An Exhaustive and Extensive Comparision of different methods of Time series Demand forecasting

A project report submitted

in partial fulfillment for the award of the degree of

Bachelor of Technology

in

Electronics and Communication Engineering(Avionics)

by

V.Adithya Krishnan

Department of Avionics

Indian Institute of Space Science and Technology

Thiruvananthapuram, India

April 2019

Certificate

This is to certify that the project report titled ***An Exhaustive and Extensive Comparision of different methods of Time series Demand forecasting*** submitted by **V.Adithya Krishnan**, to the Indian Institute of Space Science and Technology, Thiruvananthapuram, in partial fulfillment for the award of the degree of **Bachelor of Technology** in **Electronics and Communication Engineering(Avionics)** is a bona fide record of the original work carried out by him/her under my supervision. The contents of this project report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr.Sheeba Rani
Associate Professor,
ECE (Avionics), IIST

Dr.B.S.Manoj
Head of Department,
ECE (Avionics), IIST

Place: Thiruvananthapuram
Date: April 2019

Declaration

I declare that this project report titled ***An Exhaustive and Extensive Comparision of different methods of Time series Demand forecasting*** submitted in partial fulfillment for the award of the degree of **Bachelor of Technology** in **Electronics and Communication Engineering(Avionics)** is a record of the original work carried out by me under the supervision of **Dr.Sheeba Rani**, and has not formed the basis for the award of any degree, diploma, associateship, fellowship, or other titles in this or any other Institution or University of higher learning. In keeping with the ethical practice in reporting scientific information, due acknowledgments have been made wherever the findings of others have been cited.

Place: Thiruvananthapuram
Date: April 2019

V.Adithya Krishnan
(SC15B074)

This project report is dedicated to my mentor Nimai Chand Adhikari (CEO, AnalyticsLabs) for his constant belief in me and for all the knowledge and wisdom he shared with me...

Acknowledgements

I would like to express my sincere gratitude to Nimai Chand Adhikari(CEO of AnalyticLabs) who has mentored me throughout the entire duration of my project, and also for giving me the opportunity to work as an intern at AnalyticLabs, Bangalore for the completion of this project. His constant support and his sincerity and passion towards his work has inspired me to do the very same. I would also like to thank all the employees(senior members) of AnalyticLabs and my fellow interns also, cause without their support this project would not have been completed.

I would like to take this opportunity to express my thanks to my project advisor and guide, Dr.Sheeba Rani, who never put a constraint on me and was always supportive to her students. She understood my situation and helped me finish the project in time, and without this support it wouldn't have been possible.

I would also like to thank all the departments of IIST that have helped me with my project, and would also like to give a special thanks to Dr.Deepak Mishra for teaching me the courses that I required for this project and also for guiding me whenever I was stuck.

V.Adithya Krishnan

Abstract

Demand Forecasting is a Time series Forecasting problem, which forecasts or predicts the future demand or sales of any given product. It is undeniably the most important component of any organizations Supply Chain. It determines the estimated demand for the future and sets the level of preparedness that is required on the supply side to match the demand. It goes without saying that if an organization doesn't get its forecasting accurate to a reasonable level, the whole supply chain gets affected. Understandably, Over/Under forecasting has deteriorating impact on any organizations Supply Chain and thereby on the Profits and Losses. Having ascertained the importance of Demand Forecasting, it is only fair to discuss about the forecasting techniques which are used to predict the future values of demand. The input that goes in and the modeling engine which it goes through are equally important in generating the correct forecasts and determining the Forecast Accuracy. There are various advanced Statistical Time series methods for forecasting demand which are being used by Multi-national companies on a daily basis like xgBoost, combination of two or more techniques like Exponential smoothing and Auto Regressive models, and many more; but we focus here on the new and upcoming applications of machine learning and deep learning regression models on demand forecasting. Here, we present a very unique model to model exhaustive comparison of the state of art Statistical, ML and Deep learning models used in forecasting demand. The Data-set used is very clean and basic yet a very suitable one, giving the Shampoo-sales of a company over a period of three years. It is also a very standard data-set used in many competitions.

We begin by going through few of the simple yet powerful statistical methods for demand forecasting, like ARIMA, Exponential Smoothing, etc. Then we move on to more advanced Machine learning Regression methods, and we do an exhaustive analysis on both linear and non-linear models, before moving on to more advanced methods like Deep learning models (LSTMs,MLP). Then an extensive comparison of all the models is done through a common unit of measurement of performance, in our case we use the Root Mean Square Error(RMSE). We end with a complete analysis of our results and observations for all our models.

Contents

List of Figures

List of Tables

Abbreviations

ML	Machine Learning
AR	AutoRegressive
SMA	Simple Moving Average
MA	Moving Average
ARMA	AutoRegressive Moving Average
ARIMA	AutoRegressive Integrated Moving Average
SES	Simple exponential smoothing
HWES	HoltsWinter exponential smoothing
DL	Deep learning
lr	Linear Regression
en	ElasticNet
llars	LassoLARS
pa	PassiveAgressive regressor
knn	K nearest neighbors
cart	DecisionTree regressor
extra	ExtraTree regressor
SVR	Support vector regressor
SVM	Support vector machine
ada	AdaBoost regressor
bag	Bagging regressor
et	ExtraTrees regressor
rf	RandomForest regressor
gbm	Gradient Boosting regressor
LSTM	Long short term memory
MLP	Multi-layer perceptron
RMSE	Root mean squared error
API	Application programming interface

Chapter 1

Introduction

In any supply chain industry, demand forecasting plays a very vital role not only in bringing the profit but also maintaining the right quantity of products at the right time. It is one of the key driving factors in planning and decision making for any Supply Chain Management and Enterprise level. The efficiency or accuracy of the demand forecasting is taking as the major account for taking any major decisions such as capacity building, resource allocation, expansion and forward or backward integration etc.

1.1 Forecasting : A Time series problem

Forecasting is a prediction or an estimation of an actual value in a future time period .It is usual that there might be forecast error as actual results differ from the projected value, long time horizon has chance of more error. This is important to measure the forecast error for adapting corrective action plan. The development of advanced technology enables the Supply Chain Management stakeholders to share real time data and information across the network which helps dual benefit to inventory and customer service. This underlying result of the process is accuracy of forecast which firmly ensures the successful and sustainable business operations. This is called Collaborative Planning, Forecast, and Replenishment. Sales and Operations Planning (SOP) is an integration process used in business organization to ensure efficient coordination among cross functional units to align company strategy with Supply Chain planning. Various forecasting models are used to predict what the demands on the system will be in the future so that appropriate designs and operating plans can be devised. Here we present some of the well known models used widely for forecasting demand, and an extensive comparison of their performances are made and inferences are drawn.

1.2 Models and Algorithms Used

The various forecasting models that are examined can be categorized roughly into three different categories, which are, Statistical models, Machine Learning regression models and finally the newer Deep Learning models.

1.2.1 Statistical methods:

The models under Statistical methods are AutoRegressive(AR), Simple Moving Average(SMA), AutoRegressive Moving average(ARMA), AutoRegressive Integrating Moving average(ARIMA), Simple Exponential Smoothing(SES) and Holts-Winter Exponential Smoothing(HWES) model.

1.2.2 Machine Learning regression methods:

Machine Learning(ML) models again can be classifed into two categories, Linear and Non-Linear models. The Linear ML regression models used are LinearRegression(lr), Lasso, Ridge, ElasticNet(en), HuberRegressor, LassoLars(llars) and PassiveAggressiveRegressor(pa) model. The Non-Linear ML regression models are models KNeighborsRegressor, DecisionTreeRegressor, ExtraTreeRegressor, SVR, AdaBoostRegressor, BaggingRegressor, RandomForestRegressor, ExtraTreesRegressor and GradientBoostingRegressor.

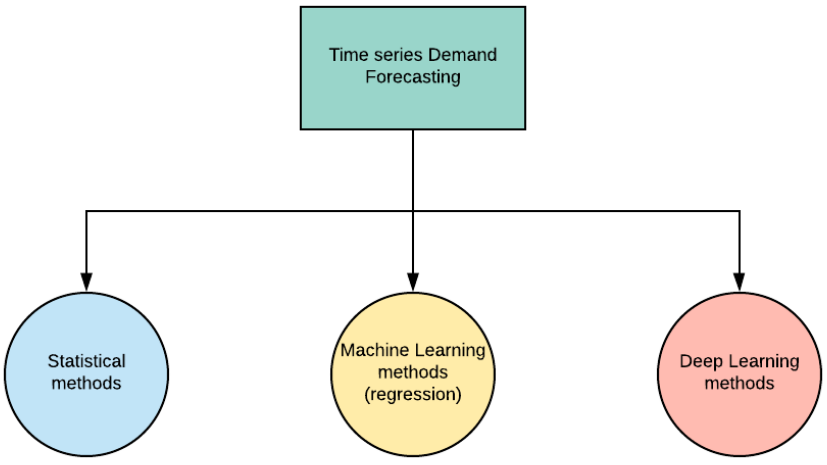


Figure 1.1: The three categories of forecasting models used

1.2.3 Deep Learning methods

Finally, the Deep Learning models used are LSTMs and MultiLayerPerceptrons(MLP). All these models are briefly described in coming sections.

1.3 Project Report outline

The remaining portion of the report is organized as follows: In Chapter 2 we discuss about the literature survey and the related work to our project. It gives a detailed description regarding all of the Demand forecasting models being used by us in this project and towards the end of this chapter, we discuss about the related literature which is related slightly to our work. In Chapter 3, we first give a description of the flowchart of activities which we intend to follow during the course of this project, followed by describing the dataset being used. Then we discuss the methodology and the experiments conducted in detail,which covers the implementation of all our models. Chapter 4 is completely based on the results and inferences drawn from our work.The results consists of the performance evaluation of all the models, the plots of the predicted values, and a table comparing all the models. Finally, Chapter 5 summarizes the entire project and highlights the results of this work

1.4 Summary

In this chapter we understood the importance of Demand forecasting in a Supply chain of any comapny and how there are multiple methods, models and algortihms to forecast demand. The main aim of this work t to evaluate these models under a common performance measure using a standard dataset. [**?**].

Chapter 2

Related Work

In this chapter, we'll look into all the models used, and give a breif introduction to them. We'll start with the Statistical methods, then go on to the linear Machine Learning models, then Non-linear Machine Learning models and finally end with deep learning models.

2.1 Statistical methods

2.1.1 AutoRegressive model (AR):

Autoregression is a time series model which specifies that the output variable depends linearly on its own previous values and on a stochastic term uses observations(an imperfectly predictable term); thus the model is in the form of a stochastic difference equation. In machine learning, an autoregressive model learns from a series of timed steps and takes measurements from previous actions as inputs for a regression model, in order to predict the value of the next time step. The notation for the model involves specifying the order of the model p as a parameter to the AR function, e.g. AR(p). For example, AR(1) is a first-order autoregression model. The method is suitable for univariate time series without trend and seasonal components. The AR(p) model is defined as :

$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i}$,
where $\varphi_1, \ldots, \varphi_p$ are the parameters of the model, c is a constant

2.1.2 Simple Moving Average model (SMA):

The moving average (MA) method models the next step in the sequence as a linear function of the residual errors from a mean process at prior time steps. A moving average is

commonly used with time series data to smooth out short-term fluctuations and highlight longer-term trends or cycles. The threshold between short-term and long-term depends on the application, and the parameters of the moving average will be set accordingly. For example, it is often used in technical analysis of financial data, like stock prices, returns or trading volumes. It is also used in economics to examine gross domestic product, employment or other macroeconomic time series. The notation for the model involves specifying the order of the model q as a parameter to the MA function, e.g. MA(q). For example, MA(1) is a first-order moving average model. The SMA forecast for the value of Y at time t+1 that is made at time t equals the simple average of the most recent m observations:

$\hat{Y}_{t+1} = \frac{Y_t + Y_{t-1} + \ldots + Y_{t-m+1}}{m}$,

where $\hat{Y}_{t+1}$ is the forecast for the timeseries Y.

