A Project Report on

SALES PREDICTION USING MACHINE LEARNING

Submitted In Partial Fulfillment of The Requirement for the Award of the Degree

MASTER OF TECHNOLOGY

in

WIRELESS NETWORKS AND COMPUTING

Submitted By

Harsh Jha (2023WNC-02)

Navneet Sagar (2023WNC-03)

Under The Supervision of

Dr. Santosh Singh Rathore

(Assistant Professor)

Atal Bihari Vajpayee
Indian Institute of Information Technology and Management (ABV-IIITM), Gwalior
(An Institute of National Importance, Ministry of Education, Government of India)

Department Of Information Technology

ATAL BIHARI VAJPAYEE INDIAN INSTITUTE OF INFORMATION TECHNOLOGY

Gwalior, MP - 474015

(Batch 2023-2025)

TABLE OF CONTENTS

ABSTRACT

In this project Store Sales Prediction using machine learning and python, the task is to predict the sales of different stores based on the attributes available in the dataset. Predictive analytics can help us to study and discover the factors that determine the number of sales that a retail store will have in the future by using different machine learning techniques and trying to determine the best algorithm suited to our particular problem statement. We have implemented normal regression techniques as well as boosting techniques in our approach and have found that the boosting algorithms have better results than the regular regression algorithms.

1. INTRODUCTION

Sales play a key role in the business. At the company level, sales prediction is the major part of the business plan and significant inputs for decision-making activities. It is essential for organizations to produce the required quantity at the specified time. For that, sales forecasting will give the idea about how an organization should manage its budgeting, workforce and resources. This prediction helps the business management to determine how much products should be manufactured, how much revenue can be expected and what could be the requirement of employees, investment and equipment. By analyzing the future trends and needs, Sales prediction helps to improve the business growth.

Sales prediction is an important part of modern business intelligence. It can be a complex problem, especially in the case of lack of data, missing data, and the presence of outliers. Sales prediction is rather a regression problem than a time series problem. Practice shows that the use of regression approaches can often give us better results compared to time series methods. Machine-learning algorithms make it possible to find patterns in the time series..

2. PROBLEM STATEMENT

Most of the business organizations heavily depend on a knowledge base and demand prediction of sales trends. Sales prediction is the process of estimating future sales. Accurate sales predictions enable companies to make informed business decisions and predict short-term and long-term performance. Companies can base their forecasts on past sales data, industry- wide comparisons, and economic trends. Sales forecasts help sales teams achieve their goals by identifying early warning signals in their sales pipeline and course correction before it's too late. The goal is to improve the accuracy from the existing project. So that the sales and profit could be increased for the companies. Choosing an efficient algorithm from comparing different algorithms to improve the prediction furthermore.

3. LITERATURE SURVEY

PAPER-1:

Intelligent Sales Prediction Using Machine Learning Techniques.

Abstract: The detailed study and analysis of comprehensible predictive models to improve future sales predictions are carried out in this research. Traditional forecast systems are difficult to deal with the big data and accuracy of sales forecasting.

Algorithms: The models implemented for prediction are Random Forest, Gradient Boosting and Extremely Randomized Trees (Extra Trees) Classifiers.

Conclusion: Random Trees was confirmed to be a very effective.

PAPER-2:

Forecasting the Retail Sales of China's Catering Industry Using Support Vector Machines.

Abstract: The forecast of China's catering retail sales was studied in this paper. The seasonal impact was considered in the forecasting. The retail sales were predicted using the seasonal auto-regressive integrated moving average (ARIMA) model.

Algorithms: ARIMA, SVM.

Conclusion: SVM method is obviously superior to the seasonal ARIMA method regardless of the long-term forecasting or the short- term forecasting.

PAPER-3:

An Intelligent Model For Predicting the Sales of a Product.

Abstract: The approach shown in this paper is a systematic, accurate

and precise model building to be used in computing and predicting current scenario and future projection of a product in market respectively.

Algorithms: Random forest algorithm, neural network.

Conclusion: Neural network.

PAPER-4:

Sales Prediction Using Machine Learning Algorithms.

Abstract: The aim of this paper is to propose a dimension for predicting the future sales of Big Mart Companies keeping in view the sales of previous years. A comprehensive study of sales prediction is done using Machine Learning models.

Algorithms: Linear Regression, K-Neighbours Regressor, XGBoost, Regressor and Random Forest Regressor.

Conclusion: Random Forest Algorithm is found to be

the most suitable

PAPER-5:

Comparison of Different Machine Learning Algorithms for Multiple Regression on Black Friday Sales Data.

Abstract: This study focuses on the field of prediction models to develop an accurate and efficient algorithm to analyze the customer spending in the past and output the future spending of the customers with same features.

Algorithms: Regression, Decision Tree, XGBoost.

Conclusion: XGBoost.

PAPER-6 :

Forecasting of Walmart Sales using Machine Learning

Abstract: The ability to predict data accurately is extremely valuable in a vast array of domains such as stocks, sales, weather or even sports. Presented here is the study and implementation of several ensemble classification algorithms employed on sales data, consisting of weekly retail sales numbers from different departments in Walmart retail outlets all over the United States of America.

Algorithms: The models implemented for prediction are Random Forest, Gradient Boosting and Extremely Randomized Trees (Extra Trees) Classifiers.

Conclusion: Random Trees was confirmed to be a very effective.

PAPER-7:

Sales Prediction For Big Mart.

Abstract: A retailer company wants a model that can predict accurate sales so that it can keep track of customers future demand and update in advance the sale inventory. In this work, we propose a technique to optimize the parameters and select the best tuning hyper parameters, further ensemble with Xgboost techniques for forecasting the future sales of a retailer company such as Big Mart and we found our model produces the better result.

