

A DATA DRIVEN APPROACH TO OPTION PRICING

ATHARVA TANKSALE

A thesis submitted to partially fulfil the requirements of the BS-MS Dual Degree Program

Department of Mathematics

Indian Institute of Science Education and Research, Pune



Thesis Advisory Committee

Dr. Anindya Goswami, *IISER Pune*

Dr. Amit Mitra, *IIT Kanpur*

Atharva Tanksale: *A Data Driven Approach to Option Pricing*,

© May 22, 2020

This thesis is published under a

CREATIVE COMMONS ATTRIBUTION-NONCOMMERCIAL 4.0

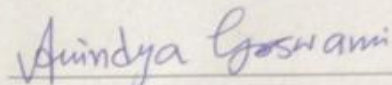
INTERNATIONAL LICENSE.

The license can be found at

<https://creativecommons.org/licenses/by-nc/4.0/>.

CERTIFICATE

This is to certify that this dissertation, titled *A Data Driven Approach to Option Pricing*, submitted to partially fulfil the requirements of the BS-MS dual degree program at the Indian Institute of Science Education and Research, Pune represents study/work carried out by Atharva Tanksale, a final year BS-MS student of the Indian Institute of Science Education and Research under the supervision of Dr. Anindya Goswami, Department of Mathematics, during the academic year 2019-2020.

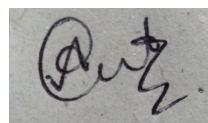

Dr. Anindya Goswami

TAC COMMITTEE

Dr. Anindya Goswami
Dr. Amit Mitra

DECLARATION

I hereby declare that the matter embodied in the thesis entitled *A Data Driven Approach to Option Pricing* is the result of the work carried out by me at the Department of Mathematics, Indian Institute of Science Education and Research, Pune, under the supervision of Dr. Anindya Goswami and the same has not been submitted elsewhere for any other degree.



Atharva Tanksale

*"The stock market is filled with individuals who know
the price of everything, but the value of nothing."
— Phillip Fisher*

Dedicated to my parents.

*When you practice gratefulness,
there is a sense of respect towards others.*

— Dalai Lama

ACKNOWLEDGMENTS

I am very grateful to Dr. Anindya Goswami for being an extremely kind and patient mentor throughout the course of this project. This project could not have been completed without the intense discussions I have had with him and Sharan Rajani. I would also like to thank Dr. Amit Mitra for agreeing to be a member of the Thesis Advisory Committee. Finally, I would also like to acknowledge the support extended to me by IISER Pune by allowing me an extensive use of the cluster computing facilities.

ABSTRACT

Fair pricing of financial instruments is at the heart of market stability. Mispricing securities can lead traders into suffering massive losses which can indirectly affect the financial health of markets. It is thus, vital to be able to derive the fair price of tradable financial instruments as this indirectly leads to optimized financial portfolios. This thesis aims to present a new approach to quantify the fair price of an *Option* contract given the underlying asset data. The models presented here would be constraint free when contrasted with traditional *Option* pricing models. We attempt to achieve the stated goal by leveraging advances in computational techniques.

CONTENTS

I GETTING TO KNOW THE BASICS

1	PRELIMINARIES :: PART A	3
1.1	Understanding Options	3
1.2	Pricing Options	4
1.3	Relationship between the Strike and the Spot	7
1.4	Thesis Roadmap	8
2	PRELIMINARIES :: PART B	9
2.1	The Machine Learning Age	9
2.2	Decision Trees	10
2.3	Moving on from Decision Trees	12
2.4	Gradient Boosted Machines	13
3	PRELIMINARIES :: PART C	15
3.1	What is Deep Learning?	15
3.2	Demystifying Deep Learning	15

II EXPERIMENTING WITH DATA

4	SETTING THE STAGE	21
4.1	Outlining the Methodology	21
4.2	Data Collection	21
4.3	Data Cleaning	23
4.4	Category Mapping	25
4.5	Model Building	27
4.6	Testing the Model	31
4.7	Miscellaneous Details	32
5	DESCRIBING VARIOUS APPROACHES	35
5.1	Approach I :: Using the Order Statistics of Time Series	35
5.2	Approach II :: Examining the Asset Time Series Closely	37
5.3	Approach III :: Including the Option Time Series	38

III UNDERSTANDING THE RESULTS

6	THE RESULTS	43
6.1	Results	43
6.2	Extending the model	46
6.3	Future Work	46

IV APPENDIX

A	APPENDIX	49
A.1	gBm and Binomial Distribution explained	49
A.2	Binomial Pricing Model	49
A.3	Hyper Parameter Optimization	50
A.4	Codes	51
B	REFERENCES	53

LIST OF FIGURES

Figure 2.1	A representative decision tree	10	
Figure 2.2	Common terms used in CART trees	11	
Figure 3.1	A general Neural Net	15	
Figure 3.2	A Feed Forward Net : Function Flow Chart	16	
Figure 3.3	A general neuron	16	
Figure 4.1	Process Flowchart	22	
Figure 4.3	Option Data Sample	23	
Figure 4.4	Asset Data Sample	23	
Figure 4.5	$\frac{C}{K}$ histogram for NIFTY50	25	
Figure 4.6	Cross-section of the Dataset after Category Mapping	26	
Figure 4.7	Python Implementation of XGBoost	28	
Figure 4.8	Python (Keras) based implementation of FNN	29	
Figure 4.9	EM visualization	32	
Figure 5.1	Cross section of an Asset Time Series	37	
Figure 5.2	Pairwise series combinations to compute the co-variances	38	
Figure 6.1	Q-Q Plot for BANKNIFTY and NIFTY50 data	44	

LIST OF TABLES

Table 5.1	Comparison of Feature Sets	39	
Table 6.1	Models trained on NIFTY50	43	
Table 6.2	Models trained on NIFTY50 and tested on BANKNIFTY	44	
Table 6.3	Models trained on both NIFTY50 and BANKNIFTY	45	
Table 6.4	Models trained on BANKNIFTY	45	
Table 6.5	Models trained on BANKNIFTY and tested on NIFTY50	45	

LISTINGS

Listing 4.1	Filtering Algorithm	24
Listing 4.2	Category Mapping	26

Part I

GETTING TO KNOW THE BASICS

An attempt is made to lay the groundwork required to understand the operative parts of the thesis. Starting off with Chapter 1 that provides the theory behind Option Pricing in sufficient detail, we move on to Chapters 2 and 3 that aim to get the reader acquainted with the theory behind decision trees models and neural networks that were used in this thesis.

*"Give me six hours to chop down a tree and I will spend
the first four sharpening the axe."*

— Abraham Lincoln

OVERVIEW The chapter begins with a primer on *Options* and then briefly discusses the models using which *Options* have been traditionally priced. The chapter ends with a brief discussion on what the thesis aims to achieve.

1.1 UNDERSTANDING OPTIONS

In financial markets there exist a variety of instruments that one can invest their money in. The most common financial instruments are stocks, bonds, commodities, derivatives etc. This thesis focuses on *Option* contracts, a special class of financial instruments. In order to better understand what *Options* are, it is vital to know what *derivatives* are. Put simply, *derivatives* are "securities" whose worth is dependant on the value of an underlying asset. *Option* contracts too are built upon the structure of derivatives. In practical terms, an *Option* is a contract that gives the *holder* (of the contract) the *right* to buy or sell the underlying asset, at a later time, from the *writer* (of the contract) at a predetermined price. The holder of the contract may choose to exercise (or not) the specified right. We go into more specific details regarding this a later in this chapter. *Option* contracts can be of two types as detailed below.

Much of the introductory discussion has been inspired by Investopedia

Note that an Option contract involves 2 parties. A holder and a writer

1. *Put Options* :: *Put* contracts gives the holder, the right to *sell* the underlying asset at the predetermined price.
2. *Call Options* :: *Call* contracts give the holder, the right to *buy* the underlying asset at the predetermined price.

It is important to clarify that the contracts are valid only within a specified timeframe. The choice of exercise of the contracts by the holder must be made *within* the timeframe. This nuance regarding the timeframe and the exercise of the contracts leads us to another important distinction in the type of *Option* contracts that are available in the markets. Specifically we have two types of contracts that can be written -

1. *American Options* :: These contracts can be exercised *anytime* within the timeframe specified.
2. *European Options* :: These contracts can be exercised only at the *end* of the timeframe specified.

The term Options, when used henceforth, refers to European Call Options

This thesis primarily focuses on *European Call Options* and refers to them as *Options*. Before we can proceed any further, it is important to define some technical terms that are associated with a general *Options* contract.

Take note of the symbols used. They will be used frequently

1. *Strike (K)* :: This denotes the predetermined price, at which the underlying asset can be brought or sold, as specified in the contract.
2. *Spot (S)* :: This denotes the current priced of the underlying asset at the time of writing the contract.
3. *Expiry Date (ttm)* :: The value of the ttm denotes the "timeframe" of the contract. Put simply, it is a measure of how long is a given contract valid for.
4. *Premium* :: This is the amount of money paid to the *writer* (of the contract) by the holder (of the contract), in order to initially acquire an *Option*.

The terms defined above are the defining characteristics of an *Options* contract.

At this point, it is natural to wonder "Why exactly are *Options* even important?". *Options* are frequently used by fund managers and traders to hedge their risks. In fact, some traders even use *Options* as a means to speculate the future state of markets and hold positions in the market at a lesser cost. *Option* contracts allow traders to realise massive profits at a fraction of the cost of the underlying asset. This aspect makes it a very lucrative financial instrument.

1.2 PRICING OPTIONS

As previously stated, the *Premium* represents the worth of an *Options* contract. The value of the *Premium* can also be understood as the *price* of an *Option* contract. It is also possible to purchase an *Option* contract from the original buyer of the contract, but the price paid to acquire it would then depend on the current valuation of the contract. It is natural to wonder how exactly is this *Premium* calculated. Let us first try to understand the difficulties in determining the worth of *Options*.

The terms *Premium* and *Price* are used interchangeably in context of *Options*

Commonly traded securities in the market, like Stocks and Bonds derive their value from the health of the company or the market sector they are based on. *Options* on the other hand derive their value based on the price of the underlying asset (usually stocks) and the anticipated future price of the asset. This complicated relationship between the value of an *Options* contract and the market sector/company is difficult to model. Moreover, the *Premium* of an *Option* contract, can be influenced by factors like the interest rates of bonds, the duration of the validity of the contract etc. All these factors, while being dynamic quantities themselves, interact with each other, in a non linear fashion making it difficult to price an *Option* contract. To add to this complexity, we need to factor in the fact that, the writer of the contract, aims to minimize the risk to him (he seeks to minimize the maximum loss he could suffer), while the holder

of the contract looks for a deal that maximizes the potential gains. The ideal scenario for a writer of the contract is to pocket the *Premium* and pay nothing to the holder of the contract while the holder desires to earn the maximum possible payoff and pay the least possible *Premium*. The intricacies illustrated above make *Option* contract valuation a hot topic of research.

Numerous attempts have been made to determine the price of an *Option* contract. Some prominent ones include the Black Scholes Merton (BSM) model and Cox Ross Rubenstein (Binomial) model. The following paragraphs give a brief summary of both the models.

1.2.1 Black Scholes Merton Model

In 1973, Black and Scholes, published a work on derivatives that considered instruments like futures, swaps, forwards and *Option* contracts. The approach showed that it was possible to obtain a unique price for *Options* regardless of the risk of the underlying security. The model was based on a number of assumptions. We list a few of them here :

1. The value of the underlying asset (eg. stock) is a continuous random variable.
2. The risky asset does not pay any dividend.
3. The risky asset price follows the dynamics of geometric Brownian motion.
4. The market is arbitrage free.
5. The transactions made in the market are frictionless implying there is no cost associated with a transaction.
6. The ideal bank has a constant interest rate.

