Ensemble Learning-based rental apartment price prediction model using stacking technique

Submitted in partial fulfilment of the requirements for the degree of

Bachelor's of Technology
In
Computer Science and Engineering

By
K Mary Nikitha 18BCE0457
Mudit Jantwal 18BCE0622

Under the guidance of

Dr. Rajkumar S
VIT Vellore

SCHOOL OF COMPUTER SCIENCE & ENGINEERING

Winter Semester 2021-22

DECLARATION

I hereby declare that the thesis entitled ”**Ensemble Learning-based rental apartment price prediction model using stacking technique**" submitted by me, for the award of the degree of Bachelor of Technology in Programme to VIT is a record of bonafide work carried out by me under the supervision of Dr. Rajkumar S.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore
Date: 15/05/22

K Mary Nikitha
Mudit Jantwal

Signature of candidate

CERTIFICATE

This is to certify that the thesis entitled "**Ensemble Learning-based rental apartment price prediction model using stacking technique**" submitted by K Mary Nikitha (18BCE0457) and Mudit Jantwal (18BCE0622), SCOPE, VIT, for the award of the degree of Bachelor of Technology in Computer Science and Engineering, is a record of bonafide work carried out by them under my supervision during Fall Semester 2021-22, as per VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfills the requirements and regulations of the University and in my opinion, meets the necessary standards for submission.

Place: Vellore
Date: 15/05/22

Signature of Guide

Internal Examiner **External Examiner**

ACKNOWLEDGEMENTS

It is our privilege to express our sincerest regards to our project coordinator, **Prof. Rajkumar S**, for his valuable inputs, able guidance, encouragement, whole-hearted cooperation and constructive criticism throughout the duration of ourproject.

We deeply express our sincere thanks to our Head of Department **Dr. Prabhu Sevuga** for encouraging and allowing us to present the project on the topic "**Ensemble Learning-based rental apartment price prediction model using stacking technique**" at our department for the partial fulfillment of the requirementsleading to the award of B-Tech degree.

We take this opportunity to thank all our lecturers who have directly or indirectly helped our project. We pay our respects and love to our parents and all other family members and friends for their love and encouragement throughout our career. Last but not the least we express our thanks to our friends for their cooperation and support.

EXECUTIVE SUMMARY

Over the last few years, there has been such an unprecedented dependency on the internet and the social space has been constantly evolving with more networks available to marketers than ever before. Especially in this era of the COVID19 pandemic, with restrictions of various degrees being enforced all over the country, real estate is one such field, where people have resorted to using online listings to look and compare rental prices based on a number of features like locality, built-up area, number of rooms, bathrooms, etc.

The major focus of the project is to create an effective house prediction model based on a number of real-time factors using machine learning. In this project, online rental listings of the city of Hyderabad are used as a data source for mapping house rent. Data points were scraped from one of the popular Indian rental websites www.nobroker.in. With the collected information, models of rental market dynamics were developed and evaluated using regression and boosting algorithms such as AdaBoost, CatBoost, LightGBM, XGBoost, KRR, ENet, and Lasso regression. An ensemble machine learning algorithm of the best combination of the aforementioned algorithms was also implemented using the stacking technique. The results of these algorithms were compared using several performance metrics such as Coefficient of determination (R^2 score), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and accuracy in order to determine the most effective model. According to further examination of results, it is clear that the ensemble machine learning algorithm does outperform the others in terms of better accuracy and reduced errors.

As rent changes are also indicators of urban transformation and social phenomena, our results can be integrated with additional information to be used in various urban studies and can also serve as practical references for homeowners and renters.

LIST OF CONTENTS

CONTENTS

LIST OF FIGURES AND TABLES

LIST OF ABBREVIATIONS

- LightGBM - Light Gradient Boosting Machine
- AdaBoost - Adaptive Boosting
- CatBoost - Category Boosting
- XGBoost - Extreme Gradient Boosting
- KRR - Kernel Ridge Regression
- ENet - Elastic Net Regression
- ML - Machine Learning
- TOM - Time on the Market
- SVM - Support Vector Machine
- AI - Artificial Intelligence
- API - Application programming interface
- CPU - Central processing unit
- GPU - Graphics processing unit
- CNTK - Microsoft Cognitive Toolkit
- MSE - Mean Squared Error
- RMSE - Root Mean Squared Error

1. INTRODUCTION

1.1 Objective

The objective of the project is to anticipate real estate pricing based on previously collected, relevant information.[4] The dataset is obtained by scraping a popular Indian rental listing website with 18 variables including the number of rooms, bathrooms, amenities available, and size among others which are factors that can influence the price of a house. This data is pre-processed and trained using seven regression algorithms and boosting methods namely AdaBoost, CatBoost, LightGBM, XGBoost, KRR, ENet Lasso regression and Stacked Regressor - formed by the best combination of the aforementioned algorithms, which is an ensemble model used for obtaining better accuracy.

1.2 Motivation

The real estate industry is growing by the minute and hence advanced and accurate predictions of prices of listings are the need of the hour. The market is changing all the time, and today many software juggernauts are turning to artificial intelligence for improved decisions and resolving certain complex real-world problems using large amounts of data. Machine learning has now become interdisciplinary and can be incorporated into a variety of industries including real estate so that it can aid not only investors to increase their business throughput but also give individuals access to accurate information about properties within their financial limits. As a result, the ML models have the potential to take into account a wide range of criteria while analyzing patterns in order to provide efficient results in terms of complexity and accuracy[3]

1.3 Background

Rent is always a key variable when it comes to explaining urban phenomena, whether theoretically or empirically. Many real estate market participants, including investors, regulators, and policymakers, rely on accurate rental price forecasting. Rent predictions, for example, are critical in the property valuation used in discounted cash-flow models. The imputed rent is a significant component of the estimate of gross domestic income. For public housing policy, a thorough understanding of the structure and evolution of rents in local and national housing markets is also essential. Furthermore, a general idea of the features of a

piece of real estate that were most likely to affect the price during the pandemic can play a major role in the real estate market [1].The parties involved in the real estate industry often tend to depend on external valuation or various other internal approaches. Due to the large number of parameters involved, these metrics are unreliable and imperfect because they do not take into account all of the property's unique qualities during calculations.[2]

2. LITERATURE REVIEW

A few studies raised the question of what apartment attributes or variables are most likely to influence its price. Swarali M. Pathak et al [5] explored the correlation between house price and a number of attributes and came to the conclusion that Location and size of the apartment had strong links with the house price. On the other hand, Andrius, G et al [6] concluded that the TOM (Time on the market) variable was by far the most dominant and consistent variable for price forecasting.

Next, studies focusing on using machine learning for predicting the prices of houses and apartments were explored. The datasets usually have a lot of nonlinearity. Mu, J. et al [7], aiming at tackling this problem of nonlinearity, effectively concluded that SVM and LSSVM were the most effective models in terms of prediction effect and learning ability. Addressing the same issue, Čeh, M. et al [8] established that in complicated urban forms, the random forest method can detect and predict the fluctuation in apartment values more successfully than multiple regression.