Algorithms: Xgboost techniques.

Conclusion: Experimental analysis found our technique produce more accurate

PAPER-8:

A Deep Learning Approach for the Prediction of Retail Store Sales.

Abstract: The purpose of this research is to construct a sales prediction model for retail stores using the deep learning approach, which has gained significant attention in the rapidly developing field of machine learning in recent years. Using such a model for analysis, an approach to store management could be formulated **.**

Algorithms: Logistic regression model

Conclusion: The accuracy decreased by around 13% when the logistic regression model was used.

4. MODULES

4.1 DATA COLLECTION:

The dataset has been collected from https://www.kaggle.com/ . The training dataset contains 9 columns and 66900 rows. The Test dataset contains 9 columns and 33450. The dataset contains 9 variables which includes Store, DayOfWeek, Date, Sales, Customers, Open, Promo, StateHoliday, SchoolHoliday.

Attribute of Store Datasets-

Id - an Id that represents a (Store, Date) duple within the test set

Store - a unique Id for each store

Sales - the turnover for any given day (this is what you are predicting)

Customers - the number of customers on a given day

Open - an indicator for whether the store was open: 0 = closed, 1 = open

StateHoliday - indicates a state holiday. Normally all stores, with few exceptions, are closed on state holidays. Note that all schools are closed on public holidays and weekends. a = public holiday, b = Easter holiday, c = Christmas, 0 = None

SchoolHoliday - indicates if the (Store, Date) was affected by the closure of public schools

StoreType - differentiates between 4 different store models: a, b, c, d

Assortment - describes an assortment level: a = basic, b = extra, c = extended

CompetitionDistance - distance in meters to the nearest competitor store

CompetitionOpenSince[Month/Year] - gives the approximate year and month of the time the nearest competitor was opened

Promo - indicates whether a store is running a promo on that day

Promo2 - Promo2 is a continuing and consecutive promotion for some stores: 0 = store is not participating, 1 = store is participating

Promo2Since[Year/Week] - describes the year and calendar week when the store started participating in Promo2

PromoInterval - describes the consecutive intervals Promo2 is started,

naming the months the promotion is started anew. E.g. "Feb,May,Aug,Nov" means each round starts in February, May, August, November of any given year for that store

4.2 DATA PREPROCESSING:

This step is an important step in the data mining process. Because it improves the quality of the experimental raw data.

i) Removal of Null values:

In this step, the null values in the fields Product Category2 and Product Category3 are filled with the mean value of the feature.

ii) Converting Categorical values into numerical:

Machine learning deals with numerical values easily because of the machine readable form. Therefore, the categorical values like Product ID, Gender, Age and City Category are converted to numerical values.

Step1: Based on its datatype, we have selected the categorical values.

Step2: By using python, we have converted the categorical values into numerical values.

iii) Separate the target variable:

Here, we have to separate the target feature in which we are going to predict. In this case, purchase is the target variable.

Step1: The target label purchase is assigned to the variable ‘y’.

Step2: The preprocessed data except the target label purchase is assigned to the variable ‘X’.

iv) Standardize the features:

Here, we have to standardize the features because it arranges the data in a standard normal distribution. The standardization of the data is made only for training data most of the time because any kind of transformation of the features only be fitted on the training data.

Step1: Only trained data was taken.

Step2: we have standardize the features.

```
X_train.describe()
```

✓ 0.1s

	Store	Customers	CompetitionDistance	Promo	Promo2	StateHoliday	StoreType	Assortment	AvgSales	AvgCustomers	...	MedSalesPerCustomer	DayOfWeek	Week
count	46484.000000	46484.000000	46484.000000	46484.000000	46484.000000	46484.000000	46484.000000	46484.000000	46484.000000	46484.000000	...	46484.000000	46484.000000	46484.0
mean	557.153214	808.864319	5476.848378	0.518695	0.475497	0.001592	1.184580	0.949832	7753.249143	808.711877	...	9.945555	3.512542	23.941786
std	321.506595	426.428865	7866.627157	0.499656	0.499405	0.052860	1.355622	0.993785	2654.517009	377.226618	...	2.051354	1.718889	23.382089
min	1.000000	82.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	3004.629630	249.481481	...	3.627509	1.000000	1.0
25%	280.000000	544.000000	680.000000	0.000000	0.000000	0.000000	0.000000	0.000000	5908.056604	581.754717	...	8.390059	2.000000	3.0
50%	556.000000	715.000000	2320.000000	1.000000	0.000000	0.000000	0.000000	0.000000	7430.222222	726.566038	...	9.802048	3.000000	5.0
75%	836.000000	952.000000	6880.000000	1.000000	1.000000	0.000000	3.000000	2.000000	8977.314815	922.851852	...	11.291318	5.000000	50.0
max	1115.000000	4962.000000	75860.000000	1.000000	1.000000	2.000000	3.000000	2.000000	21820.666667	3360.287879	...	16.915789	7.000000	52.0

4.3 ALGORITHMS

Pseudocode of the Algorithm

load_data()
clean_data(train)
clean_data(store)
filter_data(train)
filter_data(store)
visualize_data(train)
feature_engineering(train, store)
split_data(train)
build_decision_tree_model(X_train, y_train)
evaluate_model(decision_tree, X_train, y_train, X_test, y_test)
build_random_forest_model(X_train, y_train)
evaluate_model(random_forest, X_train, y_train, X_test, y_test)
build_xgboost_model(X_train, y_train)
evaluate_model(xgboost_tree, X_train, y_train, X_test, y_test)
predict_test_data(test, store, xgboost_tree)
save_predictions(predictions)

XGBoost:

XGBoost also known as Extreme Gradient Boosting has been used in order to get an efficient model with high computational speed and efficacy. The formula makes predictions using the ensemble method that models the anticipated errors of some decision trees to optimize last predictions. Production of this model also reports the value of each feature's effects in determining the last building performance score prediction.