Assumption 1 – 3 are made on the nature of the underlying asset, while the rest of the assumptions impose conditions on the nature of the markets. Black-Scholes came up with a partial differential equation that modeled the price formation dynamics of *European Options*. The equation is given below without the derivation -

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} = rV - rS \frac{\partial V}{\partial S} \quad (1.1)$$

V Price of the *Option*
S Price of the underlying asset

where t the current date
r the risk free interest rate
 σ the volatility of the market

The discussion in this section is inspired from www.tinyurl.com/smtcdh5

A derivation of the equation can be found in John Hull's book on Options

The Black-Scholes formulae is a solution to the above partial differential equation. For a *Call Option* we obtain the following:

$$V(S, T) = \max(0, S - K) \quad (1.2)$$

We only focus on the
value of the
European Call
Options

where

K Strike price of the *Option* contract

T the expiry time (maturity) of the *Option* contract

Note that this is the terminal boundary condition for which the Black Scholes PDE (1.1) is supposed to be solved. Black Scholes in their paper showed that the analytical solution for the PDE is :

$$V(S, t) = N(d_1)S - N(d_2)Ke^{-r(T-t)} \quad (1.3)$$

where,

$$N(d) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^d e^{-\frac{1}{2}x^2} dx \quad (1.4)$$

represents the cdf of the standard Normal distribution and the values of d_1 and d_2 are given by :

$$d_1 = \frac{\ln\left(\frac{S}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T - t)}{\sigma\sqrt{T - t}} \quad (1.5)$$

$$d_2 = \frac{\ln\left(\frac{S}{K}\right) + \left(r - \frac{\sigma^2}{2}\right)(T - t)}{\sigma\sqrt{T - t}} \quad (1.6)$$

One could think of the Black Scholes formulae as the probability weighted sum of what the holder of the contract would get and what the holder would pay for the contract (a negative value). As mentioned previously, the Black Scholes model assumes that the market is frictionless. Merton in 1976 extended the model to account for transaction costs.

LIMITATIONS OF BLACK SCHOLE'S MERTON MODEL The BSM model makes assumptions that do not hold in real life scenarios. For example, the assumption that the price of the underlying asset is a continuous function is far from reality as price movements seen in the markets are discrete in nature. Moreover, trading takes place in sessions, not continuously as assumed by the model. Also, note that the PDE involves terms like σ and r . Both of which are subject to estimation themselves. The BSM model also assumes that the asset dynamics are generated by a stationary process, leading to underestimation of extreme moves in the price dynamics. These deficiencies make it difficult to apply the BSM model with confidence.

1.2.2 Binomial Pricing Model

This model was proposed by William Sharpe and was mathematically described by Cox, Rubenstein and Ross in 1979.¹ Unlike the BSM model, where the price dynamics of the underlying asset is assumed to be generated from a geometric Brownian process, the binomial pricing model assumes that the asset pricing dynamics follow a binomial distribution. This assumption mitigates the drawbacks of the continuous pricing assumption made in the BSM model. The model assumes importance as it shows us a way to do away with closed form solutions for complicated PDEs as in the BSM model.

A brief description of geometric Brownian motion and binomial distribution is given in the Appendix section

The worth of *Option* contracts in the Binomial Pricing model is evaluated by following the two step process, which are described in brief below :

1. *Generation of the binomial price tree* :: An assumption is made that the underlying asset could have only two possible moves (UP or DOWN) at each discrete time step with a distinct probability for each of the moves. Simply put, the log return values for each move are taken from a set of only two constants, one positive, the other negative. A massive price tree is constructed that allows us to trace the possible paths the asset could take.
2. *Finding the Option value at each node of the price tree* :: This is done by finding the expected value of the future payoff of the underlying asset after discounting it by the risk free rate.

The exact equations that detail the pricing mechanism is given in the Appendix. Because of the way the model is constructed, it can be shown that this variant of *Option* valuation is a special case of explicit finite difference method to solve the Black Scholes PDE².

LIMITATIONS OF THE BINOMIAL PRICING MODEL The binomial model is severely ineffective when asked to value *Options* that are valid for longer periods of time. The massive price tree that needs to be built is computationally expensive too.

1.3 RELATIONSHIP BETWEEN THE STRIKE AND THE SPOT

At this juncture after examining two prominent models used to value *Option* contracts, it is important to illustrate the type of *Options* that can exist depending on the relationship between the values of the Strike price (K) and the Spot price (S). We can have :

1. Case I : When $K < S$. Such *Option* contracts are called In-the-Money (*ITM*) *Options*

¹ This subsection is inspired from Paul Wilmott's Book "Wilmott on Quantitative Finance"

² Georgiadis, Evangelos (2011). "Binomial options pricing has no closed-form solution". Algorithmic Finance

2. Case II : When $K = S$. Such *Option* contracts are called At-the-Money (*ATM*) *Options*
3. Case III : When $K > S$. Such *Option* contracts are called Out-of-the-Money (*OTM*) *Options*

The focus of this thesis is to develop a pricing model for ATM Options. The methodology proposed in the thesis can be extended to include ITM and OTM *Option* contracts too. More regarding this is detailed later in the thesis.

1.4 THESIS ROADMAP

EXPECTATIONS FROM A MODEL As seen previously, both the BSM model and the binomial pricing model have their shortcomings. This is primarily due to the models making few overarching assumptions that do not hold ground in reality. It is therefore pertinent to realise that models after all are theoretical in nature. They frequently make assumptions that seek to ignore some practicalities in order to preserve mathematical rigor. It is therefore highly desirable to come up with a constraint free approach to price *Option* contracts.

ROADMAP After laying the groundwork required to understand *Option* contracts, the later parts of the thesis attempt to explain the theory behind the computational algorithms used to achieve the stated objective. The chapters following that, delve into the specifics of how the market derived price of *Option* contracts is obtained by using classification algorithms. The final chapter talks about the results and gives a brief overview of the future studies that are in progress.

*"The less I understood of this farrago,
the less I was in a position to judge of its importance"*

— R.L Stevenson

OVERVIEW This chapter attempts to present the basics of the machine learning techniques used to solve categorization problems. It also gives an overview of decision trees along with a specific adaptation of such methods (XGBoost).

2.1 THE MACHINE LEARNING AGE

A buzzword these days, Machine learning, strictly speaking is the study of algorithms and models that allow a computer to perform a specific task without explicitly outlining the outcome of the task. It is an attempt to make computers "learn" to perform a task by recognizing patterns in data. Specific examples of this would be an algorithm that allow computers to recognize if an email is important or not to the user, based on previously made decisions by the user for similar emails.

It is important to point out that conventional algorithms can not perform such tasks as it is almost impossible to design a rule set that covers all possible use cases for a subjective task like email importance. It is here that Machine Learning algorithms shine and show their superior inferential power.

The earliest Machine Learning technique was derived from the application of Bayes Theorem, wherein the probability of an event happening given historical data was calculated. Thereafter, as research continued into making machines self sufficient, researchers developed algorithms that could learn "strategies" from previous mistakes (this forms the basis of modern reinforcement learning). Now algorithms that could beat the world's best players in games like chess, and the Chinese board game Go could be designed. The process of machine learning could be summarized as, the machine takes in some *training* data and *learns* the objective of the task. This "learning" is achieved by trying to minimize a *loss* function. *Loss* functions are easy to define when contrasted with a subjective description of a task. For example, we could define a function that calculates the points lost in a chess match. The objective here is to *learn* the correct way to minimize the loss. The "goodness" of the model obtained can be found by *testing* the model on data that wasn't used for training the model and measuring the accuracy.



A good reference for the basics of Machine Learning is the book "The Elements of Statistical Learning" by Hastie et al.

As research continued in the field of machine learning, sophisticated algorithms began to be developed for specific tasks in areas like forecasting, categorizing, clustering etc. In fact, broadly speaking, two separate classes of machine learning algorithms emerged. *Supervised* and *Unsupervised* algorithms. The difference between them being that *Unsupervised* algorithms can infer pattern in a data without being explicitly told what to look for.

In this thesis, we bring to focus a particular class of supervised machine learning algorithm known as decision trees. We build up the concepts from decision trees to gradient boosting and end with a brief note on XGBoost.

2.2 DECISION TREES

Decision tree algorithms are some of the most versatile machine learning algorithms that are in use. They can be used for both regression as well as classification problems. Decision trees seek to mimic the decision making process used by humans. They let the user see the logic at the base level and allows us to draw useful inferences at every step of the logic. In the following discussion we focus on decision trees that are used for classification problems. The reason for doing so would become clear once chapter 4 is read. Let us first formally try to define what decision trees are -

Decision Trees are a special kind of flowchart where each node of the flowchart describes a test on a specific attribute which is determined by a rule that forms the link between nodes. The ultimate node, or the leaf represents the outcome

Formally, given a feature space X , the classification tree model aims to partition the feature space into a set of n non overlapping regions (R_1, R_2, \dots, R_n) .

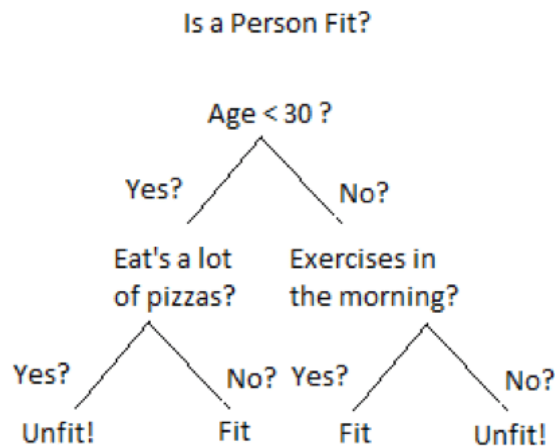


Figure 2.1: A representative decision tree

As an example of how decision trees could look like, the image 2.1¹ depicts a decision tree for a task that seeks to determine if a person is fit or not. While the given example is trivial, it is important to know how decision trees are built. There exist a many algorithms that serve the purpose, however we briefly describe the *CART* algorithm as an example of how trees are built.

CLASSIFICATION USING CART The *CART* (Classification and Regression Trees) algorithm uses the Gini index as a criterion to build tree. Refer figure 2.2 for context.

CART trees are built in a manner that the parent nodes are split to satisfy the equation -

$$\text{Goodness of Split} = 2P_L P_R \cdot \sum_{k=0,1} |P(k|L) - P(k|R)| \quad (2.1)$$

where

L	The left child node
R	The right child node
P_L	$= \frac{\text{The number of records in the left child node}}{\text{Total number of records}}$
P_R	$= \frac{\text{The number of records in the right child node}}{\text{Total number of records}}$
$P(k L)$	$= \frac{\text{The number of class "k" records in the left child node}}{\text{Total number of records in the left child node}}$
$P(k R)$	$= \frac{\text{The number of class "k" records in the right child node}}{\text{Total number of records in the right child node}}$

The term Goodness of Split refers to a quantity that determines whether the node is to be split or not. The goal of *CART* algorithms is to maximize the value of the Goodness of Split at each node. The above process is performed repeatedly till we reach a state where the decision tree ends. The tree is split in a greedy manner from the top down using binary splits. As can be seen in section 2.4, algorithms similar to *CART* find their use in a technique known as *XGBoost*.²

XGBoost is explained in detail a bit later in this chapter

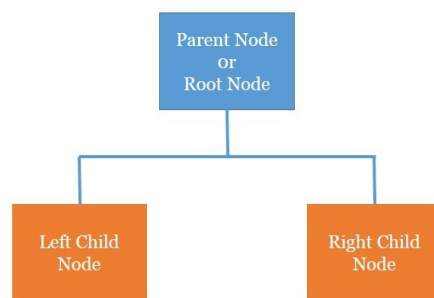


Figure 2.2: Common terms used in *CART* trees

¹ The image was taken from <https://dimensionless.in/building-blocks-of-decision-tree/>
² An example of the above equation in action can be seen at <http://ucanalytics.com/blogs/decision-tree-cart-retail-case-example-part-5/>

2.3 MOVING ON FROM DECISION TREES

While decision trees are relatively easy to understand, they become extremely difficult to construct when dealing with large amounts of data. Moreover, the trees are very susceptible to data modifications. Any small addition or deletion to the data could mean the creating the entire tree again. This is a major limitation of decision trees making it imperative to further develop the concept of decision trees.

