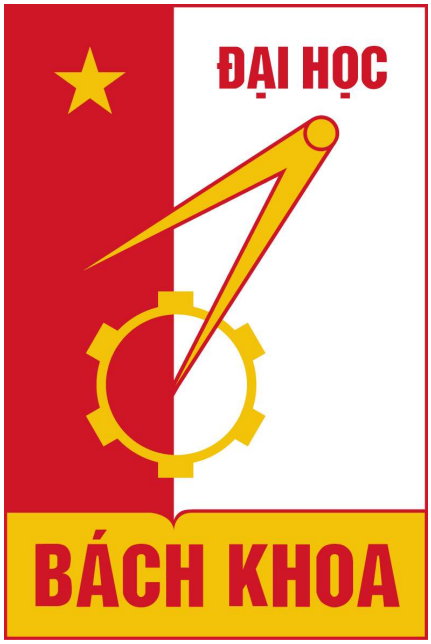
Hanoi University of Science and Technology
School of Information and Communication Technology



Capstone Project Report

Boston Housing Prices Prediction

Course: IT4242E Machine Learning and Data Mining
Class: 131081
Instructor: Nguyen Nhat Quang, Ph.D

Students: Bui Manh Tu 20194870
Truong Tuan Nghia 20194811
Tran Ngoc Dai Quang 20194827

Hanoi, June 2022

IT4242 Machine Learning Capstone Report

Project: Boston Housing Prices Prediction

Group 19

Name	ID
Bùi Mạnh Tú	20194870
Trương Tuấn Nghĩa	20194811
Trần Ngọc Đại Quang	20194827

Table of Contents

Problem Statement

Problem description

Given a data set where each record contains 12 explanatory attributes (specified later) which can affect the housing price of a neighborhood in Boston and a response attribute which tells us about the median of the housing price (medv) in that neighborhood. Our task is to determine the `MEDV` of a new neighborhood given all the 12 related attributes of that neighborhood.

Input & Output description

> Input: Given a numeric vector of 12 explanatory variables.

> Output: A number (`MEDV`) represents the mean value of owner-occupied homes in that neighborhood.

Dataset

Overview

This dataset contains information collected by the *U.S. Census Service* concerning housing in the area of Boston Mass. It was obtained from the StatLib archive, and has been used extensively throughout the literature to benchmark algorithms. However, these comparisons were primarily done outside Delve and are thus somewhat suspect. The dataset is small with only 506 cases.

The data was originally published by Harrison, D. and Rubinfeld, D.L. *Hedonic prices and the demand for clean air*, J. Environ. Economics & Management, vol.5, 81-102, 1978.

This dataset is obtained from the `scikit-learn` library through:

```
sklearn.datasets.load_boston()
```

Description

The Boston data frame has 506 rows and 14 columns.

There are 14 attributes in each case of the dataset. They are:

Attribute	Description
`CRIM`	per capita crime rate by town
`ZN`	proportion of residential land zoned for lots over 25,000 sq.ft.
`INDUS`	proportion of non-retail business acres per town.
`CHAS`	Charles River dummy variable (`1` if tract bounds river; `0` otherwise)
`NOX`	nitric oxides concentration (parts per 10 million)
`RM`	average number of rooms per dwelling
`AGE`	proportion of owner-occupied units built prior to 1940
`DIS`	weighted distances to five Boston employment centres
`RAD`	index of accessibility to radial highways
`TAX`	full-value property-tax rate per $10,000
`PTRATIO`	pupil teacher ratio by town
`B`	`1000(Bk - 0.63)^2` where `Bk` is the proportion of blacks by town
`LSTAT`	% lower status of the population
`MEDV`	median value of owner-occupied homes in $1000's

Algorithms & Approaches

Tree-based algorithms

In this approach we will simply build multiple tree-based models such as **XGBoost**, **Decision Tree** and **Random Forest**; and choose the best one.

K-means clustering

In this approach:

