonlinearity such as tanh or ReLU. s_{-1}, which is required to calculate the first hidden state, is typically initialized to all zeroes.
- o_t is the output at step t. For example, if we wanted to predict the next word in a sentence it would be a vector of probabilities across our vocabulary. $o_t = \mathrm{softmax}(Vs_t)$.

Recurrent means the output at the current time step becomes the input to the next time step. At each element of the sequence, the model considers not just the current input, but what it remembers about the preceding elements.

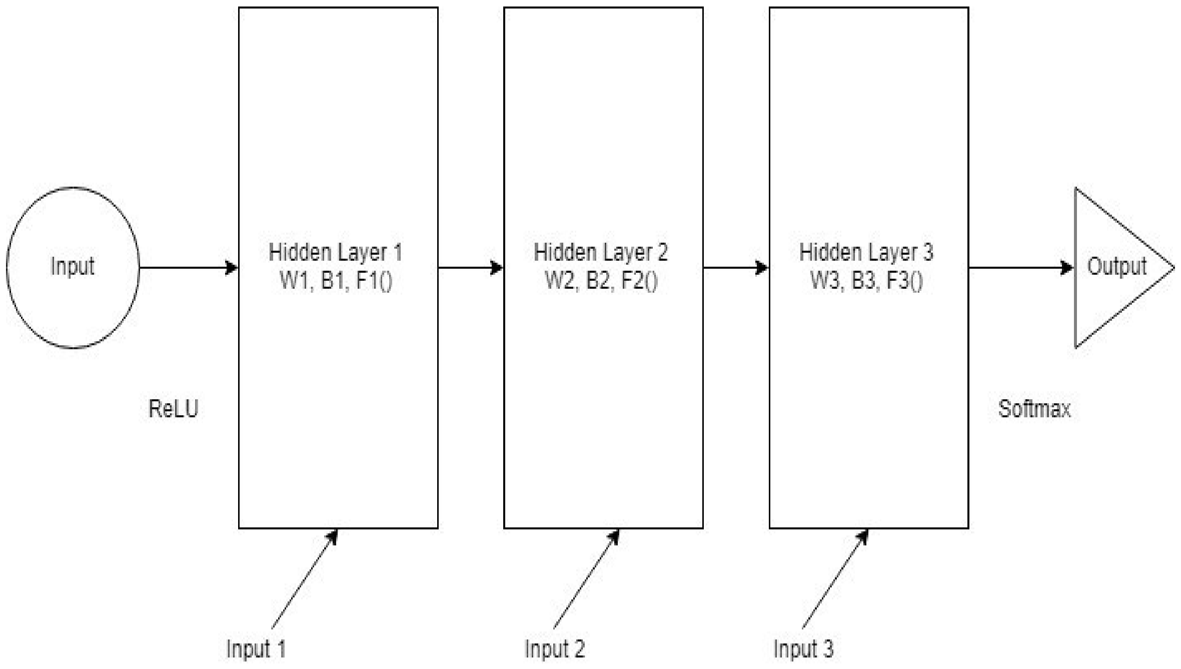


This memory allows the network to learn *long-term dependencies* in a sequence which means it can take the entire context into account when making a prediction, whether that be the next word in a sentence, a sentiment classification, or the next temperature measurement. A RNN is designed to mimic the human way of processing sequences: we consider the *entire sentence when forming a response instead of words by themselves.* For example, consider the following sentence:

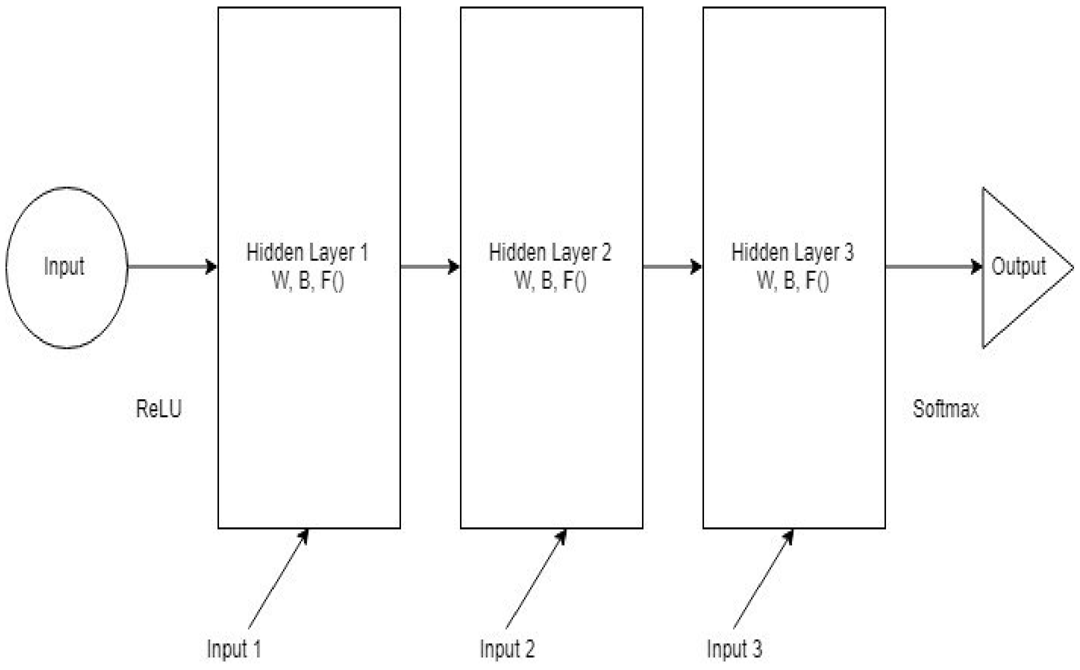
The LSTM has 3 different gates and weight vectors: there is a "forget" gate for discarding irrelevant information; an "input" gate for handling the current input, and an "output" gate for producing predictions at each time step. However, as Chollet points out, it is fruitless trying to assign specific meanings to each of the elements in the cell.

The function of each cell element is ultimately decided by the parameters (weights) which are learned during training. Feel free to label each cell part, but it’s not necessary for effective use! Recall, the benefit of a Recurrent Neural Network for sequence learning is it maintains a memory of the entire sequence preventing prior information from being lost.

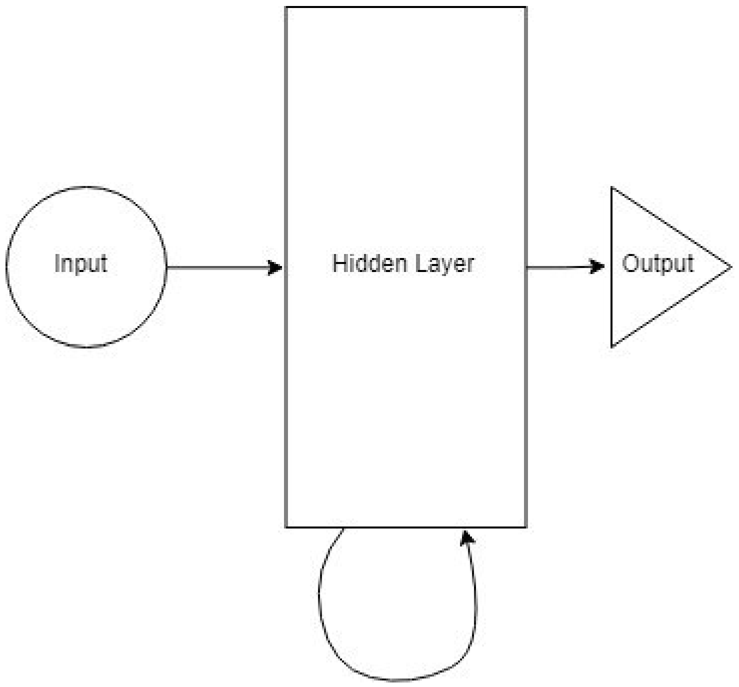
If we increase the number of layers in the above example, input layer takes the input. Then the first hidden layer does the activation passing onto the next hidden layers and so on. Finally it reaches the output layer which gives the output. Each hidden layer has its own weights and biases. Now the question is can we input to the hidden layers



Each layer has its own weight (W), biases (B), Activation Functions (F). These layers behave differently and technically would be challenging to merge together. To be able to merge them, lets replace all the layers with the same weights and biases. It will look something like this.



