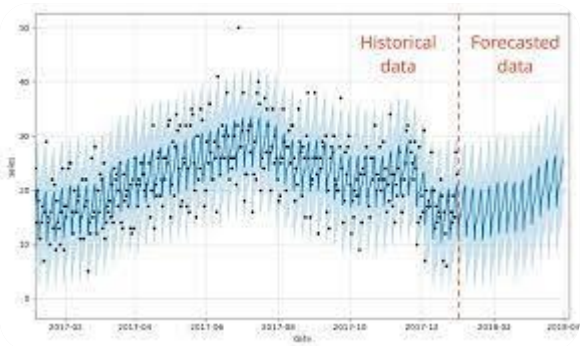




**PHASE 3**

ABSTRACT

**Phase 3: Modeling and Error Analysis**

Soumen Chatterjee

Phase 3 – 12-Jan-2022

At this stage we will use the dataset we have created at our EDA stage and use the same for ML modelling.
The initial main data set that is “train.csv” which contains the details of Units sold, has imbalance dataset. That is maximum of the rows have 0 units sold and very few (comparitiveliy) rows have at least one unit sold.
The main data set "train.csv" has 4617600 rows of data. Among the total rows we have 4498904 rows where 0 units only sold. This shows how imbalanced data set is.
To overcome the issue of imbalanced dataset we can refer the below techniques & others –

- Choose Proper Evaluation Metric
- Resampling (Oversampling and Under sampling)
- Synthetic Minority Oversampling Technique or SMOTE/ Synthetic Minority Over-Sampling Technique for Regression with Gaussian Noise SMOGN
- BalancedBaggingClassifier etc.

But here in this case we have used a classifier algorithm first, just to check whether query point's output is 0 or not. If 0 then it will share the same information as an output against the query point raised, if the initial classification model predict ‘1’ then it will pass back all the query point(s) to Regression model(s).

We are going to use **Logistic Regression as initial classification model**, as its accuracy & F1-Score is better compare to other classifiers as per the below shared details (https://analyticsindiamag.com/7-types-classification-algorithms/)

| Classification Algorithms | Accuracy | F1-Score |
|---|---|---|
| Logistic Regression | 84.60% | 0.6337 |
| Naïve Bayes | 80.11% | 0.6005 |
| Stochastic Gradient Descent | 82.20% | 0.5780 |
| K-Nearest Neighbours | 83.56% | 0.5924 |
| Decision Tree | 84.23% | 0.6308 |
| Random Forest | 84.33% | 0.6275 |
| Support Vector Machine | 84.09% | 0.6145 |

Details of matrixes used for the above mentioned comparison –

- Accuracy: (True Positive + True Negative) / Total Population
- Accuracy is a ratio of correctly predicted observation to the total observations. Accuracy is the most intuitive performance measure.
- True Positive: The number of correct predictions that the occurrence is positive
- True Negative: The number of correct predictions that the occurrence is negative

- F1-Score: (2 x Precision x Recall) / (Precision + Recall)
- F1-Score is the weighted average of Precision and Recall used in all types of classification algorithms. Therefore, this score takes both false positives and false negatives into account. F1-Score is usually more useful than accuracy, especially if you have an uneven class distribution.
- Precision: When a positive value is predicted, how often is the prediction correct?
- Recall: When the actual value is positive, how often is the prediction correct?

Below are some of the advantages and disadvantages for the classification models we considered and based on the below reasons we have picked Logistic Regression over other models.

- **Logistic Regression**

**Definition:** Logistic regression is a machine learning algorithm for classification. In this algorithm, the probabilities describing the possible outcomes of a single trial are modelled using a logistic function.

**Advantages:** Logistic regression is designed for this purpose (classification), and is most useful for understanding the influence of several independent variables on a single outcome variable.

**Disadvantages:** Works only when the predicted variable is binary, assumes all predictors are independent of each other and assumes data is free of missing values.

- **Naïve Bayes**

**Definition:** Naive Bayes algorithm based on Bayes' theorem with the assumption of independence between every pair of features. Naive Bayes classifiers work well in many real-world situations such as document classification and spam filtering.