Ming, Y. et al [9] in their research used RandomForestRegressor, XGBoost, and LightGBM to predict and fit 12 key features and a comparative analysis was done based on the results provided by all three models. According to the results, the XGBoost model had a greater prediction effect than that of the other two models, making a valuable addition to Chengdu's housing rent prediction study. Satish, G. N at al [10] in their paper proposed that machine learning could also be expandable to sectors like real estate to effectively predict housing prices. The machine algorithms used were Linear Regression, Lasso Regression, and Gradient Boosting. From the results, it could be concluded that the best performing algorithm is Gradient Boosting with an accuracy of 91.14% while the Lasso and Linear Regression algorithms have almost the same results with an accuracy of 76.14% and 76.15% each. For further work, they suggested incorporating parallel computation to decrease processing time.

Lu, S. et al [11] in their study used a hybrid regression technique for house prices prediction. For the implementation, the machine learning algorithms taken into consideration were Ridge, Lasso, and Gradient Boosting. They decided to implement a hybrid algorithm of Lasso and Gradient Boosting. Among all the combinations of the hybrid regression model, 65% Lasso and 35% Gradient Boosting performed the best, and hence they concluded that a

hybrid regression algorithm is better than a regression algorithm applied separately. Modi Maharshi et al [12] proposed to make use of ensemble machine learning methods for house price prediction. The dataset used in this application was from Kaggle.com which has 22 attributes and upon which a number of algorithms have been trained and tested. The algorithms used were Logistic Regression, SVM, Naive Bayes, Stochastic Gradient Descent, Extra Tree, K nearest Neighbors, and an ensemble machine learning algorithm that uses all the above-listed algorithms as weak learners. The performance of these algorithms was compared using various performance metrics, which concluded that the ensemble ML algorithm performed best overall followed by Extra Tree, Logistic Regression, and Naive Bayes.

Kansara, D. et al [13] in their paper used a stacked regressor formed by combining three regression algorithms namely Multiple Linear Regression, Random Forest Regressor and XGBoost. The stacked regressor takes into account each algorithm's weaknesses and strengths and balances them out to produce the final output, thus also improving the total accuracy. Because of rigorous cross-validation in action, shortcomings coming from one model are balanced out by strengths exhibited by another model, reducing error rates and balancing error rates.[14]

3. OVERVIEW OF WORK

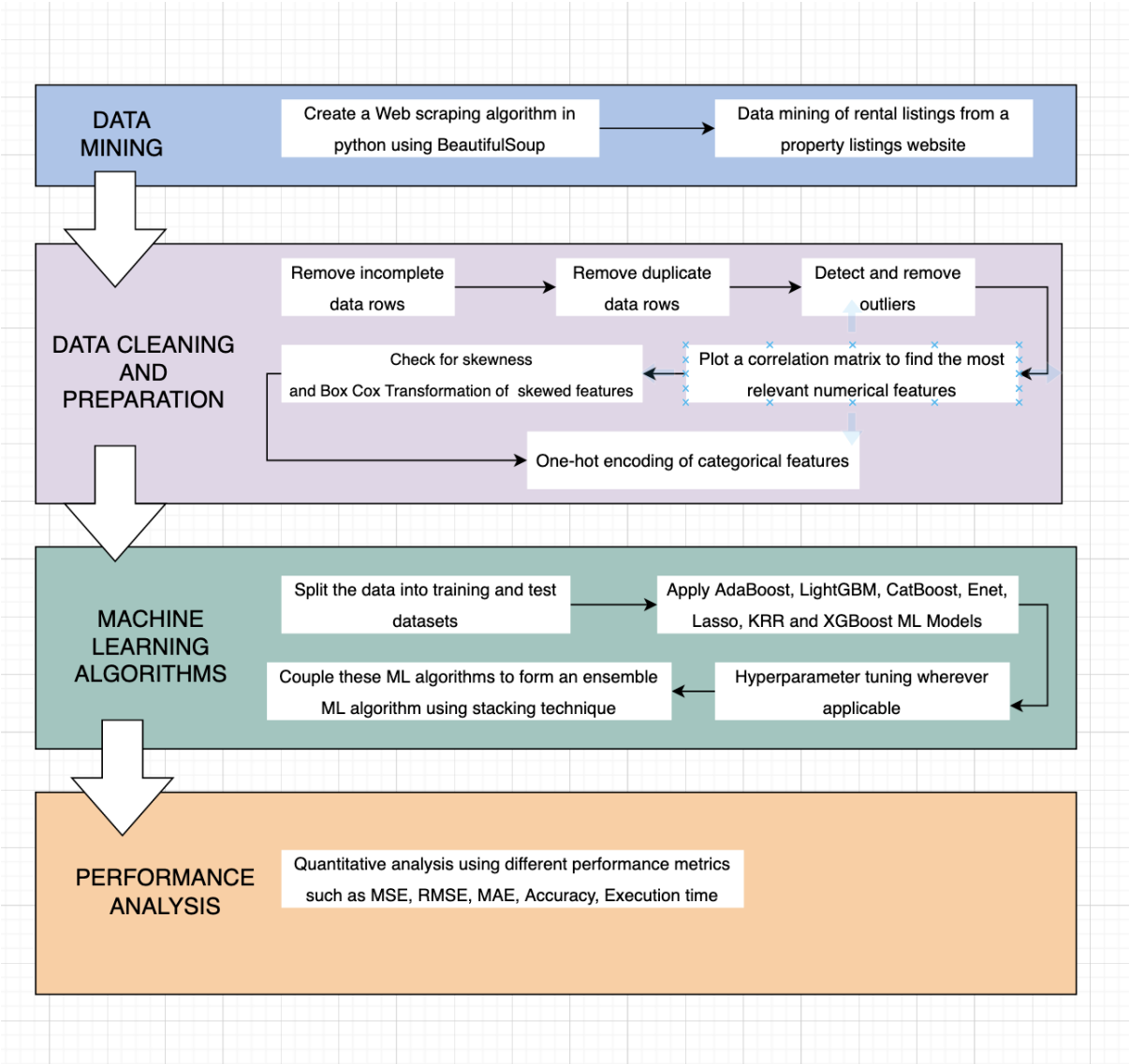


Fig 1: Research framework

4. TECHNICAL SPECIFICATION

4.1 Software Requirements: Code is written in python using its various libraries as shown below:
NumPy - NumPy is a popular Python toolkit for processing massive multi-dimensional arrays and matrices using a large number of high-level mathematical operations. It comes in handy for basic scientific computations in Machine Learning. Its linear algebra, Fourier transform, and random number skills are particularly valuable. NumPy is used internally by high-end libraries like TensorFlow to manipulate Tensors.

Scikit-learn - Scikit-learn is a popular machine learning library for traditional machine learning methods. It is based on two fundamental Python libraries, NumPy and SciPy. Most supervised and unsupervised learning algorithms are supported by Scikit-learn. Scikit-learn may also be used for data mining and analysis, making it an excellent tool for those new to machine learning.