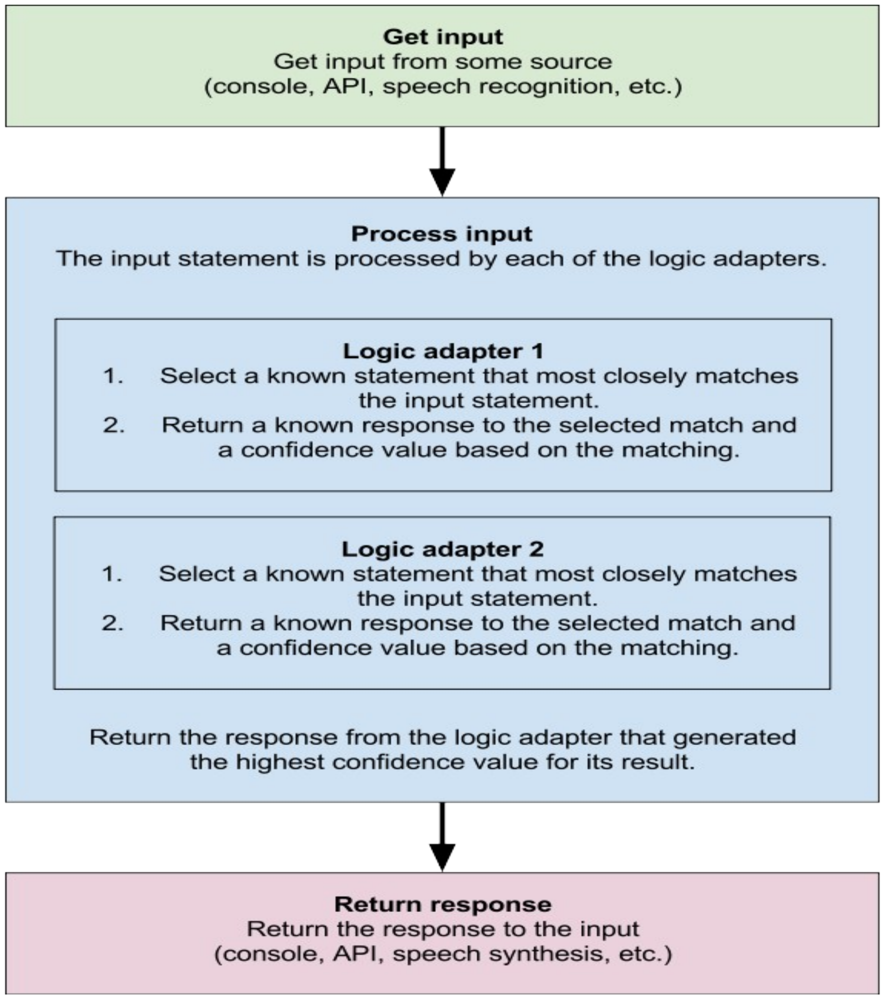
Now we can merge all the layers together. All the hidden layers can be combined into a single recurrent layer. So they start looking somewhat like this:



We will provide input to the hidden layer at each step. A recurrent neuron now stores all the previous step input and merges that information with the current step input. Thus it also captures some information regarding the correlation between current data step and the previous steps. The decision at a time step t-1 affects the decision taken at time t. This is very much like how we as humans take decisions in our life. We combine the present data with recent past to take a call on a particular problem at hand. This example is excessively rudimentary but in principle it aligns with our decision making capability. This really intrigues me as to whether we as humans are intelligent or we have a very advanced neural network model. Our decisions are just the training data that we have been collecting throughout our life. Thus can we digitise our brains once we have a fairly advanced model and systems capable of storing and computing them in reasonable time periods.

Simple ChatBot Application

```
user: Good morning! How are you doing?
bot:  I am doing very well, thank you for asking.
user: You're welcome.
bot:  Do you like hats?
```


Get input
Get input from some source
(console, API, speech recognition, etc.)
Process input
The input statement is processed by each of the logic adapters.
Logic adapter 1
1. Select a known statement that most closely matches the input statement.
2. Return a known response to the selected match and a confidence value based on the matching.
Logic adapter 2
1. Select a known statement that most closely matches the input statement.
2. Return a known response to the selected match and a confidence value based on the matching.
Return the response from the logic adapter that generated the highest confidence value for its result.
Return response
Return the response to the input
(console, API, speech synthesis, etc.)