**Advantages:** This algorithm requires a small amount of training data to estimate the necessary parameters. Naive Bayes classifiers are extremely fast compared to more sophisticated methods.

**Disadvantages:** Naive Bayes is known to be a bad estimator.

- **Stochastic Gradient Descent**

**Definition:** Stochastic gradient descent is a simple and very efficient approach to fit linear models. It is particularly useful when the number of samples is very large. It supports different loss functions and penalties for classification.

**Advantages:** Efficiency and ease of implementation.

**Disadvantages:** Requires a number of hyper-parameters and it is sensitive to feature scaling.

- **K-Nearest Neighbors**

**Definition:** Neighbors based classification is a type of lazy learning as it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the k nearest neighbors of each point.

**Advantages:** This algorithm is simple to implement, robust to noisy training data, and effective if training data is large.

**Disadvantages:** Need to determine the value of K and the computation cost is high as it needs to compute the distance of each instance to all the training samples.

- **Decision Tree**

**Definition:** Given a data of attributes together with its classes, a decision tree produces a sequence of rules that can be used to classify the data.

**Advantages:** Decision Tree is simple to understand and visualise, requires little data preparation, and can handle both numerical and categorical data.

**Disadvantages:** Decision tree can create complex trees that do not generalise well, and decision trees can be unstable because small variations in the data might result in a completely different tree being generated.

- **Random Forest**

**Definition:** Random forest classifier is a meta-estimator that fits a number of decision trees on various sub-samples of datasets and uses average to improve the predictive accuracy of the model and controls over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement.

**Advantages:** Reduction in over-fitting and random forest classifier is more accurate than decision trees in most cases.

**Disadvantages:** Slow real time prediction, difficult to implement, and complex algorithm.

- **Support Vector Machine**

**Definition:** Support vector machine is a representation of the training data as points in space separated into categories 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.

**Advantages:** Effective in high dimensional spaces and uses a subset of training points in the decision function so it is also memory efficient.

**Disadvantages:** The algorithm does not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.

**Metrics:**

For the classification model we have used F1-Score as main metric, along with others. The reason of selecting F1-Score as main metric is –

**The accuracy** of a classifier is the total number of correct predictions by the classifier divided by the total number of predictions. This may be good enough for a well-balanced class but not ideal for the imbalanced class problem. The other metrics such as **precision** is the measure of how accurate the classifier's prediction of a specific class and **recal**l is the measure of the classifier's ability to identify a class.

**For an imbalanced class dataset F1 score is a more appropriate metric**. It is the harmonic mean of precision and recall and the expression is –

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

So, if the classifier predicts the minority class but the prediction is erroneous and false-positive increases, the precision metric will be low and so as F1 score. Also, if the classifier identifies the minority class poorly, i.e. more of this class wrongfully predicted as the majority class then false negatives will increase, so recall and F1 score will low. F1 score only increases if both the number and quality of prediction improves.

F1 score keeps the balance between precision and recall and improves the score only if the classifier identifies more of a certain class correctly.

- **Setting up Classification Model**

- Creating a new dataset by merging 3 different data sets

Merging Data sets for the Classification Model

```
[ ]  df_train_key = df_train.merge(df_key, how='left', on='store_nbr')
     df_train_key_weather = df_train_key.merge(df_weather_cleaned, how='left', on=['station_nbr','date'])
     #df_train_key_weather.head(2)
```

- As per EDA removing not required columns, outliers, setting up new feature “flag” which is going to hold ‘0’ if units = 0 and going to hold ‘1’ is units > 0.

```
df_Classifier_cleand.loc[df_Classifier_cleand['units'] == 0, 'flag'] = 0
df_Classifier_cleand.loc[df_Classifier_cleand['units']  > 0, 'flag'] = 1
df_Classifier_cleand.flag = df_Classifier_cleand.flag.astype('int64')
bkupdf_Classifier_cleand = df_Classifier_cleand.copy()
df_Classifier_cleand.head(2)
```

