Capstone Project

Stock Index Predictor

SECTION I: PROJECT OVERVIEW

Domain Background:

Investment firms, hedge funds and even individuals have been using financial models to better understand stock market behavior and make profitable investments and trades. A wealth of information is available in the form of historical stock prices and company performance data, suitable for machine learning algorithms to process.

In this project I propose to study various machine learning models for predicting future prices of Stock Index Nifty50 of NSE, India. NSE is the leading stock exchange in India and the fourth largest in the world by equity trading volume in 2015, according to World Federation of Exchanges (WEF). It is the largest stock exchange in India in terms of total and average daily turnover for equity shares every year since 1995, based on annual reports of SEBI. Below is the official website of Nifty where details can be found. (https://www.nseindia.com/global/content/about_us/about_us.htm)

This prediction can help Investors, Traders and others to take informed decision about their trading strategy and may benefit financially. We will be using daily stock price data of Nifty50 from NSE India website. Using this raw data we will create features that will help us in predicting future price of Nifty50. (https://www.nseindia.com/products/content/equities/indices/historical_index_data.htm)

Reference Studies:

There are several studies on Stock market prediction using a variety of algorithms. These algorithms range from regression analysis, classification, neural network, trading system based on reinforcement learning and many many more. This is very active area of study by many researchers and numerous papers can be found out over internet. Below mentioned are some papers for reference:

https://www.sciencedirect.com/science/article/pii/S1877050918307828

http://www.ijaerd.com/papers/special_papers/RTDE014.pdf

Problem Statement/Task:

In this project, we will try to predict future/next day closing price of Nifty50 Index. We will model this as a Time Series forecasting problem and predict by using Regression Learning Algorithms. Nifyt50 price exhibits different trends over period of time and may exhibit particular behavior before reversing price in other direction. For accurate prediction, it is important to capture maximum information from prices. We will use various machine learning algorithms for our analysis which will capture different behaviors of prices and finally create a Stacker Model which will use predictions from these base algorithms for training and give out final prediction.

This prediction will help someone looking for short term trading ideas for a horizon of few days to weeks and may not be suitable for years of or long term price forecasting.

Datasets and Inputs:

Dataset that will be used for our project is freely available to download from NSE India official website. We will be using daily trading data from 04th Jan 2010 to 29th Dec 2016 for our analysis. Dataset will be split in chronological order so that we will avoid any peeking into the future. More on this in project design section.

We will do feature addition based on these raw data. We will be dropping the close price from our features dataset as this will be our predicted (target) variable. Raw data is available from NSE India official website in the form of csv file from the below link. This file consists of daily stock prices. (https://www.nseindia.com/products/content/equities/indices/historical_index_data.htm)

Capstone Project

Stock Index Predictor

The raw data csv file contains daily stock data as seen in Table 1:

Date	Open	High	Low	Close	Shares Traded	Turnover (Rs. Cr)
04-Jan-10	5200.9	5238.45	5167.1	5232.2	148652424	6531.61
05-Jan-10	5277.15	5288.35	5242.4	5277.9	240844424	7969.62

Table 1: Snapshot of downloaded csv file containing daily stock data

We will also use few fundamental indicators like P/E ratio, P/B value, Dividend yield and also volatility index (Vix) as features for training and testing of models. These can be downloaded from NSE India website. (https://www.nseindia.com/products/content/equities/indices/historical_pepb.htm)

Solution Statement:

For forecasting future prices we will use Stacking of learners. The solution consists of using several base learners and a stacker model. We will train different base models with historical data of Nifty50. For single model, it might be difficult to capture various price trends and behavior. Hence we use several base models with the intent of getting maximum information from training features. Using the historical data we will create additional features that will have more predictive power. Base models will be trained on the input dataset using GridSearchCV and TimeSeries split cross validation. Output prediction of these base models will then be fed to the stacker. Stacker will be trained on these first level predictions using GridSearchCV and cross validation dataset and will perform the task of optimization. Output predictions of Stacker will be used as our final predictions. This solution design is shown in Fig.1 below.

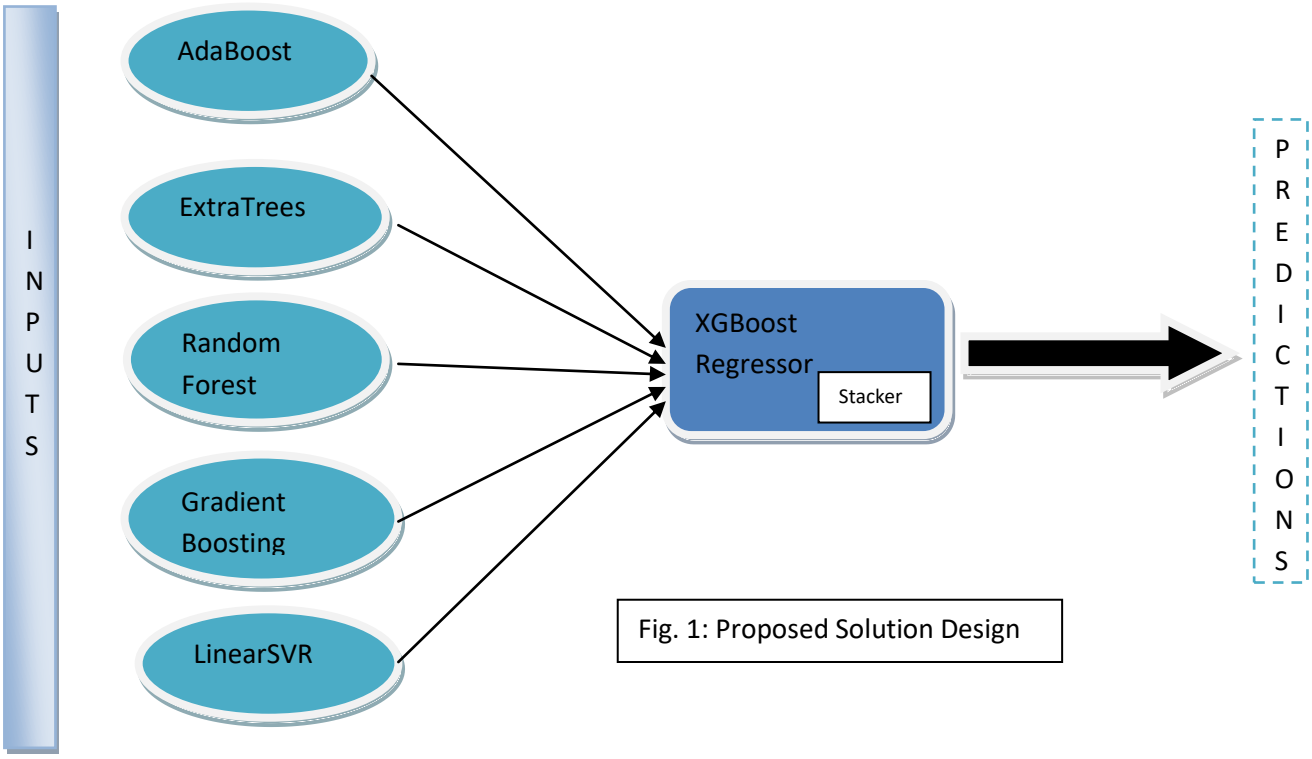


Fig. 1: Proposed Solution Design

Capstone Project

Stock Index Predictor

For base models we will use several boosting trees like AdaBoost Regressor, Random Forest Regressor, ExtraTrees Regressor, GradientBoosting Regressor, and SVR. These models incrementally build an ensemble of weak learners (Decision Stumps) into strong learner by reducing errors made by weak learners. These models improve upon errors made by weak learner and tend to reduce bias thereby increasing accuracy of the final predictions.

For Stacker we will use XGBoost Regressor. XGBoost is a very powerful algorithm and this will be trained on the predictions made by base models. This gives a chance to optimize the errors of base models and increase overall accuracy of the solution.

For comparison, we will also study MLP model which is a deep neural network and report its performance.

As final solution for our project, we will have a Stacking solution model, a benchmark model and another comparison model.

Benchmark Model

We will consider Linear Regression from sklearn as our benchmark model and report its RMSE Error and R2 score. We will be evaluating other model against RMSE error and R2 score of Linear Regression model.

Evaluation Metrics:

The key metrics that we will use for evaluation will be RMSE error. The two most widely used error metrics for Regression analysis are RMSE (Root Mean Squared Error) and MAE (Mean Absolute Error).