What Is A Chatbot?

A chatbot also known as a chatterbot, bot, artificial agent, etc is basically software program driven by artificial intelligence which serves the purpose of making a conversation with the user by texts or by speech. Famous examples include Siri, Alexa, etc.

These chatbots are inclined towards performing a specific task for the user. Chatbots often perform tasks like making a transaction, booking a hotel, form submissions, etc. The possibilities with a chatbot are endless with the technological advancements in the domain of artificial intelligence.

Almost 30 percent of the tasks are performed by the chatbots in any company. Companies employ these chatbots for services like customer support, to deliver information, etc. Although the chatbots have come so far down the line, the journey started from a very basic performance. Let's take a look at the evolution of chatbots over the last few decades.

Evolution Of Chatbots

It started in 1966 when Joseph Weizenbaum made a natural language conversational program that featured a dialog between a user and a computer program. With this great breakthrough came the new age chatbot technology that has taken an enormous leap throughout the decades.

Traditional Bots	**Current Bots**	**Future Bots**
System Driven	Driven by back-and-forth communication	Communication at multiple-levels
Automation based	The automation is at the task level	Automation at the service level
Minimal Functionality	Maintains system context	Ability to maintain task, system and people context
Maintained only system context	Maintains task context as well	Introduction to master bots and eventually a bot OS as well.

Limitations With A Chatbot

With increasing advancements, there also comes a point where it becomes fairly difficult to work with the chatbots. Following are a few limitations we face with the chatbots.

- **Domain Knowledge –** Since true artificial intelligence is still out of reach, it becomes difficult for any chatbot to completely fathom the conversational boundaries when it comes to conversing with a human.

- **Personality –** Not being able to respond correctly and fairly poor comprehension skills has been more than frequent errors of any chatbot, adding a personality to a chatbot is still a benchmark that seems far far away. But we are more than hopeful with the existing innovations and progress-driven approaches.

How Does It Work?

We can define the chatbots into two categories, following are the two categories of chatbots:

1. **Rule-Based Approach –** In this approach, a bot is trained according to rules. Based on this a bot can answer simple queries but sometimes fails to answer complex queries.

2. **Self-Learning Approach –** These bots follow the machine learning approach which is rather more efficient and is further divided into two more categories.

    - **Retrieval-Based Models –** In this approach, the bot retrieves the best response from a list of responses according to the user input.

    - **Generative Models –** These models often come up with answers than searching from a set of answers which makes them intelligent bots as well.

Let us try to make a chatbot from scratch using the chatterbot library in python.

ChatterBot Library In Python

ChatterBot is a library in python which generates responses to user input. It uses a number of machine learning algorithms to produce a variety of responses. It becomes easier for the users to make chatbots using the ChatterBot library with more accurate responses.

Language Independence

The design of ChatterBot is such that it allows the bot to be trained in multiple languages. On top of this, the machine learning algorithms make it easier for the bot to improve on its own using the user's input.

How Does It work?

ChatterBot makes it easy to create software that engages in conversation. Every time a chatbot gets the input from the user, it saves the input and the response which helps the chatbot with no initial knowledge to evolve using the collected responses.

With increased responses, the accuracy of the chatbot also increases. The program selects the closest matching response from the closest matching statement that matches the input, it then chooses the response from the known selection of statements for that response.

How To Install ChatterBot In Python?

Run the following command in the terminal or in the command prompt to install ChatterBot in python

```
pip3 install chatterbot
```

Trainer For Chatbot

Chatterbot comes with a data utility module that can be used to train the chatbots. At the moment there is training data for more than a dozen languages in this module.

we get a response from the chatbot according to the input that we have given. Let us try to build a rather complex flask-chatbot using the chatterbot-corpus to generate a response in a flask application.

5. Findings and Conclusions

5.1 Conclusion

-XGBoost Algorithm gives best accuracy and gives prediction approximate to real values

5.2 Future Work

-Performance of Prediction System can be improved again with some deep learning technique like LSTM with more parameter tunning

- Chatbot Application can be improved with deep learning algorithms like NLTK, and ANN