Project Report: House Price Prediction

ECS 308: Data Science and Machine Learning

Under Guidance of: Dr Kushal Shah

Name: Ajay Choudhury **Roll no.:** 18018 **Date:** 4th May 2021

Objective: To predict prices of houses based on certain given features like the crime rate in the locality, number of rooms, etc. using machine learning algorithms.

Group Members and Contribution:

1. Ajay Choudhury (18018).
    a. Linear Regression.
    b. Decision Tree Regression.
    c. Support Vector Regression.
    d. Random Forest Regression.
    e. Unsupervised Learning Algorithm(KMeans Cluster).
2. Raj Pritam Gupta (18381).
    a. Logistic Regression Classification.
    b. Feedforward Neural Network.

Brief Description of Dataset:

The dataset used here is the Boston Housing Dataset from the UCI repository. The dataset can be easily downloaded from its official site, or directly in .csv format from here.

We have slightly modified the dataset for logistic regression classifier, the modified dataset can be downloaded from here.

Original dataset: The original dataset("data.csv") consists of 506 instances. It contains 14 attributes in total among which 13 are continuous attributes (including the 'class' attribute "MEDV") and 1 is a binary attribute. There are no missing attribute values in the dataset.

The information about the attributes are:

1. CRIM:	per capita crime rate by town.
2. ZN:	proportion of residential land zoned for lots over 25,000 sq. ft.
3. INDUS:	proportion of non-retail business acres per town.
4. CHAS:	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).
5. NOX:	nitric oxides concentration (parts per 10 million).
6. RM:	average number of rooms per dwelling.
7. AGE:	proportion of owner-occupied units built before 1940.
8. DIS:	weighted distances to five Boston employment centres.
9. RAD:	index of accessibility to radial highways.
10. TAX:	full-value property-tax rate per $10,000.
11. PTRATIO:	pupil-teacher ratio by town.
12. B:	1000(Bk - 0.63)2 where Bk is the proportion of blacks by town.

13. LSTAT:	percentage of the lower status of the population.
14. MEDV:	median value of owner-occupied homes in $1000's.

Modified dataset for Logistic Regression analysis: The modified dataset(“data_log.csv”) is the same as the original dataset except for the class attribute “MEDV” which is a binary attribute here.

The values of “MEDV” are either 0 if the price of the house is below $30,000 and 1 if the price of the house is above $30,000.

A. Regression Models

Data Analysis for Regression Models:

Understanding each aspect of the available data is very important before proceeding to build any machine learning model upon the data.

First of all, we have imported the dataset into the Pandas Dataframe.

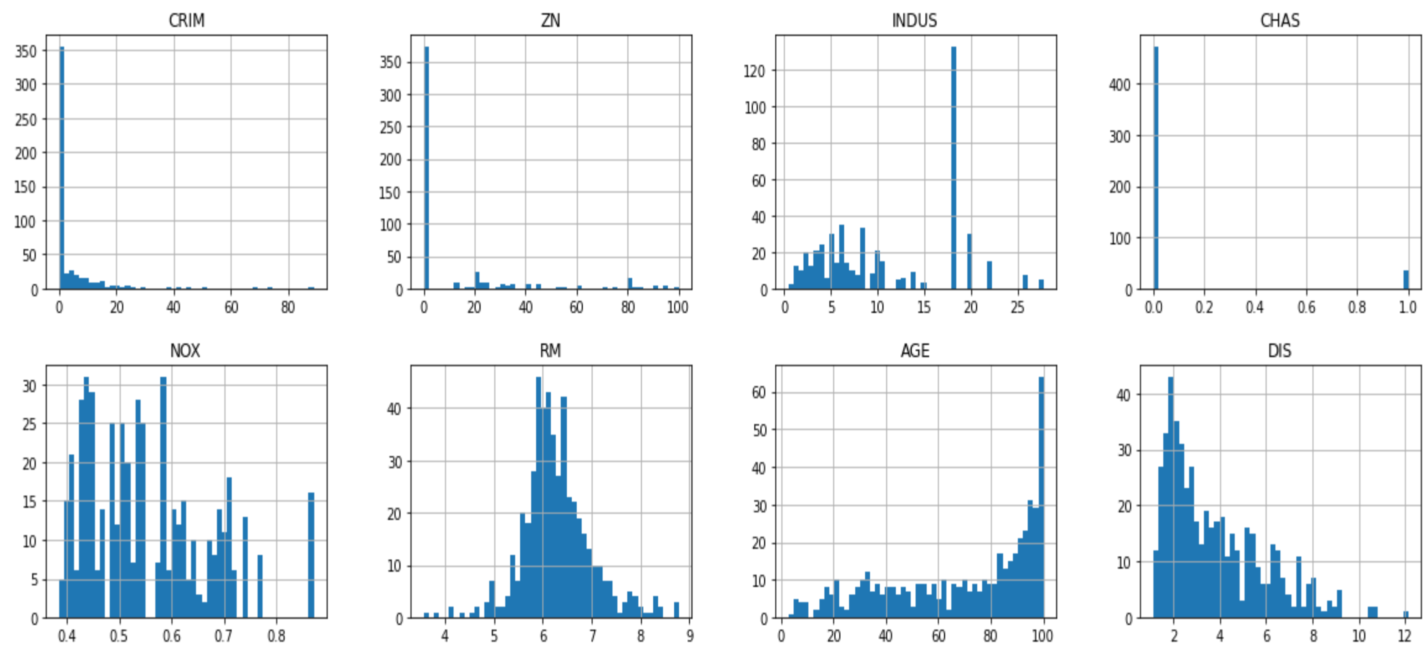
```
housing = pd.read_csv("data.csv")
```

Next, to see the data in a visual form, we have used the matplotlib library to plot a histogram for each attribute in the dataset.

```
# For plotting histogram of each feature in the dataset

housing.hist(bins=50, figsize=(20, 15))
plt.show()
```