2.1.3 AutoRegressive Moving Average model (ARMA):

The Autoregressive Moving Average (ARMA) method models the next step in the sequence as a linear function of the observations and resiudal errors at prior time steps. It combines both Autoregression (AR) and Moving Average (MA) models. The notation for the model involves specifying the order for the AR(p) and MA(q) models as parameters to an ARMA function, e.g. ARMA(p, q). The method is suitable for univariate time series without trend and seasonal components. The AR(p) model is defined as :

$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}$,

where $\varphi_1, \ldots, \varphi_p$ and $\theta_1, \ldots, \theta_q$ are the parameters of the model, c is a constant and and the $\varepsilon_t, \varepsilon_{t-1}, \ldots$ are white noise error terms

2.1.4 AutoRegressive Integrated Moving Average model (ARIMA):

The Autoregressive Integrated Moving Average (ARIMA) method models the next step in the sequence as a linear function of the differenced observations and residual errors at prior time steps. The acronym ARIMA is very descriptive, capturing the key aspects of the model itself. Briefly, they are:

- AR: Autoregression. A model that uses the dependent relationship between an observation and some number of lagged observations.

- I: Integrated. The use of differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary.

- MA: Moving Average. A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

The notation for the model involves specifying the order for the AR(p), I(d), and MA(q) models as parameters to an ARIMA function, e.g. ARIMA(p, d, q). An ARIMA model can also be used to develop AR, MA, and ARMA models. The method is suitable for univariate time series with trend and without seasonal components. A nonseasonal ARIMA model is classified as an "ARIMA(p,d,q)" model, where:

- p is the number of autoregressive terms,

- d is the number of nonseasonal differences needed for stationarity, and

- q is the number of lagged forecast errors in the prediction equation.

2.1.5 Simple Exponential Smoothing model (SES):

Exponential smoothing is a time series forecasting method for univariate data that can be extended to support data with a systematic trend or seasonal component. Exponential smoothing forecasting methods are similarto that where a prediction is a weighted sum of past observations, but this model explicitly uses an exponentially decreasing weight for past observations. It is a powerful forecasting method that may be used as an alternative to the popular Box-Jenkins ARIMA family of methods. Let α denote a "smoothing constant" (a number between 0 and 1). One way to write the model is to define a series L that represents the current level (i.e., local mean value) of the series as estimated from data up to the present. The value of L at time t is computed recursively from its own previous value like this:

$\hat{Y}_{t-1} = L_t = \alpha Y_t + (1 - \alpha) L_t$,

where $\hat{Y}_{t-1}$ is the forecast for the next period which is simply the current smoothed value.

2.1.6 Holt Winter's Exponential Smoothing model (HWES):

The Holt Winter's Exponential Smoothing (HWES) also called the Triple Exponential Smoothing method models the next time step as an exponentially weighted linear function of observations at prior time steps, taking trends and seasonality into account. The method is suitable for univariate time series with trend and/or seasonal components. The component form for the additive metod is:

$$\begin{aligned}
\hat{y}_{t+h|t} &= \ell_t + hb_t + s_{t+h-m(k+1)} \\
\ell_t &= \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1}) \\
b_t &= \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1} \\
s_t &= \gamma(y_t - \ell_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m},
\end{aligned}$$

where k is the integer part of $(h-1)/m$, which ensures that the estimates of the seasonal indices used for forecasting come from the final year of the sample. The level equation shows a weighted average between the seasonally adjusted observation $(y_t - s_{t-m})$ and the non-seasonal forecast $(\ell_{t-1}+b_{t-1})$ for time t. The trend equation is identical to Holt's linear method. The seasonal equation shows a weighted average between the current seasonal index, $(y_t - \ell_{t-1} - b_{t-1})$ and the seasonal index of the same season last year (i.e., m time periods ago).

2.2 Linear Machine Learning regression methods

2.2.1 LinearRegression model:

In statistics, linear regression is a linear approach to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models. Like all forms of regression analysis, linear regression focuses on the conditional probability distribution of the response given the values of the predictors.

Given a data set $\{y_i, x_{i1}, \ldots, x_{ip}\}_{i=1}^n$ of n statistical units, a linear regression model assumes that the relationship between the dependent variable y and the p-vector of regressors x is linear. This relationship is modeled through a disturbance term or error variable ε —

an unobserved random variable that adds "noise" to the linear relationship between the dependent variable and regressors. Thus the model takes the form:

$y_i = \beta_0 1 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip} + \varepsilon_i = x_i^T \beta + \varepsilon_i$,

where T denotes the transpose, so that $x_i^T \beta$ is the inner product between vectors x_i and β.

2.2.2 Lasso regression model:

In statistics and machine learning, lasso (least absolute shrinkage and selection operator; also Lasso or LASSO) is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the statistical model it produces. It was originally introduced in geophysics literature in 1986 by Robert Tibshirani.Lasso improves prediction error by shrinking large regression coefficients in order to reduce overfitting,and it also performs covariate selection. Lasso is able to achieve both of these goals by forcing the sum of the absolute value of the regression coefficients to be less than a fixed value, which forces certain coefficients to be set to zero, effectively choosing a simpler model that does not include those coefficients.

Consider a sample consisting of N cases, each of which consists of p covariates and a single outcome. Let y_i be the outcome and $x_i := (x_1, x_2, \ldots, x_p)^T$ be the covariate vector for the i^{th} case. Then the objective of lasso is to solve:

$\min_{\beta_0,\beta} \{\frac{1}{N} \sum_{i=1}^{N} (y_i - \beta_0 - x_i^T \beta)^2)\}$,

Here t is a prespecified free parameter that determines the amount of regularisation. The model is bassically a Linear Model trained with L1 prior as regularizer

2.2.3 Ridge regression model:

In statistics, the method is known as ridge regression, in machine learning it is known as weight decay, and it is a technique for analyzing multiple regression data that suffer from multicollinearity. When multicollinearity occurs, least squares estimates are unbiased, but their variances are large so they may be far from the true value. By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors. This model solves a regression model where the loss function is the linear least squares function and regularization is given by the l2-norm. Also known as Ridge Regression or Tikhonov

regularization. This estimator has built-in support for multi-variate regression (i.e., when y is a 2d-array of shape [n_samples, n_targets]).

The model is bassically a Linear least squares with l2 regularization.

2.2.4 ElasticNet regression model:

In statistics and, in particular, in the fitting of linear or logistic regression models, the elastic net is a regularized regression method that linearly combines the L1 and L2 penalties of the lasso and ridge methods. To overcome the limitaions of Lasso, the elastic net adds a quadratic part to the penalty ($\|\beta\|^2$). The objective function to minimize is in this case:

$\min_w \frac{1}{2n_{samples}}||Xw - y||_2^2 + \alpha\rho||w||_1 + \frac{\alpha(1-\rho)}{2}||w||_2^2$

It is a model of Linear regression with combined L1 and L2 priors as regularizer. This combination allows for learning a sparse model where few of the weights are non-zero like Lasso, while still maintaining the regularization properties of Ridge. We control the convex combination of L1 and L2 using the l1_ratio parameter. Elastic-net is useful when there are multiple features which are correlated with one another. Lasso is likely to pick one of these at random, while elastic-net is likely to pick both. A practical advantage of trading-off between Lasso and Ridge is it allows Elastic-Net to inherit some of Ridge's stability under rotation.

2.2.5 Huber regression model:

Huber regression is basically a Linear regression model that is robust to outliers. The Huber Regressor optimizes the squared loss for the samples where |(y - X'w) / sigma| < epsilon and the absolute loss for the samples where |(y - X'w) / sigma| > epsilon, where w and sigma are parameters to be optimized. The parameter sigma makes sure that if y is scaled up or down by a certain factor, one does not need to rescale epsilon to achieve the same robustness. Note that this does not take into account the fact that the different features of X may be of different scales. This makes sure that the loss function is not heavily influenced by the outliers while not completely ignoring their effect.

2.2.6 LassoLars regression model:

LassoLars is basiaclly a Lasso model fit with Least Angle Regression a.k.a. Lars. It is a Linear Model trained with an L1 prior as regularizer similar to Lasso. In statistics, least-angle

regression (LARS) is an algorithm for fitting linear regression models to high-dimensional datai. Instead of giving a vector result, the LARS solution consists of a curve denoting the solution for each value of the L1 norm of the parameter vector. The algorithm is similar to forward stepwise regression, but instead of including variables at each step, the estimated parameters are increased in a direction equiangular to each one's correlations with the residual.

2.2.7 PassiveAgressive regression model:

The passive-aggressive algorithms are a family of algorithms for large-scale learning. They are similar to the Perceptron in that they do not require a learning rate. However, contrary to the Perceptron, they include a regularization parameter C.

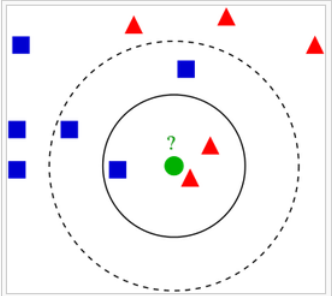
2.3 Non-Linear Machine Learning regression methods

2.3.1 K-Neighbors regression model:

K-Neighbors regression model is based on K- Nearest Neigbors. In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output is the property value for the object. This value is the average of the values of k nearest neighbors. k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms. Neighbors-based regression can be used in cases where the data labels are continuous rather than discrete variables. The label assigned to a query point is computed based on the mean of the labels of its nearest neighbors. The neighbors are taken from a set of objects for which object property value is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

2.3.2 Decision Tree regression model:

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. For instance, in the example below, decision trees learn from data to approximate a sine curve with a set of



Example of *k*-NN classification. The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If *k* = *3* (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If *k* = *5* (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

Figure 2.1: K nearest neighbors example (classification)

if-then-else decision rules. The deeper the tree, the more complex the decision rules and the fitter the model.

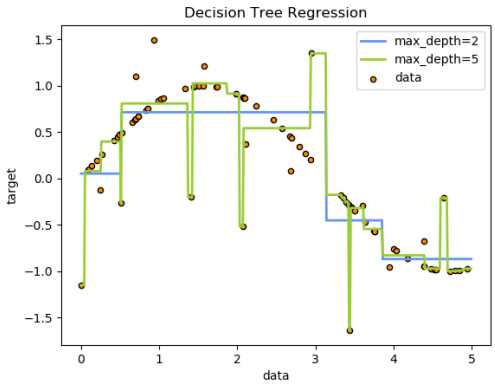


Figure 2.2: Decision Tree example

2.3.3 ExtraTrees regression model:

ExtraTrees is an extremely randomized tree regressor. Extra-trees differ from classic decision trees in the way they are built. When looking for the best split to separate the samples of a node into two groups, random splits are drawn for each of the max_features randomly

selected features and the best split among those is chosen. When max_features is set 1, this amounts to building a totally random decision tree. The Extra-Tree method (standing for extremely randomized trees) was proposed in [GEW06], with the main objective of further randomizing tree building in the context of numerical input features, where the choice of the optimal cut-point is responsible for a large proportion of the variance of the induced tree.