- Creating time based train-test split, fitting the model & predicting

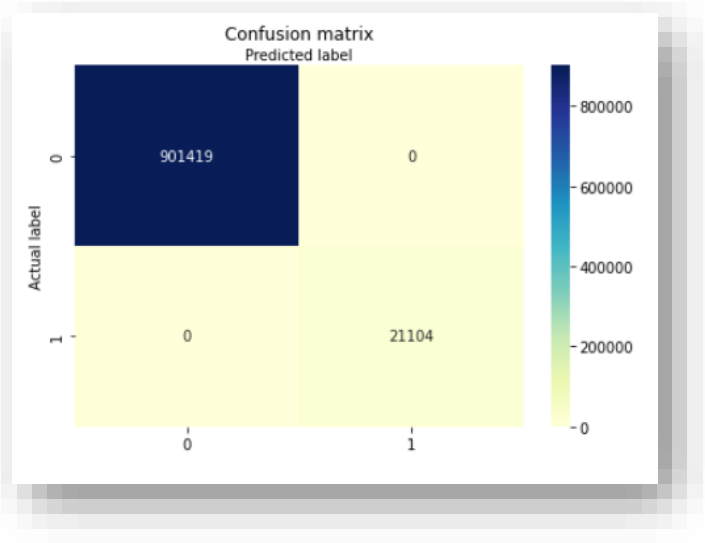
```
# create training and testing vars
X_train, X_test, y_train, y_test = train_test_split(df_Classifier_LgRgresn, y, test_size=0.2,shuffle=False,stratify=None)
```

```
model = LogisticRegression(solver='liblinear', random_state=6)
model.fit(X_train, y_train)
```

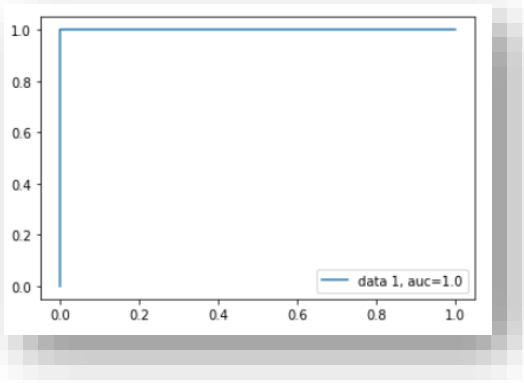
```
LogisticRegression(random_state=6, solver='liblinear')
```

```
y_pred=model.predict(X_test)
```

- Checking the metrics




```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-Score: 1.0
```




After using the classification model (Logistic regression), which we used to segregate between sales or no sales to deal with imbalanced data, we will now use the regression models.
Starting with easier one to complex one.

Below are some of the Regression models & corresponding advantages and dis advantages.

| Regression Model | Advantages | Disadvantages |
|---|---|---|
| Linear Regression | 1. Works well irrespective of the dataset size.<br>2. Gives information about the relevance of features. | 1. The assumptions of Linear Regression. |
| Polynomial Regression | 1. Works on any size of the dataset.<br>2. Works very well on non-linear problems. | 1. We need to choose the right polynomial degree for good bias/ variance tradeoff. |
| Support Vector Regression | 1. Easily adaptable.<br>2. Works very well on non-linear problems.<br>3. Not biased by outliers (object that deviates significantly from the rest). | 1. Compulsory to apply feature scaling.<br>2. Not well known.<br>3. Difficult to understand. |
| Decision Tree Regression | 1. Interpretability.<br>2. Works well on both linear and non-linear problems.<br>3. No need to apply feature scaling. | 1. Poor results on small datasets.<br>2. Overfitting can easily occur. |
| Random Forest Regression | 1. Powerful.<br>2. Accurate.<br>3. Good performance on many problems including non-linear. | 1. No interpretability.<br>2. Overfitting can easily occur.<br>3. We need to choose the number of trees. |

Ref: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-different-regression-models/