Random Forest:

Random forest is referred to as a supervised machine learning ensemble method, which uses the multiple decision trees. It involves the technique called Bootstrap aggregation also known as bagging which aims to reduce the

complexity of the models that overfit the training data . In this algorithm, rather than depending on the individual decision tree it will combine the multiple decision trees to find the final outcome.

Decision Trees Algorithm:

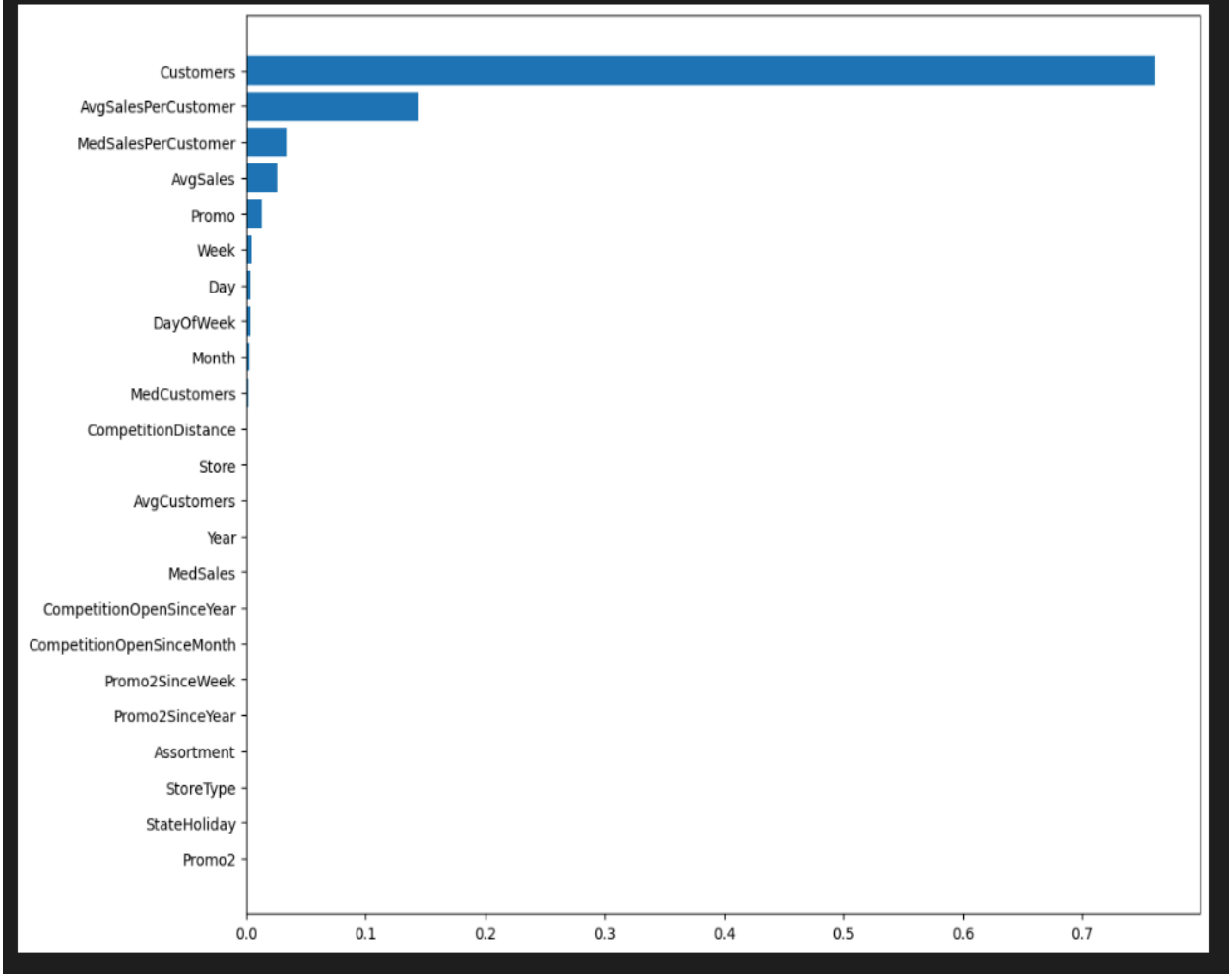
This algorithm works by creating a large number of unpruned decision trees from the training dataset. Predictions are made by averaging the prediction of the decision trees in the case of regression or using majority voting in the case of classification.

Feature Selection:

Decision Tree

```
from sklearn.tree import DecisionTreeRegressor
decision_tree = DecisionTreeRegressor()
decision_tree.fit(X_train, y_train)
plot_importance(decision_tree)
```

✓ 21.7s




```
y_hat = decision_tree.predict(X_test)
score(decision_tree, X_train, y_train, y_test, y_hat)
```