SciPy -SciPy is a popular Python library for Machine Learning aficionados since it includes modules for optimization, linear algebra, integration, and statistics.

Pandas - Pandas is a popular Python data analysis toolkit. Pandas comes in helpful because it was designed expressly for data extraction and preparation. It provides high-level data structures as well as a comprehensive range of data analysis capabilities. Many built-in methods for searching, merging, and filtering data are available.

Matplotlib - Matplotlib is a well-known Python data visualisation package. It, like Pandas, has nothing to do with Machine Learning. It's very useful when a coder needs to see how data patterns are represented. It's a 2D plotting library for making graphs and plots in 2D space. Python's pyplot module makes plotting simple for programmers by allowing them to customise line styles, font characteristics, and axes formatting, among other things. It includes histograms, error charts, bar charts, and other graphs and plots for data visualisation.

TensorFlow - The Google Brain team developed TensorFlow, a prominent open-source toolkit for high-performance numerical computing. Tensorflow is a framework for defining and conducting tensor-based calculations, as the name implies. It has the ability to train and run deep neural networks, which may be utilised to create a variety of AI applications. In the realm of deep learning research and application, TensorFlow is frequently used.

Keras -Keras is a well-known Python Machine Learning package. It's a high-level neural network API that works with TensorFlow, CNTK, and Theano. It can run on both the CPU and the GPU. Keras makes building and designing a Neural Network a breeze for ML newbies. One of the greatest things about Keras is how simple and quick prototyping is.

5. SYSTEM DESIGN

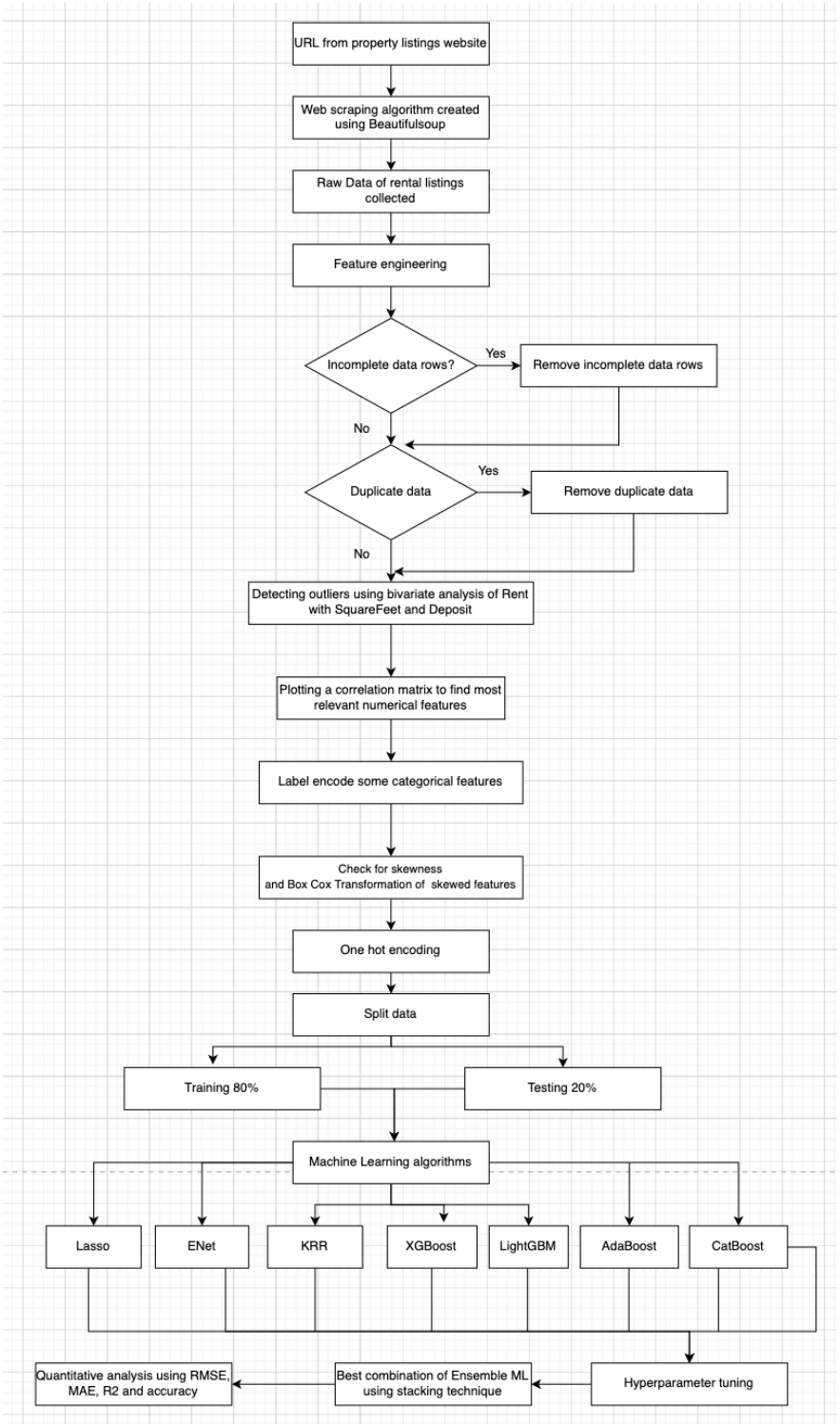


Fig 2: System Architecture

5.1 Methodology

The methodology of this project consisted of three steps: (1) data mining, (2) data cleaning and preparation (3) machine learning methods. For better understanding, the entire research framework is depicted as above.

5.1.1 Data mining

For data collection, we have made use of the web scraping technique. In this project, the BeautifulSoup package of the Python programming language was used to write an algorithm to collect data about apartment listings in the city of Hyderabad, from the popular rental website www.nobroker.in.The data was obtained from scraping over 150 pages of the website and ultimately stored in a CSV file.

5.1.2 Data Cleaning and Preparation:

After extensive data collection, a total of 737107 apartment listings with at most 18 features (such as number of rooms, number of bathrooms, square feet, age, total floors of the building, floor on which apartment is located, lift availability, deposit required, negotiability, maintenance availability, Longitude, Latitude, Gym availability, furnishing status, parking availability, facing direction, locality and rent) were collected.

The data cleaning and preparation process is as follows:

- The rows with duplicate data are dropped.
- The rows with incomplete information are dropped
- Outliers are detected and removed using bivariate analysis of Rent with square feet
- A correlation matrix is plotted to find the most relevant numerical features and the irrelevant columns are dropped
- Some categorical features are label encoded
- The skewness of all numerical features is checked and Box-Cox Transformation is applied to (highly) skewed features
- The remaining categorical features are one hot encoded

After the cleaning and preparation to use for subsequent stages, the resultant data frame had a shape of (4165, 1169)

5.1.3 Machine learning methods:

The ML process we have adopted had three distinct stages.