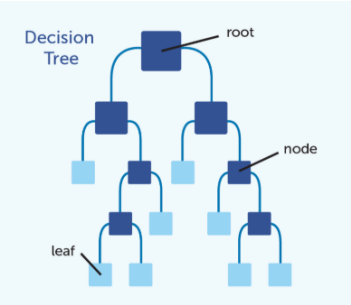


Figure 2.3: Decision Tree

2.3.4 Support Vector Regressor model(SVR):

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection. The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The model produced by Support Vector Regression depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear

classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

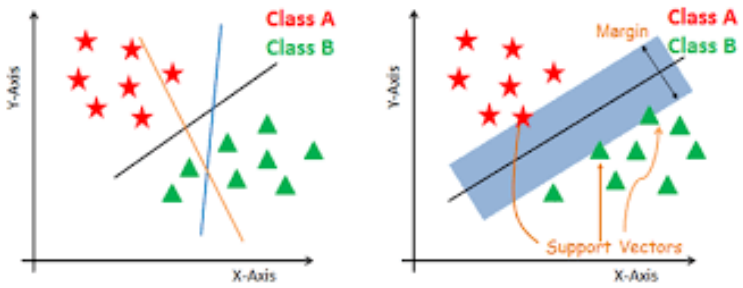


Figure 2.4: SVM algorithm

2.3.5 AdaBoost regression model:

The core principle of AdaBoost is to fit a sequence of weak learners (i.e., models that are only slightly better than random guessing, such as small decision trees) on repeatedly modified versions of the data. The predictions from all of them are then combined through a weighted majority vote (or sum) to produce the final prediction. The data modifications at each so-called boosting iteration consist of applying weights $w_1, w_2, \ldots, w_N$ to each of the training samples. Initially, those weights are all set to $w_i = 1/N$, so that the first step simply trains a weak learner on the original data. For each successive iteration, the sample weights are individually modified and the learning algorithm is reapplied to the reweighted data. At a given step, those training examples that were incorrectly predicted by the boosted model induced at the previous step have their weights increased, whereas the weights are decreased for those that were predicted correctly. As iterations proceed, examples that are difficult to predict receive ever-increasing influence. Each subsequent weak learner is thereby forced to concentrate on the examples that are missed by the previous ones in the sequence

2.3.6 Bagging regression model:

In ensemble algorithms, bagging methods form a class of algorithms which build several instances of a black-box estimator on random subsets of the original training set and then

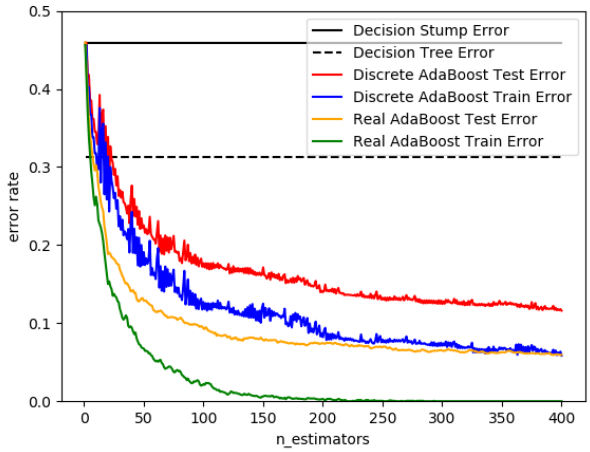


Figure 2.5: AdaBoost example (comparison)

aggregate their individual predictions to form a final prediction. These methods are used as a way to reduce the variance of a base estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it. In many cases, bagging methods constitute a very simple way to improve with respect to a single model, without making it necessary to adapt the underlying base algorithm. As they provide a way to reduce overfitting, bagging methods work best with strong and complex models (e.g., fully developed decision trees), in contrast with boosting methods which usually work best with weak models (e.g., shallow decision trees).

2.3.7 RandomForest regression model:

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. In random forests, each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set. In addition, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. Instead, the split that is picked is the best split among a random subset of the features. As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model.

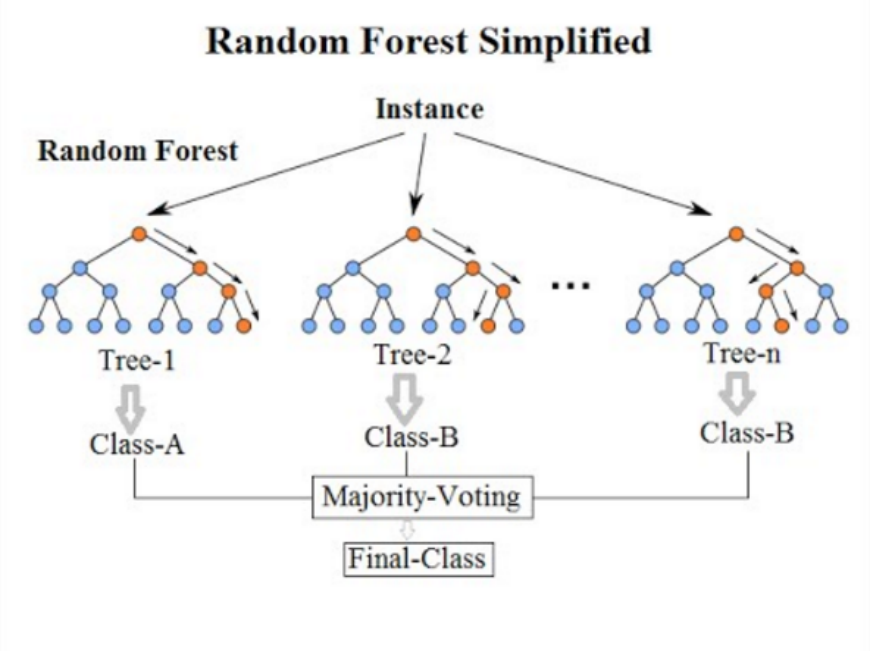


Figure 2.6: simple explanation of Random Forest

2.3.8 GradientBoosting regression model:

Gradient Tree Boosting or Gradient Boosted Regression Trees (GBRT) or simply GradientBoosting(GB) is a generalization of boosting to arbitrary differentiable loss functions. GBRT is an accurate and effective off-the-shelf procedure that can be used for both regression and classification problems. Gradient Tree Boosting models are used in a variety of areas including Web search ranking and ecology. The advantages of GBRT are:

- Natural handling of data of mixed type (= heterogeneous features)
- Predictive power
- Robustness to outliers in output space (via robust loss functions)

2.4 Deep Learning methods

2.4.1 Multi-Layer Perceptrons(MLP):

A multilayer perceptron (MLP) is a class of feedforward artificial neural network. A MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation

function. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.

Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a function $f(\cdot) : R^m \rightarrow R^o$ by training on a dataset, where m is the number of dimensions for input and is the number of dimensions for output. Given a set of features $X = x_1, x_2, ..., x_m$ and a target , it can learn a non-linear function approximator for either classification or regression. It is different from logistic regression, in that between the input and the output layer, there can be one or more non-linear layers, called hidden layers. Figure 1 shows a one hidden layer MLP with scalar output.

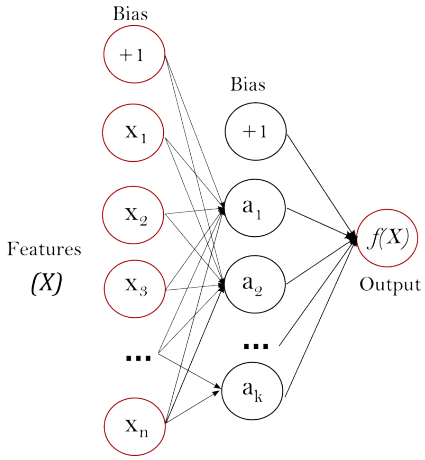


Figure 2.7: Multi-Layer perceptron network

The leftmost layer, known as the input layer, consists of a set of neurons $\{x_i | x_1, x_2, ..., x_m\}$ representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation $w_1x_1 + w_2x_2 + ... + w_mx_m$, followed by a non-linear activation function $g(\cdot) : R \rightarrow R$ - like the hyperbolic tan function. The output layer receives the values from the last hidden layer and transforms them into output values.

2.4.2 LSTM:

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections that make it a "general purpose computer". A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The

cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

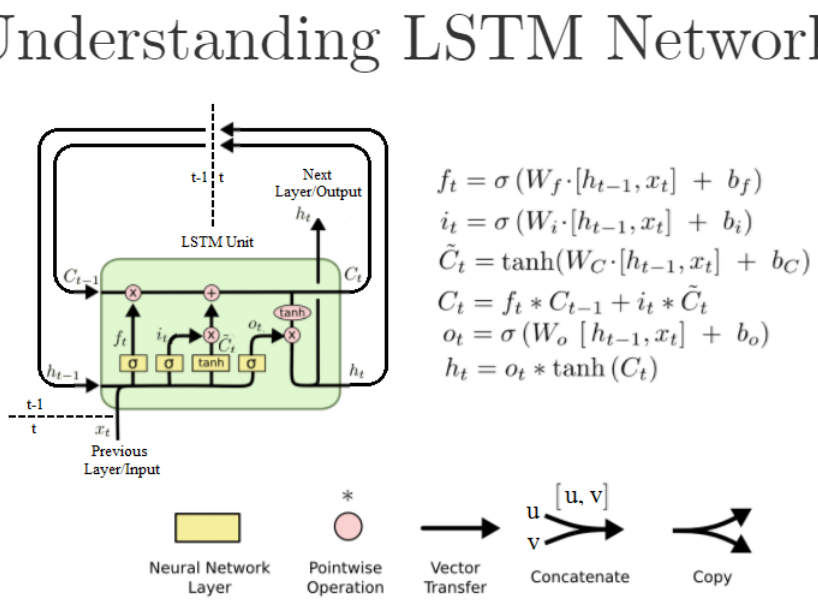


Figure 2.8: LSTM internal network and functioning

LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the exploding and vanishing gradient problems that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs.

2.5 Summary

In this chapter, we saw the breif introduction to all the Statistical, Linear and Non-linear Machine Learning regression models and the Deep Learning models used by us in this project. In the coming sections we'll discuss about the methodology and results drawn.

Chapter 3

Methodology

In this chapter we'll look into the methodology followed and the way in which our forecasting models where implemented. It also covers a brief introduction to our data (used) and the necessary pre-processing steps required for our data.

3.1 Data used and its pre-processing

Machine learning can be applied to time series datasets.These are problems where a numeric or categorical value must be predicted, but the rows of data are ordered by time. In order to fit all the models discussed in the previous section, we needed to choose a good quality standard datasets on which to train and forecast. Time series datasets that only have one variable are called univariate datasets. These datasets are a great place to get started because:

- They are so simple and easy to understand.
- You can plot them easily in excel or your favorite plotting tool.
- You can easily plot the predictions compared to the expected results.
- You can quickly try and evaluate a suite of traditional and newer methods.