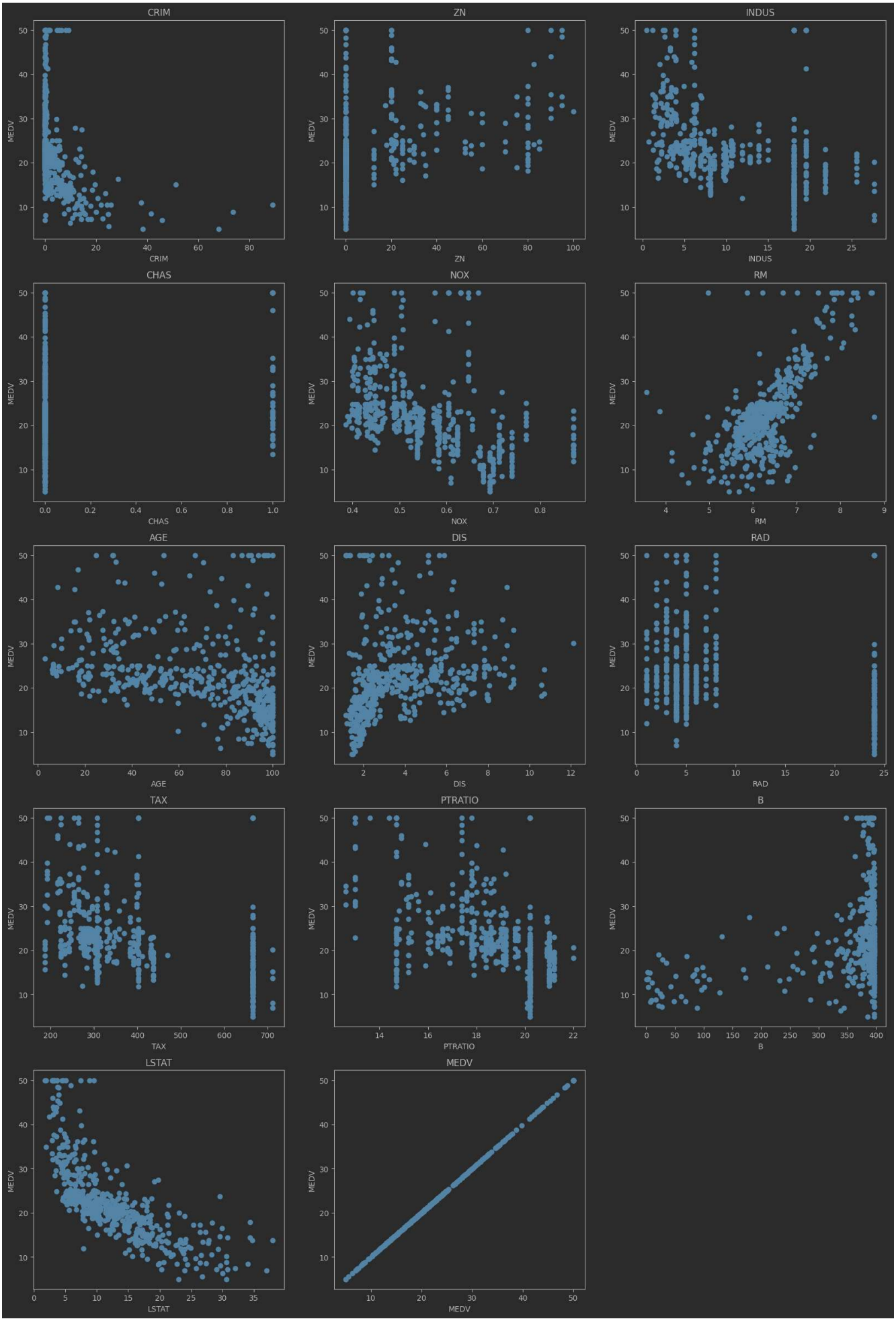
1. We apply the K-mean clustering algorithm to figure out some kind of similarity or relation between all the neighborhoods in the training set.
2. Then apply linear regression for each of the clusters to find a corresponding optimal regression function.
3. In the testing phase, given a new data sample, we will classify it into one of the clusters defined (by using K- nearest neighbours) and then apply the corresponding regression function learned.

Data Process

Outliers

Plot the scatter to study the relationship between `MEDV` and the other attributes:

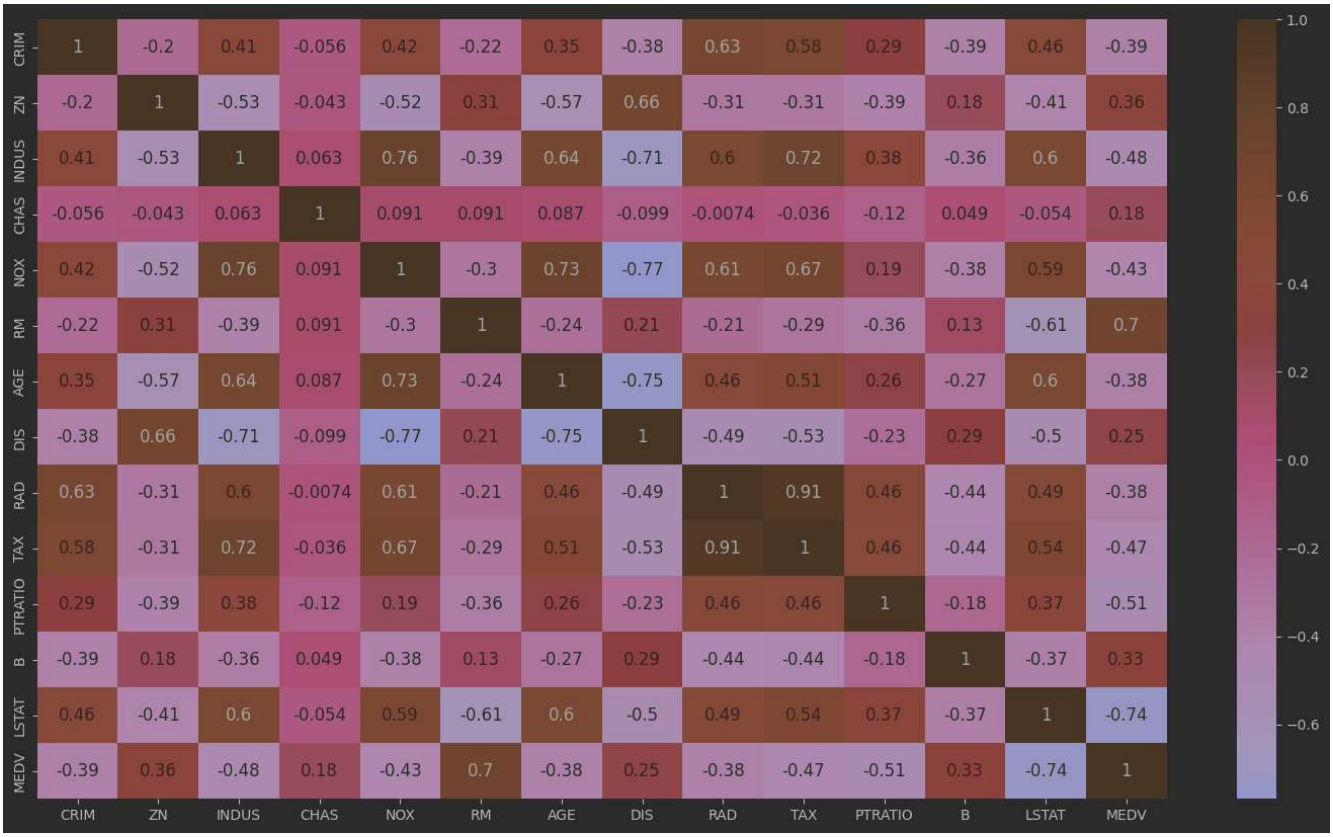
```
sns.pairplot(df, x_vars=df.columns, y_vars="MEDV")
```


CRIM
ZN
INDUS
CHAS
NOX
RM
AGE
DIS
RAD
TAX
PTRATIO
B
LSTAT
MEDV
MEDV

> The 'RM' and 'LSTAT' are in linear relationship with 'MEDV'

Correlation matrix before removing outliers:

```
sns.heatmap(df.corr(), ax=ax, annot=True, annot_kws={'size': 12})
```