5.1.3.1 Applying ML algorithms individually

In the first stage, the dataset was split into 80% and 20% training and test datasets. The 7 regression algorithms and boosting methods applied are AdaBoost, CatBoost, LightGBM, Lasso Regression, Elastic Net Regression, Kernel Ridge Regression and XGBoost.

The **AdaBoost** algorithm combines a lot of weak learners, in a sequential manner. These weak learners are called “stumps” and are very short (one-level) decision trees. Each new model in the series is constructed by taking into account the mistakes made by the model preceding it. This is accomplished by each subsequent stump taking into account the previous stump's mistakes. [16]The key distinction between the two frameworks is that **XGBoost** grows trees depth-wise, whereas **LightGBM** grows trees leaf-wise. **Catboost** cultivates a well-balanced tree. In each level of such a tree, the feature-split pair with the lowest loss (as determined by a penalty function) is chosen and applied to all nodes in that level.[17]

Adding penalties to the loss function during training encourages simpler models with smaller coefficient values, which is an extension of the **conventional linear regression**. Regularized linear regression and penalized linear regression are terms used to describe these expansions.[18]

Lasso regression is when the L1 penalty, which is using the sum of absolute values, is incorporated into the least-squares linear regression.[18] If instead, the L2 norm, which is the sum of the squares of the weights is used, we get a procedure known as ridge regression. This combined with the kernel trick is what gives rise to the **Kernel ridge regression**[20]

Elastic net is a sort of regularized linear regression that incorporates both the aforementioned penalties, the L1 and L2 penalty functions [19].

5.1.3.2 Hyperparameter tuning:

After primarily fitting these algorithms to the data, hyperparameter tuning is applied to the algorithms wherever applicable. Hyperparameter tuning is the task of identifying the perfect

model architecture, with **hyperparameters** referring to the parameters that form the model architecture.

There are a number of hyperparameter tuning methods. We make use of the GridSearch method, one of the most popular tuning methods used. Using this method, a model is created for each conceivable combination of the hyperparameter values provided. Following that, each model is examined, with the best findings being considered as the parameters for the ideal architecture.

Hyperparameter tuning for Adaboost - For getting better scores we try to tune the hyperparameters – number of estimators and learning rate using GridsearchCV. After hyperparameter tuning the algorithm provides better scores

Hyperparameter tuning for Lasso - The main purpose of tuning the hyperparameters for Lasso is to find the appropriate value of alpha for our dataset. We defined a *search space* of possible alpha values in the range of 0.0001 to 1 and maximum iterations within which the hyperparameters will be chosen based on its effective minimization of error using the GridSearchCV method.

Hyperparameter tuning for ENet - The hyperparameter tuning of ENet is very similar to the Lasso regression. We defined a *search space* of possible alpha values in the range of 0.0001 to 1, maximum iterations, and l1_ratio within which the hyperparameters will be chosen based on its effective minimization of error using the GridSearchCV method.

5.1.3.3 Creating an ensemble model using stacking technique

In the second stage, we seek to implement a model ensembling technique called stacking. Ensemble learning is a machine learning paradigm in which several models (commonly referred to as "weak learners") are trained to handle the same problem and then integrated to improve results. The main idea is that by correctly combining weak models, we can get more accurate and/or resilient models. Learning several weak learners and then combining them by training a meta-model to output predictions based on the various predictions provided by these weak models is known as stacking. [21]

Due to its smoothing nature and its ability to highlight and balance out each of the models' strengths and weaknesses, the stacked model (also known as the 2nd-level model) often

outperforms each of the separate models. As a result, stacking works best when the base models are drastically different.

Steps involved in Ensemble stacking technique:

1. Divide the data for training into two folds
2. Choose L weak learners and fit them to first-fold data
3. For each of them, produce predictions for observations in the second fold
4. Fit the meta-model on the second fold using predictions made by the weak learners as inputs.

[21]

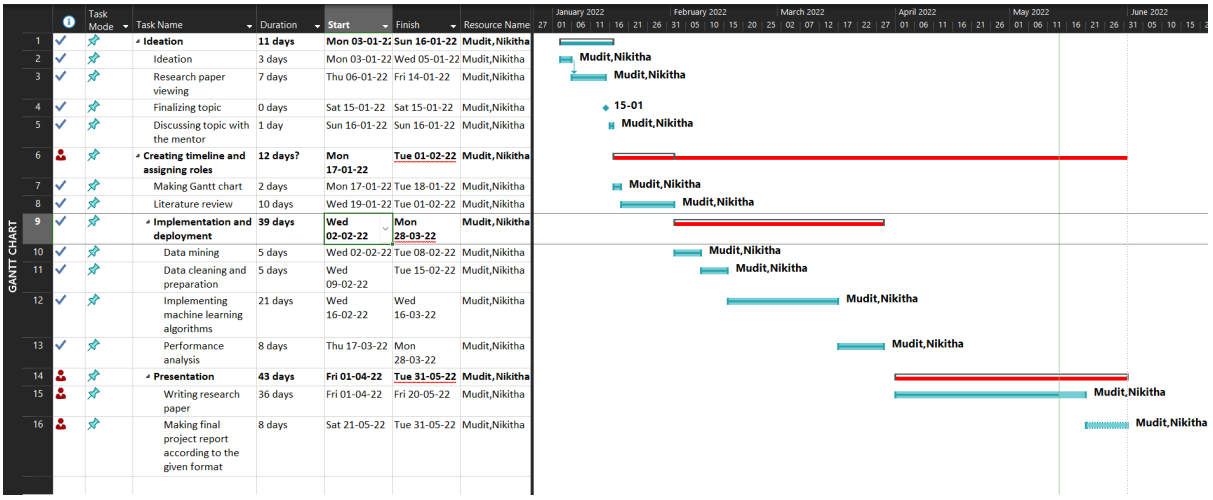
We implemented this technique for a number of combinations from the 7 algorithms. After evaluating the results, the combination of CatBoost, XGBoost and LightGBM was the one that performed the best.

5.2 Application:

This code is hosted on the cloud and can be accessible by anyone, no matter the location. Other developers of similar projects can use this code to make suitable conclusions about their personal datasets. An ideal dataset would consist of many real time variables such as square feet, total floors of the building, floor on which apartment is located, amenities availability, furnishing status, parking availability, locality and rent. As stated previously, our results can be integrated with additional information to be used in various urban studies and can also serve as practical references for homeowners and renters.

Using this dataset, the code can be tweaked accordingly to either predict the rent based on the other available real time factors or vice versa to predict any of the factors such as square feet, amenities availability, furnishing status or parking availability based on the rent and other factors.