RMSE error is given by:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

MAE error is given by:

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

As can be seen from above equations, RMSE is the square of differences between actual and predicted values whereas MAE is the absolute difference between the actual and predicted values.

Our choice of RMSE over MAE is due to the way RMSE and MAE handles large errors. MAE takes the absolute value of difference in residuals which makes it robust against outlier. RMSE takes square of

difference in residuals and amplifies the error due to squared component thereby increasing the values of error given by outliers and punishing it more unlike MAE. Due to this, RMSE consider outliers present in the data and not robust to it.

In our Dataset of stock index prediction, there can be some extreme values or outliers which are very important values as it shows some kind of buying frenzy or panic selling. Prices move in certain trends up, down or sideways. Before changing the trend, prices sometimes tend to go to extreme values. These extreme price moves are also sometimes accompanied by extreme values in volume known as buying or selling climax before changing trend. These extreme values or outliers in our data is of significant importance and cannot be ignored.

In our Machine learning algorithms, to find the best possible solution we try to reduce the error given by the model. For reducing error, we use gradient descent method, which takes tiny steps in the negative direction of the gradient of error function. For taking these gradients, the error function has to be differentiable. RMSE error being square of residuals provides perfectly differentiable function whereas MAE doesn't.

For above mentioned reasons, we will choose RMSE as our error function in evaluating models. RMSE error can be calculated using mean square error function from sklearn as follows:

```
RMSE error = np.sqrt(mean_squared_error(y_true, y_pred)
```

Along with RMSE we will also report R2 score of all models.

R2 score: This is coefficient of determination. Its value is between 0 and 1. Score of 1 indicates that target variable predictable by the features used and a Score of 0 mean that the model in no better than the one which always gives average of the features. It can also have a negative value which means the model is worse than the model which always gives average. R2 score can be calculated using below mentioned function from sklearn.

```
Score = r2_score(y_true, y_pred)
```

Setup:

Software's and libraries that will be used for this project are:

Python, Numpy, Pandas, Scipy, Matplotlib, Seaborn, Scikit Learn, Keras, Tensorflow, XGBoost, Jupyter Notebook

Capstone Project

Stock Index Predictor

SECTION II: PROJECT ANALYSIS

Data Exploration

Data Collection, setup and cleaning:

In this step we have download all necessary data from NSE India website in form of csv files. We will download daily trading data file along with fundamental indicator data file from links given in section 1. These csv files will then be loaded as DataFrame in our workspace using Pandas and then joined to form raw data. This raw data will be checked for any inconsistencies, Nan or missing values etc.
For loading of dataset we have created a load_dataset function in utility folder which will take paths of csv files, read the files and join them. After merging the file this function will split the file into features and labels and also drop any missing or Nan values. As we are forecasting next day price, for training labels we have to shift it by one day in our dataset which will be taken care by our load_dataset function using **dataframe.shift(-1) function**.

This is how price or target variable looks like as shown in Fig. 2. The blue color graph is our training sample and red color graph is our testing sample.

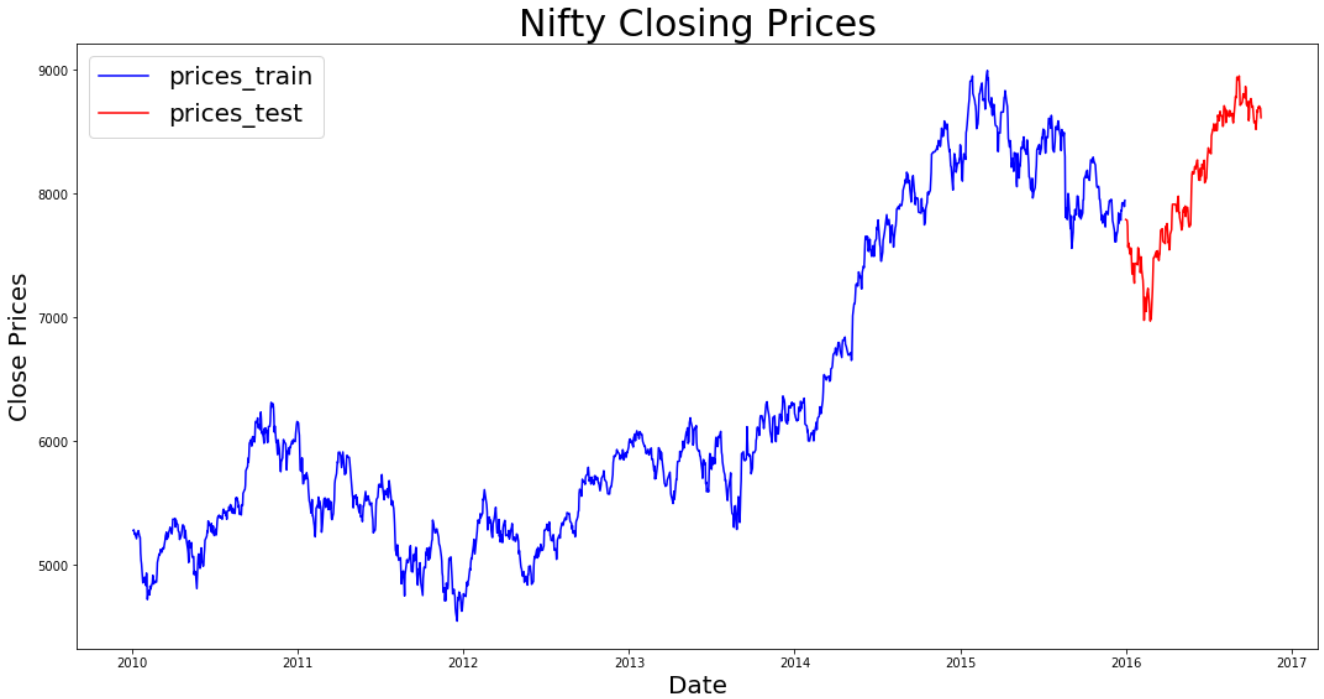


Fig. 2: Nifty50 Close Prices

Splitting of Data into Training and Testing:

We have created training and testing datasets from our data by splitting it in chronological order so as to avoid any peeking into future.
For our training data we have used samples from year 2010 to 2015.
For testing data we have used samples of year 2016.

Calculating Statistics:

We have calculated statistics of our dataset using **data.describe()** function.

Capstone Project

Stock Index Predictor

Close Prices Statistics

	Close Price
count	1491
mean	6277.68
std	1219.05
min	4544.2
25%	5337.35
50%	5870.1
75%	7568.15
max	8996.25

Table 2: Close price Statistics

Features Statistics

	Open	H-L	Volume	Vix
count	1491	1491	1491	1491
mean	6279.865	81.40859	158176400	19.843489
std	1221.478	45.28412	48678080	4.849977
min	4623.15	16.2	6555703	11.565
25%	5343.15	52.8	126740000	16.19625
50%	5872.75	73.2	150661400	18.925
75%	7561.175	100.025	181220600	22.58
max	9109.15	927.15	437039200	37.705

Table 3: Features Statistics using .describe()

As can be seen from above Table 2 & 3, different statistics can be easily calculated from our dataframe using describe function. These statistics are very important in understanding the nature of our dataset. Let's take max value of 'H-L' column; it says the maximum Intraday volatility in the given period is 927.15 points on Nifty50 Index. Let's check mean value of Volume column, it is 158176400. This value can be of significance to many traders, they will consider the trend going on if the volume if above this average. If prices are going up on low volume, the trend is not so strong and can reverse going ahead. This level indicates some sort of caution to many investors or traders and allows them to take informed decision about their investments.

<u>Checking Correlation between features:</u>

Let us check correlation plots of the features to see how related our features are to each other. We can use function from pandas library to do this as **pandas.plotting.scatter_matrix(data)**. Refer to Fig. 3; it can be observed that some features like open, high, low are perfectly correlated.