The output produced is:



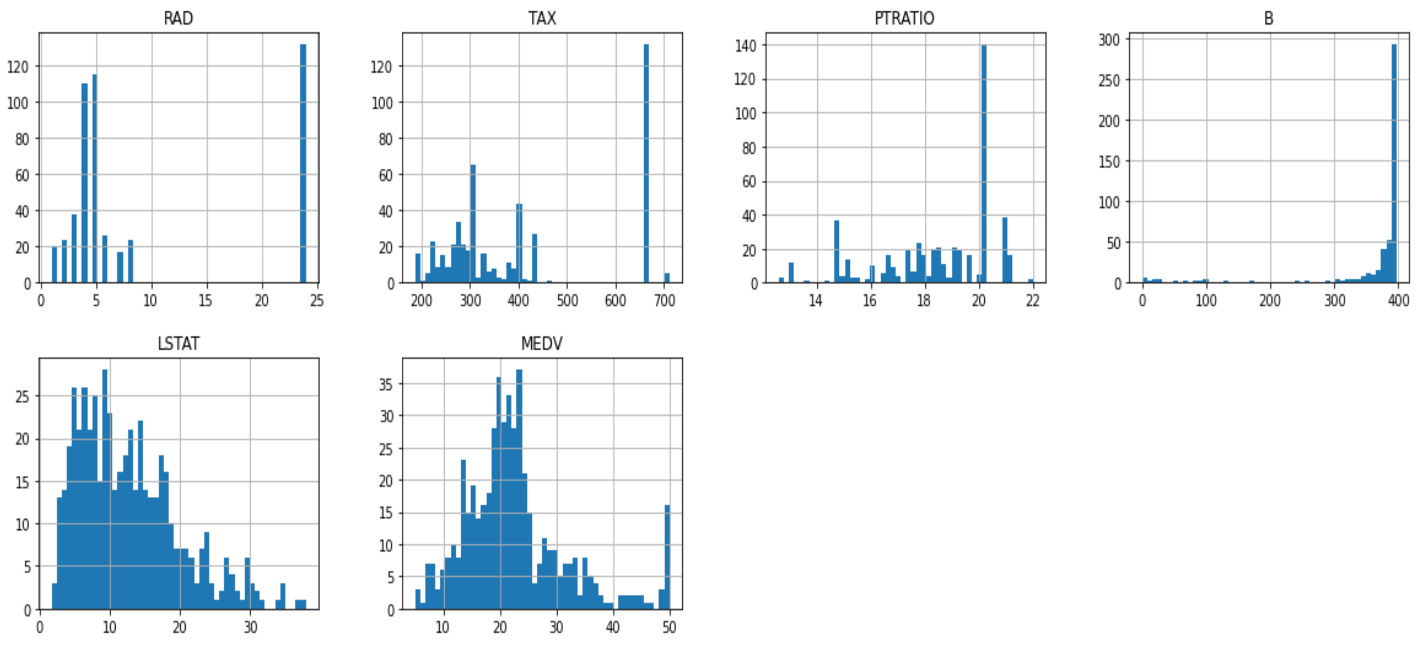


Fig.1: Histograms for each attribute of the dataset.

Now, while we create the training and test dataset from the given dataset, we need to look into two important facts here. First, our model should have a fixed training dataset and it should not shuffle the data from the whole dataset. If it does so, the model will be able to get a view of the whole dataset and it will end up getting overfitted to the given dataset.

Second, when we look into the attribute 'CHAS' (which is a binary attribute), we get to know that there are very few cases when the value is 1 and in most of the cases it is 0. So, if we perform a simple train-test split then there is a possibility that the training set gets all 0 values for 'CHAS' and all the values consisting of 1 goes to the test set. As a result, we will end up building an inefficient model.

To avoid this situation, we have applied Stratified Sampling to the attribute 'CHAS' and we have split the dataset into training and test dataset in the ratio of 80:20 percentage. StratifiedShuffleSplit from the sklearn library divides the 0 and 1 valued instances to both the training and test dataset in an equal ratio such that the model works efficiently.

```
# Splitting the dataset for Training and Testing

from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing['CHAS']):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

Now, we have separated the class attribute from the rest of the attributes.

```
housing = strat_train_set.drop("MEDV", axis=1)
housing_labels = strat_train_set["MEDV"].copy()
```

Again, to understand the importance of the attributes we have also viewed the correlations of each attribute with the predicted price of the house. This is important in case some values are missing and if we decide to drop the row or whole attribute. Correlation can give us a better picture of whether to drop the attribute or rows of data or just put some generated value in the missing place.

Larger correlations (both positive or negative) implies that the attribute is much important for the model and we should not drop them or their rows/values whereas we can do so for attributes with small values of correlation.

```
# Looking for correlations of features with the final price
corr_matrix = housing.corr()
print(corr_matrix['MEDV'].sort_values(ascending=False))