**a) We have first used the simple Linear Regression Model as starter:**
Accuracy of this model is not good

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# Model initialization
model_lnReg = LinearRegression()
# Fit the data(train the model)
model_lnReg.fit(X_train, y_train)
# Predict
y_predlnReg = model_lnReg.predict(X_test)
# printing values
#print('Slope:' ,model_lnReg.coef_)
#print('\nIntercept:', model_lnReg.intercept_)
Evaluation_Metrics (model_lnReg,y_test,y_predlnReg,"Linear Regression")
```

```
'Mean Absolute Error(MAE)' for the model -  Linear Regression is:  20.51493283305906
'Mean Squared Error(MSE)' for the model -  Linear Regression is:  672.9435992193156
'Root Mean Squared Error(RMSE)' for the model -  Linear Regression is:  25.94115647420746
NO 'Root Mean Squared Log Error(RMSLE)' for the model -  Linear Regression  as, prediction have negative values.
'R Squared (R2)' for the model -  Linear Regression is:  0.10794408408210399

----------------------------------------------------------------------------------------

'Model Train Score' for the model -  Linear Regression is:  0.12265193249704864

----------------------------------------------------------------------------------------

'Model Test Score' for the model -  Linear Regression is:  0.10794408408210399
```

```
[ ] cross_validate(LinearRegression())

    Mean RMSE: 30.547 (0.091)

    ----------------------------------------------------------------------------------------
    Mean RMSLE: nan (nan)

    ----------------------------------------------------------------------------------------
    Mean MAE: 24.365 (0.096)

    ----------------------------------------------------------------------------------------
```

**b) Then we used Lasso Regression – that Linear regression with L1 regularization**
Again accuracy was not that good.

A brief details of regularization and why Lasso is used –
In order to create less complex (parsimonious) model when you have a large number of features in your dataset, some of the Regularization techniques used to address over-fitting and feature selection are:
1. L1 Regularization
2. L2 Regularization

**A regression model that uses L1 regularization technique is called Lasso Regression and model which uses L2 is called Ridge Regression**.
The key difference between these two is the penalty term.

Ridge regression adds “squared magnitude” of coefficient as penalty term to the loss function. Here the highlighted part represents L2 regularization element.

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Cost function

Here, if lambda is zero then you can imagine we get back OLS. However, if lambda is very large then it will add too much weight and it will lead to under-fitting. Having said that it’s important how lambda is chosen. This technique works very well to avoid over-fitting issue.

**Lasso Regression (Least Absolute Shrinkage and Selection Operator)** adds “absolute value of magnitude” of coefficient as penalty term to the loss function.

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Cost function

Again, if lambda is zero then we will get back OLS whereas very large value will make coefficients zero hence it will under-fit.
The key difference between these techniques is that **Lasso shrinks the less important feature’s coefficient to zero thus, removing some feature altogether. So, this works well for feature selection in case we have a huge number of features**.
Traditional methods like cross-validation, stepwise regression to handle overfitting and perform feature selection work well with a small set of features but these techniques are a great alternative when we are dealing with a large set of features.

```
#X_train, X_test, y_train, y_test
from sklearn import linear_model
model_lassoReg = linear_model.Lasso(alpha=1,max_iter=5,tol=0.1,fit_intercept=True)
model_lassoReg.fit(X_train,y_train)
y_pred_LassoReg = model_lassoReg.predict(X_test)
Evaluation_Metrics (model_lassoReg,y_test,y_pred_LassoReg,"Linear Lasso Regression - L1 Regularization")
```

```
'Mean Absolute Error(MAE)' for the model -  Linear Lasso Regression - L1 Regularization is:  20.836003300331274
'Mean Squared Error(MSE)' for the model -  Linear Lasso Regression - L1 Regularization is:  679.1819263910813
'Root Mean Squared Error(RMSE)' for the model -  Linear Lasso Regression - L1 Regularization is:  26.0611190548503
'Root Mean Squared Log Error(RMSLE)' for the model -  Linear Lasso Regression - L1 Regularization is:  1.23204633912395
'R Squared (R2)' for the model -  Linear Lasso Regression - L1 Regularization is:  0.09967454014787103