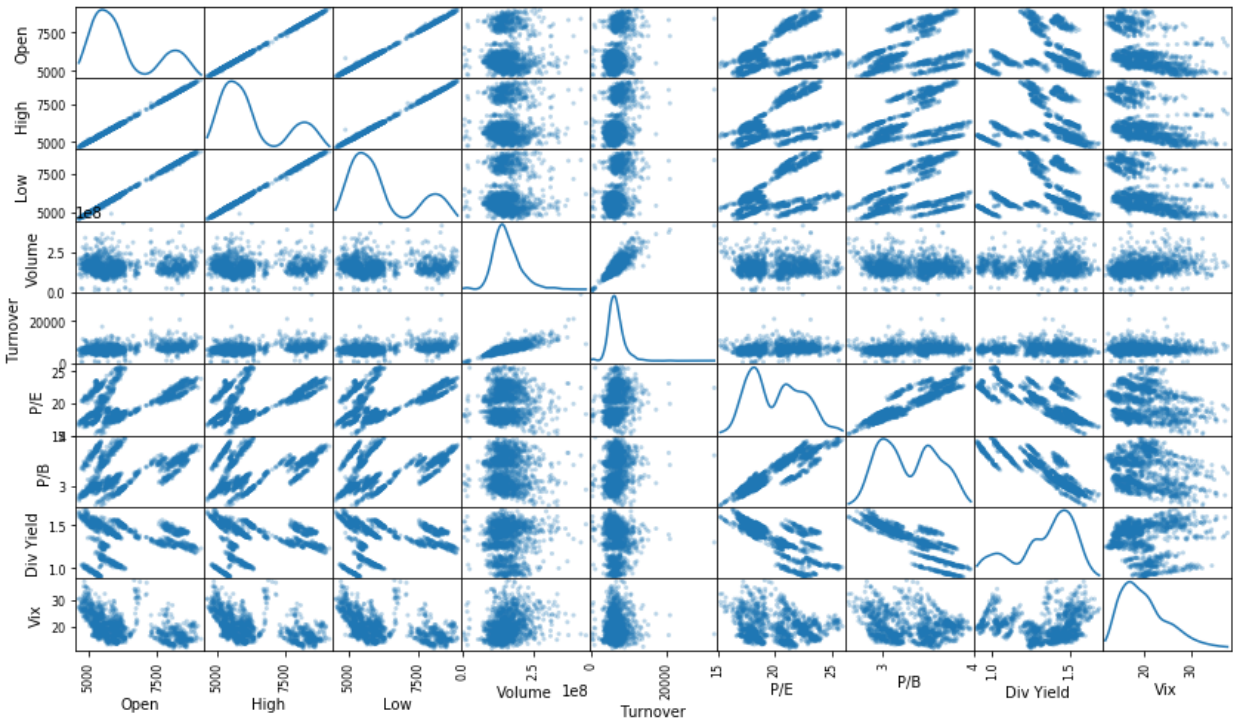


Fig. 3: Features Scatter Plot

These features create redundancy and do not carry unique information for our algorithms. The best we can do is either transform these features into new features or consider using only one of them. Also it can be seen from fig. 3 that the distribution of some of the features and not normal and needs to be normalized before it can be consumed by our algorithms.

We can use Seaborn plotting library to plot Pearson Correlation heatmap of features. Refer fig. 4; heatmap shows the percentage by which features are correlated. A value of 1 mean there is perfect correlation and a value of 0 or negative means no correlation. It can be seen from fig.4; that there are some features like open, high, low which are highly correlated and needs to be processed. We will discuss processing in feature engineering sections where we will use some features, drop some features and also generate new features using old ones.

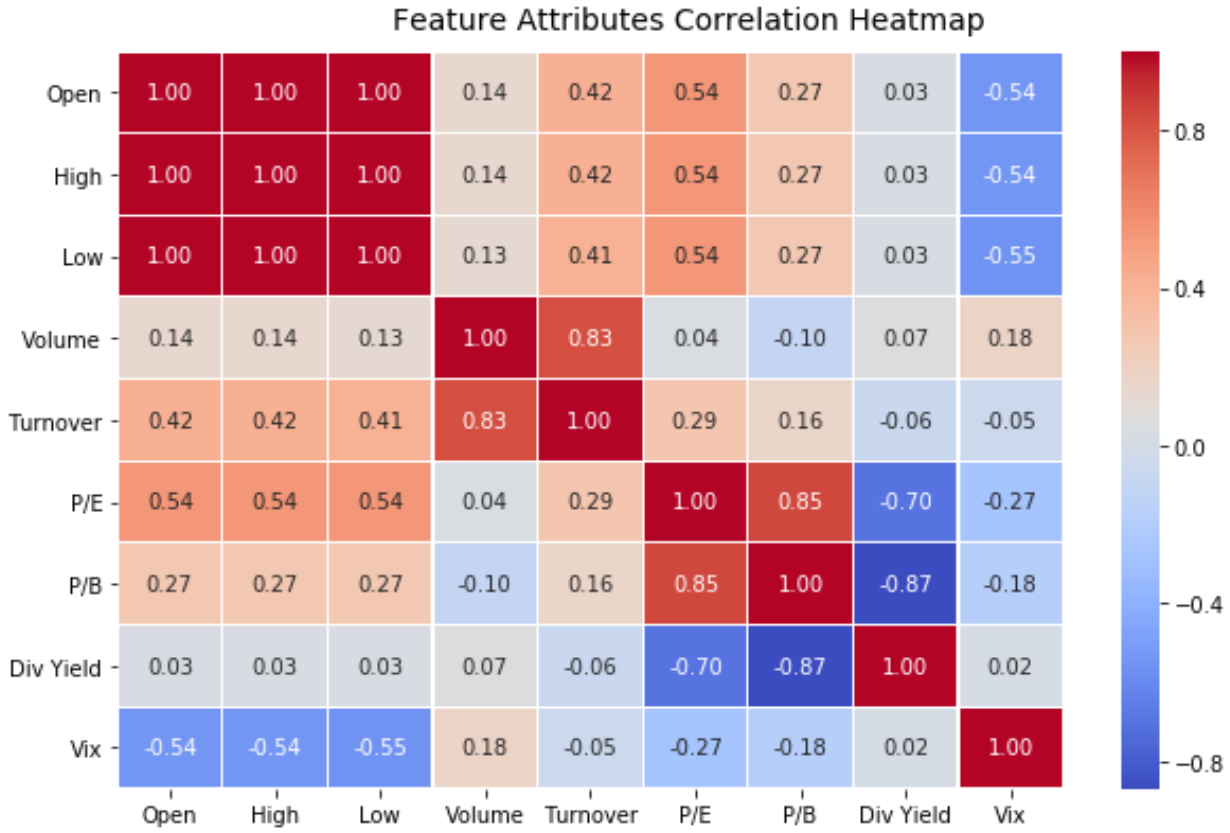


Fig. 4: Pearson's Correlation Heatmap

Algorithms and Techniques:

In this project, we are using stacking of learners for predicting future prices. The main idea is to build ensemble of learners using stacking which improves accuracy of prediction over single learner. Evaluate the models and compare it with benchmark model and comparator model. The algorithms chosen in our project are as follows:

Base model: We will choose 2 tree models based on boosting which are AdaBoost Regressor and Gradient Boosting Regressor from sklearn, 2 tree models based on averaging method which are Random Forest Regressor and ExtraTrees Regressor from sklearn and a linearSVR from sklearn.
The idea behind using these models is to introduce diversity. Some of these models are robust against outliers and some are not. Some models we train with greater depth whereas other models we keep

the depth low/middle. By diversification we are trying to get new information and capture different price behaviors.
Stacker: We will use XGBoost Regressor for stacking of our model.
Comparator Model: We will use MLP for Comparator model

Adaboost Regressor:
AdaBoost is an Ensemble of learners based on boosting methods, where base estimators are builds sequentially and one tries to reduce the bias of the combined estimator. It combines several weak models to produce a powerful/strong learner by weighted majority vote. It begins by fitting a Regressor on the original dataset and then fits additional copies of the Regressor on the same dataset but where the weights of instances are adjusted according to the error of the current predictions. As such, subsequent Regressor focuses more on difficult cases.
In sklearn, this can be imported as: from sklearn.ensemble import AdaBoostRegressor.
By default, weak learners are decision stumps. The main parameters to tune to obtain good results are
n_estimators: number of weak learners
max_depth: base estimators depth and
min_samples_split: minimum required number of samples to consider a split
Strength of AdaBoost lies in its simple to implement structure which requires few parameters to tune. It also has implicit feature selection and is resistant to overfitting.
Weakness of AdaBoost is, this algorithm requires a termination condition, sensitive to noise

Gradient Boosting Regressor:
GBR is also an Ensemble of learners based on boosting methods. GBR produces a prediction model in the form of an ensemble of weak learners typically decision trees. GBR builds an additive model in a forward stage-wise fashion, and it generalizes them by allowing optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function.
The main parameters to tune to obtain good results are:
n_estimators: number of weak learners
max_depth: base estimators depth and
min_samples_split: minimum required number of samples to consider a split
Strength of GBR is its Predictive power and robustness to outliers. GBR is weak in scaling due to its sequential nature of boosting.