As research continued into decision trees, methods like *Bagging* and *Boosting* were developed. *Bagging* basically involved constructing a large number of decision trees and then choosing the final output as the one given by a majority of the decision trees that were constructed. This method was again found to be limited by the same issues that plague a simple decision tree model. However, bagging made the process of classification a tad bit more robust than a single decision tree based model.

Then came techniques like *Random Forest* classifier, which used a small subset of randomly selected features (the factors that make up a model, one can think of them as the columns that make up a dataset. Each column is a feature for a prospective model) from the entire feature set and constructed a large number of decision trees. This collection of many small decision trees is then subjected to majority vote on similar lines like *Bagging*. The technique *Random Forest* too is sensitive to dataset modifications as it is after all a technique built upon *Bagging*, but is more robust.

A detailed exposition of Bagging and Boosting can be found in the textbook by Hastie et al.

It is important to state here that decision trees are what are termed as strong learners. What this term means is that one could use the output of a decision tree with a very high degree of reliability. This reliability stems from the fact that the trees make use of *all* the data in the dataset to come up with a rule based system that generates an output. One could say that (in the case of decision trees) the strength of a learner increases with the increase in the depth of the tree.

The above described techniques can be described as variants of *aggregate* decision tree methods, as multiple trees are constructed and an output is obtained after applying some rule to the collection of trees. A new take on decision tree methods came from *ensemble* techniques like *Boosting* which involved the sequential construction of multiple decision trees in a manner that the new tree builds upon the gains in prediction from the previous tree that was built. What this means is, aggregate methods construct multiple trees, all of which are independent from each other, however, ensemble methods construct trees one after the other taking in the feedback from the previously constructed collection of trees. Practically speaking, *Boosting* basically combines a sequence of weak learners (decision trees with limited depths) such that the errors from the previous sequence models are minimized. *Ensemble* techniques are faster than the *aggregate* techniques as building weak learners is much easier, faster

and computationally inexpensive when compared to strong learners. In the following section we describe a particular class of *ensemble* method known as *Gradient Boosting Machines* (GBM).

2.4 GRADIENT BOOSTED MACHINES

Boosting techniques result in a additive (sequential) tree based model as described by the following equation -

$$f(x) = \sum_{m=0}^M f_m(x) = f_0(x) + \sum_{m=1}^M \theta_m \phi_m(x) \tag{2.2}$$

where $f(x)$ describes the final model, $f_m(x)$ is the m^{th} learner, $f_0(x)$ is the initial guess (the first weak learner), θ_m is the weight of the m^{th} learner and ϕ_m is the base estimator at the m^{th} iteration. Most *Boosting* algorithms aim to optimize the values of $\{\phi_m, \theta_m\}$

Gradient Boosting was introduced by Friedman in 2001. This method basically constructs an additive model by implementing a gradient descent (implying that the target outcome for each step is based on the gradient of the errors in prediction, wrt to the actual values) in the function space. Each new weak learner takes a step in the direction that minimizes the prediction error. Recall that every supervised machine learning algorithm "learns" by achieving the objective of minimizing the loss function (refer the last section of chapter 3 for more about Loss functions). Assume that we have a loss function defined as below -

It is highly recommended that one read the paper by Friedman

$$L(y_i, y_i^p) = \sum (y_i - y_i^p)^2 \quad \forall i \in N \tag{2.3}$$

where, $L(y_i, y_i^p)$ is the Loss function, y_i is the i^{th} target value and y_i^p is the predicted i^{th} value.

The goal is to minimize the value of this Loss function. We do so by optimizing the Loss function using gradient descent. Think of it as trying to find the lowest point in a valley. Gradient descent is used to update the weights of the nodes. Formally, the algorithm finds the local minimum of a differentiable function as given below.

$$L(y_i, y_i^p) = \sum (y_i - y_i^p)^2 \tag{2.4}$$

$$L(y_i, y_i^p) = \sum (y_i - (mx_i + b))^2 \tag{2.5}$$

We replace the value y_i^p by $mx_i + b$ where m is the weight (the importance given to a node) of the x_i^{th} node and b is the bias. The gradient is obtained by-

$$L'(y_i, y_i^p) = \begin{bmatrix} \frac{d(L(y_i, y_i^p))}{dm} \\ \frac{d(L(y_i, y_i^p))}{db} \end{bmatrix} = \begin{bmatrix} \sum -2x_i(y_i - (mx_i + b)) \\ \sum -2(y_i - (mx_i + b)) \end{bmatrix}$$

To solve for the gradient, we iterate through our data points using our new m and b values and compute the partial derivatives. This new gradient tells us the slope of our loss function at the current iteration and the direction the next move should be made in order to update the weights. The size of our update is controlled by the learning rate as follows.

$$y_i^p = y_i^p + \alpha \frac{\partial (y_i - y_i^p)^2}{\partial y_i^p} \quad (2.6)$$

where α is the learning rate. Learning rate can be thought of as the speed with which we change the weights of the learners wrt the loss gradient.

One can interpret the above equation as trying to minimize the residuals (these are the errors wrt a prediction, refer Hastie et al. for more) to a value as close to 0 as possible and obtaining predictions that are the closest to the actual values. GBM can be intuitively understood by thinking of the entire process as an corrective model that first fits a simple model and then updates itself to a more complicated one in order to minimize the errors that were obtained in the simple model fit.

XGBoost is so famous that most entries on Kaggle use it and achieve a respectable submission score. Improving the score further depends on how well can one engineer the features

XGBOOST XGBoost is an extension of gradient boosting machines. Developed by Tianqi Chen in 2016, this variant of the GBM has become the mainstay of every machine learning scientist. XGBoost primarily became popular because of its ability to work with almost every type of Loss function. Moreover, the algorithm can be deployed to solve problems in regression and classification with ease as it couples some impressive hardware and software optimizations "under the hood" to achieve a really fast execution speed. XGBoost is available as a ready to use package in PYTHON, R etc. The most useful feature of the algorithm is that it comes built in with a cross validation routine that obviates the need to manually engineer the algorithm to avoid overfitting and bias. XGBoost is also capable of "learning" the best missing values in a dataset by taking in feedback from the Loss function behavior during training. The utility value of XGBoost is so high that machine learning practitioners often give out the advice "*When in doubt, use XGBoost!*".

"There are no patents in finance."

OVERVIEW This chapter aims to build up on the previous chapter by introducing computational techniques like Deep Learning.

3.1 WHAT IS DEEP LEARNING?

Chapter 2 gives a brief history of machine learning techniques. A big breakthrough was obtained when algorithms that aimed to mimic the neural networks of the human brain were developed. Broadly speaking, one could consider Deep Learning (DL) algorithms as a subset of the broad class of Machine Learning algorithms.

Most Deep Learning algorithms are based on a variation of a general neural net. The figure 3.1 ¹

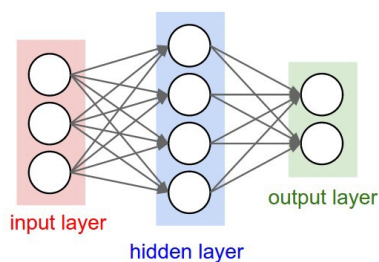


Figure 3.1: A general Neural Net

Theoretically, DL algorithms can "learn" the discrete representation of any type of input and output data. The term *Deep* tries to convey that the algorithms rely on multiple layers of neurons to extract various facets of information from the given data. The only limitation to such algorithms is the amount of data that might be required to achieve the objective. These algorithms are very useful in their specific applications but tend to be computationally intensive in general. They often require the dedicated use of a GPU. In the following sections, efforts are made to explain the exact working of neural networks.

3.2 DEMYSTIFYING DEEP LEARNING

At its core, DL is nothing but a series on encoders or decoders or a combination of both. Encoders seek to find pattern in data and create abstract interpretations,

One of the most useful references for deep learning is "Deep Learning" by Ian Goodfellow (2016). The book is freely available online at www.deeplearningbook.org

Deep can also be understood to refer to the hidden layers in 3.1

It is particularly instructive to look at the code implementation of a neural net along with the math behind it

¹ The figure is taken from Stanford's CS231 course slides.

while decoders seek to create meaningful data from the "learnt" representations. This chapter focuses on a particular class of neural nets known as Feed Forward Neural nets (FNNs). It is to be noted here that Neural Networks perform a specific task based on their architecture, ie. the way the neurons are structures and arranged in the network. Special architectures exist for tasks like image classification, object detection etc. More about this can be found in the book by Ian Goodfellow.

A Feed Forward Neural Net (FNN) is the technical term for neural networks that do not cycle information through the nodes. The network is unidirectional/linear in the sense of learning an input and giving out an output (can be seen in figure 3.1).

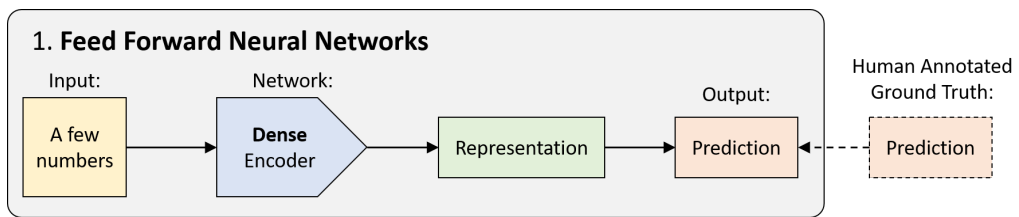


Figure 3.2: A Feed Forward Net : Function Flow Chart

The image ² 3.2 shows the general function-wise structure of what a FNN looks like. FNN's are chosen as the preferred architecture for our problem as they are excellent at constructing mappings from a set of values, referred to as features, to a categorical variable. FNN's can be trained to on a sufficiently large dataset to perform the task optimally.

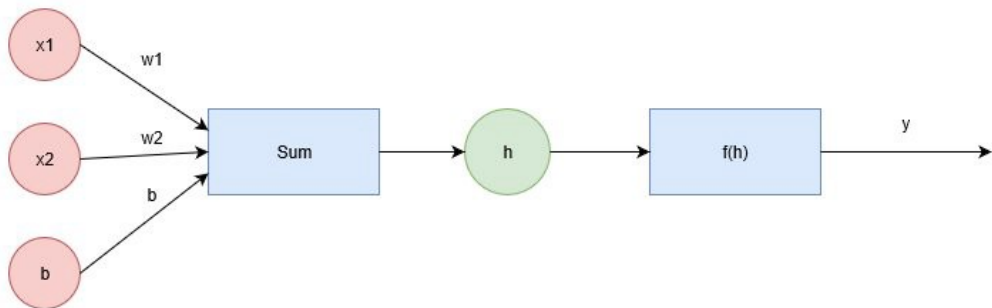


Figure 3.3: A general neuron

In order to understand how FNN's work, it is essential to get to know a few terms and their significance. Refer figure 3.3. The figure ³ shows a single neuron from the generalized neural net depicted in figure 3.1. The values x_1, x_2 are inputs to the neuron. The values w_1, w_2 are the respective weights of the inputs. The term b in the figure refers to the bias of the neural network. It is very

² Image credit to MIT DL Basics blog by Lex Fridman. <https://medium.com/tensorflow/mit-deep-learning-basics-introduction-and-overview-with-tensorflow-355bcd26baf0>

³ Image credits to Udacity DL Course

important to note that the job of a neural net is to "learn" the value of the weights and the bias. Some important terms are explained in the following paragraphs.

WEIGHTS AND BIAS They represent the importance that is given by the neuron to an input. Initially, the neurons start off with random values for weights. But as more examples are provided, the weights get adjusted accordingly to reach an optimum value that lets the network perform the assigned task well. Figure 3.3 illustrates a part labelled the *Sum*. It basically implies that the value of $w_1x_1 + w_2x_2$ is computed here. A general input can be represented as $\sum_i w_ix_i$. This value can be considered analogous to a general linear equation ($y = mx + c$), where m is the weight of the neuron.

The bias part of figure 3.3 plays the role of the intercept in this case. One can finally represent the "sum" part of the figure 3.3 as $\sum_i w_ix_i + b_i$.