----------------------------------------------------------------------------------------

'Model Train Score' for the model -  Linear Lasso Regression - L1 Regularization is:  0.11131671313921088

----------------------------------------------------------------------------------------

'Model Test Score' for the model -  Linear Lasso Regression - L1 Regularization is:  0.09967454014787103
```

We tried hyper parameter tuning for this model, but the result is not satisfactory –

Hyperparameter Tuning for Lasso reg

+ Code + Text

```
model_params = {
    'Lasso': {
        'model': linear_model.Lasso(),
        'params' : {
          'alpha': [1,10,20,30,40,50,60,70,80],
          'max_iter': [5,10,20,30,40,50],
           'fit_intercept': [True,False]
        }
    }
}
hyperparametertuning(model_params)
```

```
+----+---------+--------------+-------------------------------------------------------+
|    | model   |   best_score | best_params                                           |
|----+---------+--------------+-------------------------------------------------------|
|  0 | Lasso   |     0.106515 | {'alpha': 1, 'fit_intercept': True, 'max_iter': 5} |
+----+---------+--------------+-------------------------------------------------------+
```

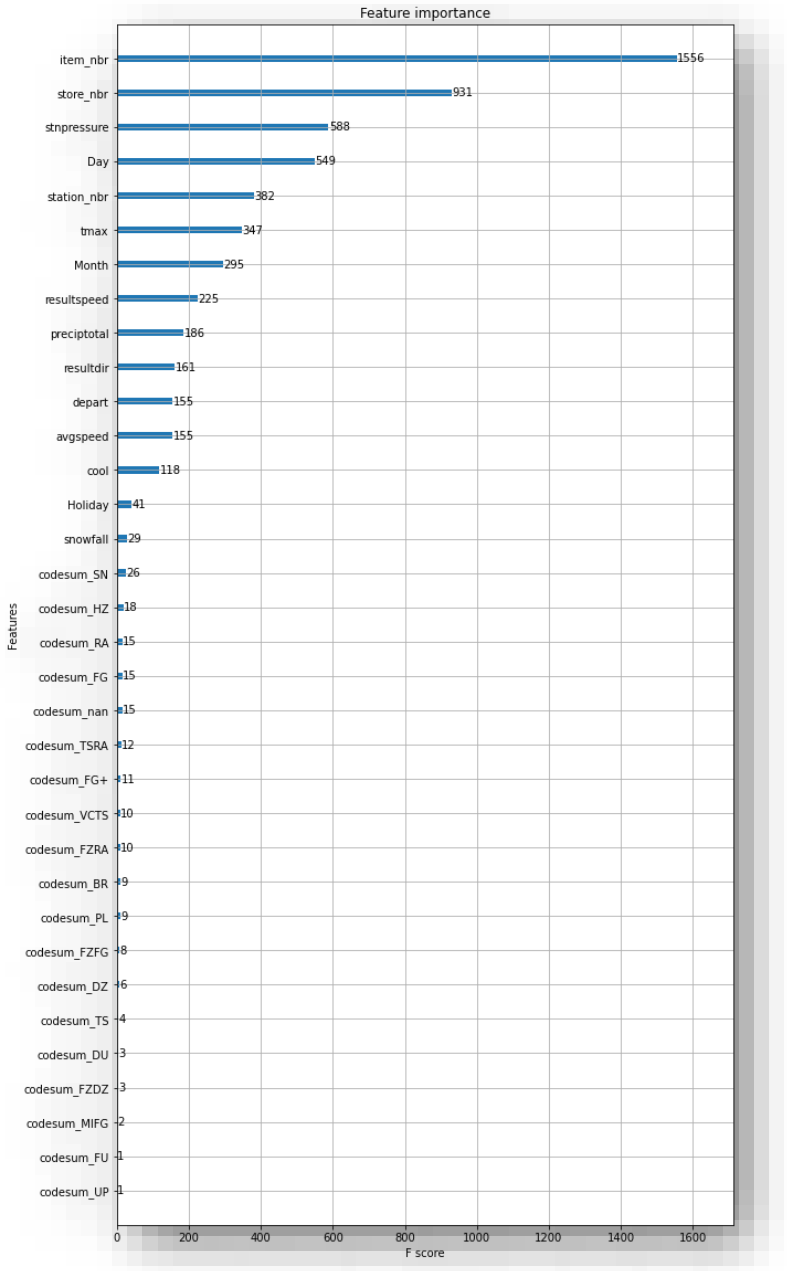
```
[ ] cross_validate(model_lassoReg)
```