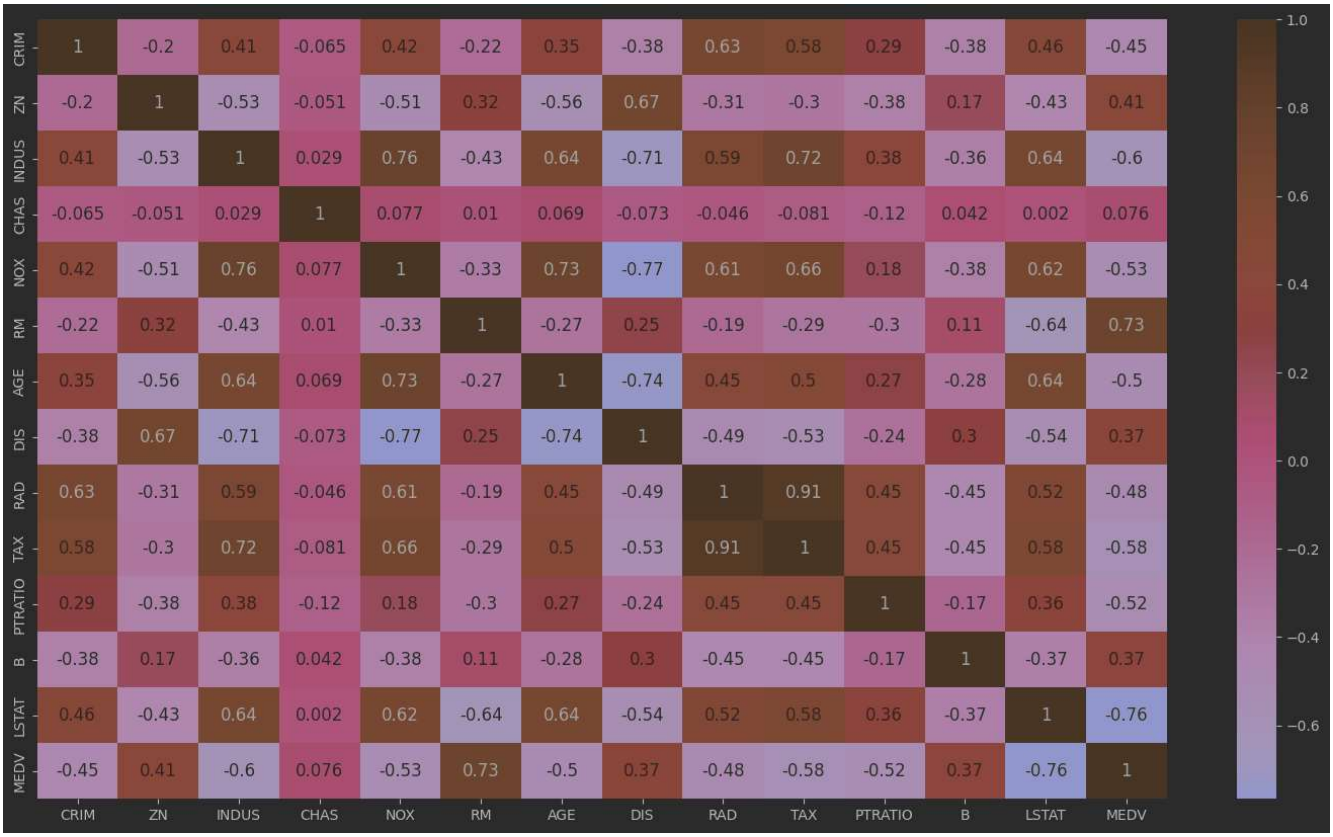


> The higher of the absolute correlation value between two features, the more they depend on each other. Note that for the negative values, it means that they grow in opposite direction.

We can deduce that `RM` has the greatest impact on `MDEV`. Therefore, we remove the outliers in `RM`, namely:

```
RM < 4 or RM > 8.4
```

After removing outliers:

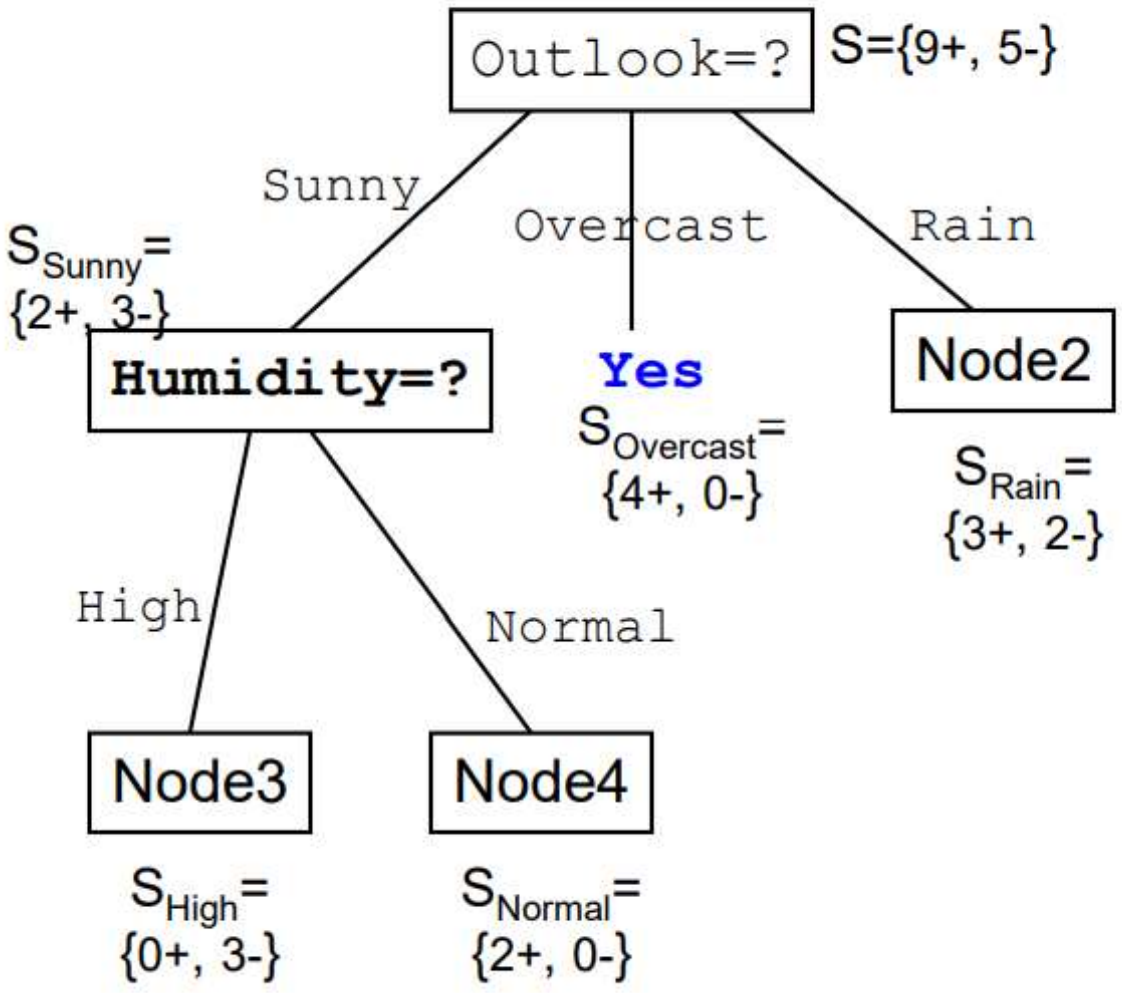

CRIM 1 -0.2 0.41 -0.065 0.42 -0.22 0.35 -0.38 0.63 0.58 0.29 -0.38 0.46 -0.45
ZN -0.2 1 -0.53 -0.051 -0.51 0.32 -0.56 0.67 -0.31 -0.3 -0.38 0.17 -0.43 0.41
INDUS 0.41 -0.53 1 0.029 0.76 -0.43 0.64 -0.71 0.59 0.72 0.38 -0.36 0.64 -0.6
CHAS -0.065 -0.051 0.029 1 0.077 0.01 0.069 -0.073 -0.046 -0.081 -0.12 0.042 0.002 0.076
NOX 0.42 -0.51 0.76 0.077 1 -0.33 0.73 -0.77 0.61 0.66 0.18 -0.38 0.62 -0.53
RM -0.22 0.32 -0.43 0.01 -0.33 1 -0.27 0.25 -0.19 -0.29 -0.3 0.11 -0.64 0.73
AGE 0.35 -0.56 0.64 0.069 0.73 -0.27 1 -0.74 0.45 0.5 0.27 -0.28 0.64 -0.5
DIS -0.38 0.67 -0.71 -0.073 -0.77 0.25 -0.74 1 -0.49 -0.53 -0.24 0.3 -0.54 0.37
RAD 0.63 -0.31 0.59 -0.046 0.61 -0.19 0.45 -0.49 1 0.91 0.45 -0.45 0.52 -0.48
TAX 0.58 -0.3 0.72 -0.081 0.66 -0.29 0.5 -0.53 0.91 1 0.45 -0.45 0.58 -0.58
PTRATIO 0.29 -0.38 0.38 -0.12 0.18 -0.3 0.27 -0.24 0.45 0.45 1 -0.17 0.36 -0.52
B -0.38 0.17 -0.36 0.042 -0.38 0.11 -0.28 0.3 -0.45 -0.45 -0.17 1 -0.37 0.37
LSTAT 0.46 -0.43 0.64 0.002 0.62 -0.64 0.64 -0.54 0.52 0.58 0.36 -0.37 1 -0.76
MEDV -0.45 0.41 -0.6 0.076 -0.53 0.73 -0.5 0.37 -0.48 -0.58 -0.52 0.37 -0.76 1
CRIM ZN INDUS CHAS NOX RM AGE DIS RAD TAX PTRATIO B LSTAT MEDV
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

Theory Review

Decision Tree

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches, each representing values for the attribute tested. Leaf node represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called root node. The decision nodes are evaluated based on information gain.



Decision Tree Algorithm

In practice, there are multiple decision tree algorithms:

1. **ID3** creates a multiway tree, finding for each node, in a greedy manner, the categorical feature that will yield the largest information gain for categorical targets. Trees are grown to their maximum size and then a pruning step is applied to improve the ability of the tree to generalize to unseen data.
2. **C4.5** removed the IDE3's restriction that features must be categorical by dynamically defining a discrete attribute that partitions the continuous attribute value into a discrete set of intervals. C4.5 converts the trained trees (i.e. the output of the ID3 algorithm) into sets of if-then rules. The accuracy of each rule is then evaluated to determine the order in which they should be applied. Pruning is done by removing a rule's precondition if the accuracy of the rule improves without it.
3. **CART** (Classification and Regression Trees) is very similar to C4.5, but it differs in that it supports numerical target variables (regression) and does not compute rule sets. CART constructs binary trees using the feature and threshold that yield the largest information gain at each node.