ACTIVATION FUNCTION Once the inputs are processed, the inputs are passed through an activation function. They are the operative part of the neuron (referred to in figure 3.3 as h) as they are responsible for the output. Some common examples of activation functions are the SIGMOID function and the TANH function. Activation functions are responsible for deciding when the neuron would give out its output. The sigmoid function, represented by-

$$h(x) = \frac{1}{1 + e^{-x}} \quad h(x) \in [0, 1] \quad (3.1)$$

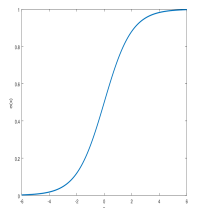
is a commonly used activation function.

Building on from the previous paragraphs, the output of a neuron in this example can be written as -

$$y = h\left(\sum_i (w_ix_i + b_i)\right) \quad (3.2)$$

A neural net generally comprises of many neurons. The outputs of all the neurons are combined to give a meaningful output. A simplified way to think of this would be considering the weights to be either 0 or 1. This would force the neurons be in wither of the states "on" or "off". The output would then depend upon whether the input triggers an "off" neuron leading to an "off" as the output and so on.

LOSS FUNCTIONS Every Machine Learning algorithms needs to have a feedback mechanism that tells the algorithm whether the "representations" learnt by it are correct or not. This feedback mechanism is coded in as the Loss function. One could think of Loss functions as type of mapping between events and real values. Depending upon the nature of the problem, one could choose to maximize or minimize the value of the Loss function. Most machine learning tasks can be considered as some variants of regression or classification problems. There exist Loss functions that are specific to the nature of the problem. Some common Loss functions are listed below-



Sigmoid function

the process of "learning" happens incrementally

If problems seek to maximize the mapping, the Loss function is called an Objective function

1. Regression Loss Functions

a) *Mean Squared Error* This is given by the expression -

$$\text{MSE} = \frac{\sum_{i=1}^n (y_i - y_i^p)^2}{n} \quad (3.3)$$

b) *Mean Absolute Error* is given by the expression -

$$\text{MAE} = \frac{\sum_{i=1}^n |y_i - y_i^p|}{n} \quad (3.4)$$

where y_i^p is the predicted value and y_i is the actual value.

2. Classification Loss Functions

a) *Hinge Loss* In this Loss function, all categories are assigned a score. The aim to make sure that the sum of all the correctly predicted categories should be greater than those incorrectly predicted categories by some margin (which is usually set as 1).

$$\text{Loss} = \sum \max(0, s_j - s_{y_i} + 1) \quad (3.5)$$

where s_j is the predicted class and s_{y_i} is the actual class.

b) *Cross Entropy* This is the most useful Loss function when it comes to classification problems. It is applied to models that predict the probability of a class being the final output. For example, if a model is trained on 3 classes, the output to be considered would be the class with the highest predicted probability. The cross entropy is given by -

$$\text{Loss} = - \sum_{c=1}^M y_{o,c} \ln(p_{o,c}) \quad (3.6)$$

where M is the number of classes, y is the binary indicator that signals if the predicted output is correct or not and $p_{o,c}$ is the probability of the instance o belonging to the class c .

Part II

EXPERIMENTING WITH DATA

This part details the process in which experiments were conducted. It begins with a chapter that describes the details behind the processing of data and follows up with a chapter that describes the rationale and the methodology of the approaches used to build models.

SETTING THE STAGE

"Be patient, good things take time."

OVERVIEW This chapter lays down the methodology in which the objective of the thesis is achieved.

4.1 OUTLINING THE METHODOLOGY

THE NEED FOR A FRESH APPROACH As chapter 1 shows, theoretical models are not enough for a trader as the price quotes obtained using traditional *Option* pricing models differ significantly from the market price of the *Option* contracts. It is highly desired that a new approach to pricing *Options* be developed. Chapters 2 and 3 lay the ground for the approach this thesis proposes. We use the powerful ML and DL algorithms described in order to predict the fair price of *Options* by learning the behavior of the market. It makes a lot of sense to learn the market perceived contract prices instead of trying to create a new theoretical model from scratch as proposing any model would require it to be mathematically robust which usually requires making unwanted assumptions. It is here that we leverage the abundance of financial data and attempt to learn the market behavior that determines the *Option* contract prices. Furthermore, having a *data - driven* model makes tweaking the inputs to the model much more easier than any theoretical model on a fundamental level.

here inputs refer to the parameters that make up the model

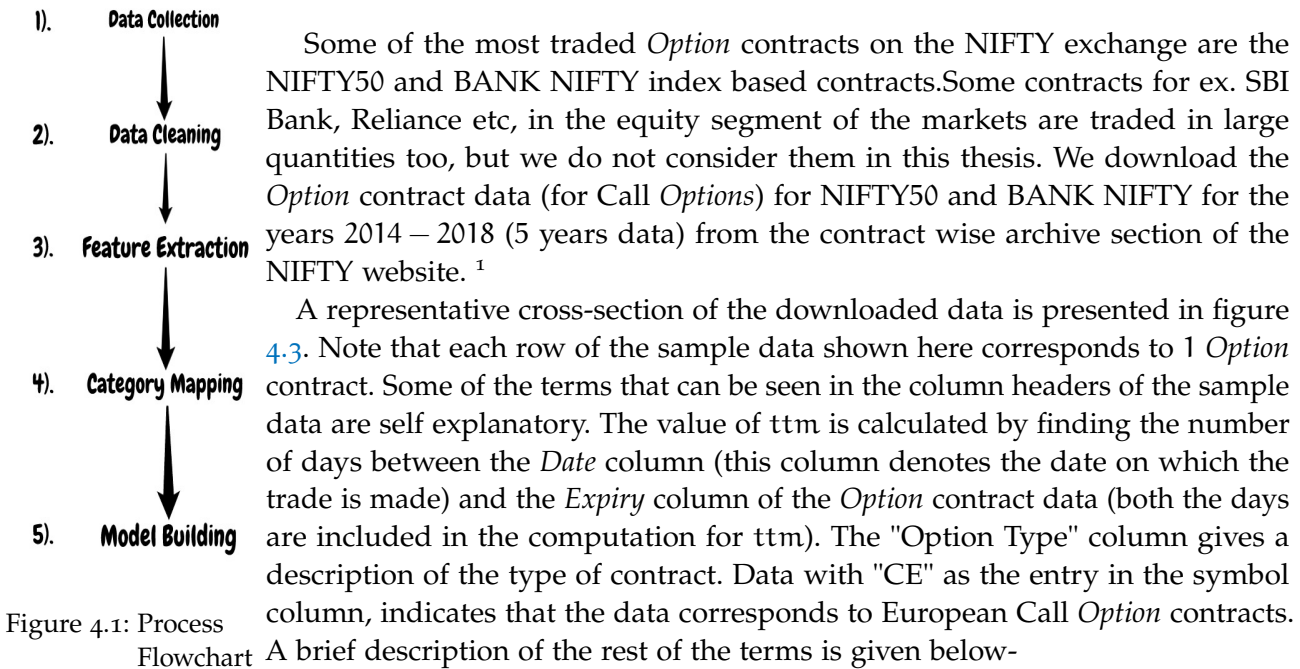
HOW DO WE GO ABOUT IT? If we want to build a *data-driven* model, we must first obtain data, clean it, process it, feed it to a ML/DL algorithm and study the results based on which we tweak the strategy used to "learn" the behavior. Now each of these steps in itself merit a section or a chapter devoted to them. We also describe in detail the specifics of how the models are set up and evaluated along with the rationale behind each aspect of constructing a model.

CHAPTER ROADMAP What this chapter does is, it provides a discussion regarding parts 1, 2, 4 & 5 as detailed in figure 4.1. We discuss part 3 in the following chapters as it forms the crux of the thesis.

4.2 DATA COLLECTION

In line with the objective of the thesis, we seek to train a ML/DL model to learn the market perceived *Option* contract prices. This would require large amounts of data. We would need data corresponding to the price quotes of *Options* market and the *Asset* market. Keeping in mind the requirement of large amounts of data, we need to choose indices on which large volumes of *Option* contracts are traded.

Now as discussed in chapter 1, *Option* contracts can be based on any sort of underlying security. NIFTY, an Indian exchange, trades large volumes of *Option* contracts on stocks and various indices. Having large volumes is essential as it is generally correlated with high trader participation which offers us a better chance to "learn" the market behavior.



Some of the most traded *Option* contracts on the NIFTY exchange are the NIFTY50 and BANK NIFTY index based contracts. Some contracts for ex. SBI Bank, Reliance etc, in the equity segment of the markets are traded in large quantities too, but we do not consider them in this thesis. We download the *Option* contract data (for Call *Options*) for NIFTY50 and BANK NIFTY for the years 2014 – 2018 (5 years data) from the contract wise archive section of the NIFTY website.¹

A representative cross-section of the downloaded data is presented in figure 4.3. Note that each row of the sample data shown here corresponds to 1 *Option* contract. Some of the terms that can be seen in the column headers of the sample data are self explanatory. The value of *ttm* is calculated by finding the number of days between the *Date* column (this column denotes the date on which the trade is made) and the *Expiry* column of the *Option* contract data (both the days are included in the computation for *ttm*). The "Option Type" column gives a description of the type of contract. Data with "CE" as the entry in the symbol column, indicates that the data corresponds to European Call *Option* contracts. A brief description of the rest of the terms is given below-

1. *Open/Close* : Indicates the first/last price quote respectively (note that here we compute the volume weighted average of the trades executed in the first/last half hour of the trading session) at which the *Option* contract was purchased on a trading day.
2. *High/Low* : Indicates the highest/lowest price quotes respectively at which the *Option* contract was purchased as during the trading day.
3. *LTP* : This is the actual last price at which a contract was sold. Most days the value of *LTP* equals the value of *Close*, but some *Option* contracts are traded in a short post trading session. This might cause the value of *LTP* to differ from the value of *Close*.
4. *Settlement Price* : This is the average price at which the *Option* contract was traded during the open and close hours of the trading session.
5. *Open Interest* : This quantity denotes the number of active contracts that exist in the market. An analogue for Volume of the contract.

Having seen a sample of what the *Options* data looks like, we present a sample of the underlying asset data in figure 4.4. Most of the terms would be familiar to the reader and are self explanatory.

¹ The data can be downloaded from the link - https://www1.nseindia.com/products/content/derivatives/equities/historical_fo.htm

Symbol	Date	Expiry	Option Type	Strike Price	Open	High	Low	Close	LTP	Settle Price	No. of contracts	Turnover In Lacs	Premium Turnover In Lacs	Open Int	Change In OI	Underlying Value
NIFTY	01-Jan-2014	30-Jan-2014	CE	7000	0.95	1.10	0.85	1.00	1.10	1.00	2444	8555.16	-	811950	10750	6301.65
NIFTY	01-Jan-2014	30-Jan-2014	CE	7050	0.00	0.00	0.00	0.25	0.25	1.40	0	0.00	-	350	0	6301.65
NIFTY	01-Jan-2014	30-Jan-2014	CE	6800	3.15	3.95	2.85	3.60	3.90	3.60	15039	51157.12	-	1403700	12400	6301.65
NIFTY	01-Jan-2014	30-Jan-2014	CE	6850	0.00	0.00	0.00	2.75	2.75	6.85	0	0.00	-	15350	0	6301.65
NIFTY	01-Jan-2014	30-Jan-2014	CE	6900	1.40	1.85	1.30	1.55	1.85	1.55	9156	31594.85	-	778250	39950	6301.65

Figure 4.3: Option Data Sample

Date	Open	High	Low	Close	Shares Traded	Turnover (Rs. Cr)
01-Oct-2014	15370.70	15377.60	15270.95	15316.20	20668460	1323.06
07-Oct-2014	15214.25	15344.60	15157.70	15180.25	20601315	1574.35
08-Oct-2014	15130.35	15371.80	15130.35	15344.00	19840589	1410.06
09-Oct-2014	15434.35	15795.65	15434.35	15741.20	27780819	1784.65
10-Oct-2014	15603.45	15628.75	15439.35	15453.80	24179823	1362.02

Figure 4.4: Asset Data Sample

4.3 DATA CLEANING

Obtaining data is just one part of the entire model building process. While in most cases, it is the most time consuming part, the financial sector generates data in large volumes everyday sparing us a lot of trouble in finding reliable sources of data. However, as important is the collection of the data it is equally important to closely monitor how the obtained data is processed. Often it is found that the data obtained contains a lot of missing entries and a decision is to be made regarding them. Too many missing data points make it difficult perform any sort of analysis on the data. Often, the rows that contain missing values are discarded if doing so does not significantly affect the characteristics of the data. One can check this by computing the column statistics of the raw data and then computing the same for a dataset that is devoid of any missing entry containing row. If the column statistics do not change by a significant value, it is generally safe to proceed ahead with "cleaned" dataset.