```
Mean RMSE: 30.729 (0.080)

----------------------------------------------------------------------------------------
Mean RMSLE: 1.384 (0.010)

----------------------------------------------------------------------------------------
Mean MAE: 24.780 (0.091)

----------------------------------------------------------------------------------------
```

### c) Next Model tested was XGBRegressor:

```
model = XGBRegressor(eta=.1,gamma=1,max_depth=6)
model.fit(X_train, y_train)
# plot feature importance
```

We checked- which features are the least important, below is our findings.



XGBRegressor performed well –

```
y_predXGBRegressor = model.predict(X_test)
Evaluation_Metrics (model,y_test,y_predXGBRegressor,"XGBRegressor")
```

```
'Mean Absolute Error(MAE)' for the model -  XGBRegressor is:  16.433493532214836
'Mean Squared Error(MSE)' for the model -  XGBRegressor is:  627.0443722182928
'Root Mean Squared Error(RMSE)' for the model -  XGBRegressor is:  25.040854063276132
NO 'Root Mean Squared Log Error(RMSLE)' for the model -  XGBRegressor  as, prediction have negative values.
'R Squared (R2)' for the model -  XGBRegressor is:  0.16878822767722979

----------------------------------------------------------------------------------------------

'Model Train Score' for the model -  XGBRegressor is:  0.7771205501968009

----------------------------------------------------------------------------------------------

'Model Test Score' for the model -  XGBRegressor is:  0.16878822767722979
```

```
cross_validate(model)
```

```
Mean RMSE: 15.745 (0.141)

----------------------------------------------------------------------------------------------
Mean RMSLE: nan (nan)

----------------------------------------------------------------------------------------------
Mean MAE: 10.406 (0.101)

----------------------------------------------------------------------------------------------
```

We tried hyper parameter tuning for this model, below is the result –

```
model_params = {
    'XGBRegressor': {
        'model': XGBRegressor(),
        'params' : {
          'eta': [.1,.2],
          'gamma': [0,1],
           'max_depth': [4,6,8]
           #'min_child_weight':[1,2]
        }
    }
}
hyperparametertuning(model_params)
```

| | model | best_score | best_params |
|---|---|---|---|
| 0 | XGBRegressor | 0.730481 | {'eta': 0.1, 'gamma': 1, 'max_depth': 6} |

### d) Next we tried with SVR (Support vector regressor):

```
[ ] #X_train, X_test, y_train, y_test
    from sklearn.svm import SVR
    regressorSVR = SVR(kernel = 'rbf') #kernel = 'rbf'
    regressorSVR.fit(X_train, y_train)

    SVR()

[ ] y_predSVR = regressorSVR.predict(X_test)
    Evaluation_Metrics (regressorSVR,y_test,y_predSVR,"SVR (Support vector regressor)")

    'Mean Absolute Error(MAE)' for the model -  SVR (Support vector regressor) is:  18.818406797103272
    'Mean Squared Error(MSE)' for the model -  SVR (Support vector regressor) is:  673.7867578789894
    'Root Mean Squared Error(RMSE)' for the model -  SVR (Support vector regressor) is:  25.95740275680503
    NO 'Root Mean Squared Log Error(RMSLE)' for the model -  SVR (Support vector regressor)  as, prediction have negative values.
    'R Squared (R2)' for the model -  SVR (Support vector regressor) is:  0.10682639060631793

    ----------------------------------------------------------------------------------------