6. GANTT CHART



	Task Mode	Task Name	Duration	Start	Finish	Resource Name
1		**Ideation**	**11 days**	**Mon 03-01-22**	**Sun 16-01-22**	**Mudit,Nikitha**
2		Ideation	3 days	Mon 03-01-22	Wed 05-01-22	Mudit,Nikitha
3		Research paper viewing	7 days	Thu 06-01-22	Fri 14-01-22	Mudit,Nikitha
4		Finalizing topic	0 days	Sat 15-01-22	Sat 15-01-22	Mudit,Nikitha
5		Discussing topic with the mentor	1 day	Sun 16-01-22	Sun 16-01-22	Mudit,Nikitha
6		**Creating timeline and assigning roles**	**12 days?**	**Mon 17-01-22**	**Tue 01-02-22**	**Mudit,Nikitha**
7		Making Gantt chart	2 days	Mon 17-01-22	Tue 18-01-22	Mudit,Nikitha
8		Literature review	10 days	Wed 19-01-22	Tue 01-02-22	Mudit,Nikitha
9		**Implementation and deployment**	**39 days**	**Wed 02-02-22**	**Mon 28-03-22**	**Mudit,Nikitha**
10		Data mining	5 days	Wed 02-02-22	Tue 08-02-22	Mudit,Nikitha
11		Data cleaning and preparation	5 days	Wed 09-02-22	Tue 15-02-22	Mudit,Nikitha
12		Implementing machine learning algorithms	21 days	Wed 16-02-22	Wed 16-03-22	Mudit,Nikitha
13		Performance analysis	8 days	Thu 17-03-22	Mon 28-03-22	Mudit,Nikitha
14		**Presentation**	**43 days**	**Fri 01-04-22**	**Tue 31-05-22**	**Mudit,Nikitha**
15		Writing research paper	36 days	Fri 01-04-22	Fri 20-05-22	Mudit,Nikitha
16		Making final project report according to the given format	8 days	Sat 21-05-22	Tue 31-05-22	Mudit,Nikitha

Fig 3: Gantt Chart

7. IMPLEMENTATION

AdaBoost:

AdaBoost

```
%%time
from sklearn.ensemble import AdaBoostRegressor
model_adb = AdaBoostRegressor()
# fit the model on all available data
model_adb.fit(X_train, y_train)
# make predictions
adb_pred = model_adb.predict(X_test)
# calculate the absolute errors
errors = abs(adb_pred - y_test)

# R2 Score
r2 = r2_score(y_test,adb_pred)
print('R2:',r2)

# Mean Absolute Error
print('MAE:', round(mean_absolute_error(y_test, adb_pred), 2))

# Mean Squared Error
mse = mean_squared_error(y_test, adb_pred)
print('MSE:', round(mse, 2))

# Calculate and display root mean squared error (RMSE)
rmse = np.sqrt(mse)
print('RMSE:', round(rmse, 2))

# Calculate and display root mean squared log error (RMSLE)
rmsle = np.log(rmse)
print('RMSLE:', round(rmsle, 2))

# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / y_test)

# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
R2: 0.8308555080435549
MAE: 0.76
MSE: 0.97
RMSE: 0.99
RMSLE: -0.01
Accuracy: 96.37 %.
Wall time: 5.83 s
```

Fig 4: AdaBoost Regressor implementation (without hyperparameter tuning)

```
%%time
model_adb = AdaBoostRegressor(learning_rate=0.01, n_estimators=50)
# fit the model on all available data
model_adb.fit(X_train, y_train)
# make predictions
adb_pred = model_adb.predict(X_test)
# Calculate the absolute errors
errors = abs(adb_pred - y_test)

# R2 Score
r2 = r2_score(y_test,adb_pred)
print('R2:',r2)

# Mean Absolute Error
print('MAE:', round(mean_absolute_error(y_test, adb_pred), 2))

# Mean Squared Error
mse = mean_squared_error(y_test, adb_pred)
print('MSE:', round(mse, 2))

# Calculate and display root mean squared error (RMSE)
rmse = np.sqrt(mse)
print('RMSE:', round(rmse, 2))

# Calculate and display root mean squared log error (RMSLE)
rmsle = np.log(rmse)
print('RMSLE:', round(rmsle, 2))

# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / y_test)

# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
R2: 0.8302128378175351
MAE: 0.69
MSE: 0.98
RMSE: 0.99
RMSLE: -0.01
Accuracy: 96.71 %.
Wall time: 6.8 s
```

Fig 5: AdaBoost Regressor implementation (with hyperparameter tuning)

CatBoost:

CatBoost

```
%%time
from catboost import CatBoostRegressor
model_cat = CatBoostRegressor()
# fit the model on all available data
model_cat.fit(X_train, y_train)
# make predictions
cat_pred = model_cat.predict(X_test)
# Calculate the absolute errors
errors = abs(cat_pred - y_test)

# R2 Score
r2 = r2_score(y_test,cat_pred)
print('R2:',r2)

# Mean Absolute Error
print('MAE:', round(mean_absolute_error(y_test, cat_pred), 2))

# Mean Squared Error
mse = mean_squared_error(y_test, cat_pred)
print('MSE:', round(mse, 2))

# Calculate and display root mean squared error (RMSE)
rmse = np.sqrt(mse)
print('RMSE:', round(rmse, 2))

# Calculate and display root mean squared log error (RMSLE)
rmsle = np.log(rmse)
print('RMSLE:', round(rmsle, 2))

# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / y_test)

# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
988:    learn: 0.5910778        total: 8.35s    remaining: 92.8ms
989:    learn: 0.5910399        total: 8.35s    remaining: 84.4ms
990:    learn: 0.5910034        total: 8.36s    remaining: 75.9ms
991:    learn: 0.5909662        total: 8.37s    remaining: 67.5ms
992:    learn: 0.5909288        total: 8.37s    remaining: 59ms
993:    learn: 0.5908921        total: 8.38s    remaining: 50.6ms
994:    learn: 0.5905328        total: 8.39s    remaining: 42.1ms
995:    learn: 0.5904995        total: 8.39s    remaining: 33.7ms
996:    learn: 0.5904635        total: 8.41s    remaining: 25.3ms
997:    learn: 0.5901101        total: 8.41s    remaining: 16.9ms
998:    learn: 0.5900739        total: 8.42s    remaining: 8.43ms
999:    learn: 0.5900359        total: 8.43s    remaining: 0us
R2: 0.9090784838067242
MAE: 0.5
MSE: 0.52
RMSE: 0.72
RMSLE: -0.32
Accuracy: 97.64 %.
Wall time: 9.18 s
```

Fig 6: CatBoost Regressor implementation

Elastic Net:

Elastic Net Regression

```
%%time
model_enet = make_pipeline(RobustScaler(), ElasticNet())
# fit the model on all available data
model_enet.fit(X_train, y_train)
# make predictions
enet_pred = model_enet.predict(X_test)
# Calculate the absolute errors
errors = abs(enet_pred - y_test)

# R2 Score
r2 = r2_score(y_test,enet_pred)
print('R2:',r2)

# Mean Absolute Error
print('MAE:', round(mean_absolute_error(y_test, enet_pred), 2))

# Mean Squared Error
mse = mean_squared_error(y_test, enet_pred)
print('MSE:', round(mse, 2))

# Calculate and display root mean squared error (RMSE)
rmse = np.sqrt(mse)
print('RMSE:', round(rmse, 2))

# Calculate and display root mean squared log error (RMSLE)
rmsle = np.log(rmse)
print('RMSLE:', round(rmsle, 2))

# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / y_test)

# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
R2: 0.674786929080126
MAE: 1.06
MSE: 1.87
RMSE: 1.37
RMSLE: 0.31
Accuracy: 94.97 %.
Wall time: 335 ms
```