✓ 3.1s

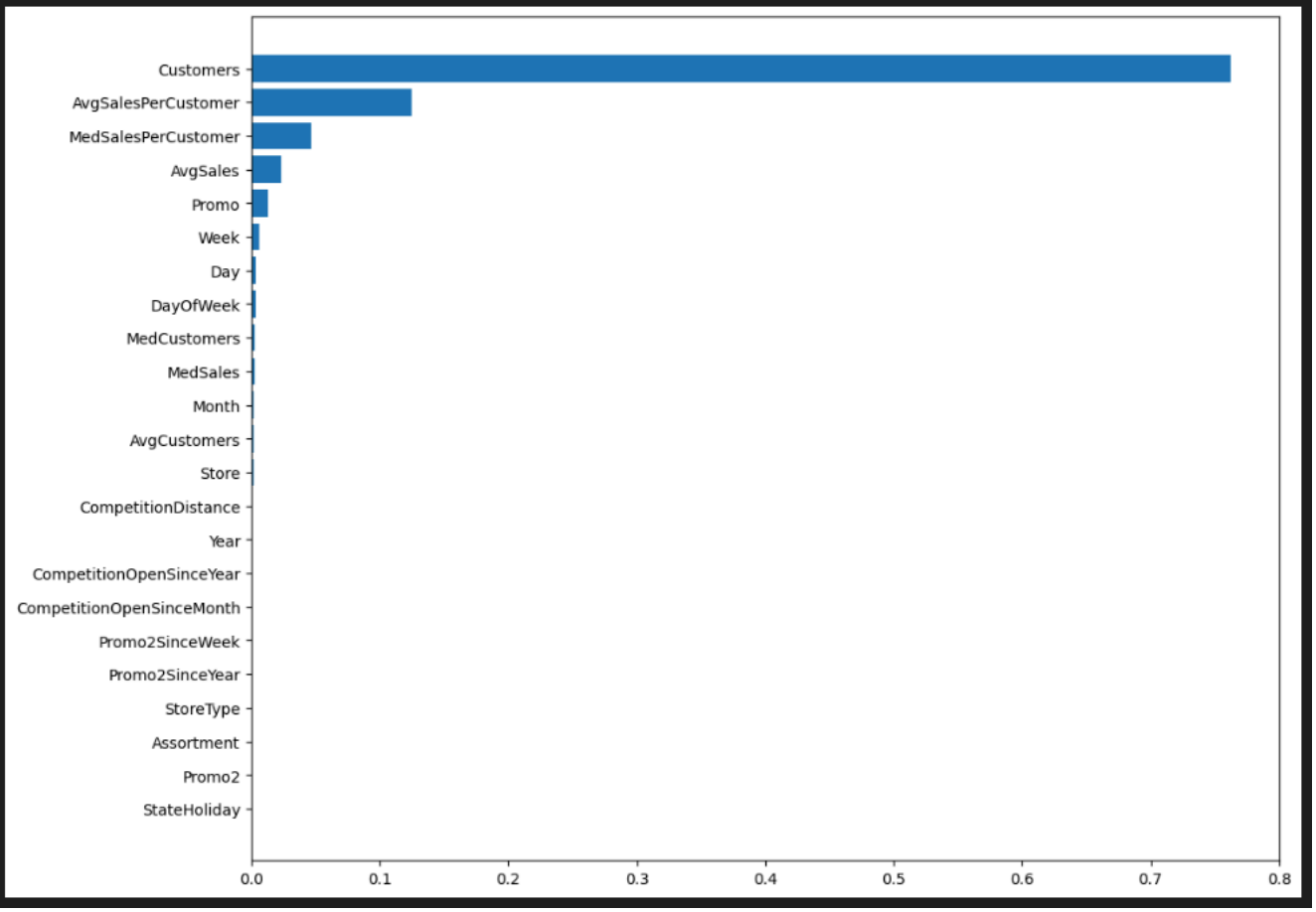
```
Mean -0.08527742714808229
Variance 1.0467347419952391e-06
RMSPE 0.08278304338385223
```

Random forest

```
from sklearn.ensemble import RandomForestRegressor
randomForest = RandomForestRegressor(n_estimators=25, n_jobs=-1, verbose=1)
randomForest.fit(X_train, y_train)
plot_importance(randomForest)
```

✓ 27.5s

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    3.6s finished
```




```
y_hat = randomForest.predict(X_test)
score(randomForest, X_train, y_train, y_test, y_hat)
```

✓ 15.6s