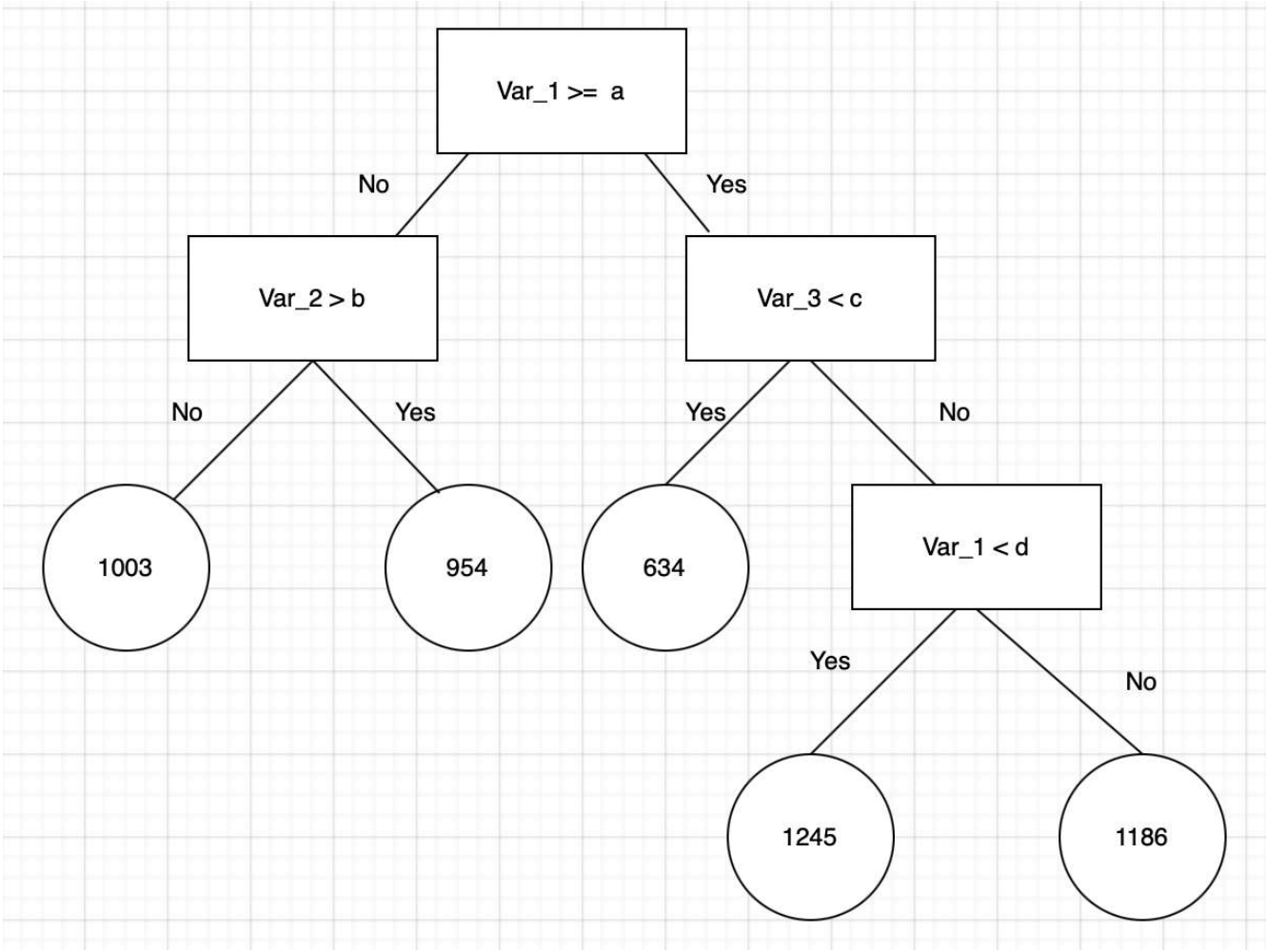
In this project, we use `scikit-learn`, scikit-learn uses an optimized version of the `CART` algorithm.

Random Forest

In random forests, each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set.

Furthermore, when splitting each node during the construction of a tree, the best split is found either from all input features or a random subset of size `max_features`.

The purpose of these two sources of randomness is to decrease the variance of the forest estimator. Indeed, individual decision trees typically exhibit high variance and tend to over-fit. The injected randomness in forests yield decision trees with somewhat decoupled prediction errors. By taking an average of those predictions, some errors can cancel out. Random forests achieve a reduced variance by combining diverse trees, sometimes at the cost of a slight increase in bias. In practice the variance reduction is often significant hence yielding an overall better model.


Var_1 >= a
No
Yes
Var_2 > b
Var_3 < c
No
Yes
Yes
No
1003
954
634
Var_1 < d
Yes
No
1245
1186

Models

In this approach, we build three different regression tree-based models, which are: the decision tree regressor, the random forest regressor, and the XGBoost regressor. Then perform hyperparameter tuning on them by using grid search.