Random Forest:
Random Forest is an Ensemble of learners based on averaging methods, where it builds several estimators independently and then averages their predictions. On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced. Each tree is built from a sample drawn with replacement from the training set. The best split will be the split which is picked among a random subset of features. Random forest uses averaging to improve the predictive accuracy and control over-fitting. The main parameters to tune to obtain good results are:
n_estimators: number of weak learners
max_depth: base estimators depth and
min_samples_split: minimum required number of samples to consider a split
Random Forest seldom overfits, not much parameter tuning, can also work in parallel. These models are difficult to interpret and sensitive to noise.

Capstone Project

Stock Index Predictor

ExtraTrees Regressor:
ExtraTrees Regressor is also an Ensemble of learners based on averaging methods, differing from Random Forest in the way splits are computed. As in random forests, a random subset of candidate features is used, but instead of looking for the most discriminative thresholds, thresholds are drawn at random for each candidate features and the best of these randomly-generated thresholds is picked as the splitting rule. This Regressor fits a number of randomized decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.
The main parameters to tune to obtain good results are:
n_estimators: number of weak learners
max_features: size of the random subsets of features to consider when splitting a node and
min_samples_split: minimum required number of samples to consider a split
ExtraTrees Regressor controls overfitting. These are high on memory. To reduce memory consumption, the complexity and size of the trees should be controlled by setting those parameter values.

LinearSVR:
This algorithm is similar to SVR with parameter kernel='linear, but implemented in terms of liblinear rather than libsvm. So it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples. The main parameters to tune to obtain good results are:
C: Penalty parameter of the error term.
Max_iter: The maximum number of iterations to be run.
epsilon = epsilon parameter in the epsilon-insensitive loss function.

XGBoost Regressor:
XGBoost Regressor is a sklearn wrapper interface for XGBoost. This is an advanced implementation of gradient boosting algorithm which builds ensemble of weak learners into a strong learner. By adding models sequentially, the errors of previous model are corrected by the next predictor. It fits new model to new residuals of the previous prediction and then minimizes the loss when adding the latest prediction. This helps in fast and accurate way of solving or predicting problems.
The main parameters to tune to obtain good results are:
Max_depth: Maximum tree depth for base learners
Learning_rate: Boosting learning rate
N_estimators: Number of boosted trees to fit
Gamma: Minimum loss reduction required to make a further partition on a leaf node of the tree
Reg_alpha: L1 regularization term on weights
Reg_lambda: L2 regularization term on weights
Some of the advantages in using XGBoost is parallel computing, regularization, high flexibility, handling missing values, built-in cross-validation, continue of existing model, tree pruning etc.

MLP:
A Multilayer perceptron (MLP) is a deep, artificial neural network. An MLP consists of three layers of nodes: an input layer which receives the signal, a hidden layer for computation and an output layer for prediction. Each node is a neuron except input nodes that uses a nonlinear activation function. MLP utilizes backpropagation for training. In keras library, MLP can be build using sequential model and layers with Dense or fully connected layer for output.

Capstone Project

Stock Index Predictor

SECTION III: METHODOLOGY

Data Preprocessing:

Feature Engineering:

In this project, the dataset which we have used consists of features which are opening price, high price, low price, close price, volume, turnover, pe ratio, pb ratio, dividend yield and vix.

After closely watching correlations between the features in Fig 4, we can conclude that there are some features which are highly correlated and needs to be dropped or transformed.

1. **Open, high, low:** Since these features are high correlated, it makes sense to use only one of them instead of using all. Out of all these features, Opening price of the stock looks more significant as this price will be used in determining the daily returns of stock. Also it is the first price at which stock trades in a day. This is specifically significant in many ways, as it indicates all of the news flow and sentiments from last day's close till the open. It also visually shows jump or inactivity between prices of two day which keeps special importance to chartist or technical analysts for gap analysis and certain chart patterns. So we will use opening prices in our features dataset and for other prices we will calculate new features such as **Open-Low, High-low and High-Open**. As can be seen from below heatmap that these features are not highly correlated and can be used for our machine learning dataset.
2. **Volume, turnover:** As volume represents the number of shares traded in a day and turnover represent its equivalent amount in terms of money required to buy that number of shares, we can use only one feature out of this. Here we will be using Volume as a feature in our dataset.

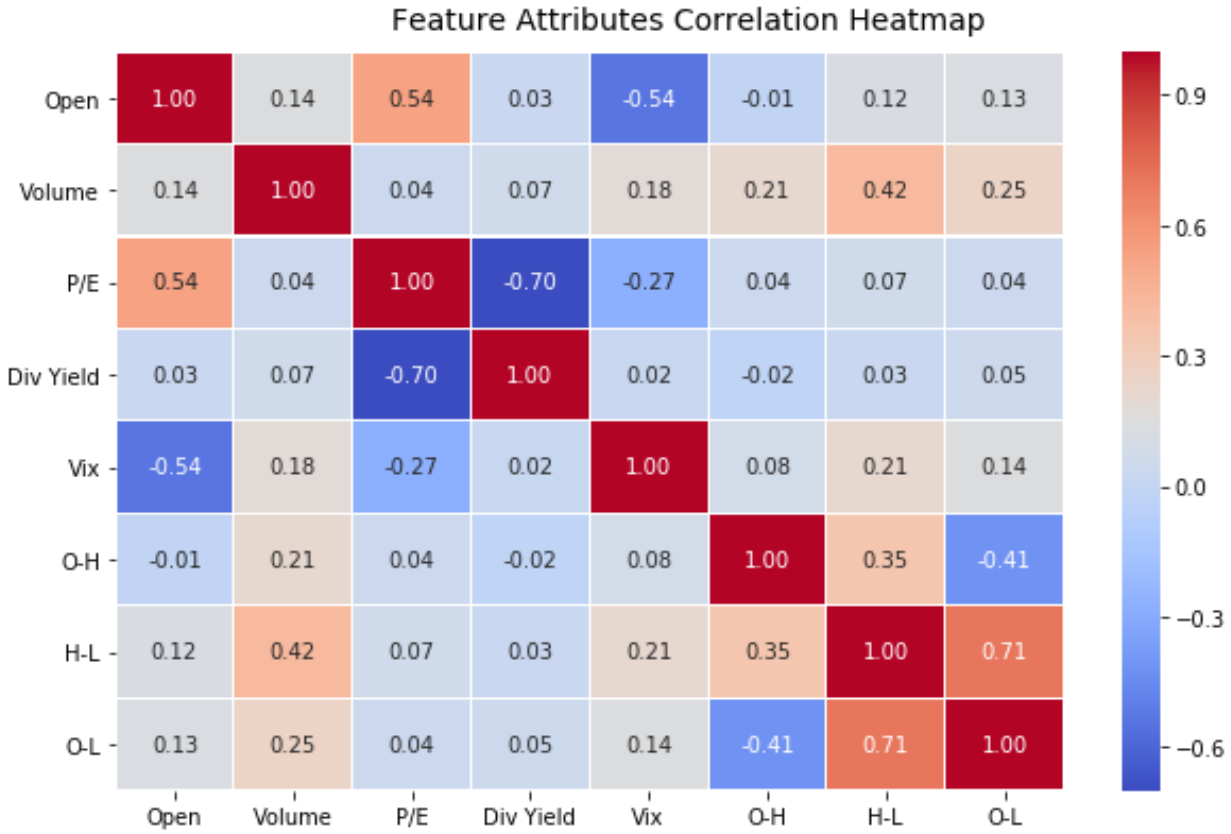


Fig 5. Pearson's correlation heatmap after features engineering

3. P/E, P/B: As can be seen from our heatmap (refer Fig. 4) that these features our highly correlated. P/E represents the price to earnings ratio and P/B represent price to book value of all the stock composition of Index. We will be using P/E ratio for our dataset.

After feature addition, our feature correlation heatmap as shown in Fig.5 looks good.

Features Transformation:

As can be seen from below plot in Fig. 6, our features are skewed and don't have normal distribution. We will be applying log transformation to these features. Here we will use a very simple approach for reducing skewness which applies natural logarithm to every sample in our dataset by using numpy function like numpy.log(x+1) where x is the data point. Here 1 is added to the data point to deal with 0 issue.