```
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  25 out of  25 | elapsed:    0.0s finished
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    3.0s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  25 out of  25 | elapsed:    0.0s finished
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    3.1s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  25 out of  25 | elapsed:    0.0s finished
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    2.9s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  25 out of  25 | elapsed:    0.0s finished
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    2.9s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  25 out of  25 | elapsed:    0.0s finished
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent workers.
Mean -0.06084524457718714
Variance 6.694684626283388e-07
RMSPE 0.05989317288655422
```

Xgboost

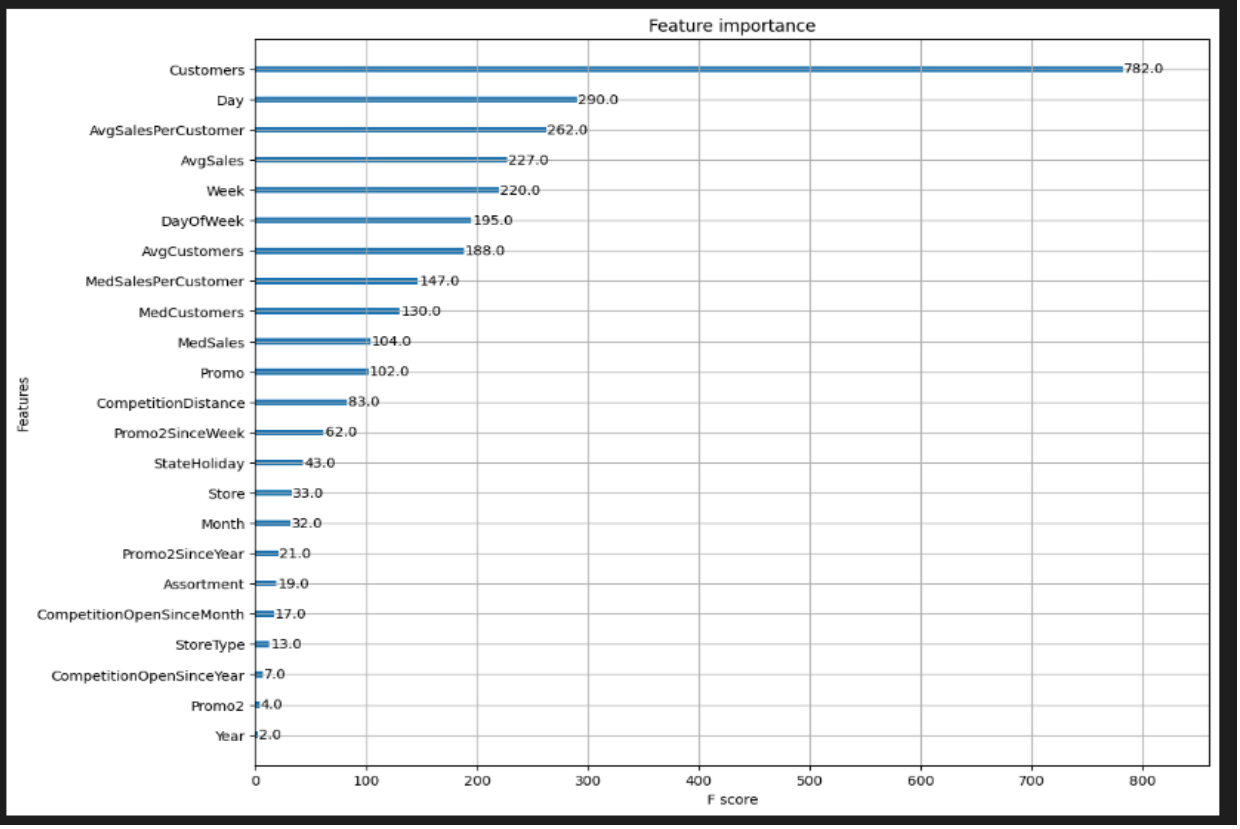
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.8, device=None, early_stopping_rounds=None,
             enable_categorical=False, eta=0.1, eval_metric=None,
             feature_types=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=None, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=2,
             max_leaves=None, min_child_weight=2, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=1000,
             n_jobs=-1, num_parallel_tree=None, ...)
```

```
print("Note that this is not in percentage, thus not to scale of graphs above")
xgb.plot_importance(xgboost_tree)
```

✓ 0.5s

```
Note that this is not in percentage, thus not to scale of graphs above

<Axes: title={'center': 'Feature importance'}, xlabel='F score', ylabel='Features'>
```




```
def rmspe_exp(y, y_hat):
    return rmspe(np.expm1(y), np.expm1(y_hat))

rmpse_xg_scorer = make_scorer(rmspe_exp, greater_is_better = False) # Loss function

def score(model, X_train, y_train, y_test, y_hat):
    score = cross_val_score(model, X_train, y_train, scoring=rmpse_xg_scorer, cv=5)
    print('Mean', score.mean())
    print('Variance', score.var())
    print('RMSPE', rmspe(y_test, np.expm1(y_hat)))

y_hat = xgboost_tree.predict(X_test[X])
score(xgboost_tree, X_train[X], np.log1p(y_train), y_test, y_hat)
```

✓ 38.0s

```
Mean -0.05527399756181465
Variance 2.1195917600129122e-07
RMSPE 0.054297836525485525
```

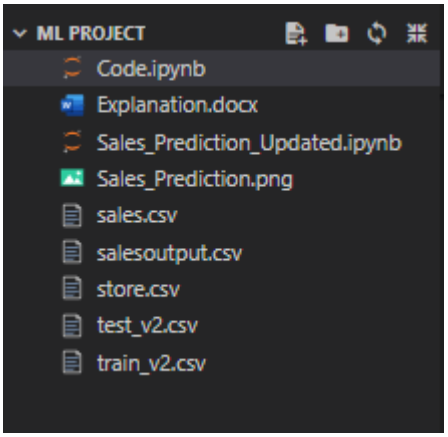
4.4 RESULTS AND DISCUSSION:

The evaluation of the machine learning algorithms is an essential part of any prediction model building. For that, we should carefully choose the evaluation metrics . These metrics are used to measure or judge the quality of the model. The performance of the machine learning algorithms are mainly focusing on accuracy. Companies use machine learning models with high accuracy for the practical business decisions.

```
test = pd.read_csv("C:\\Users\\harsh\\Desktop\\ML project\\test_v2.csv", parse_dates=[2], dtype=ty
features = build_features(test, store)

y_hat = np.expm1(xgboost_tree.predict(features[X]))

df = pd.DataFrame({"Id": range(1, len(test) + 1), 'Sales': y_hat})
df.loc[test['Open'] == 0, 'Sales'] = 1
df.to_csv('salesoutput.csv', index=False)
✓ 0.4s
```