Fig 7: Elastic Net Regression implementation (without hyperparameter tuning)

```
# Tuning model
elastic = ElasticNet()
param_grid = {'alpha': [0.0005, 0.001, 0.5, 1.0], 'max_iter' : [150, 170, 175], 'l1_ratio': [0.3, 0.4, 0.45, 0.9] }
elastic_grid = GridSearchCV(elastic, param_grid,  cv=5, verbose=True, return_train_score=True, n_jobs=-1)

elastic_grid.fit(X_train, y_train)

print (elastic_grid.best_score_, elastic_grid.best_params_)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
0.9021125975638358 {'alpha': 0.0005, 'l1_ratio': 0.3, 'max_iter': 175}
```

```
%%time
model_enet = make_pipeline(RobustScaler(), ElasticNet(alpha=0.0005, l1_ratio=0.3, max_iter=175))
# fit the model on all available data
model_enet.fit(X_train, y_train)
# make predictions
enet_pred = model_enet.predict(X_test)
# Calculate the absolute errors
errors = abs(enet_pred - y_test)

# R2 Score
r2 = r2_score(y_test,enet_pred)
print('R2:',r2)

# Mean Absolute Error
print('MAE:', round(mean_absolute_error(y_test, enet_pred), 2))

# Mean Squared Error
mse = mean_squared_error(y_test, enet_pred)
print('MSE:', round(mse, 2))

# Calculate and display root mean squared error (RMSE)
rmse = np.sqrt(mse)
print('RMSE:', round(rmse, 2))

# Calculate and display root mean squared log error (RMSLE)
rmsle = np.log(rmse)
print('RMSLE:', round(rmsle, 2))

# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / y_test)

# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
R2: 0.8999687807905447
MAE: 0.57
MSE: 0.58
RMSE: 0.76
RMSLE: -0.28
Accuracy: 97.31 %.
Wall time: 709 ms
```

Fig 8: Elastic Net Regression implementation (with hyperparameter tuning)

Kernel Ridge Regression:

Kernel Ridge Regression

```
%%time
model_krr = KernelRidge(alpha=0.6, kernel='polynomial', degree=2, coef0=2.5)
# fit the model on all available data
model_krr.fit(X_train, y_train)
# make predictions
krr_pred = model_krr.predict(X_test)
# Calculate the absolute errors
errors = abs(krr_pred - y_test)

# R2 Score
r2 = r2_score(y_test,krr_pred)
print('R2:',r2)

# Mean Absolute Error
print('MAE:', round(mean_absolute_error(y_test, krr_pred), 2))

# Mean Squared Error
mse = mean_squared_error(y_test, krr_pred)
print('MSE:', round(mse, 2))

# Calculate and display root mean squared error (RMSE)
rmse = np.sqrt(mse)
print('RMSE:', round(rmse, 2))

# Calculate and display root mean squared log error (RMSLE)
rmsle = np.log(rmse)
print('RMSLE:', round(rmsle, 2))

# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / y_test)

# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
R2: 0.8949612488427188
MAE: 0.58
MSE: 0.6
RMSE: 0.78
RMSLE: -0.25
Accuracy: 97.23 %.
Wall time: 811 ms
```

Fig 9: Kernel Ridge Regression implementation

Lasso Regression:

Lasso Regression

```
%%time
model_las = make_pipeline(RobustScaler(), Lasso())
# fit the model on all available data
model_las.fit(X_train, y_train)
# make predictions
las_pred = model_las.predict(X_test)
# Calculate the absolute errors
errors = abs(las_pred - y_test)

# R2 Score
r2 = r2_score(y_test,las_pred)
print('R2:',r2)

# Mean Absolute Error
print('MAE:', round(mean_absolute_error(y_test, las_pred), 2))

# Mean Squared Error
mse = mean_squared_error(y_test, las_pred)
print('MSE:', round(mse, 2))

# Calculate and display root mean squared error (RMSE)
rmse = np.sqrt(mse)
print('RMSE:', round(rmse, 2))

# Calculate and display root mean squared log error (RMSLE)
rmsle = np.log(rmse)
print('RMSLE:', round(rmsle, 2))

# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / y_test)

# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
R2: 0.6028697885378284
MAE: 1.19
MSE: 2.29
RMSE: 1.51
RMSLE: 0.41
Accuracy: 94.37 %.
Wall time: 372 ms
```

Fig 10: Lasso Regression implementation (without hyperparameter tuning)

```
# Tuning model
las = Lasso()
param_grid = { 'alpha': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 1], 'max_iter' : [30, 40, 45]}
las_grid = GridSearchCV(las, param_grid,  cv=5, verbose=True, return_train_score=True, n_jobs=-1)

las_grid.fit(X_train, y_train)

print (las_grid.best_score_, las_grid.best_params_)
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
0.9008485409429573 {'alpha': 0.0005, 'max_iter': 30}
```

```
%%time
# Model after tuning
model_las = make_pipeline(RobustScaler(), Lasso(alpha=0.0005, max_iter=30))
# fit the model on all available data
model_las.fit(X_train, y_train)
# make predictions
las_pred = model_las.predict(X_test)
# Calculate the absolute errors
errors = abs(las_pred - y_test)

# R2 Score
r2 = r2_score(y_test,las_pred)
print('R2:',r2)

# Mean Absolute Error
print('MAE:', round(mean_absolute_error(y_test, las_pred), 2))

# Mean Squared Error
mse = mean_squared_error(y_test, las_pred)
print('MSE:', round(mse, 2))

# Calculate and display root mean squared error (RMSE)
rmse = np.sqrt(mse)
print('RMSE:', round(rmse, 2))

# Calculate and display root mean squared log error (RMSLE)
rmsle = np.log(rmse)
print('RMSLE:', round(rmsle, 2))

# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / y_test)

# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
R2: 0.8987423815345592
MAE: 0.57
MSE: 0.58
RMSE: 0.76
RMSLE: -0.27
Accuracy: 97.3 %.
Wall time: 381 ms
```

Fig 11: Lasso Regression implementation (with hyperparameter tuning)

LightGBM

LightGBM

```
%%time
from lightgbm import LGBMRegressor
model_lgbm = LGBMRegressor()
# fit the model on all available data
model_lgbm.fit(X_train, y_train)
# make predictions
lgbm_pred = model_lgbm.predict(X_test)
# Calculate the absolute errors
errors = abs(lgbm_pred - y_test)

# R2 Score
r2 = r2_score(y_test,lgbm_pred)
print('R2:',r2)

# Mean Absolute Error
print('MAE:', round(mean_absolute_error(y_test, lgbm_pred), 2))

# Mean Squared Error
mse = mean_squared_error(y_test, lgbm_pred)
print('MSE:', round(mse, 2))

# Calculate and display root mean squared error (RMSE)
rmse = np.sqrt(mse)
print('RMSE:', round(rmse, 2))

# Calculate and display root mean squared log error (RMSLE)
rmsle = np.log(rmse)
print('RMSLE:', round(rmsle, 2))

# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / y_test)

# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
R2: 0.9067523765875072
MAE: 0.49
MSE: 0.54
RMSE: 0.73
RMSLE: -0.31
Accuracy: 97.69 %.
Wall time: 290 ms
```