# Plotting the correlations of some important features with each other
from pandas.plotting import scatter_matrix
attributes = ["MEDV", "RM", "ZN", "LSTAT"]
scatter_matrix(housing[attributes], figsize = (12,8))
plt.show()
```

The output is:

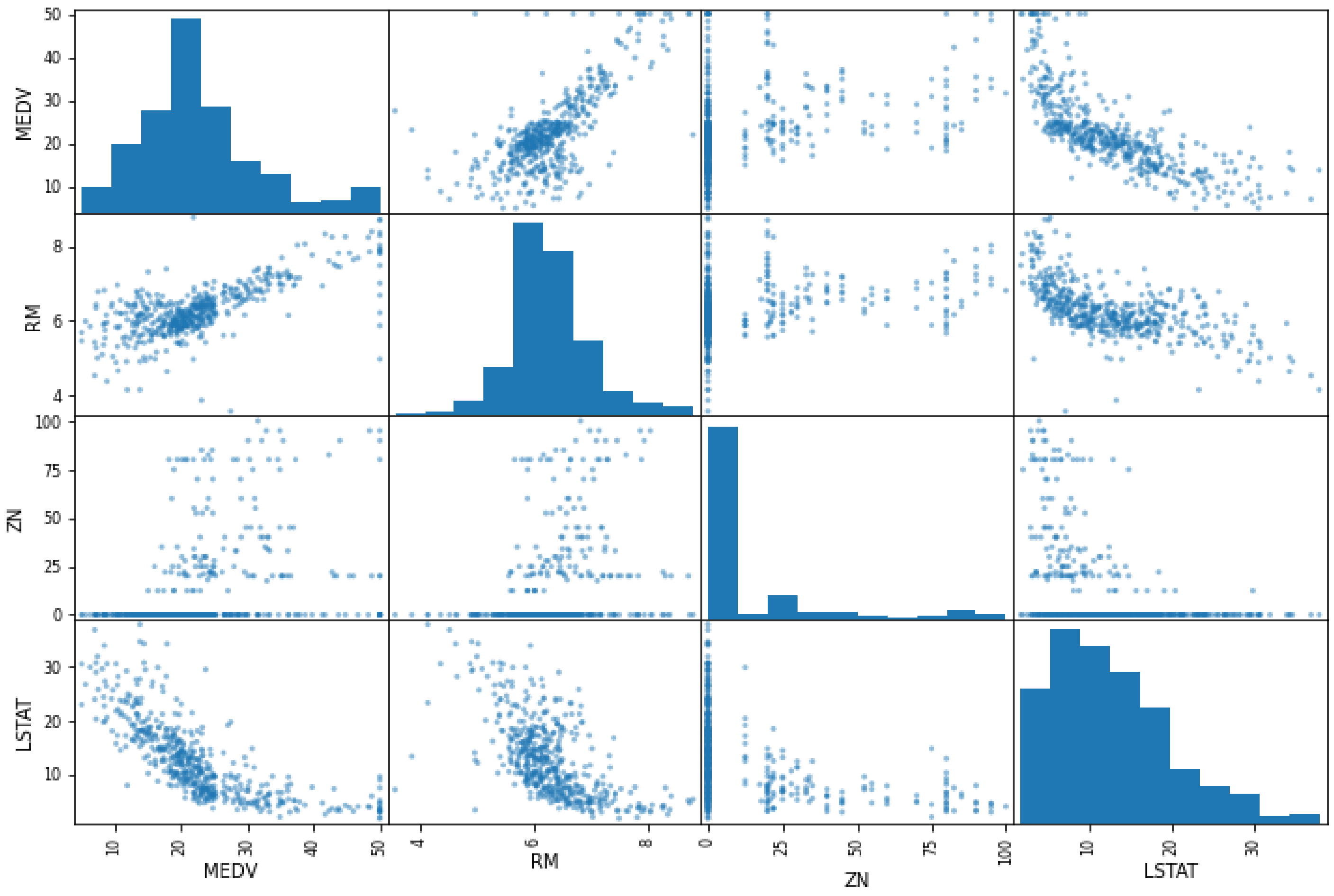


Fig.2: Correlations of some important attributes.

In case, some of the values in the dataset are missing then we will replace them with the median of the values of that attribute using the estimator Simple Imputer from the sci-kit learn library.

Also, we will use standardisation for feature scaling here. What standardisation does is it replaces every value in the dataset with (value - mean) / std. Now, we will apply the Imputer and Standardisation in our pipeline of model building.

```
# Creating the Pipeline
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
my_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('std_scaler', StandardScaler()),
])
```

We are done with data analysis now and now we just need to pass our training set through our pipeline to fit and transform the data accordingly.

```
# Passing the training dataset through the Pipeline
housing_num_tr = my_pipeline.fit_transform(housing)
```

Now, we just need to select specific models to run on our training dataset for which we will use the sklearn library. The models used are discussed in details below.

Algorithms and Machine Learning Models:

1. Linear Regression

The linear regression model is a regression model which works with a continuous dependent variable and an independent variable. The linear regression model tries to establish and fit a linear relationship between the independent variables(features) and the dependent variable(class attribute).

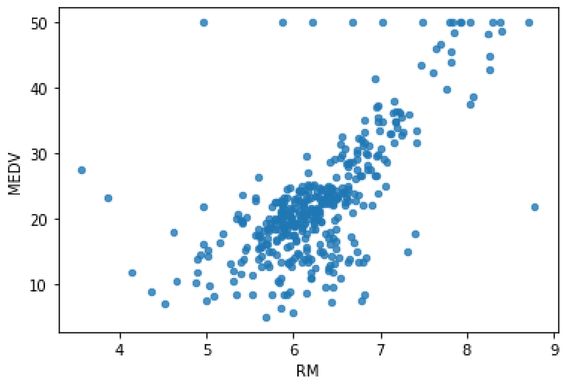


Fig.3: Graph of the linear relation between RM and MEDV.

Here, there are 13 independent variables and one dependent variable(MEDV) in the dataset which gives rise to multiple linear regression. In the correlations plot above, we have seen how the features are related to the final price of the houses. Features with high positive or negative correlations with the class attribute will have greater regression coefficients and hence will affect the final predicted price heavily.

After applying the linear regression model on the dataset we have calculated the root mean squared error and standard deviation by cross-validation evaluation and then calculated the RMSE on the test dataset.

On the training dataset by Cross-validation:

Root Mean Squared Error: 5.030437102767304
Standard deviation: 1.0607661158294832

On the test dataset:

Root Mean Squared Error: 4.143819554319329

Sample Prediction:

Predicted price: 23.95433455
Original price: 21.9

After data analysis we have applied the following code for Linear regression:

```
# Selecting the model
from sklearn.linear_model import LinearRegression

model = LinearRegression()

model.fit(housing_num_tr, housing_labels)

# Testing the model on Test dataset
X_test = strat_test_set.drop("MEDV", axis=1)
Y_test = strat_test_set["MEDV"].copy()
X_test_prepared = my_pipeline.transform(X_test)
final_predictions = model.predict(X_test_prepared)
final_mse = mean_squared_error(Y_test, final_predictions)
final_rmse = np.sqrt(final_mse)

# Printing the final predictions on test dataset
print("Predicted Prices on Test dataset:")
print(final_predictions, "\n")
print("Original Prices:")
print(list(Y_test),"\n")
print("The RMSE of the model on Test dataset:")
print(final_rmse, "\n")
```

2. Decision Tree Regression

A decision tree is a supervised machine learning model which is used to predict a target variable by learning decision rules from associated features. It is a non-linear regression technique. In the regression tree, the value obtained by a terminal node in the dataset is the mean response of observation falling in that region. Hence, if an unseen datapoint falls in that region we can make its prediction with an average or mean value.

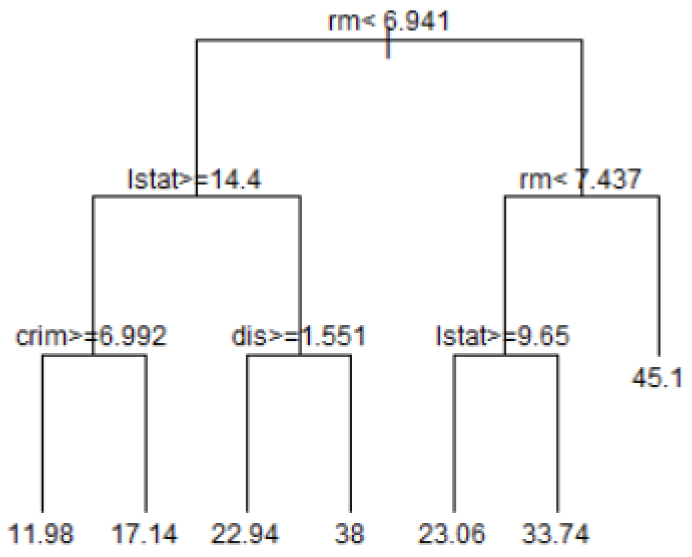


Fig.4: Understanding Decision Tree Regression.

The decision tree regression technique is used to predict continuous target variables. The decision tree works in 3 major steps:

a. Position the best attribute of the dataset at the root of the tree.
b. Split the dataset into subsets such that each subset contains homogeneous data.
c. Repeats the previous two steps until leaf nodes are found in all branches of the tree.

After applying the decision tree regression model on the dataset we have calculated the root mean squared error and standard deviation by cross-validation evaluation and then calculated the RMSE on the test dataset.

On the training dataset by Cross-validation:

Root Mean Squared Error: 4.45217631320528
Standard deviation: 0.879476619879307

On the test dataset:

Root Mean Squared Error: 3.9461075385602897

Sample Prediction:

Predicted price: 21.9
Original price: 21.9

After data analysis we have applied the following code for decision tree regression:

```
# Selecting the model
from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor()
model.fit(housing_num_tr, housing_labels)

# Testing the model on Test dataset
X_test = strat_test_set.drop("MEDV", axis=1)
Y_test = strat_test_set["MEDV"].copy()
X_test_prepared = my_pipeline.transform(X_test)
final_predictions = model.predict(X_test_prepared)
final_mse = mean_squared_error(Y_test, final_predictions)
final_rmse = np.sqrt(final_mse)

# Printing the final predictions on test dataset
print("Predicted Prices on Test dataset:")
print(final_predictions, "\n")
print("Original Prices:")
print(list(Y_test),"\n")
print("The RMSE of the model on Test dataset:")
print(final_rmse, "\n")
```

3. Support Vector Regression

When a support vector machine is applied for regression scenarios, then we call it a support vector regression model. The objective of the support vector machine is to find a hyperplane in an N-dimensional space i.e. N number of features. This distinctly classifies the data points. SVR is a linear model which works in both classification and regression problems.

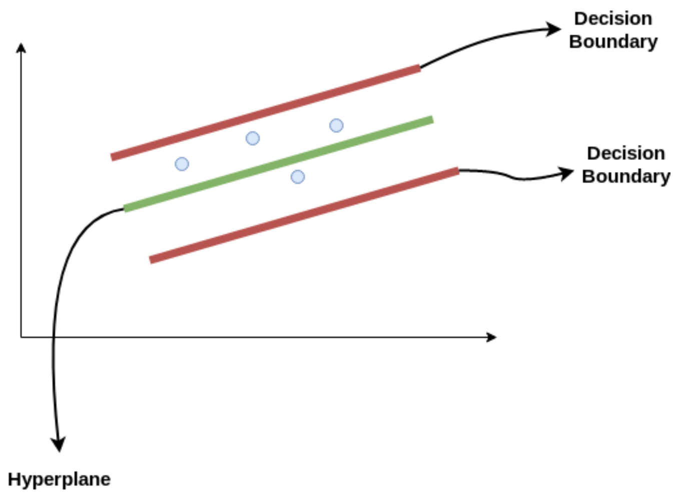


Fig.5: Understanding Support Vector Regression.

Considering the two red lines as decision boundaries and the green line as the hyperplane. When we are applying the SVR then we consider the points that are within the decision boundary line. Our best fit will be the hyperplane that has the maximum number of points.

After applying the support vector regression model on the dataset we have calculated the root mean squared error and standard deviation by cross-validation evaluation and then calculated the RMSE on the test dataset.

On the training dataset by Cross-validation:

Root Mean Squared Error: 5.606712889302848
Standard deviation: 1.70961043370698

On the test dataset:

Root Mean Squared Error: 4.4504032566741145

Sample Prediction:

Predicted price: 22.99990377
Original price: 21.9

After data analysis, we have applied the following code for support vector regression:

```
# Selecting the model
from sklearn.svm import SVR
model = SVR()
model.fit(housing_num_tr, housing_labels)

# Testing the model on Test dataset
X_test = strat_test_set.drop("MEDV", axis=1)
Y_test = strat_test_set["MEDV"].copy()
X_test_prepared = my_pipeline.transform(X_test)
final_predictions = model.predict(X_test_prepared)
final_mse = mean_squared_error(Y_test, final_predictions)
final_rmse = np.sqrt(final_mse)

# Printing the final predictions on test dataset
print("Predicted Prices on Test dataset:")
print(final_predictions, "\n")
print("Original Prices:")
print(list(Y_test),"\n")
print("The RMSE of the model on Test dataset:")
print(final_rmse, "\n")
```

4. Random Forest Regression

Random forest regression is a supervised learning model which utilizes the bagging ensemble technique. Bagging is used to create several subsets of data from the training sample chosen randomly with replacement. Each collection of subset data is used to train the decision trees.

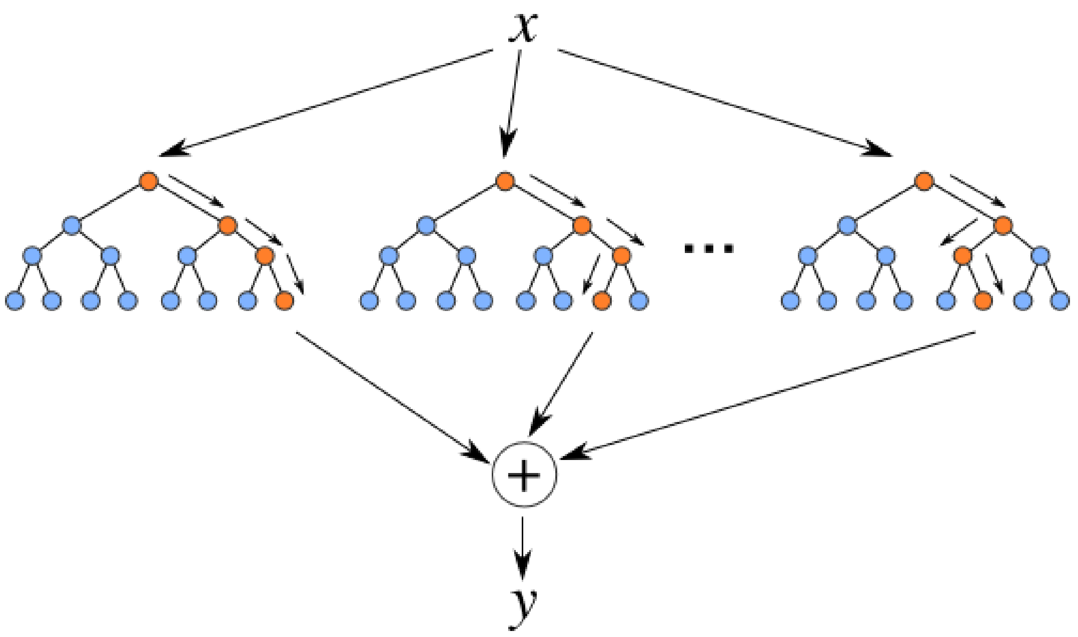


Fig.6: Understanding Random Forest Regression.

It constructs multiple decision trees at training time, runs them parallelly and combines the result to predict the final output. The steps involved in a random forest regression model are:

a. Pick K data points randomly from the training set.
b. Build decision trees associated with those data points.
c. Choose the number N of trees you want to build and repeat the steps a and b.
d. For a new data point, make each of your N trees predict the value of Y for the data point and assign the new data point the average across all of the predicted values.

As we take the average of the predictions of many decision trees our accuracy improves and we get better predictions with random forest regression. These kind of ensemble learning methods are quite stable as any change in the dataset are less likely to impact the forest of decision trees.

After applying the random forest regression model on the dataset we have calculated the root mean squared error and standard deviation by cross-validation evaluation and then calculated the RMSE on the test dataset.

On the training dataset by Cross-validation:

Root Mean Squared Error: 3.35320702603959
Standard deviation: 0.7168580437638796

On the test dataset:

Root Mean Squared Error: 2.9734602655795457

Sample Prediction:

Predicted price: 22.421
Original price: 21.9

After data analysis, we have applied the following code for random forest regression;

```
# Selecting the model
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor()
model.fit(housing_num_tr, housing_labels)

# Testing the model on Test dataset
X_test = strat_test_set.drop("MEDV", axis=1)
Y_test = strat_test_set["MEDV"].copy()
X_test_prepared = my_pipeline.transform(X_test)
final_predictions = model.predict(X_test_prepared)
final_mse = mean_squared_error(Y_test, final_predictions)
final_rmse = np.sqrt(final_mse)

# Printing the final predictions on test dataset
print("Predicted Prices on Test dataset:")
print(final_predictions, "\n")
print("Original Prices:")
print(list(Y_test),"\n")
print("The RMSE of the model on Test dataset:")
print(final_rmse, "\n")
```

Now, we are done with the supervised learning regression models. We will now try to apply the unsupervised learning method to the given Boston Housing dataset.

B. Unsupervised Learning

Data Analysis for Unsupervised Learning Model:

First of all, we have imported the dataset into the Pandas Dataframe.