Firstly, for the decision tree regressor, the mechanism of splitting is similar to that of the classification model, except for one core thing is that the algorithm is aiming at maximizing the reduction of the variance of the model result after splitting at each node instead of purity score.

Secondly, for the random forest regressor, this method builds multiple regression trees based on a strategy called bootstrap sampling, which has been included in the theory lecture.

Evaluation Metric

Here we choose the coefficient of determination and Root-Mean-Square Deviation

Coefficient of determination

> In statistics, the **coefficient of determination**, denoted `R2` or `r2` and pronounced "R squared", is the proportion of the variation in the dependent variable that is predictable from the independent variable(s).

Given the mean of the observed data:

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

then the variability of the data set can be measured with two sums of squares formulas:

1. The sum of squares of residuals, also called the **residual sum of squares**:

$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

2. The **total sum of squares** (proportional to the variance of the data):

$$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2$$

The most general definition of the coefficient of determination is

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

Best case: `SS_res = 0` and `R^2 = 1`

Root-Mean-Square Deviation

The **root-mean-square deviation** (RMSD) or root-mean-square error (RMSE) is a frequently used measure of the differences between values (sample or population values) predicted by a model or an estimator and the values observed.

Mathematically, RMSD is the square root of mean square error (MSE):

$$\mathrm{RMSD} = \sqrt{\frac{\sum_{t=1}^{T} (\hat{y}_t - y_t)^2}{T}}.$$

Decision Tree Model

```
    regressor_tree = DecisionTreeRegressor(random_state=5)
regressor_tree.fit(X_train, Y_train)
regressor_tree.predict(X_test)
```

After the first run, the model gave a high performance:

```
    R_2:    0.7078878505113222
RMSE:   5.436721546710903
```

Feature Importance

In some ML or DM problems, attributes may associate with differing costs (i.e., importance degrees). The trend of learning DTs is to use as many as possible low-cost attributes and only use high-cost attributes if it is a must, in order to achieve reliable classifications.

There are many types and sources of evaluating feature importance, although popular examples include statistical correlation scores, coefficients calculated as part of linear models, decision trees, and permutation importance scores.

In `scikit-learn` :

> The importance of a feature is computed as the (normalized) total reduction of the criterion brought by that feature. It is also known as the **Gini importance**.
>
> *see* _ scikit-learn_ .

> **Gini Importance** or *Mean Decrease in Impurity (MDI)* calculates each feature importance as the sum over the number of splits (across all tress) that include the feature, proportionally to the number of samples it splits.

```
model.feature_importances_
```

```
RM,0.557782
LSTAT,0.208201
CRIM,0.078176
DIS,0.074048
RAD,0.021333
AGE,0.017396
TAX,0.011814
NOX,0.010263
PTRATIO,0.008125
B,0.004651
INDUS,0.004263
ZN,0.002755
CHAS,0.001194
```

We decided to discard features that have `importance < 0.01`. The remaining are:

```
['RM', 'LSTAT', 'CRIM', 'DIS', 'RAD', 'AGE', 'TAX', 'NOX']
```

Tuning

In this section, we try tuning important parameters in the `DecisionTreeRegressor` model, namely `max_depth`: the maximum depth of the tree. To achieve the optimal `max_depth`, we apply `GridSearchCV` which exhaustively search over specified parameter values for an estimator, while the estimator is the DT model. To obtain a deterministic behaviour during fitting, `random_state` has to be fixed to an integer, specifically `5`. The searched model is, once again, evaluated by `r2` score.

```
cv_sets = ShuffleSplit(n_splits=10, test_size=0.20, random_state=0)
regressor = DecisionTreeRegressor(random_state=5)

params = {'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}
scoring_fnc = make_scorer(r2)

grid = GridSearchCV(estimator=regressor, param_grid=params, scoring=scoring_fnc, cv=cv_sets, verbose=False)
grid = grid.fit(x, y)
```

Then we achieved the optimal `max_depth = 5` .

Final Evaluation

Run the model again with after all optimization step, we achieve the performance:

```
     Train:
R_2: 0.9187356483217677
RMSE:  2.552670619339252

Test:
R_2: 0.8330910082968221
RMSE:  4.109624547190927
```

Random Forest

The first run's performance:

```
     Train:
R_2: 0.9779043684576689
RMSE:  1.3310609787714158

Test:
R_2: 0.8876495070791605
RMSE:  3.3717065615582396
```

Feature importance:

```
     RM,0.469441
LSTAT,0.321317
DIS,0.062087
CRIM,0.048438
NOX,0.031399
AGE,0.016784
B,0.013346
TAX,0.013319
PTRATIO,0.012697
INDUS,0.006124
RAD,0.003031
CHAS,0.001057
ZN,0.000961
```

Remove features having `importance < 0.01` . The remaining are:

```
['RM', 'LSTAT', 'DIS', 'CRIM', 'NOX', 'AGE', 'B', 'TAX', 'PTRATIO']
```

Tuning

The main parameters to adjust when using this model are `n_estimators` and `max_features`:

1. `n-estimators`: the number of trees in the forest. The larger, the better, but also the longer it will take to compute. In addition, note that results will stop getting significantly better beyond a critical number of trees.
2. `max_features`: the size of the random subsets of features to consider when splitting a node. The lower, the greater the reduction of variance, but also the greater the increase in bias.