Fig 12: LightGBM Regressor implementation

XGBoost:

XGBoost

```
%%time
model_xgb = XGBRegressor()
# fit the model on all available data
model_xgb.fit(X_train, y_train)
# make predictions
xgb_pred = model_xgb.predict(X_test)
# Calculate the absolute errors
errors = abs(xgb_pred - y_test)

# R2 Score
r2 = r2_score(y_test,xgb_pred)
print('R2:',r2)

# Mean Absolute Error
print('MAE:', round(mean_absolute_error(y_test, xgb_pred), 2))

# Mean Squared Error
mse = mean_squared_error(y_test, xgb_pred)
print('MSE:', round(mse, 2))

# Calculate and display root mean squared error (RMSE)
rmse = np.sqrt(mse)
print('RMSE:', round(rmse, 2))

# Calculate and display root mean squared log error (RMSLE)
rmsle = np.log(rmse)
print('RMSLE:', round(rmsle, 2))

# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / y_test)

# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
R2: 0.9078110522707654
MAE: 0.5
MSE: 0.53
RMSE: 0.73
RMSLE: -0.32
Accuracy: 97.64 %.
Wall time: 3.92 s
```

Fig 13: XGBoost Regressor implementation

Ensemble algorithm using stacking technique (CatBoost + LightGBM + XGBoost)

```
%%time
# define the base models
level0 = list()
level0.append(('Cat',model_cat))
level0.append(('XGB', model_xgb))
level0.append(('LGBM', model_lgbm))
# define meta learner model
level1 = LinearRegression()
model = StackingRegressor(estimators=level0, final_estimator=level1, cv=5)
# fit the model on all available data
model.fit(X_train, y_train)
# make predictions
stack_pred = model.predict(X_test)
# Calculate the absolute errors
errors = abs(stack_pred - y_test)

# R2 Score
r2 = r2_score(y_test,stack_pred)
print('R2:',r2)

# Mean Absolute Error
print('MAE:', round(mean_absolute_error(y_test, stack_pred), 2))

# Mean Squared Error
print('MSE:', round(mean_squared_error(y_test, stack_pred), 2))

# Calculate and display root mean squared error (RMSE)
rmse = np.sqrt(mse)
print('RMSE:', round(rmse, 2))

# Calculate and display root mean squared log error (RMSLE)
rmsle = np.log(rmse)
print('RMSLE:', round(rmsle, 2))

# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / y_test)

# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
988:    learn: 0.5647325        total: 5.28s    remaining: 58.7ms
989:    learn: 0.5646933        total: 5.29s    remaining: 53.4ms
990:    learn: 0.5644181        total: 5.29s    remaining: 48.1ms
991:    learn: 0.5643787        total: 5.3s     remaining: 42.7ms
992:    learn: 0.5641521        total: 5.3s     remaining: 37.4ms
993:    learn: 0.5640588        total: 5.31s    remaining: 32ms
994:    learn: 0.5640196        total: 5.31s    remaining: 26.7ms
995:    learn: 0.5639806        total: 5.32s    remaining: 21.4ms
996:    learn: 0.5637760        total: 5.32s    remaining: 16ms
997:    learn: 0.5637371        total: 5.33s    remaining: 10.7ms
998:    learn: 0.5636982        total: 5.33s    remaining: 5.34ms
999:    learn: 0.5636039        total: 5.34s    remaining: 0us
R2: 0.9126707853133492
MAE: 0.48
MSE: 0.5
RMSE: 0.73
RMSLE: -0.32
Accuracy: 97.72 %.
Wall time: 58.9 s
```

Fig 14: Ensemble algorithm using stacking technique implementation

Comparison of individual algorithms vs ensemble algorithm using BoxPlot

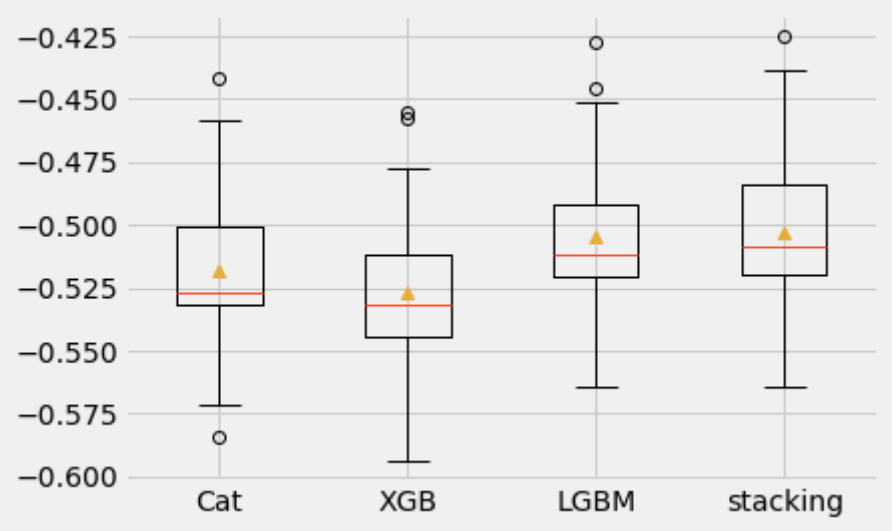


Fig 15: Comparison of individual algorithms vs ensemble algorithm using BoxPlot

8. RESULTS AND DISCUSSION

Seven algorithms are used in the prediction to determine which model has the best accuracy and so construct a system for speedier prediction. The following metrics are used to evaluate the regression model: **Coefficient of determination (R^2 score)**, **Mean Squared Error (**MSE), **Root Mean Squared Error (**RMSE), **Mean Absolute Error (**MAE), and **Accuracy.**

The coefficient of determination, Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Accuracy of the test datasets are presented below.

Algorithm	R^2	MAE	MSE	RMSE	Accuracy (%)
AdaBoost	0.830	0.69	0.98	0.99	96.71
LightGBM	0.906	0.49	0.54	0.73	97.69
CatBoost	0.909	0.5	0.52	0.72	97.64
Lasso Regression	0.898	0.57	0.58	0.76	97.3
Kernel Ridge Regression	0.894	0.58	0.6	0.78	97.23
ElasticNet Regression	0.899	0.57	0.58	0.76	97.31
XGBoost	0.907	0.5	0.53	0.73	97.64
Stacking algorithm (Catboost + LightGBM + XGBoost)	0.912	0.48	0.5	0.73	97.72

Table 1: Results of trained model on testing data

A higher value of R^2 is desirable as it indicates better results. Hence, the stacking algorithm performs the best with the lowest R^2 value of 0.912, closely followed by XGBoost and CatBoost.