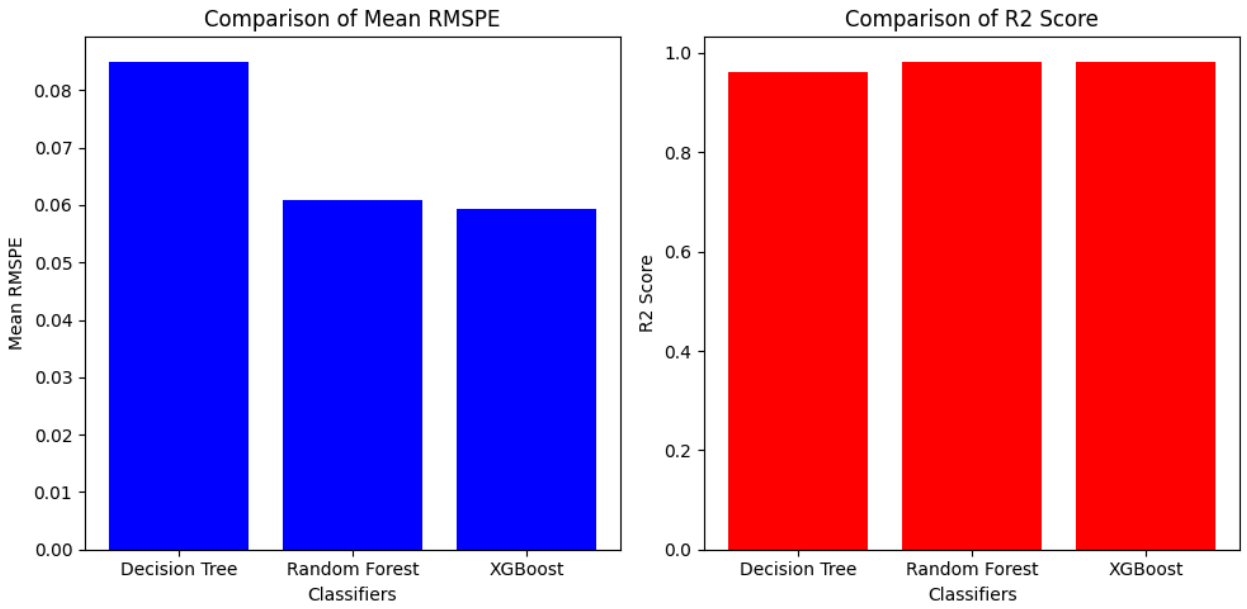


Based on the performance, we have concluded that the XGBoost and Gradient Boost algorithms are considered as the best fit compared to other algorithms. This comparative evaluation will help the organizations to choose the better and efficient machine-learning model.

Decision Tree:
Mean RMSPE: 0.08502289598872459
R2 Score: 0.9616017027548811

Random Forest:
Mean RMSPE: 0.06082037225363275
R2 Score: 0.9809883395714637

XGBoost:
Mean RMSPE: 0.059217758108770835
R2 Score: 0.9831096504954233



In summary, based on these scores:

- The Random Forest model performs the best in terms of both RMSPE and R2 score.
- The Decision Tree model performs reasonably well but has a slightly higher error rate compared to Random Forest.
- The XGBoost model has the lowest RMSPE but highest R2 score, indicating very high potential model performance. Further investigation and tuning may be necessary to improve its performance.

5. SYSTEM REQUIREMENTS

5.1 HARDWARE REQUIREMENTS

- System : i5 Processor
- Hard Disk : 500 GB
- Ram : 8GB

5.2 SOFTWARE REQUIREMENTS

- Operating system : Windows 7 or above, linux.
- Scripting Tool: Jupyter Notebook, Google colab
  Language: Python3.0 or above

6. CONCLUSION

Sales forecasting is mainly required for the organizations for business decisions. Accurate forecasting will help the companies to enhance the market growth. Machine learning techniques provide an effective mechanism in prediction and data mining as it overcomes the problem with traditional techniques. These techniques enhance the data optimization along with improving the efficiency with better results and greater predictability. After predicting the purchase amount, the companies can apply some marketing strategies for certain sections of customers so that the profit could be enhanced.

7. FUTURE SCOPE

Potential avenues for future work and improvements:

- Fine-tuning model hyperparameters for better performance.
- Incorporating additional data sources or features for enhanced prediction accuracy.
- Exploring advanced modeling techniques or deep learning approaches.

REFERENCES

[1] Sunitha Cheriyan, Shaniba Ibrahim, Saju Mohanan & Susan Treesa (2018) Intelligent Sales Prediction Using Machine Learning Techniques.

[2] Xiangsheng Xie & Gang Hu (2008). Forecasting the Retail Sales of China's Catering Industry.

[3] Avinash kumar, Neha Gopal & Jatin Rajput(2020). An Intelligent Model For Predicting the Sales of a Product.

[4] Purvika Bajaj, Renesa Ray, Shivani Shedge & Shravani Vidhate(2020). SALES PREDICTION USING MACHINE LEARNING ALGORITHMS.

[5] Ching-Seh (Mike) Wu. Pratik Patil & Saravana Gunaseelan(2018). Comparison of Different Machine Learning Algorithms for Multiple Regression on Black Friday Sales Data.

[6] Nikhil Sunil Elias, Seema Singh(2019).FORECASTING of WALMART SALES using MACHINE LEARNING ALGORITHMS.

[7] Yuta Kaneko & Katsutoshi Yada(2016). A Deep Learning Approach for the Prediction of Retail Store Sales.

[8] Gopal Behera & Neeta Nain (2019). Sales Prediction For Big Mart.