The *Options* price data collected by us contains more than 1.2 crores rows. This is expected as on each trading day large volumes of *Option* contracts with various different Strike values (K) are traded (giving rise to ITM/ATM/OTM type contracts). We first remove rows that contain settlement data as they generally are rows that correspond to *Option* contracts that are expiring (technically have a ttm of 0). These rows are generally of little use to us as the price of such contracts on the day of expiry tend to near 0 values. This is because there is almost no uncertainty left in the price dynamics of the underlying asset.

As far as the asset price data is concerned, unlike *Option* contract data, there are no dates on which the asset is "settled". Furthermore, using the Index equity as the underlying security helps us in making a model as the index does not give out any dividend. This leads to a continuity in the pricing of the Index based equity.

FILTERING OUT THE ATM OPTIONS To summarize thus far, we have two datasets, one that contains data corresponding to all possible *Options* contracts between 2014 – 2018 and a dataset with the daily price value of the underlying Index. We now seek to filter out the ATM *Option* contracts from the raw *Option* contracts dataset. It must be specified here that for now we are interested in *Option* contracts that are ATM in nature. The method proposed in this thesis can be extended to other *Option* contract types.

As was discussed in chapter 1, an ATM contract is the one which satisfies the relation $K = S$. However, contracts that are perfectly ATM are rarely observed in real life. What we see instead are *near ATM* contracts. The reason for this is the Strike prices (K) of the contracts have a granularity of 50 points while the underlying index price movement is granular to 0.01 points. This disparity in the granularity makes it rare for a situation to arise when the Strike of contract exactly matches the Spot of the index. To account for this, we create a filtering algorithm to screen out near ATM contracts from the raw data. A pseudo code version of the same is given below.

Listing 4.1: Filtering Algorithm

```
##### Filtering Algorithm #####
create filtered.dataframe = []

for (row) in dataset:
    value = |1 - Strike/Spot|
    if (value <= 0.02):
        append row to filtered.dataframe
    else:
        pass
#####
```

What the above filtering algorithm does is compute the value of $p (= |1 - \frac{K}{S}|)$, where p is the *filtering parameter*, and check if the value (of p) obtained is less than the predetermined value 0.02. This implies that we seek *Option* contracts with a Strike-Spot tolerance of 2%. The value 2% was decided after experiment with different values of the filtering parameter. Setting 2% as the filtering parameter gave us just the right amount of data needed for the steps that follow "data cleaning", while allowing us to retain (to some extent) the rigour of the theoretical definition of an ATM *Option*.

4.4 CATEGORY MAPPING

Refer figure 4.1. Category mapping is the 4th step in the process flowchart. Step 3 will be covered in detail in the following chapter. For now we assume that we have generated the feature set. We first briefly describe what feature sets are. One can think of them as the inputs to a ML/DL algorithm. They are generated from the raw dataset and basically attempt to present the information contained in the raw dataset in a more meaningful way. What constitutes a feature set depends solely on the type of problem being solved.

Assuming that we already have a feature set, It is now important for us to *label* the data. Doing so helps create a feedback mechanism for the ML/DL algorithms through the Loss functions. A natural question that arises at this point would be "How do we label the price of an *Option* contract?", as price quotes are integers, not attributes that can be labelled easily. This can be resolved by observing the filtered data for ATM *Option* contracts. One can see that there exists a very wide range for the price quotes of the contracts. Furthermore, if we aim to "learn" the market perceived pricing, we cant really use the value of *Option* prices directly as the value of the underlying asset does not remain a constant over the years. In fact the index value has significantly increased. What this means is that a price quote of 30 Rs. in 2015 does not have the same market significance as a price quote of 30 Rs. in 2018. What we could however use is the ratio of the contract price and the Strike price of the contract as it takes care of the changing nature of both the K and S. Studying the data shows that over the years, the distribution of the ratios is similar.

We can verify this by plotting the ratio distributions for each year

From the discussion had so far, we can say that it is better to map the features of the *Option* contracts to the ratio of the Close price C and the Strike price K of the contract than just the Close price of the contract. Refer figure 4.5

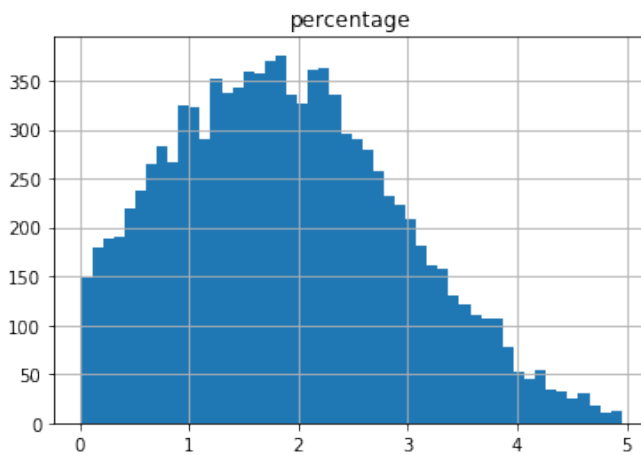


Figure 4.5: $\frac{C}{K}$ histogram for NIFTY50

In figure 4.5, the x-axis and y-axis represent the $(\frac{C}{K} \times 100)$ and the number of observations respectively. The histogram was plotted with the number of bins set as 50 as after experimenting with different values for the number of bins it

was found that using 50 bins gave us a segregation where there was just enough data points per bin and yet have enough number of categories (ie. bins) to make the model robust.

Once the value of $\frac{C}{K} \times 100$ is calculated for each *Option*, we map the *Option* features to its respective "bin". For example, the first bin contains all *Option* contracts that have a $\frac{C}{K} \times 100$ value in the range $(0, 0.1]$. It is important to clarify here that the bins here are arranged in the intervals given by the expression, $(0.1(n - 1), 0.1(n)]$, where n represents the number of bins decided. All the *Option* contracts are assigned their respective bin values in this manner by comparing the value of $\frac{C}{K} \times 100$ and finding the bin interval in which the value lies.

ttm	atmError	percentage	repoRate	yield03	yield1	Target
28.0	1.008089	3.045968	7.75	8.30	8.293	30
28.0	1.013538	3.422432	7.75	8.30	8.293	34
28.0	1.019046	3.717391	7.75	8.30	8.293	37
28.0	0.986866	1.819211	7.75	8.30	8.293	18

Figure 4.6: Cross-section of the Dataset after Category Mapping

Figure 4.6 gives a small subset of one of the Feature sets that is generated. For now, let's just focus on two of the columns shown. Columns *Target* and *Percentage*. As just discussed, the value of *Percentage* is calculated for all rows. The *Target* column shows how each percentage value is mapped to a bin in the histogram. Other columns represent some of the features that are used in the ML/DL algorithms. For the sake of completeness, we give the pseudo code version of *Category Mapping*.

Listing 4.2: Category Mapping

```
##### Category Mapping #####

ratio_list = []
for (row) in filtered.dataset:
    ratio = (Close_Option)/(Strike_Option) * 100
    ratio_list.append(ratio)

plot.hist (ratio, bins = 50)

# We obtain the histogram and the bin boundaries
# Bin1 = 0-0.1, Bin2 = 0.1-0.2 and so on...

for j in Bins:
    for i in ratio_list:
        if (i >= Bins_lower_limit) and (Bin_upper_limit > i):
            append target = j to feature_set row
```

#####

To summarize, what we do in this step is map the features of each *Option* contract to its respective bin by computing the value of $\frac{C}{K}$ and assigning the corresponding bin number to it as its label (*Target*). This helps us in understanding the predictions of the models.

4.5 MODEL BUILDING

This is the most important section of the entire chapter. What we have done so far with the datasets is just build up the required background for this section. In this section we describe how the models are implemented in specific details with some code snippets to illustrate the point better. Do note that the feature sets generated as described in chapter 5 are common for the DL and ML models. We begin by describing the process of model building and then give details about how a model is evaluated.

The process of model building broadly involves the following steps -

1. *Creating the Test Train Set* : This involves splitting the feature data set into two parts, the *test* and the *train* dataset. The feature set is usually split in a 80 : 20 ratio.
2. *Training the Model* : Once a suitable model is selected, ie. a suitable algorithm is identified, the *train* dataset is fed into it. The model then attempts to "learn" the intended objective from the *training* data.
3. *Testing* : Once the model has "learnt" the objective, it is desirable to determine how well has the model performed its task. We do so, by testing the model on a dataset that the model has not been trained on. By testing, it is implied that the model is fed in the *test* dataset as the input while the correct outputs are hidden. The model predictions are then compared with the correct outputs and a measure is designed to determine how well the model has "learnt" the objective.

This summarizes the process of model building. We now explain the specifics of the listed steps.

Step 1 of the process is generally straightforward. Most ML/DL problems ensure that the *test-train* data is split randomly to prevent over fitting ². However, for the problem being tackled by this thesis, Step 1 would entail splitting the feature set that has been ordered date-wise in the ratio of 80 : 20, ie. the test data set contains the first 80% of the data point, while the train data set contains the rest 20%. This is because our problem does not require any sort of randomized splitting as it could lead to training a model with future data points, making no sense for a trading model as a trader has no way to generate the future data

² overfitting is when a model that is fit so well to a custom dataset that its predictions fail when exposed to newer data points

points. A trader can obtain only the past data points. Furthermore, randomizing the dataset splitting could lead to the models creating higher order interpolations between the future of an *Option* contract and the past leading to an unintended biasing of the model.

We now look at Step 2 in detail. The next two subsections, describe two different approaches for Step 2.

4.5.1 Implementing XGBoost

XGBoost or Extreme Gradient Boosting, is a special implementation of the Gradient Boosting algorithm developed by Tianqi Chen. The algorithm makes use of a lot of "under the hood" optimizations to make the execution of the algorithm very fast. The paper ³ by Tianqi Chen details the said optimizations.

XGBoost can be thought of as a modified version of the GBM algorithm that combines many weak learners to create a strong learner. Implementations of XGBoost can be found in Python and R easily. Refer figure 4.7. The figure shows a naive implementation of a XGBoost model. As the code snippet shows, we first create a blank XGBoost classifier model. The model is then trained on the datasets *X_train*. The feedback to the model is given via the dataset *Y_train*. Here the dataset X and Y contain the features and the corresponding labels respectively. To put it simply, the dataset *X_train* contains the feature set section on which we wish to train the XGBoost classifier model. The dataset *Y_train* contains the row wise mapping of each of the feature set rows to *Target* variable

the algorithm constructs a mapping of the feature set row to the bin number it belongs to

```

2 model = XGBClassifier()
3 %time model.fit(X_train, Y_train)
4 print(model)
5
6

```

CPU times: user 38.5 s, sys: 56 ms, total: 38.5 s
Wall time: 41.3 s
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=3, min_child_weight=1, missing=None, n_estimators=100, n_jobs=1, nthread=None, objective='multi:softprob', random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None, silent=None, subsample=1, verbosity=1)

Figure 4.7: Python Implementation of XGBoost

As can be seen in figure 4.7, the model `XGBClassifier` contains a lot of function arguments like `booster`, `gamma`, `learning_rate` etc. These function arguments are what are termed as model hyperparameters. The model `XGBClassifier` shown in the figure is a naive one. Tuning the hyperparameters can help prevent overfitting, increase the accuracy, speed up the algorithm etc. Each of the hyperparameters mentioned, have a range over which they can be set. The process of tuning the hyperparameters is beyond the scope of this thesis, but has been done to prevent the overfitting. Refer the Appendix for more details about this.