One such univariate standard time series dataset used in many competetions is the Shampoo-sales dataset. This dataset describes the monthly number of sales of shampoo over a 3-year period. The units are a sales count and there are 36 observations. The original dataset is credited to Makridakis, Wheelwright, and Hyndman

For applying our Statistical models like ARIMA, SES, etc, we can directly fit these models onto our dataset. The dataset first should be divided into train and test data, in order to calculate the performance evaluation metric, Root mean square error(RMSE). Root

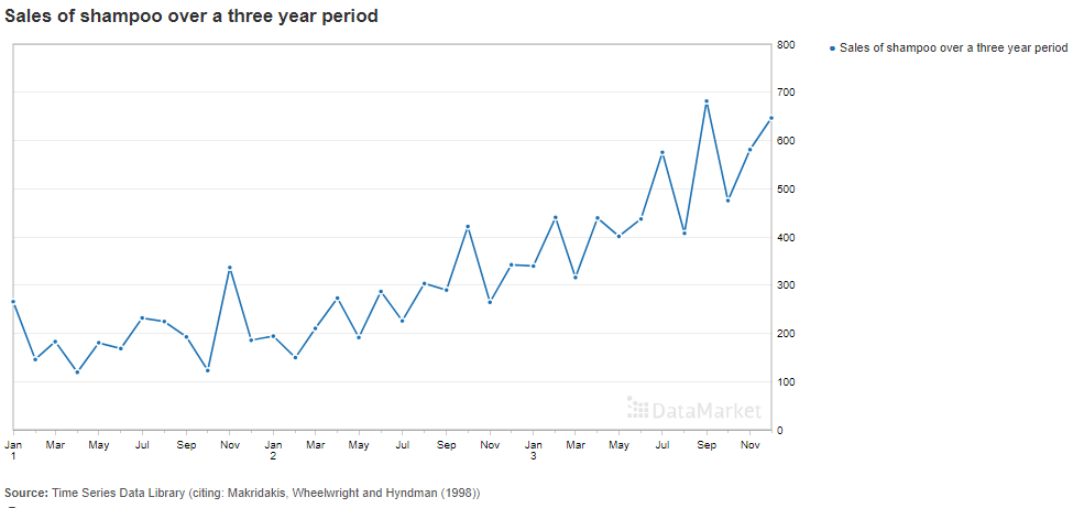


Figure 3.1: Plot of our Shampoo-sales dataset

```
"Month","Sales of shampoo over a three year period"
"1-01",266.0
"1-02",145.9
"1-03",183.1
"1-04",119.3
"1-05",180.3
```

Figure 3.2: Head of our Shampoo-sales dataset

Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit. Root mean square error is commonly used in climatology, forecasting, and regression analysis to verify experimental results. The RMSD of predicted values $\hat{y}_t$ for times t of a regression's dependent variable y_t, with variables observed over T times, is computed for T different predictions as the square root of the mean of the squares of the deviations:

$$\text{RMSE} = \sqrt{\frac{\sum_{t=1}^{T}(\hat{y}_t - y_t)^2}{T}},$$

The effect of each error on RMSD is proportional to the size of the squared error; thus larger errors have a disproportionately large effect on RMSD. Consequently, RMSD is sensitive to outliers. As the data used is a clean and standard dataset, thus the errors or outliers in the dataset are very low, hence initial pre-processing isnt required.

When we use the same data on Machine learning and Deep learning models then a few pre-processing steps are required namely:

- Transform the time series into a supervised learning problem
- Transform the time series data so that it is stationary.
- Transform the observations to have a specific scale.

The LSTM model in Keras assumes that your data is divided into input (X) and output (y) components, i.e. a supervised learning problem. For a time series problem, we can achieve this by using the observation from the last time step (t-1) as the input and the observation at the current time step (t) as the output. We can achieve this using the shift() function in Pandas that will push all values in a series down by a specified number places.

```
            0           0
0    0.000000  266.000000
1  266.000000  145.899994
2  145.899994  183.100006
3  183.100006  119.300003
4  119.300003  180.300003
```

Figure 3.3: Supervised learning Shampoo-sales dataset

The Shampoo Sales dataset is not stationary. This means that there is a structure in the data that is dependent on the time. Specifically, there is an increasing trend in the data. Stationary data is easier to model and will very likely result in more skillful forecasts. The trend can be removed from the observations, then added back to forecasts later to return the prediction to the original scale and calculate a comparable error score. A standard way to remove a trend is by differencing the data. That is the observation from the previous time step (t-1) is subtracted from the current observation (t). This removes the trend and we are left with a difference series, or the changes to the observations from one time step to the next.

```
Name: Sales, dtype: float64
0   -120.1
1     37.2
2    -63.8
3     61.0
4    -11.8
dtype: float64
```

Figure 3.4: Differenced Shampoo-sales dataset

Like other neural networks, LSTMs expect data to be within the scale of the activation function used by the network. The default activation function for LSTMs is the hyperbolic tangent (tanh), which outputs values between -1 and 1. This is the preferred range for the time series data. To make the experiment fair, the scaling coefficients (min and max) values must be calculated on the training dataset and applied to scale the test dataset and

any forecasts. This is to avoid contaminating the experiment with knowledge from the test dataset, which might give the model a small edge. We can transform the dataset to the range [-1, 1] using the MinMaxScaler class.

```
Name: Sales, dtype: float64
0   -0.478585
1   -0.905456
2   -0.773236
3   -1.000000
4   -0.783188
dtype: float64
```

Figure 3.5: Scaled and Differenced Shampoo-sales dataset

Now that the dataset has been loaded, pre-processed and fit into a dataframe, we can start fitting our models one by one in order to forecast the future sales. But before going on to fit our models, we need to setup a baseline, to compare the models with. We'll be talking about it in the next section.

3.2 Baseline prediction

Establishing a baseline is essential to any time series forecasting problem. A baseline's performance gives you an idea of how well all other models will actually perform on your problem. It provides a point of comparison of the forecast performance. It is also a point of reference for all other modeling techniques on your problem. If a model achieves performance at or below the baseline, the technique should be abandoned, as the baseline techniques doesnt consider any details about the data and if an algorithm which does consider is performing poorer thanthe baseline, then it should be discarded.

The technique used to generate a forecast to calculate the baseline performance must be easy to implement and naive of problem-specific details. The goal is to get a baseline performance on your time series forecast problem as quickly as possible so that you can get to work better understanding the dataset and developing more advanced models. Three properties of a good technique for making a baseline forecast are:

- Simple: A method that requires little or no training or intelligence.
- Fast: A method that is fast to implement and computationally trivial to make a prediction.
- Repeatable: A method that is deterministic, meaning that it produces an expected output given the same input.

A common algorithm used in establishing a baseline performance is the persistence algorithm.

The persistence algorithm uses the value at the previous time step (t-1) to predict the expected outcome at the next time step (t+1). This satisfies the three above conditions for a baseline forecast. Using this we find out the Baseline for our Shampoo-sales dataset, and the RMSE comes out to be 136.71, and hence below this our models can be abandoned. A line plot of the test dataset (blue) compared to the predicted values (orange) is also created showing the persistence model forecast in context.

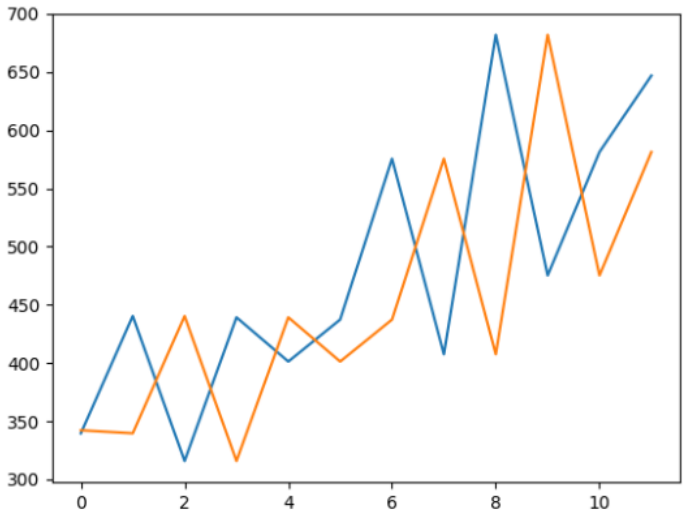


Figure 3.6: Persistance forecast of observed vs predicted

Now that we have our data as well as our baseline ready, we start by fitting our dataset on the Staistical models, then Machine Learning and then finally Deep Learning models.

3.3 Fitting of models and finding forecasts

3.3.1 Statistical models:

An ARIMA model can be created using the statsmodels library as follows:

- Define the model by calling ARIMA() and passing in the p, d, and q parameters.
- The model is prepared on the training data by calling the fit() function.
- Predictions can be made by calling the predict() function and specifying the index of the time or times to be predicted.

A value of 0 can be used for a parameter, which indicates to not use that element of the model. This way, the ARIMA model can be configured to perform the function of an ARMA model, and even a simple AR, I, or MA model.

But in order to get the best forecast output, we need to optimize the hyperparameters p,d and q while fitting the model on our datasaet. There are two ways in which this can be performed:

- Box-Jenkins Methodology:- In summary, the steps of this process are as follows:-
  - Model Identification. Use plots and summary statistics to identify trends, seasonality, and autoregression elements to get an idea of the amount of differencing and the size of the lag that will be required.
  - Parameter Estimation. Use a fitting procedure to find the coefficients of the regression model.
  - Model Checking. Use plots and statistical tests of the residual errors to determine the amount and type of temporal structure not captured by the model.

  The process is repeated until either a desirable level of fit is achieved on the in-sample or out-of-sample observations (e.g. training or test datasets). This is the conventional method of fitting an Arima model, but there is one more way of computationally grid searching the hyperparameters.

- Grid Search HyperParameter optimization: In this process, we will fine tune the parameters p, d and q using our system, instead of manually checking for the best fit. This can be achieved in two ways:-
  - Splitting the Dataset into train and test and then iterating for multiple combination values for pdq and checking for the error in fit (RMSE) by using predicted and expected in order to choose the best parameters.
  - Using library statsmodel.tsa.arima_model to fit the arima model (on entire dataset) using multiple combination values for pdq again, but this time using the AIC parameter that is returned, in order to find the best fit (minimum AIC).

Grid Search Hyperparameter optimization was chosen to reduce human effort, and in it the first way is more robust and it also gives us an idea about the error in fit, hence that was chosen.

A similar procedure can be used in optimizing the hyperparameters of HoltWinters model (triple exponential smoothing) and for fitting the model on our dataset.

3.3.2 Machine Learning(ML) regression models:

Most machine learning models do not directly support the notion of observations over time. Instead, the lag observations must be treated as input features in order to make predictions. This is a benefit of machine learning algorithms for time series forecasting. Specifically, that they are able to support large numbers of input features. These could be lag observations for one or multiple input time series. Other general benefits of machine learning algorithms for time series forecasting over classical methods include:

- Ability to support noisy features and noise in the relationships between variables.
- Ability to handle irrelevant features.
- Ability to support complex relationships between variables.

So after converting the unsupervised learning problem into a supervised learning problem by setting the lags as input features, all the linear and the non-linear ML models can be fit on our dataset and the forecast can be made. The models where listed in a dictionary (as shown in the fig.) and then called into a fit_model function in order to fit into our dataset. The linear models where imported from sklearn.linear_model and the non-linear models from sklearn.ensemble, sklearn.tree, sklearn.svm and sklearn.neighbors.

```
# prepare a list of ml models
def get_models(models=dict()):
    # linear models
    models['lr'] = LinearRegression()
    models['lasso'] = Lasso()
    models['ridge'] = Ridge()
    models['en'] = ElasticNet()
    models['huber'] = HuberRegressor()
    models['llars'] = LassoLars()
    models['pa'] = PassiveAggressiveRegressor(max_iter=1000, tol=1e-3)
    models['sgd'] = SGDRegressor(max_iter=1000, tol=1e-3)
    print('Defined %d models' % len(models))
    return models
```

Figure 3.7: get_models() function: linear

Then the plots for actual vs predicted where made and the RMSE values for each model was recorded.