APPENDIX

```
# %%
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sb
# Size the plot appropriately for online display
plt.rcParams['figure.figsize'] = (12.0, 10.0)

# %%
types = {'StateHoliday': np.dtype(str)}
train = pd.read_csv("C:\\Users\\harsh\\Desktop\\ML project\\train_v2.csv", parse_dates=[2], nrows=66901,
dtype=types)
store = pd.read_csv("C:\\Users\\harsh\\Desktop\\ML project\\store.csv")

# %%
not_open = train[(train['Open'] == 0) & (train['Sales'] != 0)]
print("No closed store with sales: " + str(not_open.size == 0))

# %%
no_sales = train[(train['Open'] == 1) & (train['Sales'] <= 0)]
print("No open store with no sales: " + str(no_sales.size == 0))

# %%
train = train.loc[train['Sales'] > 0]
assert(train[train['Sales'] == 0].size == 0)

# %%
dates = pd.to_datetime(train['Date'], format="%Y%m%d:%H:%M:%S.%f").sort_values()
dates = dates.unique()
start_date = dates[0]
end_date = dates[-1]
print("Start date: ", start_date)
print("End Date: ", end_date)
date_range = pd.date_range(start_date, end_date).values
assert(all(dates == date_range))

# %%
f, ax = plt.subplots(7, sharex=True, sharey=True)
for i in range(1, 8):
    mask = train[train['DayOfWeek'] == i]
    ax[i - 1].set_title("Day {0}".format(i))
    ax[i - 1].scatter(mask['Customers'], mask['Sales'], label=i)

plt.legend()
plt.xlabel('Customers')
```

```
plt.ylabel('Sales')
plt.show()

# %%
plt.scatter(train['Customers'], train['Sales'], c=train['DayOfWeek'], alpha=0.7,
cmap=plt.cm.get_cmap('viridis'))

plt.xlabel('Customers')
plt.ylabel('Sales')
plt.show()

# %%
for i in ["0", "a", "b", "c"]:
    data = train[train['StateHoliday'] == i]
    if (len(data) == 0):
        continue
    plt.scatter(data['Customers'], data['Sales'], label=i)

plt.legend()
plt.xlabel('Customers')
plt.ylabel('Sales')
plt.show()

# %%
for i in [0, 1]:
    data = train[train['SchoolHoliday'] == i]
    if (len(data) == 0):
        continue
    plt.scatter(data['Customers'], data['Sales'], label=i)

plt.legend()
plt.xlabel('Customers')
plt.ylabel('Sales')
plt.show()

# %%
for i in [0, 1]:
    data = train[train['Promo'] == i]
    if (len(data) == 0):
        continue
    plt.scatter(data['Customers'], data['Sales'], label=i)

plt.legend()
plt.xlabel('Customers')
plt.ylabel('Sales')
plt.show()

# %%
train['SalesPerCustomer'] = train['Sales'] / train['Customers']

avg_store = train.groupby('Store')[['Sales', 'Customers', 'SalesPerCustomer']].mean()
```

```
avg_store.rename(columns=lambda x: 'Avg' + x, inplace=True)
store = pd.merge(avg_store.reset_index(), store, on='Store')
store.head()

# %%
for i in ['a', 'b', 'c', 'd']:
    data = store[store['StoreType'] == i]
    if (len(data) == 0):
        continue
    plt.scatter(data['AvgCustomers'], data['AvgSales'], label=i)

plt.legend()
plt.xlabel('Average Customers')
plt.ylabel('Average Sales')
plt.show()

# %%
for i in [0, 1]:
    data = store[store['Promo2'] == i]
    if (len(data) == 0):
        continue
    plt.scatter(data['AvgCustomers'], data['AvgSales'], label=i)

plt.legend()
plt.xlabel('Average Customers')
plt.ylabel('Average Sales')
plt.show()

# %%
# fill NaN values
store["CompetitionDistance"].fillna(-1)
plt.scatter(store['CompetitionDistance'], store['AvgSales'])

plt.xlabel('CompetitionDistance')
plt.ylabel('Average Sales')
plt.show()

# %%
def build_features(train, store):
    # Convert string types into integers
    store['StoreType'] = store['StoreType'].astype('category').cat.codes
    store['Assortment'] = store['Assortment'].astype('category').cat.codes
    train["StateHoliday"] = train["StateHoliday"].astype('category').cat.codes

    merged = pd.merge(train, store, on='Store', how='left')

    # remove NaNs
    NaN_replace = 0
    merged.fillna(NaN_replace, inplace=True)

    merged['Year'] = merged.Date.dt.year
```

```
    merged['Month'] = merged.Date.dt.month
    merged['Day'] = merged.Date.dt.day
    merged['Week'] = merged.Date.dt.isocalendar().week

    # Number of months that competition has existed for
    merged['MonthsCompetitionOpen'] = \
        12 * (merged['Year'] - merged['CompetitionOpenSinceYear']) + \
        (merged['Month'] - merged['CompetitionOpenSinceMonth'])
    merged.loc[merged['CompetitionOpenSinceYear'] ==
               NaN_replace, 'MonthsCompetitionOpen'] = NaN_replace


    merged['WeeksPromoOpen'] = \
    12 * (merged['Year'] - merged['Promo2SinceYear']) + \
    (merged['Date'].dt.isocalendar().week - merged['Promo2SinceWeek'])


    toInt = [
        'CompetitionOpenSinceMonth',
        'CompetitionOpenSinceYear',
        'Promo2SinceWeek',
        'Promo2SinceYear',
        'MonthsCompetitionOpen',
        'WeeksPromoOpen'
    ]
    merged[toInt] = merged[toInt].astype(int)

    return merged

med_store = train.groupby('Store')[['Sales', 'Customers', 'SalesPerCustomer']].median()
med_store.rename(columns=lambda x: 'Med' + x, inplace=True)

store = pd.merge(med_store.reset_index(), store, on='Store')
features = build_features(train, store)
features.head()

# %%
from sklearn.model_selection import train_test_split
X = [
    'Store',
    'Customers',
    'CompetitionDistance',

    'Promo',
    'Promo2',

#     'SchoolHoliday',
    'StateHoliday',
    'StoreType',
    'Assortment',
```

```
    'AvgSales',