³ The paper can be found here <https://arxiv.org/abs/1603.02754>

4.5.2 Implementing a FNN

Over the last few years, it has become extremely easy to set up neural nets. From complicated (in terms of neural net architecture) ones like Convolution Neural Nets (CNN) to relatively simple ones like FNN's. There exist plenty of packages in Python and R that make setting up a neural net very easy.

Refer figure 4.8. The figure shows the construction of a very basic FNN with just 1 hidden layer. There is a lot to unpack in the figure. We will deal with everything in a step by step manner. Let us first understand the architecture out of the way. The Appendix contains some important information regarding the neural net configurations used.

THE NET STRUCTURE Examine the code snippet. The part that says *model* is to be focused on. We first initialize a sequential model by making use of the keras package in PYTHON. This implies that neurons are arranged sequentially in a layer after layer fashion. As shown in the snippet we add neurons to the initialized model to form 3 layers.

```

1 opt=optimizers.Adam(lr=0.05);
2 model = keras.models.Sequential()
3 model.add(keras.layers.Dense(units=128, activation='relu',input_dim=X_train.shape[1]))
4 model.add(keras.layers.Dense(units=64, activation='relu'))
5 model.add(keras.layers.Dense(units=50, activation='softmax'))
6 model.summary()
7 model.compile(loss='categorical_crossentropy',
8               optimizer=opt,
9               metrics=['accuracy'],)
10 history=model.fit(X_train, Y_train,
11                  batch_size=16,
12                  epochs=45,
13                  verbose=1,
14                  class_weight=class_weights,
15                  validation_data=(X_test, Y_test),
16                  shuffle=True)

```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 128)	3456
dense_2 (Dense)	(None, 64)	8256
dense_3 (Dense)	(None, 50)	3250

Total params: 14,962
Trainable params: 14,962
Non-trainable params: 0

Train on 8304 samples, validate on 2076 samples
Epoch 1/45
8304/8304 [=====] - 3s 363us/step - loss: 3.9126 - accuracy: 0.0305 - val_loss: 3.9618 - val_accuracy: 0.0289

Figure 4.8: Python (Keras) based implementation of FNN

Each of the layers contain 128, 64 and 50 neurons each. To confirm this, we can look at the output of the code. The terms *dense_1* to *dense_3* contain the exact number of neurons specified. As described previously, we are trying to map the feature set data to 50 bins. This is the reason why the last layer has 50 neurons. If we change the number of bins we are trying to map data to, we would need to change the network structure, specifically alter the number of neurons in the last layer. The rationale behind this is that the last layer is structured

Confirm this by looking at the Output Shape column of the output

in a manner that allows each bin/category to have a neuron dedicated to it. Activation of that particular neuron would imply that the features are mapped to the corresponding category.

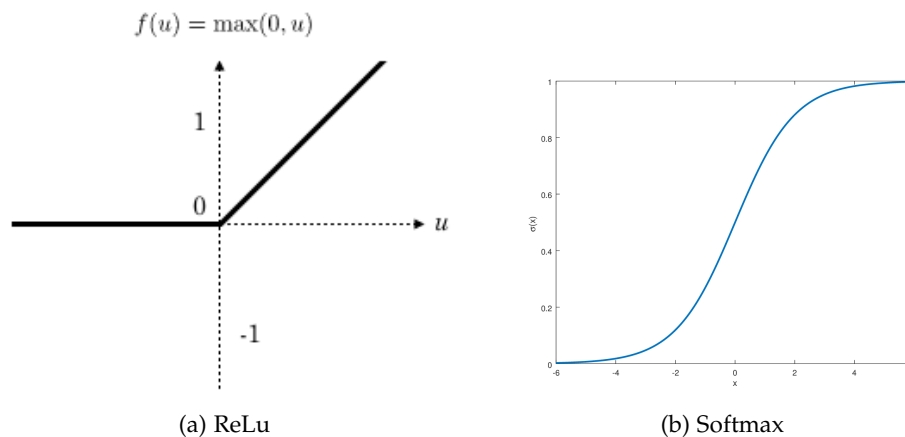
OPTIMIZER ^{4 5} Before even the model was initialized, we set a parameter called the Optimizer to Adam Optimizer. Recall from chapter 2 that the task of a ML/DL algorithm is to minimize the Loss function. Optimizer functions help to do just that. In order to minimize the Loss function, one needs to update the values of weights and the biases to reach the state with the lowest value of Loss. The Optimizer is what helps one do that. It basically is a strategy to update the weights of the FNN after every iteration of training. There are many optimizers that one can use. Few of the prominent ones are Adam, AdaGrad, AdaDelta, SGD (Stochastic Gradient Descent) etc. The details of the Optimizer algorithms are beyond the scope of the thesis.

Simply put,
Optimizers are just
different ways to
execute the gradient
descent algorithms

LOSS AND TRAINING As can be seen, the model uses *categorical cross-entropy* as the Loss function. The Loss function has been explained in chapter 3. We focus on the activation functions for each of the layers. The first and the second layers use an activation function called "ReLU" or "rectified linear unit". A graphical representation of "ReLU" is given in figure 4.9a

The "ReLU" function can be written as $f(x) = \max(0, x)$ and is a commonly used activation function. The last layer uses the "softmax" or the sigmoid activation function.

Refer chapter 3 for
more about the
sigmoid function



TRAINING THE MODEL Neural networks are trained in iterations. The first iteration starts off with random weights and the next ones take feedback from the value of the Loss functions and adjust the weights accordingly based on the scheme prescribed by the Optimizer. The number of iterations during which the model is trained is termed as epochs. Care must be taken to not set the value of epochs as too high as it might lead to overfitting the model. The model

⁴ An excellent resource on Optimizer function is the blog post that can be found here <https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f>

⁵ We do not delve into much detail regarding Optimizers as it is beyond the scope of the thesis

is *validated* after every epoch to compute the value of training error. One can consider training error as a metric that describes how well is a model learning the objective. Note that the validation set is *NOT* used to adjust the weights of the neurons in the model. Moving on to the term batch size, its a computational addition to the FNN algorithm to reduce memory constraints. We wont delve into the specific details in this thesis.

4.6 TESTING THE MODEL

Once a model is trained, it is imperative to test the performance of the model by exposing it to data that it hasn't been trained on (the test dataset) and study the predictions. The most common way to evaluate the predictions is to examine the value of the accuracy metric. It is defined as -

$$A = \frac{C_i}{N} \quad (4.1)$$

where C_i are the number of correct predictions and N is the total number of prediction. However, using the accuracy metric holds very little value for this kind of classification problem with a large number of bins. This is because, if we obtain a model that outputs a prediction close to the actual value of the input, the accuracy metric would discard the importance of this "flawed" prediction. We are, however interested in understanding how flawed the prediction is. We desire a range of values, over which a trader could claim, with some confidence, to be the market derived fair price of an *Option* contract. It is therefore imperative to come up with a metric does a better job than the accuracy metric. We do so by proposing a metric called the *Error Metric (EM)*. We compute EM as -

$$EM = \left(\frac{w}{T} \sum_{i=0}^{i=T} |C_i - P_i| \right) \times 100 \quad (4.2)$$

where,

w the *binwidth*, which is the width of each bin

T number of terms in the *test* dataset

C_i the ground truth category value

P_i the model predicted category value

The figure 4.9 attempts to visually present the aim of the EM metric. It is possible to have a model that is able to capture the market perception the *Option* contract prices. But a particular prediction is off by a bin or two or more. This might happen for a lot of *Option* contracts. The EM metric allows us to gain an insight into how *off* the predictions by the model are. It is desired that we have a model that has the value of EM metric as low as possible. This signals a departure from the classic approach used in ML models. During the course of building the models, we aim to minimize the value of the EM metric not maximize the value of the *Accuracy* metric. Doing so would help a trader understand the predictions of the model better.

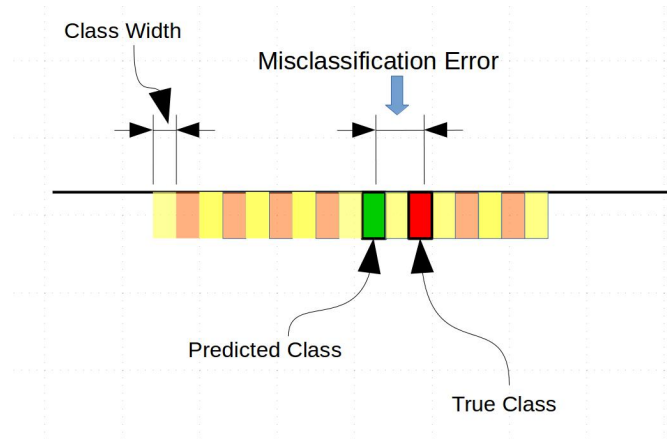


Figure 4.9: EM visualization

4.7 MISCELLANEOUS DETAILS

This section covers some operation details regarding model building. The following paragraphs can be read independently.

SCALARS A feature set when generated would contain the actual values of the independent features that are computed. Say for example, one of the features we consider is the value of *Close* price of the asset. For our use case, we consider data spanning over 4 years. The price of the index in the beginning of the data was in 7000 range and moved up to 11000 range by the end. This could bias the model to recognizing towards the lower or higher data ranges. This is very possible in the DL approach as the neurons are sensitive to the scale of the data being fed to it. In order to reduce the chances of such kinds of bias, we scale the data using the *Robust Scalar* method. This is basically scaling each data point as per the equation given below-

There are many types of scaling methods available, but it was found that using Robust scalar gave us the best results

$$X_i = \frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)} \tag{4.3}$$

where

X_i the new value of the i^{th} observation

x_i the old value of the i^{th} observation

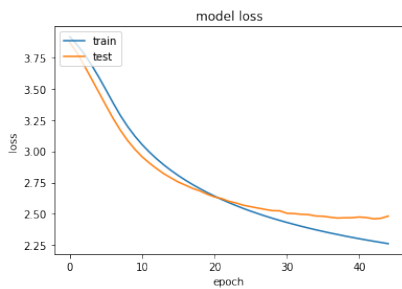
Q_i the value of the i^{th} Quartile range of the entire data column

The term Q_i is basically the value of the feature at the i^{th} percentile of the data column (represented as x without any label). The scalar is applied to every column in the feature set leading to the values in all the columns being scaled appropriately.

ONE HOT ENCODERS This is an operational aspect of presenting data to DL algorithms. What one hot encoding does is create a tuple and uses the 1 – 0 representation to indicate the class value. For example, if we had a three class classification problem and we wanted to represent the second class, *one hot encoding* would give an output of [0 1 0] for an input of 2.

UNDERSTANDING TRAINING Refer figure 4.8. In the output section of the figure terms like *loss*, *accuracy*, *val_loss*, *val_accuracy* have been mentioned. The quantities with the suffix "val" are computed on the test data set. Note that during validation, the model weight are NOT changed. The values of all the quantities mentioned are calculated every iteration or *epoch* of the training period. It is very instructive to plot the values of the losses (*loss* and *val_loss*) vs. the number of *epochs*. Doing so will help in understanding the progress of the training process for the model. It also gives us an indication of whether the model is being over fit or not. We don't explain the detailed interpretation of the figure 4.10a, but a good resource on this can be found at the link in the footnote ⁶. The figure can also be used to estimate the correct number of *epochs* that are needed to train a model without overfitting it. Refer the link mentioned above for more.

CLASSIFICATION REPORTS Along with the *accuracy* and EM metric, one particularly useful tool is the classification report. Figure 4.10b shows a sample classification report for a model. What it basically does is breakdown the prediction of the model for each class. Using the value of F1 scores could help us in figuring out if a model is performing well across all classes.