```
# Loading the dataset
housing_uns = pd.read_csv("data.csv")
```

Here, we have decided to build a KMeans cluster for the relation between 'MEDV' and 'NOX' attributes. Many other attributes can be used for KMeans Clustering but the relationship should be meaningful. The plot for the relation of 'MEDV' and 'NOX' is plotted using the code below:

```
# Plotting a relation to see if clustering is meaningful

plt.scatter(housing_uns['NOX'], housing_uns['MEDV'])
plt.show()
```

The graph depicting the relation is:

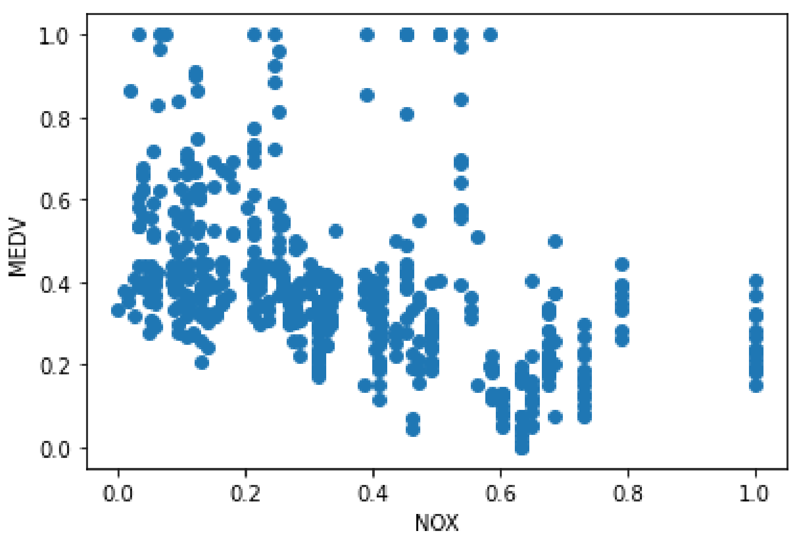


Fig.7: NOX vs MEDV relation.

Here we have applied the MinMaxScaler which is also called normalisation as it scales down every value to a number between 0 and 1. MinMaxScaling replaces every value with (value - min) / (max - min). It is derived from the sklearn preprocessing library.

```
# Implementing MinMaxScaler to transform the data
scaler = MinMaxScaler()
housing_uns['MEDV'] = scaler.fit_transform(housing_uns[['MEDV']])
housing_uns['NOX'] = scaler.fit_transform(housing_uns[['NOX']])
```

Algorithm for Unsupervised Machine Learning Model:

KMeans Clustering:

KMeans clustering is an iterative unsupervised machine learning model which is used to form K pre-defined cluster of non-distinct and non-overlapping subgroups. The model assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid is at the minimum. The less variation within the clusters, the more homogeneous the data points are within the same cluster.

The steps to be followed in the KMeans algorithm are:

a. Specify the number of clusters.
b. Initialize centroids by randomly selecting K points without replacement.
c. Iterate continuously until there is no change to the centroids i.e assignment of data points to clusters doesn't change.
d. Compute the sum of squared distance between the data points and centroids.
e. Assign each data point to the closest cluster.
f. Compute the centroids for the clusters by taking the average of all data points belonging to the same cluster.

We have implemented the KMeans clustering as follows:

```
# Deciding the number of clusters
km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(housing_uns[['NOX', 'MEDV']])

housing_uns['cluster'] = y_predicted
# print("The cluster centers are: ", km.cluster_centers_, "\n")
# The centres of clusters

df1 = housing_uns[housing_uns.cluster==0]
df2 = housing_uns[housing_uns.cluster==1]
df3 = housing_uns[housing_uns.cluster==2]

plt.scatter(df1.NOX, df1['MEDV'], color='green')
plt.scatter(df2.NOX, df2['MEDV'], color='red')
plt.scatter(df3.NOX, df3['MEDV'], color='yellow')
plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:,1],
color='purple', marker='*', label='centroid')

plt.xlabel('NOX')
plt.ylabel('MEDV')
plt.legend()
plt.show()
```

The plot of the KMeans clustering model on the given dataset is:

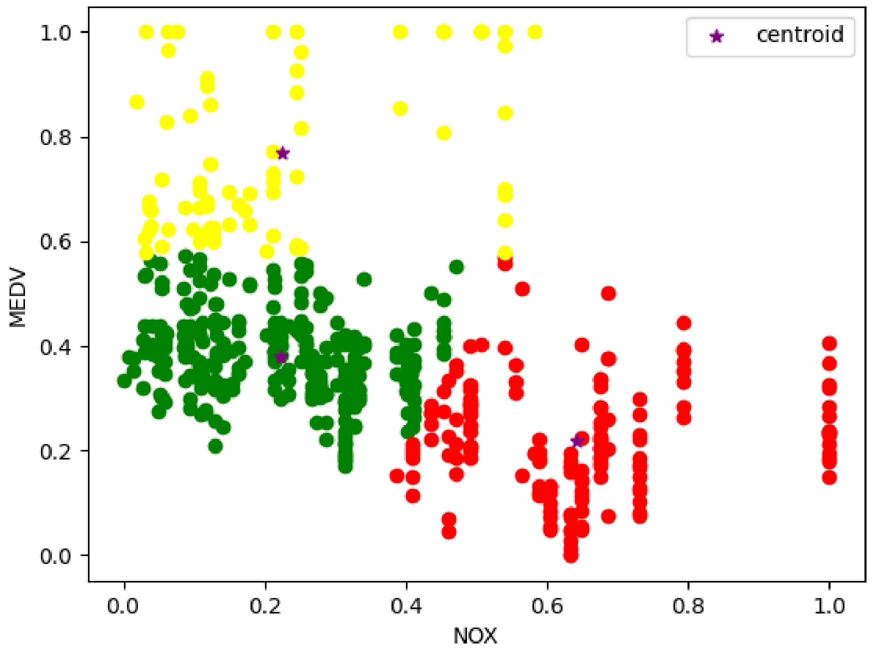


Fig.8: KMeans Clustering on NOX vs MEDV.

To plot the relationship between the sum of squared errors and K we have applied the following code:

```
# To plot the relationship between the sum of squared errors and K
k_rng = range(1,10)
sse = []

for k in k_rng:
    km = KMeans(n_clusters=k)
    km.fit(housing_uns[['NOX', 'MEDV']])
    sse.append(km.inertia_)

print("The array of SSE: ", sse, "\n")
plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.plot(k_rng,sse)
plt.show()
```

The output graph is:

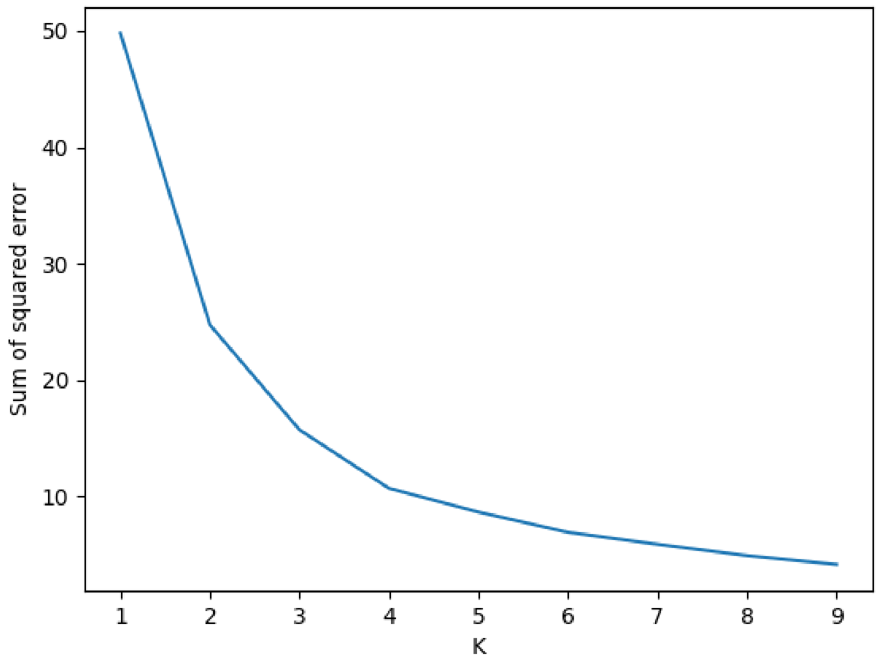


Fig.9: SSE vs K for KMeans Clustering.

C. Neural Network

Data Analysis for Feedforward Neural Network Model:

First of all, we need to import the dataset in the pandas Dataframe and then split it into the training and test set but these operations were already performed in the case of regression models so we will use the same techniques here until the pipeline implementation (we won't apply Standard Scaler here).

Instead of applying the Standard Scaler, we will apply the normalisation method here using the preprocessing library of sklearn as follows:

```
housing = preprocessing.normalize(housing)
X_test = preprocessing.normalize(X_test)
```

Algorithm for Feedforward Neural Network Model:

Neural network models deal with basically three layers of nodes which are:

a. Input layer
b. Hidden layers
c. Output layer

Hidden layers here consists of many layers of nodes. More number of hidden layers generally improves the accuracy of the decision.

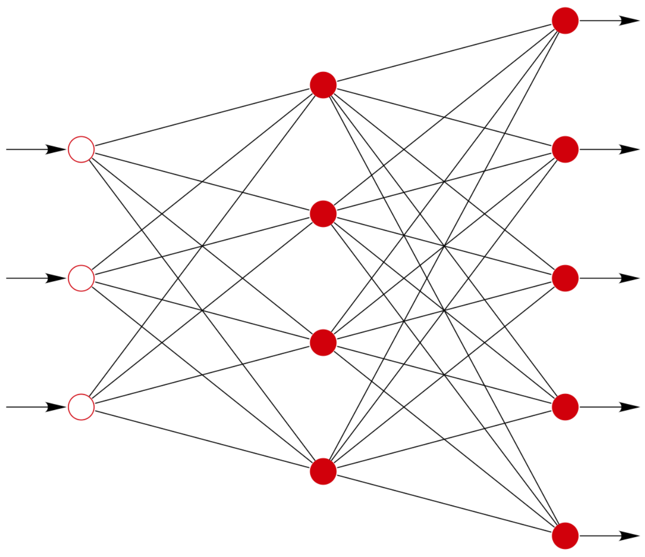


Fig.10: Feedforward Neural Network.

Feedforward neural network is a network of forward-moving nature which means if a signal has moved forward then it doesn't iterate or come back into the previous layer, it continues to move forward i.e. there is no feedback. The activation function used here is ReLU which is a piecewise linear function.

Here, we have used the Sequential modelling from the Keras library and RMSprop optimizer to build our model.

After applying the Feedforward neural network model on the dataset we have calculated the root mean squared error(RMSE) on the test dataset.

RMSE of the model on the test dataset:

Root Mean Squared Error: 4.060599385963832

Sample Prediction:

Predicted price: 22.643642
Original price: 21.9

We have implemented the Feedforward neural network as follows:

```python
# Creating the Neural Network Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
def HousePricePredictionModel():
    model = Sequential()
    model.add(Dense(128,activation='relu',input_shape=(housing[0].shape)))
    model.add(Dense(64,activation='relu'))
    model.add(Dense(32,activation='relu'))
    model.add(Dense(1))
    model.compile(optimizer='rmsprop',loss='mse',metrics=['mae'])
    return model

k = 4
num_val_samples = len(housing)
num_epochs = 50
all_scores = []

model = HousePricePredictionModel()
history = model.fit(x=housing, y=housing_labels, epochs=num_epochs,
batch_size=1, verbose=1, validation_data=(X_test,Y_test))

# Testing the model on Test dataset
final_predictions = model.predict(X_test)
final_mse = mean_squared_error(Y_test, final_predictions)
final_rmse = np.sqrt(final_mse)