A low MAE value indicates better performance of the model. Hence Stacking algorithm performs the best, closely followed by LightGBM, the CatBoost algorithm

For MSE, a lower value is better and 0 indicates a perfect model. Hence, the stacking algorithm performs the best with the lowest MSE value of 0.5.

A lower value of RMSE is desirable as it indicates better results. Hence CatBoost performs the best, closely followed by LightGBM, XGBoost and the stacking algorithm

The greater the accuracy, the better the model. Hence Stacking algorithm performs the best with an accuracy of 97.72%

In conclusion, taking into consideration the seven individual algorithms implemented, it can be seen that all algorithms have an accuracy of above 95%, with the best being LightGBM closely followed by Catboost and XGboost. Coincidentally, these algorithms proved to be the best combination to be used in the ensemble machine learning algorithm using the stacking technique, giving an increased accuracy of 97.72%. Another observation is that AdaBoost has the worst performance in terms of MSE, RMSE, and MAE but the stacking algorithm outperforms all the others in terms of error metrics having an MSE of 0.5, RMSE of 0.73 and MAE of 0.4 and R^2 score of 0.912

Model	Execution Time
AdaBoost	6.8 s
LightGBM	290 ms
CatBoost	9.18 s
Lasso Regression	381 ms
Kernel Ridge Regression	811 ms
ElasticNet Regression	709 ms
XGBoost	3.92 s
Stacking algorithm	58.9 s

Table 2: Summary of models' execution times

Regarding the model's performance, there are significant differences in the model execution times. While LightGBM, Lasso, KRR, and Enet have considerably short run times while the other models could take longer, which are indicated in the table above. It is also underlined that the stacking algorithm spends more training time than any of the individual algorithms. Therefore, there is a trade-off between modes' runtime and prediction accuracy.

9. CONCLUSION

In summary, this paper seeks to implement various machine learning models and boosting methods for house price prediction which has a number of applications in the real estate industry. Firstly, the original data which is scraped from a rental listings website is prepared and transformed into a cleaned dataset ready for analysis using various techniques. Seven different machine learning algorithms and boosting methods are applied and evaluated. In addition, the best combination of the aforementioned algorithms using stacking technique is sought after. From the results, it is concluded that the combination of XGBoost, CatBoost and Light GBM provides the most optimal solution in terms of accuracy but its runtime could be an area of improvement. Therefore, it could be used for further deployment.

10. REFERENCES:

[1] Andrius, G., Vaida, P., & Alina, S. (2021). Predictive analytics using Big Data for the real estate market during the COVID-19 pandemic. *Journal of Big Data*, *8*(1).

[2] The Potential of Machine Learning Real Estate Valuation Models (2018, August 12), Ershad Chagani, Cornell Blog

[3] Reddy, T., Sanghvi, J., Vora, D., & Kanani, P. (2018). Wanderlust: A Personalized Travel Itinerary Recommender. *International Journal Of Engineering Development And Research IJEDR*, *6*(3), 78-83.

[4] Machine learning algorithms in human language (2018,August,13) Datakeen [Online] Available: https://www.datakeen.co/en/8-machine-learning-algorithms-explained-in-human-language/

[5] Andrius, G., Vaida, P., & Alina, S. (2021). Predictive analytics using Big Data for the real estate market during the COVID-19 pandemic. *Journal of Big Data*, *8*(1).

[6] Swarali M. Pathak , Archana K. Chaudhari, 2021, Comparison of Machine Learning Algorithms for House Price Prediction using Real Time Data, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) Volume 10, Issue 12 (December 2021)

[7] Mu, J., Wu, F., & Zhang, A. (2014, August). Housing value forecasting based on machine learning methods. In *Abstract and Applied Analysis* (Vol. 2014). Hindawi.

[8] Čeh, M., Kilibarda, M., Lisec, A., & Bajat, B. (2018). Estimating the performance of random forest versus multiple regression for predicting prices of the apartments. *ISPRS international journal of geo-information*, *7*(5), 168.

[9] Ming, Y., Zhang, J., Qi, J., Liao, T., Wang, M., & Zhang, L. (2020, September). Prediction and analysis of chengdu housing rent based on xgboost algorithm. In *Proceedings of the 2020 3rd International Conference on Big Data Technologies* (pp. 1-5).

[10] Satish, G. N., Raghavendran, C. V., Rao, M. S., & Srinivasulu, C. (2019). House price prediction using machine learning. *Journal of Innovative Technology and Exploring Engineering*, *8*(9), 717-722.

[11] Lu, S., Li, Z., Qin, Z., Yang, X., & Goh, R. S. M. (2017, December). A hybrid regression technique for house prices prediction. In *2017 IEEE international conference on industrial engineering and engineering management (IEEM)* (pp. 319-323). IEEE.

[12] Modi Maharshi, Sharma Ayush and Madhavan Dr. P. 2020 Applied Research on House Price Prediction Using Diverse Machine Learning Techniques *International Journal of Scientific & Technology Research* **9** APRIL

[13] Kansara, D., Singh, R., Sanghvi, D., & Kanani, P. (2018). Improving accuracy of real estate valuation using stacked regression. *Int. J. Eng. Dev. Res.(IJEDR)*, *6*(3), 571-577.

[14] Güneş, F., Wolfinger, R., & Tan, P. Y. (2017, April). Stacked ensemble models for improved prediction accuracy. In *Proc. Static Anal. Symp.* (pp. 1-19).

[15] https://machinelearningmastery.com/adaboost-ensemble-in-python/ (Accessed in April 2022)

[16] https://github.com/bhweshgaur/Health-Insurance-Cross-Sell-Prediction/ (Accessed in April 2022)

[17] https://neptune.ai/blog/xgboost-vs-lightgbm (Accessed in April 2022)

[18https://machinelearningmastery.com/ridge-regression-with-python/ (Accessed in April 2022)

[19] https://carvadia.com/what-is-partitioned-regression/ (Accessed in April 2022)

[20] https://www.mathworks.com/matlabcentral/fileexchange/63122-kernel-ridge-regression/ (Accessed in April 2022)

[21]https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205 (Accessed in April 2022)

[22] https://github.com/dhruv-mehta2604/housePricePrediction/ (Accessed in April 2022)

[23]https://towardsdatascience.com/regression-an-explanation-of-regression-metrics-and-what-can-go-wrong-a39a9793d914 (Accessed in April 2022)

[24] https://sonalsart.com/how-do-you-evaluate-a-regression-model// (Accessed in April 2022)

[25]https://manish-ks.medium.com/advantages-and-disadvantages-of-linear-regression-its-assumptions-evaluation-and-implementation-61437fc551ad/ (Accessed in April 2022)

[26]https://medium.com/@amanbamrah/how-to-evaluate-the-accuracy-of-regression-results-b38e5512afd3 (Accessed in April 2022)