(a) Loss Plot

	precision	recall	f1-score	support
0.0	0.38	0.28	0.32	39
1.0	0.41	0.38	0.40	45
2.0	0.27	0.75	0.40	53
3.0	0.00	0.00	0.00	44
4.0	0.13	0.42	0.20	45
5.0	0.25	0.02	0.03	63
--	--	--	--	--

(b) Classification Report

A brief explanation of the terms in figure 4.10b is given below.

- A. *Precision* : This is the measure corresponding to how many predictions were correct. Mathematically given by $\text{Precision} = \frac{TP}{TP+FP}$
- B. *Recall* : This measure gives us a sense of how well the model was able to catch the positive cases. Mathematically given by $\text{Recall} = \frac{TP}{TP+FN}$
- C. *F1 score* : This measure combines the values of *Precision* and *Recall* to give us an indication of how many positive predictions were correct. Mathematically given by $\text{F1} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Recall} + \text{Precision}}$
- D. *Support* : It gives the number of predictions made in every class.

Note that here TP, FP, FN mean true positives, false positives and false negatives respectively.

⁶ <https://forums.fast.ai/t/determining-when-you-are-overfitting-underfitting-or-just-right/7732/2>

*"Stay committed to your decisions,
but stay flexible in your approach."*

— Tony Robins

OVERVIEW This chapter describes three different approaches to build the feature sets that are required to build a predictive model. This chapter completes the feature generation step left for later in chapter 4.

5.1 APPROACH I :: USING THE ORDER STATISTICS OF TIME SERIES

As explained in chapter 1 the pricing of an *Options* contract is largely dependent upon the nature of the underlying security. If a newly minted trader was told this and asked to build a predictive model based on this statement, it would seem natural to him that the model should focus on the underlying asset time series. From the perspective of a trader, he'd be aware of the inputs to a Black-Scholes based pricing model, which are the following, K, S, r and ttm . The value of K (the Strike Price), S (the Spot) and the ttm is readily available from the *Option* contract itself. The closest real world approximate would be the government bonds yield or the central bank interest rates. ¹.

*r is the risk free
interest rate*

From the parameters that are available to the trader, the most important determinant of the value of the *Option* contract would be the past value of the underlying security and the price dynamics it has been following over the past few days. The trader would want to come up with a feature set that is majorly drawn from the past Spot (S) values. Directly using the Spot prices would make the values scale dependent. This problem is especially pertinent when features are computed across years. It is then much better to use the value of returns. The "return" of a time series is defined as -

$$\Delta T_i = T_i - T_{i-1} \quad (5.1)$$

where, T_i is the i^{th} term of a time series. The expression gives us the value of the *simple returns*. Simple returns too are scale dependent. However, a trader could do better by *detrending* the time series. *Detrending* of a series can be done

*Also referred to as
normalizing the time
series*

¹ All these terms have been defined in Chapter 1

by computing the *log return* values as shown in the expression below. ²

$$\Delta_{\log T} = \log(T_i) - \log(T_{i-1}) = \log \frac{T_i}{T_{i-1}} \quad (5.2)$$

One advantage of computing the *log returns* is that it renders the quantity dimensionless, making it a robust quantity to add in the feature set. The returns series can be computed for an ad-hoc number of days. It is important to set a limit. We set a limit of 20 trading days. Simply put, the fictional trader would compute the log returns for the 20 trading days and obtain 19 log return values to be used as features.

This corresponds to approximately a month including all the Sundays and holidays

At this point, the trader could use the asset log returns to compute specific features like the mean, the standard deviation, etc. But, the trader instead chooses to do the following. He adopts an extremely naive approach of letting the model compute the features. What this means is, the trader constructs an *empirical cumulative function* (the Order Statistics) from the obtained log returns and feeds it into the model as the input. He then lets the model compute the required moments from the obtained Order Statistics. Let us explain the rationale this step a bit more.

The quantities *mean* and the *standard deviation* are the first and second moments of the Order Statistics of the log returns. Computing only those and using them as features, would mean that we are ignoring the possible contribution of higher order moments to the accuracy of the model. It might so happen, that those values be actually very important in letting the model determine the market perceived price. In order to avoid excluding the higher order moments, it's better to supply the model with the Order Statistics of the log returns.

COMPUTING THE ORDER STATISTICS The Order Statistics is computed by simply arranging the log returns in either ascending or descending order. Formally stated, given n observations, $(x_1, x_2, x_3 \dots, x_n)$, we can arrange them in order as $x_{i(1)}, x_{i(2)}, x_{i(3)} \dots$ and so on, where $x_{i(1)}$ denotes the lowest amongst the n values. The final output we get obeys the relation - $x_{i(1)} \leq x_{i(2)} \leq x_{i(3)} \leq \dots \leq x_{p(n)}$. The i^{th} order statistics in this output is the i^{th} term of the series. To firmly put the point across, we take a small example. Say we have the following observations - (3, 6, 1, 2). The order statistics of the observations would be (1, 2, 3, 6), with 3 being the third order statistic.

WHAT DOES THE FEATURE SET LOOK LIKE From the preceding discussions, the features that the naive trader chooses are -

1. The 19 log return order statistics.
2. *ttm* values for each *Option* contract.

Note that 20 time series observations would result in 19 log return values

² The book on Time Series Analysis by Box, Jenkins et al. gives a comprehensive argument in favour of detrending in the chapter on ARIMA processes.

3. r , *the interest rate* : Note that we use the 1 year sovereign bond yield rates as an approximation for the risk free interest rates.
4. *atmError* : This quantity, computed as $\text{atmError} = \frac{S}{K}$ (the value of Spot on the day of purchasing the contract), gives an indication of how far from the theoretical definition of an ATM *Option* is the current contract.

theoretical ATM contracts require
 $K = S$

The features 2 – 4 are constant for all the other approaches that will be discussed in the following sections. The focus of this approach was the rationale behind finding the order statistics of the log returns.

5.2 APPROACH II :: EXAMINING THE ASSET TIME SERIES CLOSELY

In this approach, the trader decides to get more sophisticated in his analysis of the asset time series after using a very naive Order Statistics based approach. He takes a look at figure 5.1 and realises that the available asset data is not only limited to the "Close" values of the asset. A lot more information can be gleaned by taking into account the values of "High", "Low" and "Open" along with the values of "Close". The rationale behind this being adding more asset data information would only help the model understand the dynamics of the asset better. This approach can then be reduced to a problem involving 4 time series.

Open	High	Low	Close
10881.70	10923.60	10807.10	10910.10
10868.85	10895.35	10735.05	10792.50
10796.80	10814.05	10661.25	10672.25
10699.70	10741.05	10628.65	10727.35
10804.85	10835.95	10750.15	10771.80

Figure 5.1: Cross section of an Asset Time Series

It is natural to consider extending the approach taken in the previous section and feeding in the Order Statistics of each of the time series as an input to the model. But that would give us about $19 \times 4 = 76$ features. It is cumbersome to feed in that many features to the model. It would also increase the time taken by the models to train. This limitation, makes extending *Approach I* to problems with multiple time series very inefficient.

cumbersome in terms of creating the feature set

The trader, then decides to do the following. He first computes the log returns of all the series. He then computes the means for each of the log returns series. In addition to the means, he also computes the co-variance values of all possible pairs.

Formally put, the trader does the following -

- A. Compute the log returns of all 4 series.

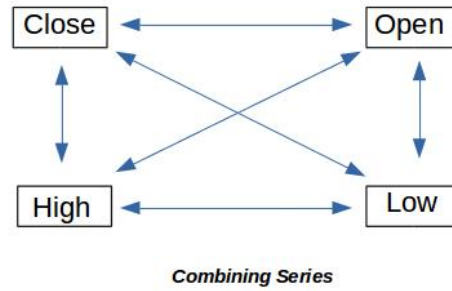


Figure 5.2: Pairwise series combinations to compute the co-variances

- b. Find the respective μ_i 's.
- c. Compute the covariance matrix for the 4 series.
 - a) Obtain the 4×4 matrix -

$$\text{cov} = \begin{bmatrix} \text{COV}_{11} & \text{COV}_{12} & \text{COV}_{13} & \text{COV}_{14} \\ \text{COV}_{21} & \text{COV}_{22} & \text{COV}_{23} & \text{COV}_{24} \\ \text{COV}_{31} & \text{COV}_{32} & \text{COV}_{33} & \text{COV}_{34} \\ \text{COV}_{41} & \text{COV}_{42} & \text{COV}_{43} & \text{COV}_{44} \end{bmatrix}$$

where

$$\text{cov}_{xy} = \frac{1}{N-1} \sum_{i=0}^N (x_i - \mu_x)(y_i - \mu_y)$$

(here, x and y denote two separate series and cov_{xy} represents the pairwise covariance of the series x and y)

- b) Consider the values in the Lower Triangular half of the covariance matrix. These values are directly used as features.

The feature set for this approach is then given by -

1. The 14 (= 4 series mean(s) and + 10 pairwise covariances) statistical quantities mentioned above.
2. *ttm* values for each *Option* contract.
3. r , the interest rate : The 1 year sovereign bond yield rates
4. *atmError*

5.3 APPROACH III :: INCLUDING THE OPTION TIME SERIES

Having implemented Approach II, the trader now realises that it is possible to use the *Option* time series itself as an input to the model. A snapshot of the *Option* Time Series is given in figure 4.3. As can be seen, the *Option* contract data contains many time series. Using an extension of *Approach I* or even *Approach*

II would be unpractical. (*Approach I* would lead to an extremely large feature set, while *Approach II* would result in a feature set that contains more than 40 features.) It is important that we choose our features wisely. Through many iterations of feature engineering, it was found that the most useful feature that actually added value to the process of model generation was the ratio of *previous* "Close" (of the contract) by the "Strike" of the *Option* contract. It is very important to state here that this ratio is very different from the quantity *atmError* described previously. The term *atmError* used the value of the Spot (obtained from Asset data) and Strike (obtained from Option contract). The newly defined ration uses the value of Close and Strike, both of which are obtained from the *Option* contract data.

In order to understand the rationale behind this, note that there exists a very complex relationship between past *Option* contract prices and the past asset price dynamics. Including the previous day's contract price would help us account for any auto-regressive feature that might be present in the *Option* price dynamics. Simply put, *Approaches I and II* aimed at trying to find an association between the asset price dynamics and the market perceived price of the contracts. In *Approach III* we knowingly include a member of a time series that we are trying to predict. This key addition makes it possible to factor in all the previously present uncertainties in the asset price dynamics by using the previously determined market price of the contract.

To summarize, when compared to *Approach II*, the only addition in *Approach III* is the addition of a feature computed as the ratio of previous *Option* contract "Close" and the *Option* Strike price. Table 5.1 gives a quick summary cum comparison of the different feature sets generated.

<i>Feature sets at a glance</i>		
<i>Approach I</i>	<i>Approach II</i>	<i>Approach III</i>
Order Statistics	μ of log returns	
ttm	Pairwise covariance	Approach II Features
Interest rate	ttm	
atmError	Interest rate	Ratio of C/K
	atmError	
<i>22 features</i>	<i>17 features</i>	<i>18 features</i>

Table 5.1: Comparison of Feature Sets

Part III

UNDERSTANDING THE RESULTS

THE RESULTS

"Realists do not fear the results of their study."
— Dostoevsky

OVERVIEW This chapter presents the results along with the inferences drawn from those results.

6.1 RESULTS

The approaches described in Chapter 5 are applied to the NIFTY50 dataset. The table 6.1, presents the EM values of all the three approaches when the models are trained on the NIFTY50 dataset.

<i>Experiment Type</i>	<i>XGB</i>	<i>NN</i>
	EM	EM
<i>Approach I</i>	27.06	23.46
<i>Approach II</i>	28.27	20.91
<i>Approach III</i>	22.60	17.90

Table 6.1: Models trained on NIFTY50

As can be seen in the table, the NN performs better than the XGB variant of the model for all approaches. A reason for this could be that the NN is able to capture the higher order dependence of the asset price dynamics better than the XGB variant.