# Printing the final predictions on test dataset
print("Predicted Prices on Test dataset:")
print(final_predictions, "\n")
print("Original Prices:")
print(list(Y_test),"\n")
print("The RMSE of the FF Neural network model on Test dataset:")
print(final_rmse, "\n")
```

D. Logistic Regression Classification

Data Analysis Logistic Regression Classification:

We have a slightly modified dataset for the application of Logistic regression classification. Here instead of price prediction, the dataset contains a binary class attribute MEDV which contains 1 if the price of the house is above $30,000 and 0 if the price is below $30,000.

For logistic regression classification, we require no special data analysis in this case. We have simply imported the dataset into the Pandas Dataframe using the code:

```
# Loading the dataset
housing_log = pd.read_csv("data_log.csv")
```

Now, we need to split the dataset into training and testing dataset from which we need to separate the class attribute from the feature attributes. We have implemented the same using the code below:

```
# Prepare the training set
X = housing_log.iloc[:, :-1]
y = housing_log.iloc[:, -1]

# Splitting the dataset itself to test the accuracy
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size= 0.2,
random_state=42)
```

Algorithm for Logistic Regression Classification:

Logistic regression is similar to linear regression except it is used to predict whether something is true or false instead of predicting some continuous value. It fits an ‘S’ shaped logistic function to the data which is a curve that goes from 0 to 1.

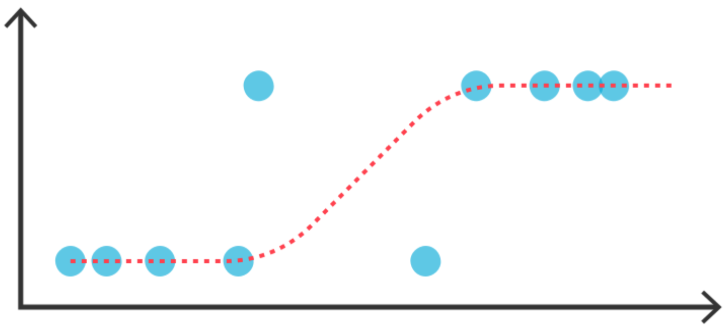


Fig.11: Logistic Regression.

Logistic regression tells the probability of an event which is used to classify the event to 0(<50% Probability) and 1(>50%) label. Here we have classified the instances into two categories 0(for the price below $30K) and 1(for the price above $30K). We have imported the logistic regression function from the sklearn library and implemented it as follows:

```
# Training the model
model = LogisticRegression()
model.fit(x_train, y_train)
```

After applying the logistic regression classification model to the dataset we have calculated the accuracy of the training dataset.

Accuracy of the model on the test dataset:

Accuracy: 0.9607843137254902

Sample Classification:

Predicted category: 0 (price below $30K)
Original category: 0 (price below $30K)

We have tested the logistic regression classification by implementing the following code:

```
# Testing the model
predictions = model.predict(x_test)

# Checking the precision, recall and f1-score
print("\nClassification report:\n", classification_report(y_test,
predictions), "\n")
print("Accuracy of the Model: ", accuracy_score(y_test, predictions), "\n")

# Printing the predictions for the test dataset
print("The predictions for the Test dataset are: ", predictions, "\n")
```

Comparative Analysis

We have applied a total of four regression models namely linear regression, decision tree regression, support vector regression and random forest regression.

Linear regression tries to fit the model linearly and as we can see in plots of the features some features or independent variables do not show linear relation with the dependent variable. Hence, we get an RMSE of 4.143 on the test dataset.

Decision tree regression is a non-linear regression technique. In the regression tree, the value obtained by a terminal node in the dataset is the mean response of observation falling in that region. Hence, if an unseen datapoint falls in that region we can make its prediction with an average or mean value. The decision tree is a non-linear regression model that fits better to the given dataset and we get an RMSE of: 3.94610 on the test dataset.

Support vector regression is also a linear model that finds a hyperplane in an N-dimensional space i.e. N number of features. The decision boundary line here works less efficiently than the non-linear decision tree regression model. The RMSE produced using SVR is 4.450 on the test dataset.

The random forest regression model utilizes bagging type of ensemble learning. It applies various decision trees at the training time and runs them parallelly to compute more accurate predictions. This model fits the best among all other regression models used and produces an RMSE of 2.9667 on the test dataset.

The Feedforward neural network seems to outperform the linear regression model because of the non-linear nature of some of the attributes. Here in the feedforward neural network model, a linear activation function is used in the output layer as it is regression analysis. The RMSE obtained on the test dataset is 4.060.

The KMeans clustering is an unsupervised machine learning model which is used to segregate and cluster a group of data points from a relation between a dependent and an independent attribute. Here we have performed KMeans clustering upon NOX and MEDV relation which gives us 3 clusters of the price which can be useful in trend analysis of price in relation to the nitric oxide concentration in the environment.

Using the logistic regression classifier, we have classified the instances of the dataset (data_log.csv) into two categories: 0(for the price below $30K) and 1(for the price above $30K).

A complete overview of all the regression models when evaluated using cross-validation is listed below. It consists of the mean of root mean squared error(RMSE) and the standard deviation computed on the training dataset.

1. Linear Regression
   Mean of RMSE: 5.030437102767304
   Standard deviation: 1.0607661158294832

2. Decision Tree
   Mean of RMSE: 4.218344806994929
   Standard deviation: 0.7928741428323183

3. SVR
   Mean of RMSE: 5.606712889302848
   Standard deviation: 1.70961043370698

4. Random Forest Regressor
   Mean of RMSE: 3.3597803308307173
   Standard deviation: 0.7295102580579761

5. Feedforward Neural Network
    RMSE: 4.060599385963832

6. Logistic Regression Classification
    Accuracy: 0.9607843137254902