3.3.3 Deep Learning models:

Similar to ML models, for a data to be fit on a LSTM or a MLP network, the lag observations must be treated as input features in order to make predictions, i.e. supervised learning problem is essential. So again, after converting the unsupervised learning problem into

```python
# prepare a list of ml models
def get_models(models=dict()):
    # non-linear models
    models['knn'] = KNeighborsRegressor(n_neighbors=7)
    models['cart'] = DecisionTreeRegressor()
    models['extra'] = ExtraTreeRegressor()
    models['svmr'] = SVR()
    # # ensemble models
    n_trees = 100
    models['ada'] = AdaBoostRegressor(n_estimators=n_trees)
    models['bag'] = BaggingRegressor(n_estimators=n_trees)
    models['rf'] = RandomForestRegressor(n_estimators=n_trees)
    models['et'] = ExtraTreesRegressor(n_estimators=n_trees)
    models['gbm'] = GradientBoostingRegressor(n_estimators=n_trees)
    print('Defined %d models' % len(models))
    return models
```

Figure 3.8: get_models() function: non-linear

a supervised learning problem by setting the lags as input features, we can structure the model by using the Sequential Keras API to define the network, and then fitting the dataset over it.

3.3.3.1 Multi-Layer Perceptron (MLP):

We will use a base MLP model with 1 neuron hidden layer, a rectified linear activation function on hidden neurons, and linear activation function on output neurons. A batch size of 4 is used where possible, with the training data truncated to ensure the number of patterns is divisible by 4. In some cases a batch size of 2 is used. Normally, the training dataset is shuffled after each batch or each epoch, which can aid in fitting the training dataset on classification and regression problems. Shuffling was turned off for all experiments as it seemed to result in better performance. The model will be fit using the efficient ADAM optimization algorithm and the mean squared error loss function. Now we'll run various experiments in order to find optimized hyperparameters. Each experimental scenario will be run 30 times (in order to account for randomness) and the RMSE score on the test set will be recorded from the end each run and a mean RMSE will be calculated.

The experiments conducted on the MLP network are:-

- varying the number of epochs,
- varying the number of hidden layer neurons,
- varying the hidden layer neurons with lag

In the first experiment, we will investigate varying the number of training epochs for a simple MLP with one hidden layer and one neuron in the hidden layer. We will use a batch size of 4 and evaluate training epochs 50, 100, 500, 1000, and 2000. Another angle to

consider with a network configuration is how it behaves over time as the model is being fit. We can evaluate the model on the training and test datasets after each training epoch to get an idea as to if the configuration is overfitting or underfitting the problem. We will use this diagnostic approach on the top result from each set of experiments. A total of 10 repeats of the configuration will be run and the train and test RMSE scores after each training epoch plotted on a line plot. In this case, we will use this diagnostic on the MLP fit for 1000 epochs.

In the second experiment, we will look at varying the number of neurons in the single hidden layer. Increasing the number of neurons can increase the learning capacity of the network at the risk of overfitting the training data. We will explore increasing the number of neurons from 1 to 5 and fit the network for 1000 epochs. Again a diagnostic on network's behaviour over time is run.

```
model = Sequential()
model.add(Dense(neurons, activation='relu', input_dim=X.shape[1]))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X, y, epochs=nb_epoch, batch_size=batch_size, verbose=0, shuffle=Fa
return model
```

Figure 3.9: MLP Keras Sequential API structure

In the last experiment, we will look at increasing the lag observations as input, whilst at the same time increasing the capacity of the network. Increased lag observations will automatically scale the number of input neurons. For example, 3 lag observations as input will result in 3 input neurons. The added input will require additional capacity in the network. As such, we will also scale the number of neurons in the one hidden layer with the number of lag observations used as input. We will use odd numbers of lag observations as input from 1, 3, 5, and 7 and use the same number of neurons respectively. The change to the number of inputs affects the total number of training patterns during the conversion of the time series data to a supervised learning problem. As such, the batch size was reduced from 4 to 2 for all experiments in this section. A total of 1000 training epochs are used in each experimental run. And the time behaviour diagnostics is also run.

3.3.3.2 LSTM:

By default, an LSTM layer in Keras maintains state between data within one batch. A batch of data is a fixed-sized number of rows from the training dataset that defines how many patterns to process before updating the weights of the network. State in the LSTM layer between batches is cleared by default, therefore we must make the LSTM stateful.

This gives us fine-grained control over when state of the LSTM layer is cleared, by calling the reset_states() function. The LSTM layer expects input to be in a matrix with the dimensions: [samples, time steps, features].

- Samples: These are independent observations from the domain, typically rows of data.
- Time steps: These are separate time steps of a given variable for a given observation.
- Features: These are separate measures observed at the time of observation.

We have some flexibility in how the Shampoo Sales dataset is framed for the network. We will keep it simple and frame the problem as each time step in the original sequence is one separate sample, with one timestep and one feature. Once compiled, it can be fit to the training data. Because the network is stateful, we must control when the internal state is reset. Therefore, we must manually manage the training process one epoch at a time across the desired number of epochs.

```
model = Sequential()
model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1], X.shape[2]), sta
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
```

Figure 3.10: LSTM Keras Sequential API structurer

Now again, like in the case of MLP, we'll perform some experiments to tune the hyperparameters in order to optimize them. Each experimental scenario will be run 10 times. The reason for this is that the random initial conditions for an LSTM network can result in very different results each time a given configuration is trained. Again a diagnostic approach will be also be used to investigate model configurations. This is where line plots of model skill over time (training iterations called epochs) will be created and studied for insight into how a given configuration performs and how it may be adjusted to elicit better performance. The model will be evaluated on both the train and the test datasets at the end of each epoch and the RMSE scores saved. The train and test RMSE scores at the end of each scenario are printed to give an indication of progress. The series of train and test RMSE scores are plotted at the end of a run as a line plot. Train scores are colored blue and test scores are colored orange.The experiments conducted on the MLP network are:-

- varying the number of epochs,
- varying the number of hidden layer neurons,

- varying the hidden layer neurons with lag

The first LSTM parameter we will look at tuning is the number of training epochs. The model will use a batch size of 4, and a single neuron. We will explore the effect of training this configuration for different numbers of training epochs, starting with 500, then 1000, 2000, 4000 and 6000. This concludes the first experiment.

Batch size controls how often to update the weights of the network. Importantly in Keras, the batch size must be a factor of the size of the test and the training dataset. In the previous experimentn exploring the number of training epochs, the batch size was fixed at 4, which cleanly divides into the test dataset (with the size 12) and in a truncated version of the test dataset (with the size of 20). In this section, we will explore the effect of varying the batch size. We will hold the number of training epochs constant at 1000. This will be the second experiment.

In the last experiment, we will investigate the effect of varying the number of neurons in the network. The number of neurons affects the learning capacity of the network. Generally, more neurons would be able to learn more structure from the problem at the cost of longer training time. More learning capacity also creates the problem of potentially overfitting the training data. We will use a batch size of 4 and 1000 training epochs. This concludes all our experiments. For each experiment, the mean RMSE is taken to be the performance measure.

3.4 Summary

In this chapter we saw the methodology and algorithms used in forecasting the Shampoo-sales, and we went in-depth into all the initial steps to be taken and also about the experiments which were conducted in each model. The results of this section and of the project will be followed up in the next chapter.

Chapter 4

Results and Inferences

In this chapter we compile all the results, plots and tables, which we obtained from our experiments, and by fitting all our models on the shampoo sales dataset. we again split the chapter into three sections for displaying the results of Statistical models, ML models and Deep learning models seperately. Finally we create a tabular comparison of all our models based on their RMSE scores. We end the chapter witha summary.

4.1 Statistical Model results

Here we will display all the results of our statistical models, i.e. the actuals vs predicted plot and the RMSE scores.

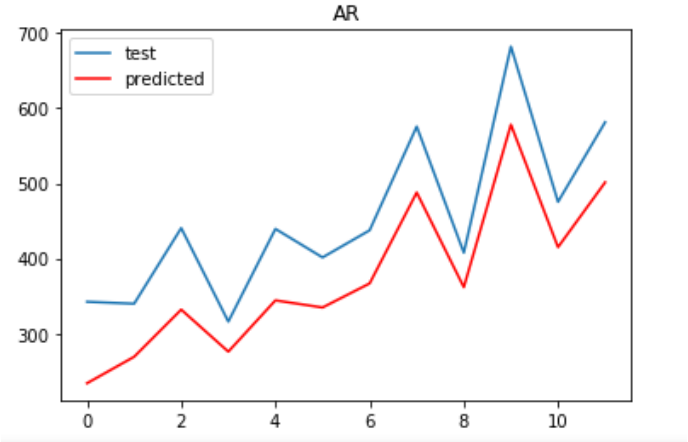


Figure 4.1: AutoRegressive model (AR): Actuals vs Predicted

From the plots above, we can see that ARIMA and HoltsWinter Exponential Smoothing(HWES) are have the best fit, as their predicted values are quite close to the expected.

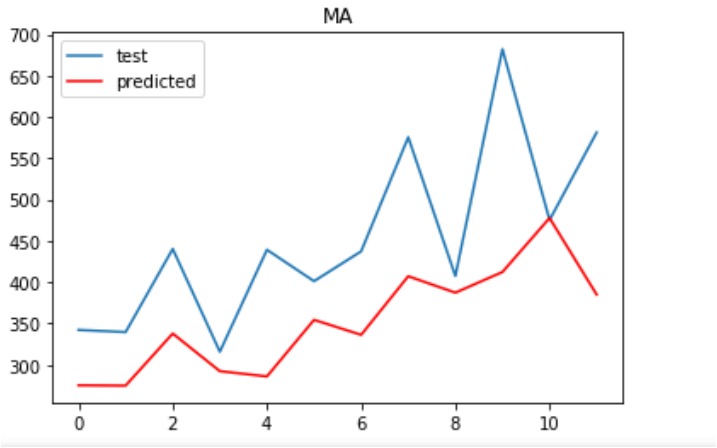


Figure 4.2: Simple Moving Average model (SMA): Actuals vs Predicted

Figure 4.3: AutoRegressive Moving average model (ARMA): Actuals vs Predicted

Hence their RMSE scores should also be the least, which is indeed the case. Below we create a table stating all the RMSE scores of the Statistical methods.