    'Model Train Score' for the model -  SVR (Support vector regressor) is:  0.09036113892631403

    ----------------------------------------------------------------------------------------

    'Model Test Score' for the model -  SVR (Support vector regressor) is:  0.10682639060631793
```

We tried hyper parameter tuning for this model, below is the result –

```
model_params = {
    'SVR': {
        'model': SVR(),
        'params' : {
            'kernel': [ 'rbf','poly'],#'linear','poly', , 'sigmoid', 'precomputed'
            'gamma': ['scale','auto'],
            'C': [1,2,3]
        }
    }
}
hyperparametertuning(model_params)
```

| | model | best_score | best_params |
|---|---|---|---|
| 0 | SVR | 0.0911346 | {'C': 3, 'gamma': 'scale', 'kernel': 'rbf'} |

### e) Then the next model tried is Random Forest:

```
# Fitting Random Forest Regression to the dataset
# import the regressor
from sklearn.ensemble import RandomForestRegressor

 # create regressor object
RFregressor = RandomForestRegressor(n_estimators = 100, random_state = 0)

# fit the regressor with x and y data
RFregressor.fit(X_train, y_train)
```

```
RandomForestRegressor(random_state=0)
```

```
y_predRFregressor = RFregressor.predict(X_test)
Evaluation_Metrics (RFregressor,y_test,y_predRFregressor,"Random Forest")
```

```
'Mean Absolute Error(MAE)' for the model -  Random Forest is:  18.529043620083392
'Mean Squared Error(MSE)' for the model -  Random Forest is:  745.7857656063524
'Root Mean Squared Error(RMSE)' for the model -  Random Forest is:  27.309078446669567
'Root Mean Squared Log Error(RMSLE)' for the model -  Random Forest is:  1.190416229245406
'R Squared (R2)' for the model -  Random Forest is:  0.011384304734749162

----------------------------------------------------------------------------------------------

'Model Train Score' for the model -  Random Forest is:  0.9684715215735444

----------------------------------------------------------------------------------------------