    'AvgCustomers',
    'AvgSalesPerCustomer',

    'MedSales',
    'MedCustomers',
    'MedSalesPerCustomer',

    'DayOfWeek',
    'Week',
    'Day',
    'Month',
    'Year',

    'CompetitionOpenSinceMonth',
    'CompetitionOpenSinceYear',
    'Promo2SinceWeek',
    'Promo2SinceYear',

#     'MonthsCompetitionOpen',
#     'WeeksPromoOpen'
]
X_train, X_test, y_train, y_test = train_test_split(
    features[X], features['Sales'], test_size=0.15, random_state=10)

# %%
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer

# Error calculating function according to kaggle
def rmspe(y, y_hat):
    return np.sqrt(np.mean(((y - y_hat) / y) ** 2))

rmpse_scorer = make_scorer(rmspe, greater_is_better = False) # Loss function

def score(model, X_train, y_train, y_test, y_hat):
    score = cross_val_score(model, X_train, y_train, scoring=rmpse_scorer, cv=5)
    print('Mean', score.mean())
    print('Variance', score.var())
    print('RMSPE', rmspe(y_test, y_hat))

def plot_importance(model):
    k = list(zip(X, model.feature_importances_))
    k.sort(key=lambda tup: tup[1])

    labels, vals = zip(*k)

    plt.barh(np.arange(len(X)), vals, align='center')
    plt.yticks(np.arange(len(X)), labels)

# %%
```

```
from sklearn.tree import DecisionTreeRegressor
decision_tree = DecisionTreeRegressor()
decision_tree.fit(X_train, y_train)
plot_importance(decision_tree)

# %%
y_hat = decision_tree.predict(X_test)
score(decision_tree, X_train, y_train, y_test, y_hat)

# %%
from sklearn.ensemble import RandomForestRegressor
randomForest = RandomForestRegressor(n_estimators=25, n_jobs=-1, verbose=1)
randomForest.fit(X_train, y_train)
plot_importance(randomForest)

# %%
y_hat = randomForest.predict(X_test)
score(randomForest, X_train, y_train, y_test, y_hat)

# %%
import xgboost as xgb

def rmspe_xg(yhat, y):
    y = np.expm1(y.get_label())
    yhat = np.expm1(yhat)
    return "rmspe", rmspe(y,yhat)

xgboost_tree = xgb.XGBRegressor(
    n_jobs = -1,
    n_estimators = 1000,
    eta = 0.1,
    max_depth = 2,
    min_child_weight = 2,
    subsample = 0.8,
    colsample_bytree = 0.8,
    tree_method = 'exact',
    reg_alpha = 0.05,
    silent = 0,
    random_state = 1023
)
xgboost_tree.fit(X_train[X], np.log1p(y_train),
                 eval_set = [(X_train[X], np.log1p(y_train)), (X_test[X], np.log1p(y_test))],
                 eval_metric = rmspe_xg,
                 early_stopping_rounds = 300
                )

# %%
print("Note that this is not in percentage, thus not to scale of graphs above")
xgb.plot_importance(xgboost_tree)

# %%
```

```
def rmspe_exp(y, y_hat):
    return rmspe(np.expm1(y), np.expm1(y_hat))

rmpse_xg_scorer = make_scorer(rmspe_exp, greater_is_better = False) # Loss function

def score(model, X_train, y_train, y_test, y_hat):
    score = cross_val_score(model, X_train, y_train, scoring=rmpse_xg_scorer, cv=5)
    print('Mean', score.mean())
    print('Variance', score.var())
    print('RMSPE', rmspe(y_test, np.expm1(y_hat)))


y_hat = xgboost_tree.predict(X_test[X])
score(xgboost_tree, X_train[X], np.log1p(y_train), y_test, y_hat)

# %%
# Define a function to calculate both RMSPE and R2 score
def calculate_scores(model, X_train, y_train, X_test, y_test):
    # Calculate RMSPE
    scores_rmspe = cross_val_score(model, X_train, y_train, scoring=rmpse_scorer, cv=5)
    mean_rmspe = scores_rmspe.mean()

    # Calculate R2 score
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    r2 = r2_score(y_test, y_pred)

    return mean_rmspe, r2

# Calculate scores for Decision Tree
decision_tree_rmspe, decision_tree_r2 = calculate_scores(decision_tree, X_train, y_train, X_test, y_test)

# Calculate scores for Random Forest
random_forest_rmspe, random_forest_r2 = calculate_scores(randomForest, X_train, y_train, X_test, y_test)

# Calculate scores for XGBoost
xgboost_rmspe, xgboost_r2 = calculate_scores(xgboost_tree, X_train[X], np.log1p(y_train), X_test[X], y_test)

# Print out the scores
print("Decision Tree:")
print("Mean RMSPE:", decision_tree_rmspe)
print("R2 Score:", decision_tree_r2)

print("\nRandom Forest:")
print("Mean RMSPE:", random_forest_rmspe)
print("R2 Score:", random_forest_r2)

print("\nXGBoost:")
print("Mean RMSPE:", xgboost_rmspe)
print("R2 Score:", xgboost_r2)
```

```
# Visualize comparison of accuracies
import matplotlib.pyplot as plt

classifiers = ['Decision Tree', 'Random Forest', 'XGBoost']
rmspe_scores = [decision_tree_rmspe, random_forest_rmspe, xgboost_rmspe]
r2_scores = [decision_tree_r2, random_forest_r2, xgboost_r2]

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.bar(classifiers, rmspe_scores, color='b')
plt.xlabel('Classifiers')
plt.ylabel('Mean RMSPE')
plt.title('Comparison of Mean RMSPE')

plt.subplot(1, 2, 2)
plt.bar(classifiers, r2_scores, color='r')
plt.xlabel('Classifiers')
plt.ylabel('R2 Score')
plt.title('Comparison of R2 Score')

plt.tight_layout()
plt.show()


# %%
test = pd.read_csv("C:\\Users\\harsh\\Desktop\\ML project\\test_v2.csv", parse_dates=[2], dtype=types)
features = build_features(test, store)

y_hat = np.expm1(xgboost_tree.predict(features[X]))

df = pd.DataFrame({"Id": range(1, len(test) + 1), 'Sales': y_hat})
df.loc[test['Open'] == 0, 'Sales'] = 1
df.to_csv('salesoutput.csv', index=False)
```