Table 4.1: RMSE scores of the Statistical models

Model	RMSE
AutoRegression(AR)	81.297
Moving Average(SMA)	127.633
AutoRegression Moving Average(ARMA)	88.089
AutoRegression Integrated Moving Average(ARIMA)	68.519
Simple Exponential Smoothing(SES)	104.762
HoltsWinter Exponential Smoothing(HWES)	71.212

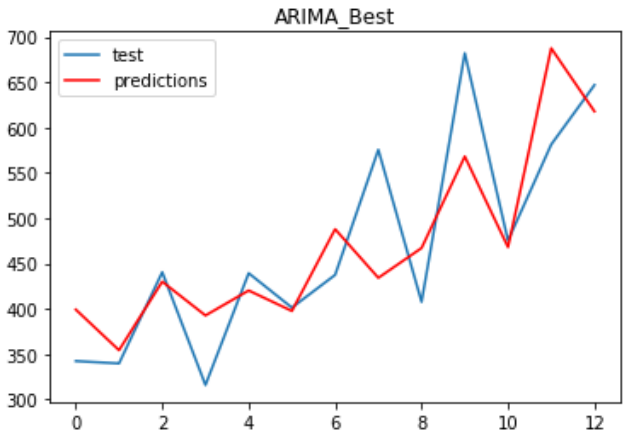


Figure 4.4: AutoRegressive Integrated Moving Average model (ARIMA): Actuals vs Predicted

```
ARIMA(0, 0, 0) MSE=52425.268
ARIMA(0, 0, 1) MSE=38145.167
ARIMA(0, 0, 2) MSE=23989.567
ARIMA(0, 1, 0) MSE=18003.173
ARIMA(0, 1, 1) MSE=9558.410
ARIMA(0, 2, 0) MSE=67339.808
ARIMA(0, 2, 1) MSE=18323.163
ARIMA(1, 0, 0) MSE=23112.958
ARIMA(1, 1, 0) MSE=7121.373
ARIMA(1, 1, 1) MSE=7003.683
ARIMA(1, 2, 0) MSE=18607.980
ARIMA(2, 1, 0) MSE=5689.932
ARIMA(2, 1, 1) MSE=7759.707
ARIMA(2, 2, 0) MSE=9860.948
ARIMA(4, 1, 0) MSE=6649.594
ARIMA(4, 1, 1) MSE=6796.279
ARIMA(4, 2, 0) MSE=7596.332
ARIMA(4, 2, 1) MSE=4694.873
ARIMA(6, 1, 0) MSE=6810.080
ARIMA(6, 2, 0) MSE=6261.107
ARIMA(8, 0, 0) MSE=7256.028
ARIMA(8, 1, 0) MSE=6579.403
Best ARIMA(4, 2, 1) MSE=4694.873
```

Figure 4.5: ARIMA hyperparameter optimization

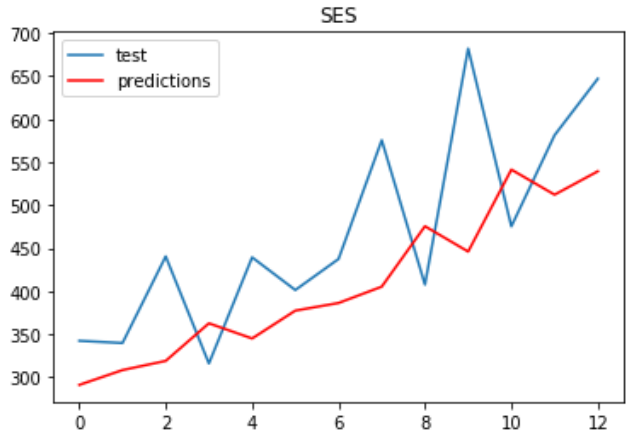


Figure 4.6: Simple Exponential Smoothing (SES): Actuals vs Predicted

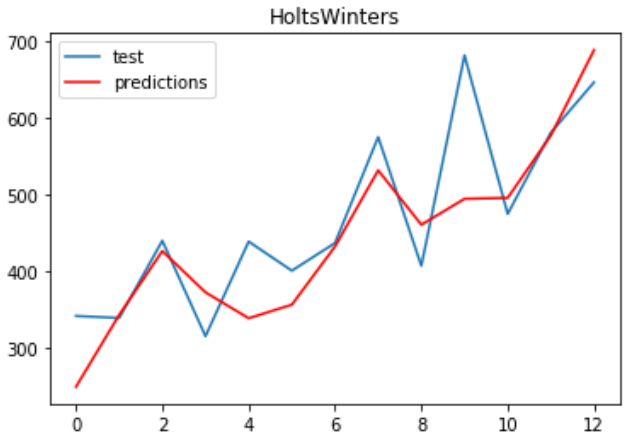


Figure 4.7: HoltsWinter Exponential Smoothing model (HWES): Actuals vs Predicted

4.2 Machine Learning model results

Here we will display all the results of our machine learning models, i.e. the actuals vs predicted plot and the RMSE scores. First we'll display the linear models, then followed by the non-linear models. Here the Expected/Actuals is the raw_values and the predicted values are given the name 'inverted'.

4.2.1 Linear ML model results:

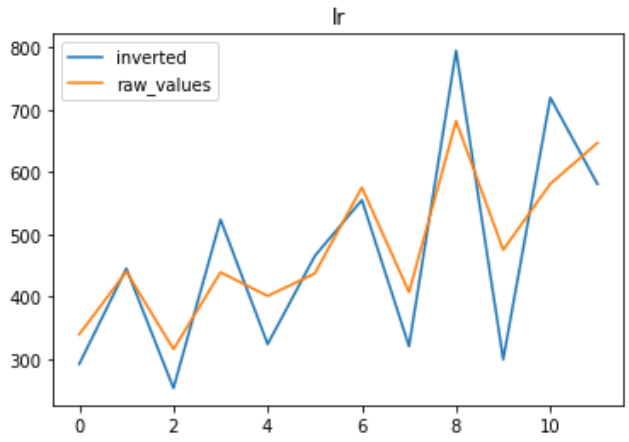


Figure 4.8: Linear Regression model (lr): Actuals vs Predicted

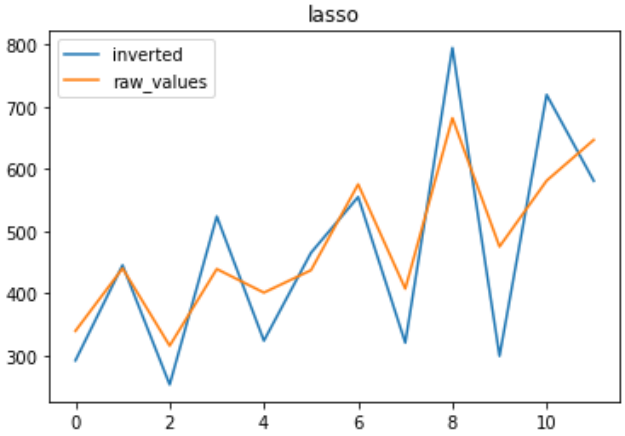


Figure 4.9: Lasoo regression model: Actuals vs Predicted

All the Machine learning linear models give almost a similar fit and when we see the RMSE scores, we can see that all the models have RMSE scores in the same range. But

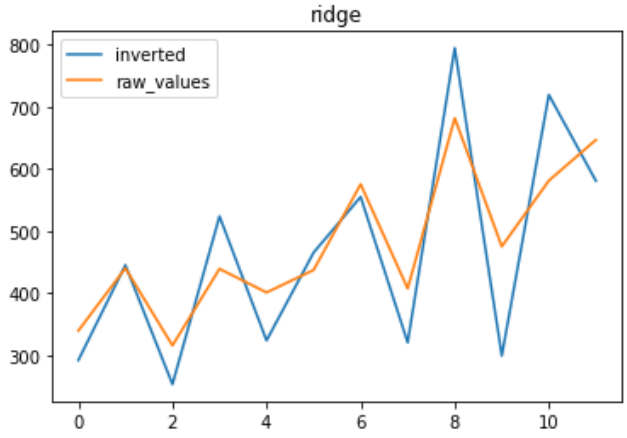


Figure 4.10: Ridge regression model: Actuals vs Predicted

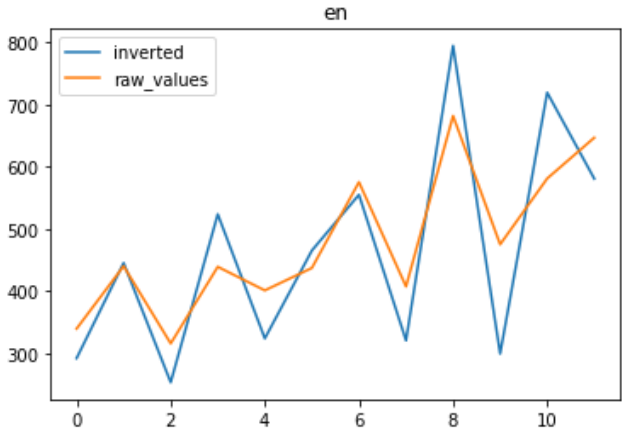


Figure 4.11: ElasticNet regression model (en): Actuals vs Predicted

the best performing model here is the online training model, PassiveAgressive regression model which is giving the best fit. We tabulate the RMSE scores of all the linear ML models to better understand and compare the models based on their performance.

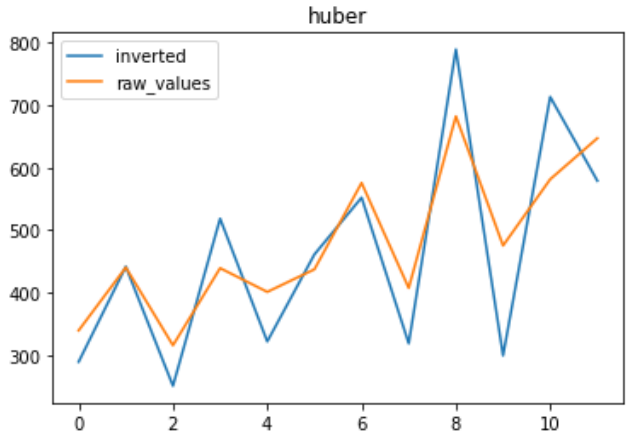


Figure 4.12: Huber regression model: Actuals vs Predicted

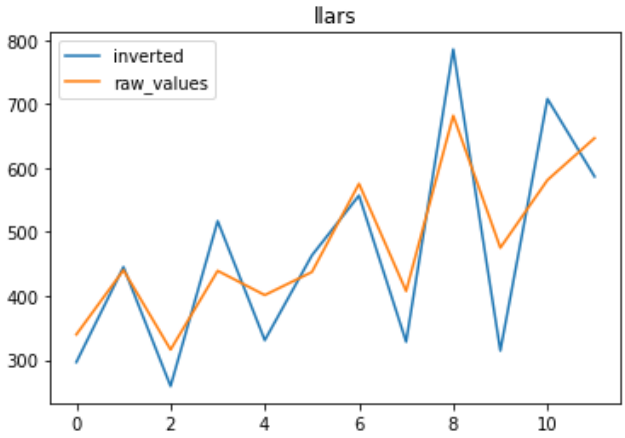


Figure 4.13: LassoLARS regresion model (llars): Actuals vs Predicted

Table 4.2: RMSE scores of the Linear ML models

Model	RMSE
Linear Regression	89.038
Lasso	89.021
Ridge	89.038
ElasticNet	89.024
Huber	87.776
LassoLars	81.591
PassiveAgressive	23.555