'Model Test Score' for the model -  Random Forest is:  0.011384304734749162
```

After trying with few models, and using cross validation, Hyper tuning we found XGBRegressor is the most suitable one –

| | model | best_score | RMSE | Cross Val - RMSE | Best_Params |
|---|---|---|---|---|---|
| 0 | Lasso Regression | 0.106515 | 26.0611 | 30.729 | {'alpha': 1, 'fit_intercept': True, 'max_iter': 5} |
| 1 | XGBRegressor | 0.730481 | 25.0408 | 15.745 | {'eta': 0.1, 'gamma': 1, 'max_depth': 6} |
| 2 | SVR (Support vector regressor) | 0.091134 | 25.9574 | 31.229 | {'C': 3, 'gamma': 'scale', 'kernel': 'rbf'} |
| 3 | Random Forest | 0.300001 | 27.3091 | 15.488 | best_params |

**Metrics preferred:**

- **Mean Absolute Error (MAE)**

Mean absolute error, also known as L1 loss is one of the simplest loss functions and an easy-to-understand evaluation metric. It is calculated by taking the absolute difference between the predicted values and the actual values and averaging it across the dataset. Mathematically speaking, it is the arithmetic average of absolute errors. MAE measures only the magnitude of the errors and doesn't concern itself with their direction. The lower the MAE, the higher the accuracy of a model.

Mathematically, MAE can be expressed as follows,

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} | y_i - \hat{y}_i |$$

Where y_i = actual value, y_hat_i = predicted value, n = sample size

**Pros of the Evaluation Metric:**

- It is an easy to calculate evaluation metric.
- All the errors are weighted on the same scale since absolute values are taken.
- It is useful if the training data has outliers as MAE does not penalize high errors caused by outliers.
- It provides an even measure of how well the model is performing.

**Cons of the evaluation metric:**

- Sometimes the large errors coming from the outliers end up being treated as the same as low errors.
- MAE follows a scale-dependent accuracy measure where it uses the same scale as the data being measured. Hence it cannot be used to compare series' using different measures.
- One of the main disadvantages of MAE is that it is not differentiable at zero. Many optimization algorithms tend to use differentiation to find the optimum value for parameters in the evaluation metric.
- It can be challenging to compute gradients in MAE.

- **Root Mean Squared Error (RMSE)**
  RMSE is computed by taking the square root of MSE. RMSE is also called the Root Mean Square Deviation. It measures the average magnitude of the errors and is concerned with the deviations from the actual value. RMSE value with

zero indicates that the model has a perfect fit. The lower the RMSE, the better the model and its predictions. A higher RMSE indicates that there is a large deviation from the residual to the ground truth. RMSE can be used with different features as it helps in figuring out if the feature is improving the model's prediction or not.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

**Pros of the Evaluation Metric:**

- RMSE is easy to understand.
- It serves as a heuristic for training models.
- It is computationally simple and easily differentiable which many optimization algorithms desire.
- RMSE does not penalize the errors as much as MSE does due to the square root.

**Cons of the evaluation metric:**

- Like MSE, RMSE is dependent on the scale of the data. It increases in magnitude if the scale of the error increases.
- One major drawback of RMSE is its sensitivity to outliers and the outliers have to be removed for it to function properly.
- RMSE increases with an increase in the size of the test sample. This is an issue when we calculate the results on different test samples.

As the same code we have used to evaluate the models, we have created a function namely "Evaluation_Metrics"

**Cross Validated:**

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

We have also used K Fold cross validation to measure the model's performance.

And to do so a function is defined namely "cross_validate(model)".

**Hyper parameter Tuning:**

We have also tuned the hyper parameters of the model to get the optimal result.

A Machine Learning model is defined as a mathematical model with a number of parameters that need to be learned from the data. By training a model with existing data, we are able to fit the model parameters.

However, there is another kind of parameters, known as hyper parameters that cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model such as its complexity or how fast it should learn.

Some examples of model hyper parameters include:

The penalty in Logistic Regression Classifier i.e. L1 or L2 regularization

The learning rate for training a neural network.

The C and sigma hyper parameters for support vector machines.

The k in k-nearest neighbors.

Vanilla linear regression does not have any hyperparameters. Variants of linear regression (ridge and lasso) have regularization as a hyperparameter. The decision tree has max depth and min number of observations in leaf as hyperparameters.

## References


1. https://towardsdatascience.com/strategies-and-tactics-for-regression-on-imbalanced-data-61eeb0921fca
2. https://github.com/nickkunz/smogn
3. https://analyticsindiamag.com/7-types-classification-algorithms/
4. https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8
5. https://stats.stackexchange.com/questions/48360/is-standardization-needed-before-fitting-logistic-regression
6. https://www.listendata.com/2017/04/how-to-standardize-variable-in-regression.html
7. https://realpython.com/logistic-regression-python/
8. https://www.codegrepper.com/code-examples/python/standardization+classification+sklearn
9. https://www.aionlinecourse.com/tutorial/machine-learning/support-vector-regression
10. https://www.geeksforgeeks.org/linear-regression-python-implementation/
11. https://towardsdatascience.com/5-reasons-why-you-should-use-cross-validation-in-your-data-science-project-8163311a1e79
12. https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter
13. https://scikit-learn.org/stable/modules/cross_validation.html
14. https://www.analyticsvidhya.com/blog/2021/05/know-the-best-evaluation-metrics-for-your-regression-model/
15. https://machinelearningmastery.com/regression-metrics-for-machine-learning/
16. https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/
17. https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
18. https://www.analyticsvidhya.com/blog/2021/06/5-techniques-to-handle-imbalanced-data-for-a-classification-problem/
19. https://towardsdatascience.com/choosing-the-best-classification-algorithm-f254f68cca39
20. https://analyticsindiamag.com/7-types-classification-algorithms/
21. https://www.geeksforgeeks.org/advantages-and-disadvantages-of-different-regression-models/
22. https://www.analyticsvidhya.com/blog/2021/10/evaluation-metric-for-regression-models/