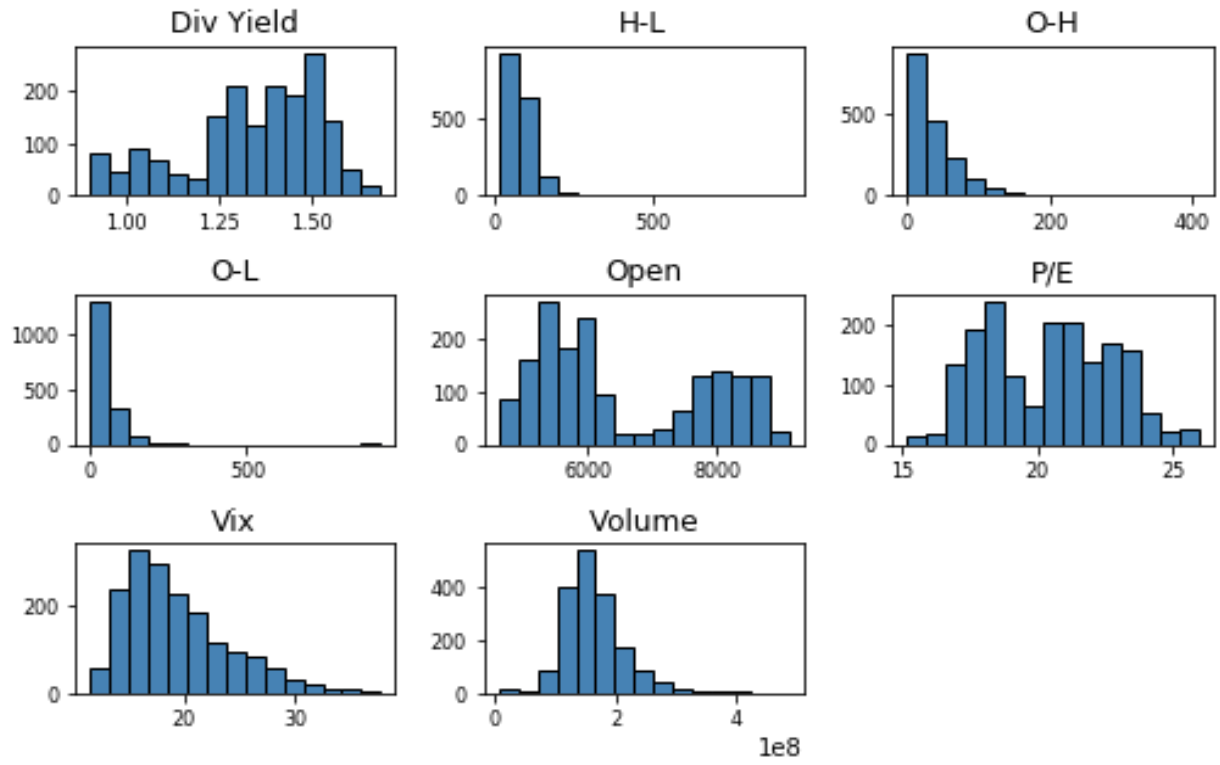


Fig. 6 Features Histogram

Features Normalization:

Our features have different min, max values and different ranges. So to plot and use our data effectively we need to normalize. Generally we use data normalization methods like MinMaxScaler or Standard Scaler for these purposes, but here we are dealing with financial time series dataset. These dataset are not stationary that is their statistical values/parameters like minimum, maximum, mean, standard deviation keeps changing over time.

To deal with this issue, I have tried to normalize the data by dividing the whole dataset with its first sample as mentioned below:

df = df/df1

In this section, we have written function to normalize and denormalize the data using above technique.

Capstone Project

Stock Index Predictor

Implementation:

Splitting the Data and converting into Numpy Arrays:

Before we feed our data to machine learning models, we need to split the data into training and testing sets and convert into numpy arrays so that our models can be trained on training dataset and tested on testing dataset. This will avoid any overfitting issues or look ahead biases.
For training set we have used data from 2010 till 2015
For testing set we will be using data from year 2016.
Finally we convert our data into numpy arrays so that it can be consumed by machine learning models.

Performance Metric:

For evaluating our model performance we have used RMSE error which gives the average combined error given by any model over the entire dataset. For this we will use **mean_squared_error** function from sklearn. Also we will be reporting R2 score of all the models. Final Model selection will be done based on lowest RMSE error. For this, we will use **r2_score** function from sklearn

Benchmark Model:

We have used Linear Regression from sklearn as our benchmark model for evaluating/scoring different models over each other. This can be imported from sklearn as:
from sklearn.linear_models import LinearRegression.
Further we use model's fit method to train on our training dataset as: **model.fit(X,y).**
Once the model is trained, it can be used to predict on testing dataset using model's predict function as: **model.predict(X_test)**. Using these predictions and actual prices we calculate RMSE error and R2 score using sklearn functions as:
RMSE error = np.sqrt(mean_squared_error(y_true, y_pred)
Score = r2_score(y_true, y_pred)
Shown below in Fig. 7 are the actual prices and predicted prices given by Linear Regression Model.

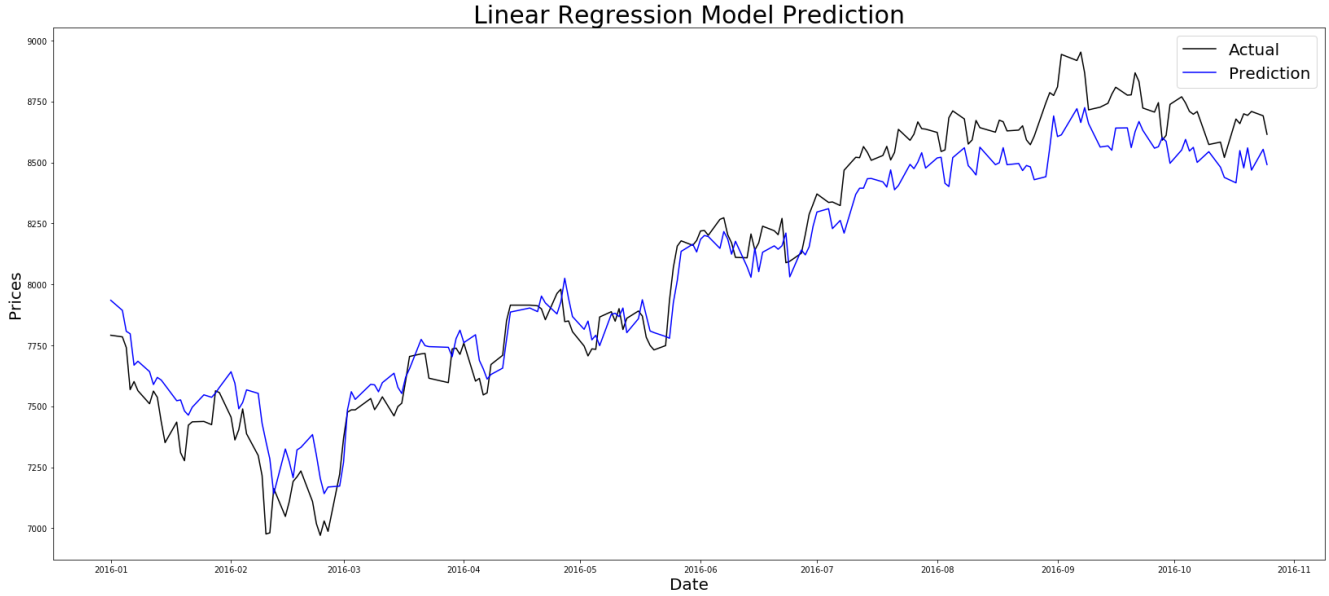


Fig 7: Linear Regression Model Predictions

Capstone Project

Stock Index Predictor

LinearRegression Model has RMSE Testing error of 138.70
LinearRegression Model has R2 score of 0.94

Model Selection and Implementation:

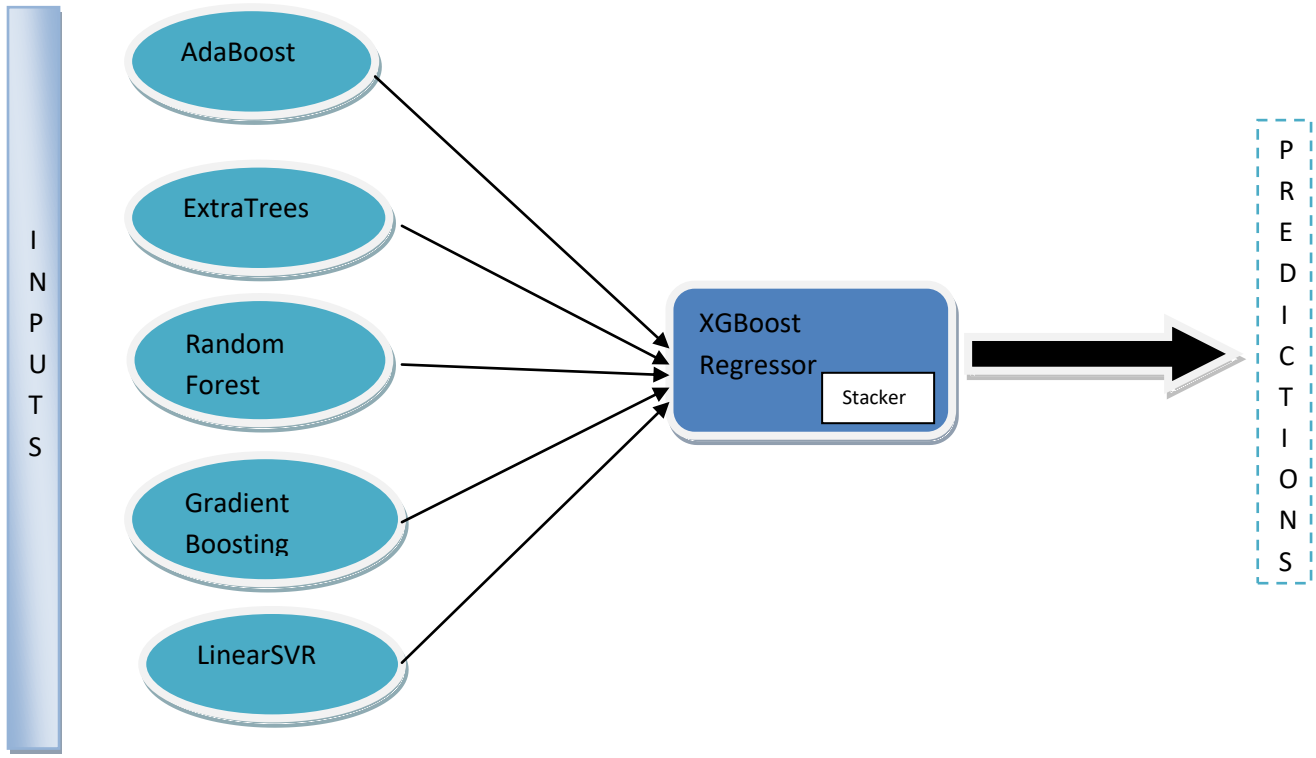
For our regression analysis of financial time series data, we will be studying 2 different models.

1. Stacking of learner
2. MLP as comparator model

1. Stacking/Ensemble of learners.

For our Stacking solution, algorithms that we have selected are as follows:
Base model: RandomForestRegressor, AdaBoostRegressor, ExtraTreeRegressor, GradientBoosting Regressor and LinearSVR.
Stacker: XGBoost Regressor

Base models are trained on input dataset. Parameters for these base models are tuned using GridSearchCV. For cross-validation dataset, we use TimeSeriesSplit and pass it to gridsearchcv. RMSE error and R2 score of these models are calculated and reported. Predictions from these base models are then saved and used as training set for stacker, which then optimizes the error given by these base models and improves upon RMSE error. Output of stacker will be our final prediction.



Capstone Project

Stock Index Predictor

Unoptimized Parameters of base models and their reported respective RMSE error and R2 score are as follow:

AdaBoost Regressor:
Params: {}
AdaBoostRegressor has Unoptimized Testing RMSE error of 122.15
AdaBoostRegressor has Unoptimized R2 score of 0.95

Gradient Boosting Regressor:
Params: {'n_estimators':[30]}
GradientBoostingRegressor has Unoptimized RMSE Testing error of 132.97
GradientBoostingRegressor has Unoptimized R2 score of 0.94

ExtraTrees Regressor:
Params: {'n_estimators':[10],'max_depth':[5]}
ExtraTreesRegressor has Unoptimized RMSE Testing error of 150.08
ExtraTreesRegressor has Unoptimized R2 score of 0.92

Random Forest Regressor:
Params: {'n_estimators':[10],'max_depth':[5]}
RandomForestRegressor has Unoptimized Testing RMSE error of 102.10
RandomForestRegressor has Unoptimized R2 score of 0.97

LinearSVR:
Params: {'C':[1],'tol':[0.01],'max_iter':[2000],'loss':['epsilon_insensitive']}
LinearSVR has Unoptimized Testing RMSE error of 289.23
LinearSVR has Unoptimized R2 score of 0.72

These parameters are then tuned using GridSearchCV and passed on to the base_model_predict function for training and predicting as below:

```
def base_model_predict(model, params):
    scorer = make_scorer(mean_squared_error)
    tscv = TimeSeriesSplit(n_splits=5)
    tscv = tscv.split(range(X_train.shape[0]))
    grid_obj = GridSearchCV(model,params,scorer,cv=tscv)
    grid_fit = grid_obj.fit(X_train,y_train)
    best_reg = grid_fit.best_estimator_
    ypred = best_reg.predict(X_test)
    ytrain = best_reg.predict(X_train)
    return best_reg, ytrain, ypred
```

base_model_predict function uses GridSearchCV, for scoring it uses mean_squared_error function from sklearn and for cross validation set it uses TimeSeriesSplit which splits the data into 5 folds. Best parameters are found using best_reg.get_params() function.

Capstone Project

Stock Index Predictor

Optimized Parameters and reported rmse error and r2 score of these base models are as follows:

Adaboost: {'base_estimator':'DecisionTreeRegressor',
'n_estimators':[50],
'learning_rate':[0.01],
'base_estimator__max_depth':[10],
'base_estimator__min_samples_split':[7]}
AdaBoostRegressor has Testing RMSE error of 97.82
AdaBoostRegressor has R2 score of 0.97

Gradient Boosting: {'n_estimators':[100],
'learning_rate':[0.3],
'max_depth':[3],
'alpha':[0.9],
'min_samples_split':[5]}
GradientBoostingRegressor has Testing RMSE error of 98.24
GradientBoostingRegressor has R2 score of 0.97

ExtraTrees: {'n_estimators':[10],
'max_depth':[10],
'min_samples_split':[2]}
ExtraTreesRegressor has Testing RMSE error of 99.32
ExtraTreesRegressor has R2 score of 0.97

RandomForest: {'n_estimators':[10],
'max_depth':[10],
'min_samples_split':[2]}
RandomForestRegressor has Testing RMSE error of 97.43
RandomForestRegressor has R2 score of 0.97

LinearSVR: {'C':[2],
'tol':[0.07],
'max_iter':[1000],
'loss':['epsilon_insensitive']}
LinearSVR has Testing RMSE error of 149.56
LinearSVR has R2 score of 0.92

Output of these base models are concatenated using numpy functions as: **np.concatenate(df1,df2,df3,df4,df5)**. These first level predictions are then fed to the stacker XGB Regressor as features for training. Parameters of XGB are tuned using GridSearchCV. Stacker also uses cross validation using TimeSeriesSplit function:

XGB parameters: {'n_estimators':[100,200,500],
'learning_rate':[0.07,0.01,0.1],
'gamma':[0.0009,0.1,0.01,0.001],
'max_depth':[3,5,7,10]}

```
def xgb_model_predict(model,params):
    scorer = make_scorer(mean_squared_error)
    tscv = TimeSeriesSplit(n_splits=5)
    tscv = tscv.split(range(X_train.shape[0]))
    grid_obj = GridSearchCV(model,params,scorer,cv=tscv)
    grid_fit = grid_obj.fit(x_train,y_train)
    best_reg = grid_fit.best_estimator_
    ypred = best_reg.predict(x_test)
    ytrain = best_reg.predict(x_train)
    return best_reg, ytrain, ypred
```

Model and parameters are passed to **xgb_model_predict** function, which uses GridSearchCV, for scoring it uses mean_squared_error function from sklearn and for cross validation set it uses TimeSeriesSplit which splits the data into 5 folds. In each fold data into train and test set where test set will always be ahead in time of train set so as to avoid any look-ahead bias or peeking in future. Training using kfold cv makes this model robust. Best parameters are found using best_reg.get_params() function.