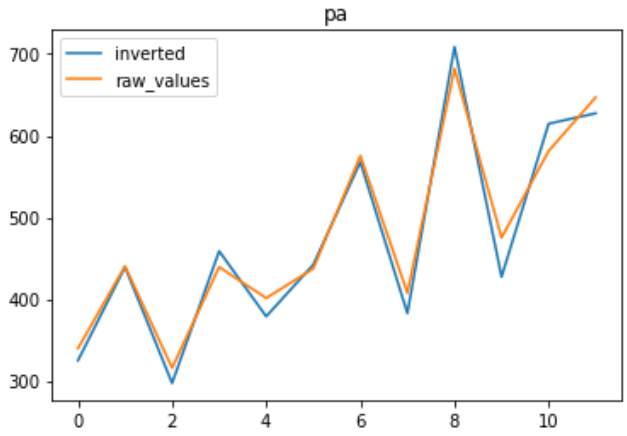


Figure 4.14: PassiveAggresive regression model (pa): Actuals vs Predicted

4.2.2 Non-Linear ML model results:

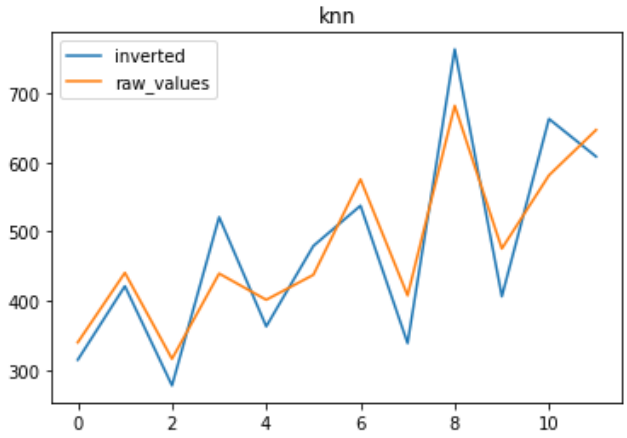


Figure 4.15: Kneighbors regression model(knn): Actuals vs Predicted

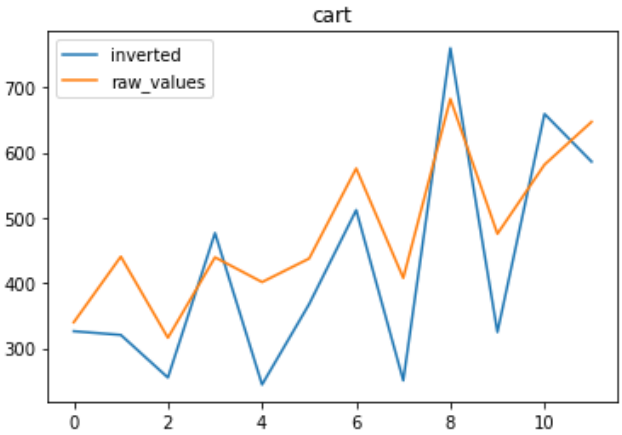


Figure 4.16: Decision Tree regression model(cart): Actuals vs Predicted

Unlike the linear ML models, the non-linear ML models have a varying fit for each model. This is more due to the complex nature of these non-linear models and the intorduction of non-linearity. Some models perform poorly, like the ExtraTree regressor, but some on the other hand outperform all the other models, like the SVR which gave a near perfect fit, and has a impressive RMSE = 6.416. Again we tabulate the RMSE scores to better understand and compare the models based on their performance.

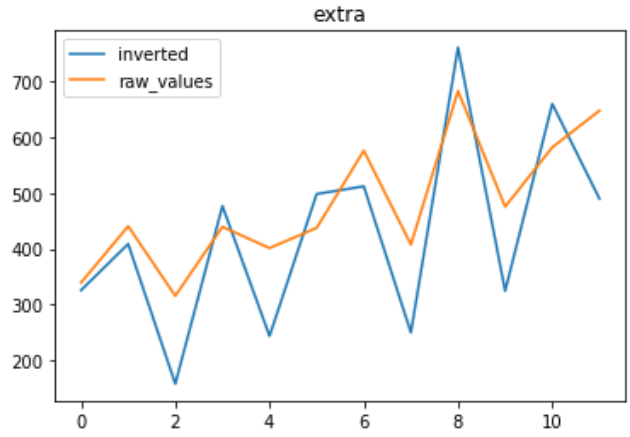


Figure 4.17: ExtraTree regression model(extra): Actuals vs Predicted

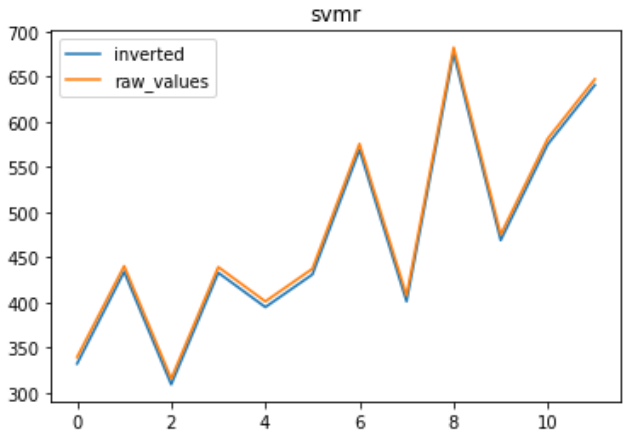


Figure 4.18: SVR regression model(svr): Actuals vs Predicted

Table 4.3: RMSE scores of the Non-Linear ML models

Model	RMSE
Kneighbors	56.397
DecisionTree	98.548
ExtraTree	109.479
SVR	6.416
AdaBoost	101.204
Bagging	76.525
RandomForest	71.502
ExtraTrees	93.872
GradientBoosting	97.140

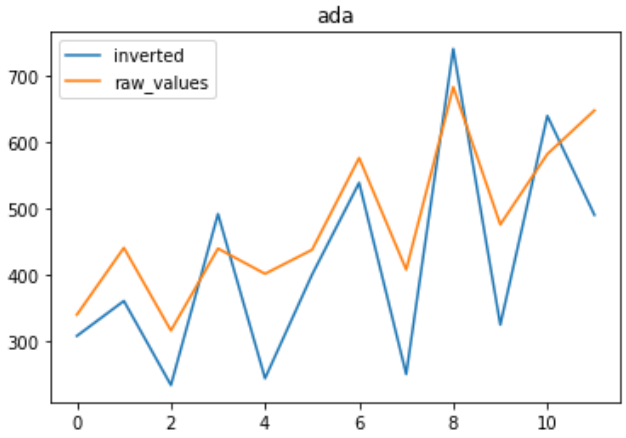


Figure 4.19: AdaBoost regression model(ada): Actuals vs Predicted

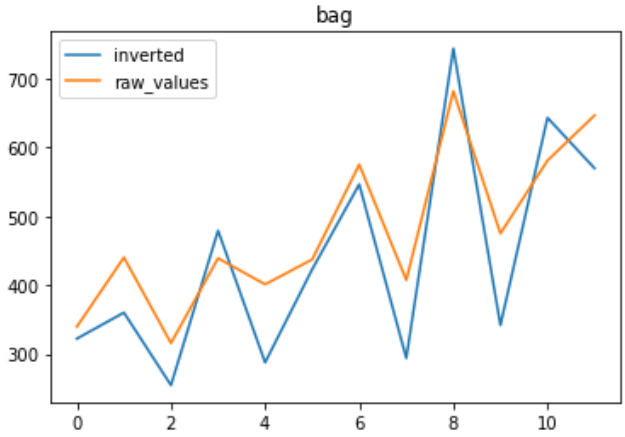


Figure 4.20: Bagging regression model(bag): Actuals vs Predicted

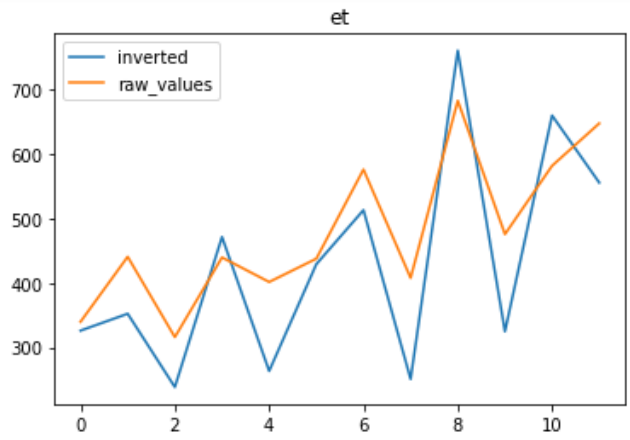


Figure 4.21: Extra Trees regression model(et): Actuals vs Predicted

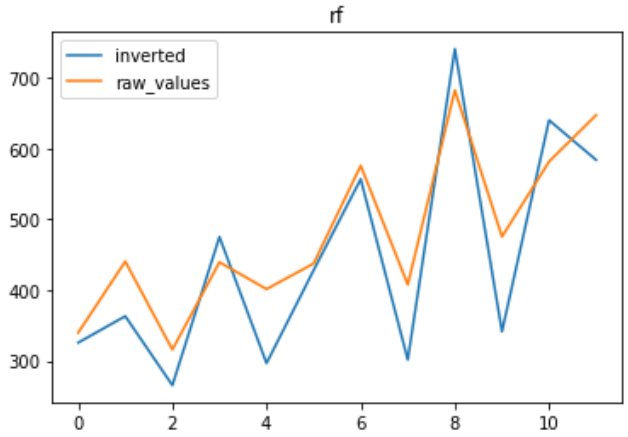


Figure 4.22: RandomForest regression model(rf): Actuals vs Predicted

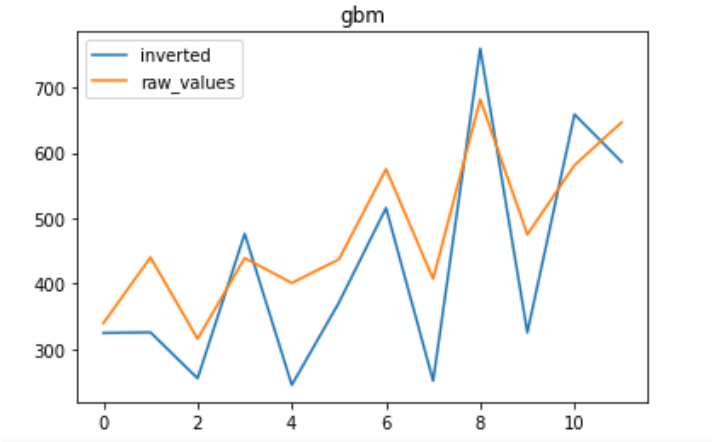


Figure 4.23: Gradient Boosting regression model(et): Actuals vs Predicted

4.3 Deep Learning model results

Here we will display all the results of our deep learning models, i.e. the experiment results.

4.3.1 MLP model results:

	50	100	500	1000	2000
count	30.000000	30.000000	30.000000	30.000000	30.000000
mean	129.660167	129.388944	111.444027	103.821703	107.500301
std	30.926344	28.499592	23.181317	22.138705	24.780781
min	94.598957	94.184903	89.506815	86.511801	86.452041
25%	105.198414	105.722736	90.679930	90.058655	86.457260
50%	129.705407	127.449491	93.508245	90.118331	90.074494
75%	141.420145	149.625816	136.157299	135.510850	135.741340
max	198.716220	198.704352	141.226816	139.994388	142.097747

Figure 4.24: MLP experiment 1 : varying epochs

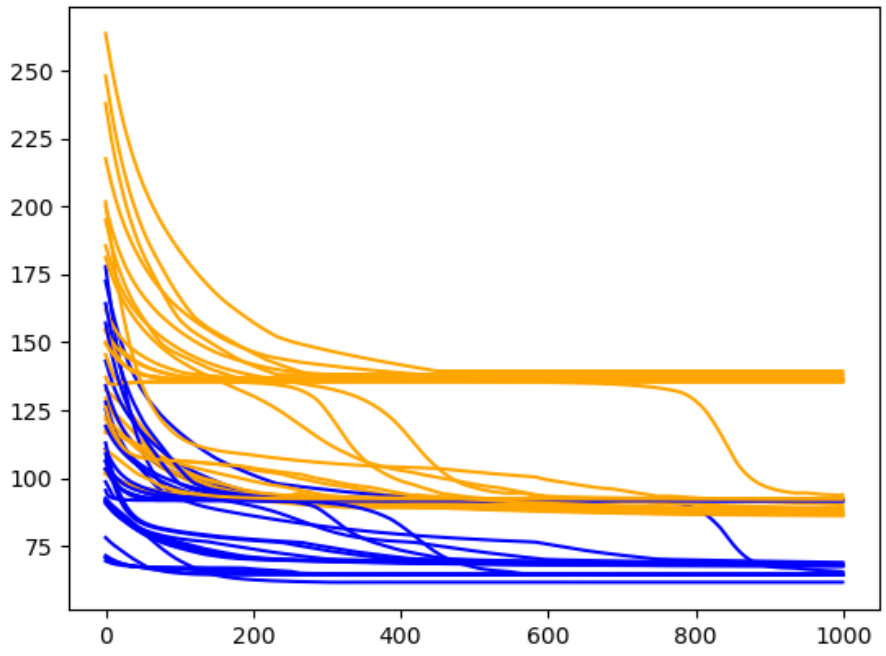


Figure 4.25: MLP experiment 1 diagnostics : varying epoch

The Diagnostic prints the final train and test RMSE for each run, and a line plot is made which shows the train RMSE (blue) and test RMSE (orange) after each training epoch. In

this case, the diagnostic plot shows little difference in train and test RMSE after about 400 training epochs. Both train and test performance level out on a near flat line. This rapid leveling out suggests the model is reaching capacity and may benefit from more information in terms of lag observations or additional neurons.