We then perform a different experiment where the models were trained on the complete NIFTY50 dataset, but tested on the entire BANKNIFTY dataset. Note that the model is never exposed to data from BANKNIFTY *Option* contracts during training. Table 6.2 presents the results for this experiment. These results allow us to make an interesting observation. The model performance does not deteriorate significantly when ported to BANKNIFTY contract data. This could imply a similar price forming mechanism. We elaborate on this a bit later in this chapter.

The results in table 6.2 motivate us to perform another variant of the same experiment wherein models were trained on 80% data of both NIFTY50 and BANKNIFTY. And then tested on the rest of the dataset. The results for this experiment are given in table 6.3. The EM metric values decrease significantly

<i>Experiment Type</i>	<i>XGB</i>	<i>NN</i>
	EM	EM
<i>Approach I</i>	34.66	32.57
<i>Approach II</i>	33.61	29.42
<i>Approach III</i>	24.60	24.59

Table 6.2: Models trained on NIFTY50 and tested on BANKNIFTY

when compared to the previous results. These results again seem to suggest that the assets NIFTY50 and BANKNIFTY have similar pricing mechanisms for their respective *Option* contracts. Another possible explanation of the improved results could be the increase in the training data. It is likely that the improved results are caused by a combination of both the factors. This observation is very significant as it suggests the possibility of creating asset agnostic models that perform very well. One can further confirm this hypothesis by examining the quantile-quantile (Q-Q) plot for BANKNIFTY and NIFTY50 datasets. Figure 6.1 shows the Q-Q plot as obtained. It is to be noted here that a Q-Q plot is principally a probability plot. It allows one to compare the probability distribution of 2 random variables. One can easily infer from the Q-Q plot how similar the probability distributions of 2 random variables are.

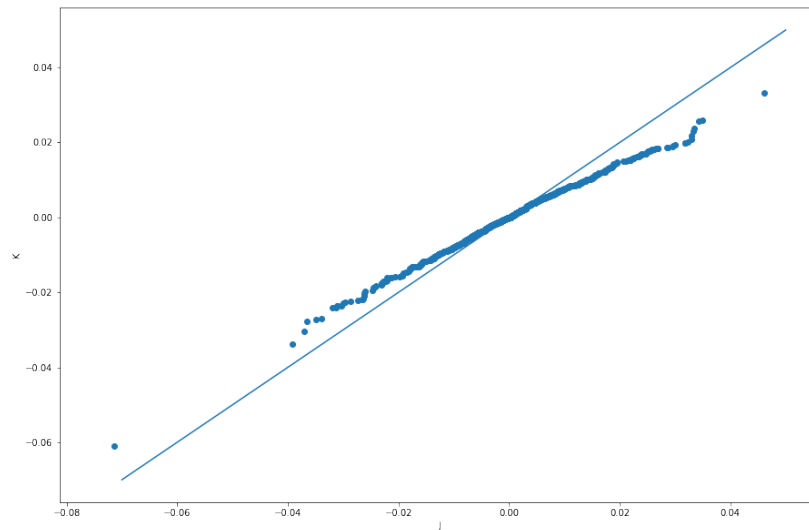


Figure 6.1: Q-Q Plot for BANKNIFTY and NIFTY50 data

As can be seen in the figure, if the distributions are very similar the Q-Q points are observed very close to the line fit on the $y = x$ axis. The figure shows a remarkable fit and lends strength to the hypothesis of the probability distributions of BANKNIFTY and NIFTY50 being very similar.

For the sake of completeness we present the results of the models trained on BANKNIFTY *Option* contracts in table 6.4. The EM values obtained lend credence to the hypothesis that having more data might result in better training

<i>Experiment Type</i>	<i>XGB</i>	<i>NN</i>
	EM	EM
<i>Approach I</i>	23.93	19.16
<i>Approach II</i>	23.46	17.85
<i>Approach III</i>	22.05	14.94

Table 6.3: Models trained on both NIFTY50 and BANKNIFTY

of the models. (BANKNIFTY data contains about 20% more ATM contract data than NIFTY50).

<i>Experiment Type</i>	<i>XGB</i>	<i>NN</i>
	EM	EM
<i>Approach I</i>	23.94	22.41
<i>Approach II</i>	22.47	18.89
<i>Approach III</i>	20.64	15.47

Table 6.4: Models trained on BANKNIFTY

We also present the results of the experiment where the BANKNIFTY trained models were ported to data from NIFTY50 asset in table 6.5

<i>Experiment Type</i>	<i>XGB</i>	<i>NN</i>
	EM	EM
<i>Approach I</i>	31.66	30.61
<i>Approach II</i>	31.82	27.37
<i>Approach III</i>	27.98	22.48

Table 6.5: Models trained on BANKNIFTY and tested on NIFTY50

USING THE MODELS Consider the scenario where the trader has an *Approach I* model that is trained on data from NIFTY50. He wishes to find the fair price of an ATM *Option* as determined by the market. He can set the value of S, K for that trading session and feed in those values along with the 19 values of the log returns of the underlying security. The model would then output a bin value. The trader can then use the bin value to find the approximate price of the contract. The EM value for *Approach I* as given in table 6.1 would determine the bandwidth in which the real value of the contract would lie. Simply put, if the

bin number is extrapolated to a price of 500 Rs for the contract, the fair value of the ATM *Option* would be 500 ± 27.06 Rs.

It is obvious that the smaller the value of the EM metric, the better the model has performed. Put simply, it would give the trader an idea of how accurate the predictions are. As table 6.3 shows, a trader would be interested in a model that gives a small EM value. The DL models seem to show the most promise for a trader.

6.2 EXTENDING THE MODEL

The focus of this thesis has been determining the market price of ATM *Option* contracts. Recall that during the process of data cleaning, we set the value of the filtering parameter p as 0.02 to imply a Strike-Spot tolerance of 2%. This choice of p allowed us to screen out near ATM *Option* contracts from the raw *Option* dataset.

However a trader might be interested in contracts that are ITM/OTM. The models described in this thesis can be extended by simply modifying the filtering algorithm to include *Option* contracts that lie within a particular range of p values. Doing so would create datasets of OTM/ITM contract types as per requirement and models can be built using the feature set generated.

6.3 FUTURE WORK

Significant progress has been made in extending the models described to develop a system that would help qualitatively diagnose the health of markets. Work has also been planned to develop variants of the models described and make the predictions asset independent. It is anticipated that the performance of the models could be improved by incorporating specialized features that are designed to convey more information about market dynamics. An example of this would be using variants of a commonly used technical indicator like the ATR (Average True Range) or the RSI (Relative Strength Index).

Part IV

APPENDIX

APPENDIX

A.1 GBM AND BINOMIAL DISTRIBUTION EXPLAINED

A.1.1 GBM

A geometric Brownian motion is a continuous time stochastic process given by the stochastic differential equation -

$$dS(t) = \mu S(t)dt + \sigma S(t)dB(t) \quad (\text{a.1})$$

where the term $S(t)$ represents the time evolution of an asset, the coefficients μ and σ represent the *drift* and *volatility* respectively and the term $B(t)$ represents a Brownian motion. We present without proof the solution of the above differential equation -

$$S(t) = S(0) \exp \left(\left(\mu - \frac{1}{2} \sigma^2 \right) t + \sigma B(t) \right) \quad (\text{a.2})$$

The above equation is used to model the movement of an asset. The stock prices obtained are then used as inputs to the BSM model described in chapter 1 to obtain the corresponding *Option* price.

A.1.2 Binomial Distribution

If a random variable X follows the binomial distribution $B(n, p)$, the pmf of the distribution is given by -

$$B(n, p) = \binom{n}{k} p^k (1-p)^{n-k} \quad (\text{a.3})$$

where n is the sample size, p is the probability of a successful outcome and k is the number of successful outcome.

A.2 BINOMIAL PRICING MODEL

The binomial model tries to factor in the future payoff of *Options* contract. It is mostly used to evaluate the worth of *American Options*. Such pricing models are also termed as path dependent models as the price of the contract is dependent on the path followed along the nodes of the binomial price tree. We briefly describe how the price of a *Call Option* is computed.

Consider an asset with the initial price S_0 . The price of the asset after n steps in the binomial price tree would then be -

$$S_n = S_0 \xi^n \quad (\text{a.4})$$

where

$$\xi = \begin{cases} u & \text{with probability } p \\ d & \text{with probability } 1 - p \end{cases}$$

The value of S_n can then be written as -

$$S_n = \begin{cases} S_0 u^r d^{n-r} & \text{with probability } p_r \\ p_r & \text{for some } r = [0, n] \text{ such that } \sum_0^n p_r = 1 \end{cases}$$

Note that the value of p_r is given by the binomial distribution.

$$p_r = \binom{n}{r} p^r (1-p)^{n-r} \quad (\text{a.5})$$

We now define a risk neutral measure p^* such that

$$p_r^* = \binom{N}{r} \pi (1-\pi)^{N-r} \quad (\text{a.6})$$

where π is defined as -

$$\pi = \frac{1+r-d}{u-d} \quad (\text{a.7})$$

Using the above, we can define the price of the *Option* contracts at time $t = 0$ as follows -

$$C_0 = E^*((S_N - K) | S_0 = s) \quad (\text{a.8})$$

which finally gives us the value as -

$$C_0 = \sum_{r=0}^N e^{-Nr} (s u^r d^{n-r} - K) p_r^* \quad (\text{a.9})$$

For more details about this, one may refer the book by Paul Wilmott (Wilmott on Quantitative Finance).

A.3 HYPER PARAMETER OPTIMIZATION

Hyper parameters are values that are set before we even begin the process of training a model. They might help in preventing overfitting of data, speed up the training process etc. Hyper parameter tuning usually refers to searching for values of the hyper parameters that output the best model evaluation metric values. There exist specialized algorithms like grid search, to perform a thorough

search for the best values of the hyper parameters. Such algorithms are computationally expensive. For the purpose of this thesis, we initially used the cluster computing facilities at IISER to brute search for the optimum values of the hyper parameters. It was later decided to discard the obtained optimum values as the models then did not port well to other assets (we obtained EM metric values that were almost half of those reported in this thesis). A possible reason for this could be that optimizing the hyper parameters makes the models over fit to a particular asset class. It is however recommended that a trader perform hyper parameter tuning and obtain the best possible model for a given asset class.

A.4 CODES

The complete set of codes and datasets required to replicate the results of the thesis will be uploaded to a GitHub public repository that can be accessed [here](#).

REFERENCES

1. John C. Hull, *Option, Futures and Other Derivatives*. 9th Edition, Prentice Hall, [2016].
2. Yves Hilpisch, *Python for Finance*. 1st Edition, O'Reilly Publications, [2015].
3. Yves Hilpisch, *Derivative Analytics with Python*. 1st Edition, Wiley Finance Series, [2015].
4. Paul Wilmott, *Paul Wilmott on Quantitative Finance*. 2nd Edition, Wiley, [2006].
5. Ian Goodfellow, Yoshua Bengio, Aaron Courville, *Deep Learning*. 1st Edition, MIT Press, [2016].
6. Marcos Lopez de Prado, *Advances in Financial Machine Learning*. Wiley, [2018].
7. Robert H. Schumway, *Time Series Analysis and its Applications*, Springer, [2000].
8. Justin Sirigano, Rama Cont, *Universal Features of Price Formation in Financial Markets: Perspectives From Deep Learning*. (March 16, 2018). Available at SSRN: <https://ssrn.com/abstract=3141294>
9. Gordon Ritter, *Machine Learning for Trading* (August 8, 2017). Available at SSRN: <https://ssrn.com/abstract=3015609>
10. *Essential Options Trading Guide* by Investopedia. Accessed at the url :: <https://www.investopedia.com/options-basics-tutorial-4583012>.
11. *Gradient Descent* by ML Cheat Sheet. Accessed at the url :: https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html

COLOPHON

This document was typeset using a modified version of the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić. `classicthesis` is available for both \LaTeX and \LyX :

<https://bitbucket.org/amiede/classicthesis/>

Final Version as of May 22, 2020 (`classicthesis v4.6`).