Optimized parameters and reported RMSE error and R2 score for XGB Regressor are:

XGB parameters: {'n_estimators':[100],
'learning_rate':[0.07],
'gamma':[0.0009],
'max_depth':[3,5]}

XGBRegressor has Testing error of 95.97

XGBRegressor has R2 score of 0.97

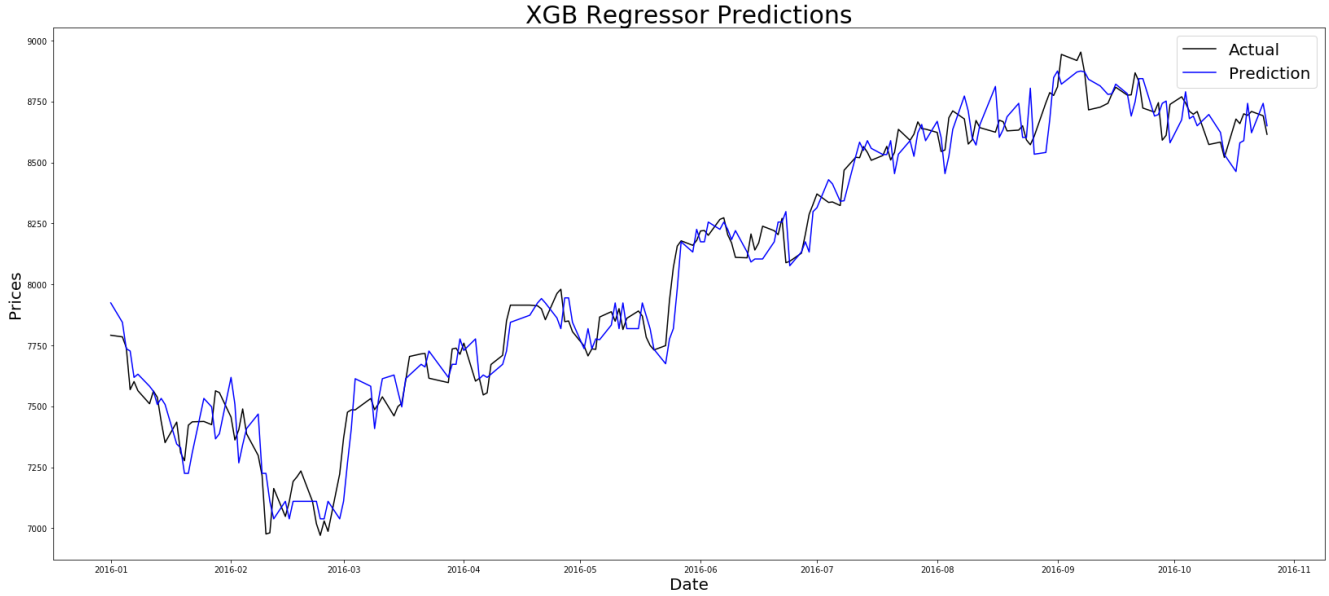


Fig. 8: XGB Regressor Predictions

For robustness, XGB Regressor model is tested on different samples of test set by splitting into 3 folds.

For test fold 1:
xtest1 = x_test[0:150,]
On test fold 1 XGBRegressor has Testing error of 96.09
On test fold 1 XGBRegressor has R2 score of 0.96

For test fold 2:
xtest2 = x_test[0:180,]
On test fold 2 XGBRegressor has Testing error of 96.07
On test fold 2 XGBRegressor has R2 score of 0.97

For test fold 3:
xtest3 = x_test[0:200,]
On test fold 3 XGBRegressor has Testing error of 95.97
On test fold 3 XGBRegressor has R2 score of 0.97
Above mentioned results shows that RMSE error and R2 score of XGB Regressor on different test set are very close, thus showing that our solution model is robust

2.MLP:

For comparing our XGB model we will build another model, Multi-Layer Perceptron, a deep neural net. For this we will use Sequential model and layers from keras library. This model consist of 1 input layer for receiving input data, 1 hidden layer with 20 neurons and relu activation and a fully connected Dense layer with 1 output and relu activation. For compiling we have used rmsprop as our optimizer along with mean squared error as our loss function. We have trained our MLP with 50 epochs of batch size of 71 samples. This can be achieved using keras library as show below:

```
From keras.model import Sequential
From keras.layers.core import Dense, Activation
Model_mlp = Sequential()
Model_mlp.add(Dense(20, activation='relu', input_shape=(8,)))
Model_mlp.add(Dense(1, activation='relu'))
Model_mlp.compile(loss='mean_squared_error', optimizer='rmsprop', metrics=['mse'])
Model_mlp.fit(X_train, y_train, epochs=50, batch_size=71, verbose=0)
Mlp_pred = model_mlp.predict(X_test)
```

MLP Model has a Testing RMSE error of 133.38
MLP Model has a R2 score of 0.94

MLP reports a RMSE score of 133.38. These performance values are better than our Benchmark Model, but not close to our XGB Regressor. It needs more hidden layers and epochs for training to get better results which would require aggressive hyper-tuning.

Please refer to Fig. 9, for MLP predictions on our test dataset.

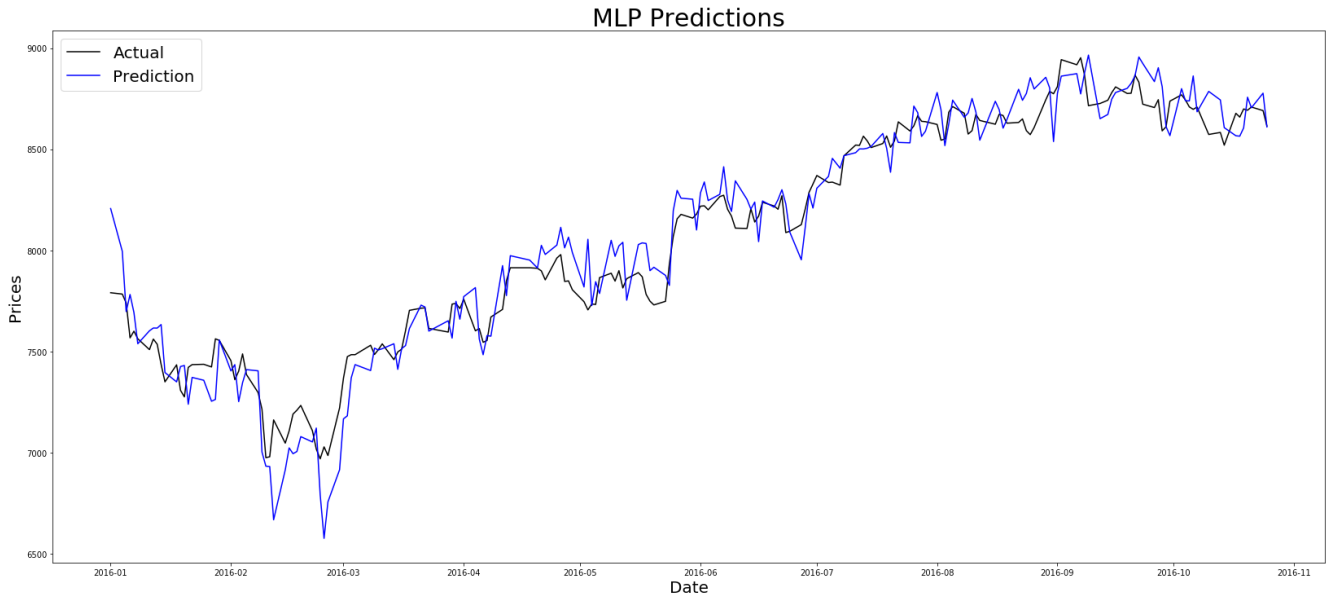


Fig. 9: MLP Predictions

REFINEMENT:

In all of our base models, the hyper parameters we have chosen have been tuned using GridSearchCV from sklearn. For cross validation set of GridSearchCV, we have passed TimeSeriesSplit as cvset with number of folds as 5. TimeSeriesSplit splits the data into n number of folds. In each fold data is split into k folds for training set and k+1th for testing set. This test set is always ahead in time of train set. After doing exhaustive search using GridSearch the final parameters are used for training our base models.

Using above methods, we were able to improve RMSE error and R2 score of models atleast by 20% as mentioned in below Table 4:

Models	Unoptimized		Optimized	
	RMSE Error	R2 Score	RMSE Error	R2 Score
AdaBoost Regressor	122.15	0.95	97.82	0.97
Gradient Boosting Regressor	132.97	0.94	98.24	0.97
ExtraTrees Regressor	150.08	0.92	99.32	0.97
Random Forest Regressor	102.10	0.97	97.43	0.97
LinearSVR	289.23	0.72	149.56	0.92

Table 4: Performance of base models

DIFFICULTIES:

The most difficult part of the project is to further lower the RMSE error. Ideally we want the error to be very low or close to zero for accurate predictions. Our model although doesn't have very low error but it captures the trend very well and generalizes well on the dataset.