Empirical good values are:

- `max_features=1.0` or `max_features=None` *(always consider all features instead of a random subset)* for regression problems
- `max_features="sqrt"` *(using a random subset of size sqrt(n_features))* for classification tasks.

Good results are often achieved when setting `max_depth=None` and `min_samples_split=2` (i.e., when fully developing the trees).

The best parameter values should always be cross-validated.

In this model, we try tuning `n_estimators`, `max_features` and `min_sample_split` with random permutation cross-validator:

```
params = {'n_estimators': [int(x) for x in range(10, 1000, 100)],
     'max_features': [0.5, 'auto', 'sqrt', 'log2'],
     'min_samples_leaf': [1, 2, 4]}
```

Then get the tuned parameters:

```
n_estimators = 710, max_features = 0.5, min_sample_leaf = 1
```

Final evaluation

Final performance evaluation:

```
     Train:
R_2: 0.9820329826755432
RMSE:  1.200280458738707

Test:
R_2: 0.9037667655049607
RMSE:  3.120504560161701
```

XGBoost

Of the two above methods, according to the theory class, and by experiment, we have proven that the random forest outperforms the original regression tree model in this problem, since out data is very small, and the random forest have a better mechanism of exploiting the data. The result show that, at the end, the random forest show a final R-squared score on test data is `0.90` while that of regression tree is only `0.85` .

However, because of our very limited data size, both of the two mentioned algorithms are very easy to over-fit, thus although the score on test data are higher than that of the regression tree, the amount of over-fitting for random forest is much greater than that of the regression tree.

That result has not satisfied our expectation, thus we are looking for a more interesting method, which is called `XGBoost` , from some articles we have read, this method has won in many competition, we have studied in details about the mathematical behind, but within the scope of this report we just explain how the algorithm works and cite the resource that we have study for the math.

The main idea behind `XGBoost` is that, it will start with an initial value, which is the mean of the response variable, the default in the library is (`0.5`), and then with that initial value, it will:

1. Compute the residual value (the difference between predicted and true value)
2. Build an XGBoost unique tree, it is in fact a regression tree to predict the residual value, but the splitting method is based on the two metrics called “similarity score” and “gain”. The splitting which results in a higher value of gain will be chosen.

$$Similarity = \frac{(sum\ of\ the\ residual\ on\ a\ leaf)^2}{total\ number\ of\ residuals\ on\ that\ leaf + \lambda}$$

$$Gain = Similarity\ of\ left\ child + Similarity\ of\ right\ child - Similarity\ of\ root$$

Intuitively, this formula can be interpreted as, if the similarity of a node is different in terms of sign, it will cancel out each other in the numerator, thus it will result in a low similarity score. Vice versa, if the residual in a leaf is the same in terms of sign, this means, the similarity will be high. Lambda here plays the role as a regularization parameter. Because, later on, we will prune the tree based on the value of similarity score on external nodes, a higher lambda results in a lower similarity score, thus leading to more pruning.

3. After predicting all the new residual values for all records, we will come back to step 1 and repeat the whole process with all the new predicted residual values until the algorithm converges.
4. The final predicted value of the XGBoost algorithm will be evaluated by:

$$Initial\ value + learning\ rate * (residual\ predicted\ by\ 1^{st}\ tree + residual\ predicted\ by\ 2^{nd}\ tree + \ldots)$$

The similarity score formula above is a gradient-based formula, it aims at reducing the residual gap between the predicted and the true value of the response variable.

And that is also the reason why the regression tree built in this algorithm called, `XGBoost` "unique" tree, it removes the "random" factor out of the algorithm.

This is an awesome algorithm, but once again, our dataset is very limited, it can not prevent over-fitting. The final result on test data is `0.91` which is better than that of random forest and regression trees and the amount of over-fit is also reduced.

References

1. XGBoost Part 1 (of 4): Regression; StatQuest
2. XGBoost Part 3 (of 4): Mathematical Details; StatQuest
3. Machine Learning Project: Predicting Boson House Prices With Regression
4. Boston Home Prices Prediction And Evaluation
5. Predicting House Prices Using Scikit Learn's Random Forest Model