	1	2	3	4	5
count	30.000000	30.000000	30.000000	30.000000	30.000000
mean	105.107026	102.836520	92.675912	94.889952	96.577617
std	23.130824	20.102353	10.266732	9.751318	6.421356
min	86.565630	84.199871	83.388967	84.385293	87.208454
25%	88.035396	89.386670	87.643954	89.154866	89.961809
50%	90.084895	91.488484	90.670565	91.204303	96.717739
75%	136.145248	104.416518	93.117926	100.228730	101.969331
max	143.428154	140.923087	136.883946	135.891663	106.797563

Figure 4.26: MLP experiment 2 : varying hidden neurons

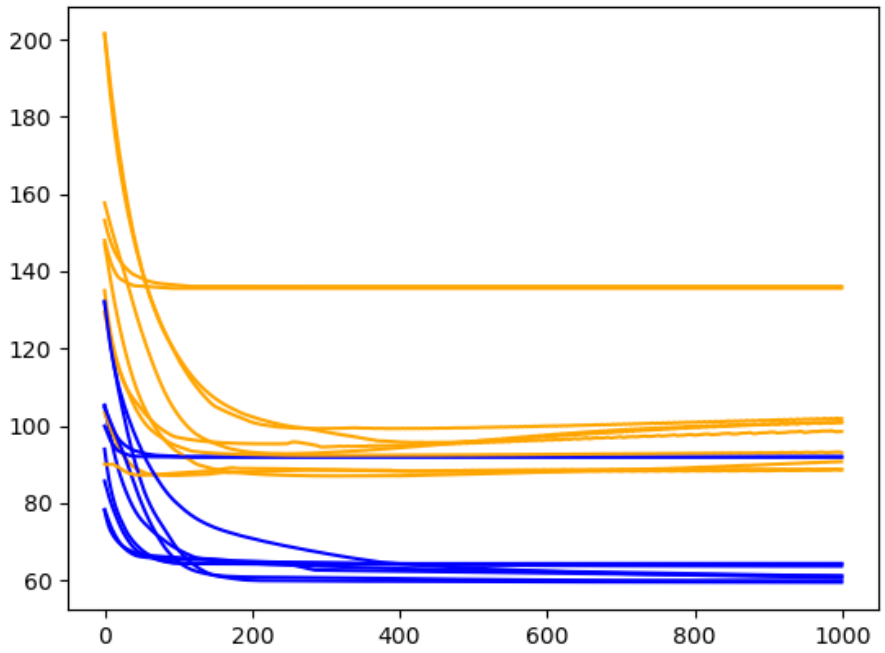


Figure 4.27: MLP experiment 2 diagnostics : varying hidden neurons

The diagnostics suggest a flattening out of model skill, perhaps around 400 epochs. The plot also suggests a possible situation of overfitting where there is a slight increase in test RMSE over the last 500 training epochs, but not a strong increase in training RMSE.

	1	3	5	7
count	30.000000	30.000000	30.000000	30.000000
mean	105.465038	109.447044	158.894730	147.024776
std	20.827644	15.312300	43.177520	22.717514
min	89.909627	77.426294	88.515319	95.801699
25%	92.187690	102.233491	125.008917	132.335683
50%	92.587411	109.506480	166.438582	145.078842
75%	135.386125	118.635143	189.457325	166.329000
max	139.941789	144.700754	232.962778	186.185471

Figure 4.28: MLP experiment 3 : varying hidden neurons with lag

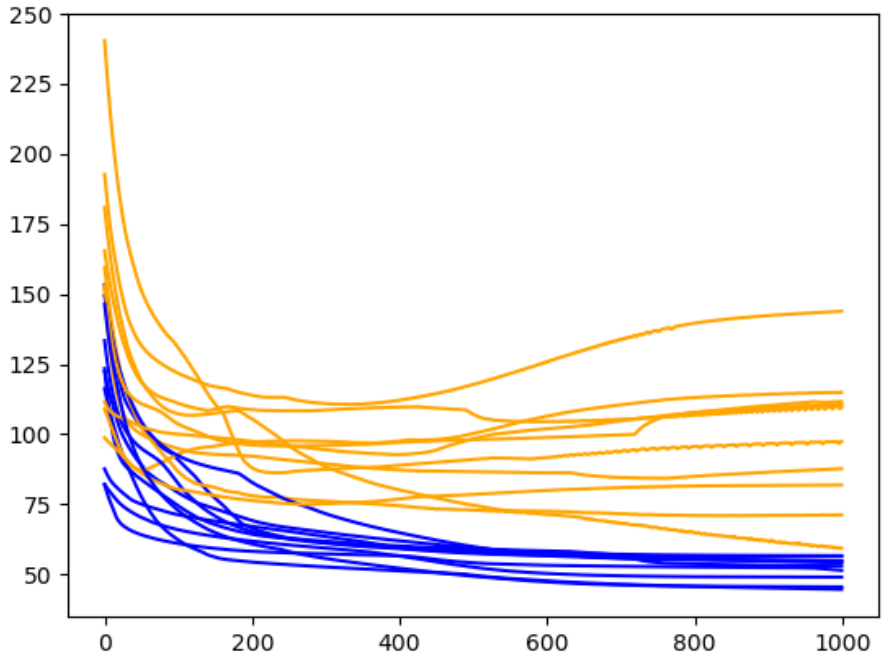


Figure 4.29: MLP experiment 3 diagnostics : varying hidden neurons with lag

The results suggest good learning during the first 500 epochs and perhaps overfitting in the remaining epochs with the test RMSE showing an increasing trend and the train RMSE showing a decreasing trend.

In conclusion, we see that after tuning the epochs, neurons and lags the best RMSE score is for 1000 epochs, 3 neurons and a lag of one, which is, RMSE = 92.675.

4.3.2 Lstm model results:

	1	2	3	4	5
count	30.000000	30.000000	30.000000	30.000000	30.000000
mean	98.344696	103.268147	102.726894	112.453766	122.843032
std	13.538599	14.720989	12.905631	16.296657	25.586013
min	81.764721	87.731385	77.545899	85.632492	85.955093
25%	88.524334	94.040807	95.152752	102.477366	104.192588
50%	93.543948	100.330678	103.622600	110.906970	117.022724
75%	102.944050	105.087384	110.235754	118.653850	133.343669
max	132.934054	152.588092	130.551521	162.889845	184.678185

Figure 4.30: Lstm experiment 1 : varying neurons

The mean gives an idea of the average expected performance of a configuration, whereas the standard deviation gives an idea of the variance. The min and max RMSE scores also give an idea of the range of possible best and worst case examples that might be expected. Looking at just the mean RMSE scores, the results suggest that an epoch configured to 1000 may be better. The results also suggest further investigations may be warranted of epoch values between 1000 and 2000.

	500	1000	2000	4000	6000
count	30.000000	30.000000	30.000000	30.000000	30.000000
mean	109.439203	104.566259	107.882390	116.339792	127.618305
std	14.874031	19.097098	22.083335	21.590424	24.866763
min	87.747708	81.621783	75.327883	77.399968	90.512409
25%	96.484568	87.686776	86.753694	102.127451	105.861881
50%	110.891939	98.942264	116.264027	121.898248	125.273050
75%	121.067498	119.248849	125.518589	130.107772	150.832313
max	138.879278	139.928055	146.840997	157.026562	166.111151

Figure 4.31: LSTM experiment 2 : varying epochs

From the mean performance alone, the results suggest a network configuration with 1 neuron as having the best performance over 1000 epochs with a batch size of 4 (was fixed earlier). This configuration also shows the tightest variance.

	1	2	4
count	30.000000	30.000000	30.000000
mean	98.697017	102.642594	100.320203
std	12.227885	9.144163	15.957767
min	85.172215	85.072441	83.636365
25%	92.023175	96.834628	87.671461
50%	95.981688	101.139527	91.628144
75%	102.009268	110.171802	114.660192
max	147.688818	120.038036	135.290829

Figure 4.32: Lstm experiment 3 : varying batchsize

From the mean performance alone, the results suggest lower RMSE with a batch size of 1. A box and whisker plot of the data was also created to help graphically compare the distributions. The plot shows the median performance as a green line where a batch size of 4 shows both the largest variability and also the lowest median RMSE.

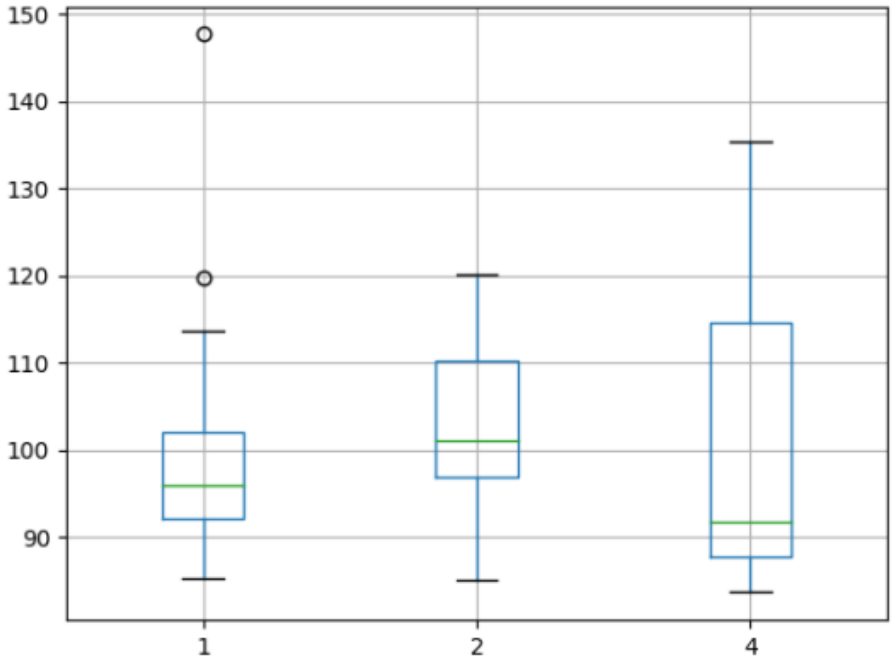


Figure 4.33: Box and Whisker plot summarizing Batch size results

Tuning a neural network is a tradeoff of average performance and variability of that performance, with an ideal result having a low mean error with low variability, meaning that it is generally good and reproducible.

We see here that after tuning the hyperparameters of the LSTM, the best RMSE comes out to be for one hidden neuron, batchsize of 4 and 1000 epochs, which is, RMSE = 98.697

4.4 Conclusions

After performing all the experiments and seeing all the results we can see that most of the Machine learning models perform as good or sometimes even better than the basic statistical methods. And Deep Learning methods perform even better than ML. After some more tuning and optimization we can get even better results for Deep learning methods, and the scope for them is ever expanding. All the models have their own merits and demerits,

and its upto the usage and the user that these models can be actually compared. But on a standard univariate dataset, we have seen which model performs in which way, but more research has to be done with more complex and multivarying datasets to actually understand the performance.