Capstone Project

Stock Index Predictor

SECTION IV: RESULTS

MODEL EVALUATION AND VALIDATION:

As shown in Table 5 below are RMSE errors and R2 scores of each model used in the project.

Model	RMSE Testing Error	R2 Score
Linear Regression	138.70	0.94
AdaBoost Regressor	97.82	0.97
Gradient Boosting Regressor	98.24	0.97
ExtraTrees Regressor	99.32	0.97
Random Forest Regressor	97.43	0.97
LinearSVR	149.56	0.92
XGBoost Regressor	95.97	0.97
MLP	133.38	0.94

Table 5: Performance of all the Models used in the project

As can be seen from above evaluation, XGB Regressor has lowest RMSE error and highest R2 score among all other models. Comparing its performance over other parameter, we were successfully able to increase model performance using Stacking by few percentage points. It can be seen from Fig 8 of XGB Regressor predictions, that our solution generalizes well and we were able to capture the trend in a nice way.

XGB Regressor Test result on kfold cv:

On test fold 1 XGBRegressor has Testing error of 96.09
On test fold 1 XGBRegressor has R2 score of 0.96
On test fold 2 XGBRegressor has Testing error of 96.07
On test fold 2 XGBRegressor has R2 score of 0.97
On test fold 3 XGBRegressor has Testing error of 95.97
On test fold 3 XGBRegressor has R2 score of 0.97

It can be observed from XGB Regressor test results on kfold that our solution is robust. We have trained our XGB Regressor using 5 fold cross validation dataset using TimeSeries split. Also we have divided our test set into 3 folds in chronological order. Output of these folds is very close, suggesting that our solution is very robust.

Capstone Project

Stock Index Predictor

JUSTIFICATION:

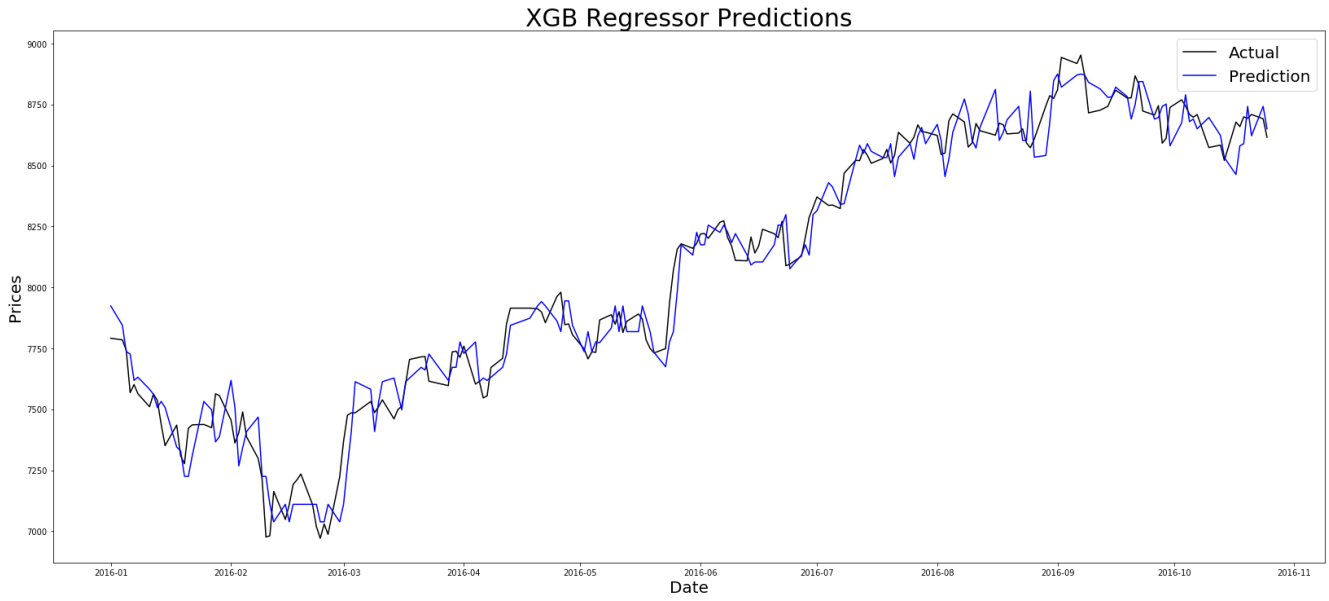
	Models	RMSE Error	R2 Score (%)
Benchmark Model	Linear Regression	138.70	94
Solution Model	Stacking Model	95.97	97
Comparing Model	MLP	133.38	0.94

Table 6: Model Performance Comparison

As per our evaluation metric as shown above in Table 6, XGBoost Regressor performs better than both benchmark model and comparator model by significant margin on both RMSE error and R2 score. This is because XGB Regressor is used as a stacker which uses predictions from several base models thereby improving on the errors made by these models. These base models might have picked up different characteristics of the dataset while training which helps the final stacker in generalizing well over unseen data and improve overall accuracy of the system.

SECTION V: CONCLUSION

FREE-FORM VISUALIZATION:



Shown above is the prediction vs actual price plot of XGB Regressor for our project. It shows that our prediction very nicely traces the trend of future prices. It clearly shows that although actual prices cannot be forecasted accurately but trends can be tracked nicely. This is very important, for eg: if we

look at the bottom made between feb and march of 2016, actual prices made a double bottom before reversing the price trend (the 'W' formation). When prices break out of this formation, it gives a signal that prices have reversed and a trade can be taken accordingly in the breakout direction. This can be tracked in our predictions too. Our prediction also formed a double bottom at the same time which reconfirms the belief of 'W' formation. Infact the price breakout happened earlier on our prediction giving an opportunity of quick entry or wait for actual price to do the same breakout and then take trading position with full conviction.

Accordingly, any technical analysis done on price chart like trendlines, support and resistance breakouts can be carried out on our prediction giving us double conviction on analysis.

REFLECTION:

In this project of Stock Index Predictor, we are predicting next day closing price of Stock Index Nifty50 of NSE. We took the daily data of stock, did feature generation and preprocessing to make the data samples suitable to feed into machine learning algorithm. We defined a performance metric which uses RMSE error to evaluate the performance of our models.

In this project we have used Stacking of learners to predict future price movement. Our stacking model used 5 different base learners (AdaBoost, Gradient Boosting, Random Forest, ExtraTrees and LinearSVR) to combine into a strong learner using XGBoost. XGBoost successfully builds upon base model and optimize/reduce the RMSE error given by these models. We have also build a benchmark model (Linear Regression) and a comparator model (MLP) for comparing performance of XGBoost. XGBoost successfully beats the performance of benchmark and comparator model and comes out as clear winner.

For benchmark model, we have used Linear Regression from sklearn library and for comparator model we have used MLP from keras library.

Finally we have evaluated all the models RMSE error and R2 score and justified our results.

Learning's from project:

It was interesting to learn from the project how accuracy can be improved using stacking methods. This was my first ever building of stacking algorithm. There was no 'AHA' moment in the project rather it was a slow and gradual process of tuning the hyperparameters to get the right solution.

It was very important learning from the project how difficult is the task of accurately forecasting Stock prices. These predictions cannot be used as a standalone method for investing/trading. Rather it has to be included in a trading system where we compare/include our predictions with various other factors for taking right decision. One way to do this is to forecast prices for n days into future, compare their trends, values and build a strategy around it.

Capstone Project

Stock Index Predictor

The most difficult part of the project was to further lower the error without introducing new features or algorithms.

IMPROVEMENT :

Forecasting can be made more accurate by adding more features to the dataset. Sentiment analysis can be introduced as one of the features that should improve the accuracy of the model.

Advanced algorithms like GAN (Generative Adverserial Network) which is an actor critic method can also be studied to do Forecasting of the prices.

One important improvement can be done in the form of smoothing the predictions by some technique like simple or exponential moving avearge. This will give a very clear picture of analysis that can be carried out on our predictions and might help uncover some trends.

Thus, we have successfully studied various machine learning algorithms and build an ensemble of learners on Nifty50 dataset for predicting future prices. Our model closely tracks the trend of future prices. As per our conclusion, for accurately forecasting next day price, our selected features are not enough and more features should be introduced.