# CHAPTER 5: Non-Parametric Methods to solve Linear Regression

## Introduction

In the previous chapter, we studied the parametric approach to solving the Linear Regression, in this look at non-parametric algorithms to solve the multiple linear regression problem.

A machine learning estimator or technique is parametric if it relies on the underlying distribution of the data in some way. most of the tools in classical statistics are parametric: fundamentally rely on assumptions about the underlying distribution of the data being studied. Techniques like least square regression are parametric. the alternative is non-parametric tools.

Nonparametric machine learning algorithms do not rely on assumptions about the shape of the underlying data to work. nonparametric techniques include support vector machines and decision trees. Nonparametric techniques tend to be natively set in the classification context. However, they can all be extended relatively easily to regression, in one way or another.

## Nonparametric Regression Algorithms

The following are some of the nonparametric regression algorithms which can solve regression problems.

* Decision Tree Regressor
* Random Forest Regressor
* Support Vector Regressor
* KNN Regressor

Let’s discuss them one by one.

## Decision Tree Regressor

The decision tree algorithm is one of the most versatile algorithms in machine learning which can perform both classification and regression analysis. It is very powerful and works great with complex datasets. Apart from that, it is very easy to understand and read. That makes it more popular to use. This Algorithm works by dividing the whole dataset into a tree-like structure based on some rules and conditions and then gives predictions based on those conditions.

When performing regression with a decision tree, we try to divide the given values of X into distinct and non-overlapping regions, e.g. for a set of possible values X1, X2,….Xp; we will try to divide them into “J” distinct and non-overlapping R1, R2,……………Rj. For a given observation falling into the region Rj, the prediction is equal to the mean of the response(y) values for each training observation (x) in the region Rj. The regions R1, R2...RJ is selected in a way to reduce the following sum of squares of residuals:

Text

Description automatically generated

Diagram

Description automatically generated

It is a tree-structured classifier with three types of nodes. The Root Node is the initial node which represents the entire sample and may get split further into further nodes. The interior Nodes represent the features of a data set and the branches represent the decision rules. Finally, The leaf Nodes represent the outcome.

### **Recursive binary splitting (Greedy approach)**

As mentioned above, we try to divide the X values into j regions, but it is very expensive in terms of computational time to try to fit every set of X values into j regions. Thus, the decision tree opts for a top-down greedy approach in which nodes are divided into two regions based on the given condition, i.e. not every node will be split but the ones which satisfy the condition are split into two branches. It is called greedy because it does the best split at a given step at that point of time rather than looking for splitting a step for a better tree in upcoming steps. It decides a threshold value(say s) to divide the observations into different regions(j) such that the RSS for Xj>= s and Xj <s is minimum.

Chart

Description automatically generated with low confidence

Here for the above equation, j and s are found such that this equation has the minimum value. The regions R1, R2 are selected based on that value of s and j such that the equation above has the minimum value. Similarly, more regions are split out of the regions created above based on some condition with the same logic. This continues until a stopping criterion (predefined) is achieved. Once all the regions are split, the prediction is made based on the mean of observations in that region.

The process mentioned above has a high chance of overfitting the training data as it will be very complex.

### **Tree Pruning**

Tree pruning is the method of trimming down a full tree (obtained through the above process) to reduce the complexity and variance in the data. Just as we regularised linear regression, we can also regularise the decision tree model by adding a new term.

Text, letter

Description automatically generated with medium confidence

Where T is the subtree which is a subset of the full tree T0 And α is the non-negative tuning parameter which penalizes the MSE with an increase in tree length. By using cross-validation, such values of α and T are selected for which our model gives the lowest test error rate. This is how the decision tree regression model works. Let’s now see the working algorithm of doing classification using a decision tree. Greedy Algorithm As per the Hands-on machine learning book “greedy algorithm greedily searches for an optimum split at the top level, then repeats the process at each level. It does not check whether the split will lead to the lowest possible impurity several levels down. A greedy algorithm often produces a reasonably good solution, but it is not guaranteed to be the optimal solution.”

### **Post-pruning**

Post-pruning, also known as backward pruning, is the process where the decision tree is generated first and then the non-significant branches are removed. Cross-validation set of data is used to check the effect of pruning and tests whether expanding a node will improve or not. If any improvement is there then we continue by expanding that node else if there is a reduction in accuracy then the node not be expanded and should be converted into a leaf node.

### **Pre-pruning**

Pre-pruning, also known as forward pruning, stops the non-significant branches from generating. It uses a condition to decide when it should terminate splitting of some of the branches prematurely as the tree is generated.

## Random Forest Regressor

Random Forest Regression is a bagging technique in which multiple decision trees are run in parallel without interacting with each other. It is an ensemble algorithm that combines more than one algorithm of the same or different kinds of regression problems. Multiple decision trees are trained over a dataset and averaged to arrive at the result.

The Decision Tree is an easily understood and interpreted algorithm and hence a single tree may not be enough for the model to learn the features from it. On the other hand, Random Forest is also a “Tree”-based algorithm that uses the qualities features of multiple Decision Trees for making decisions.

Therefore, it can be referred to as a “Forest” of trees and hence the name “Random Forest”. The term “Random” is because this algorithm is a forest of “Randomly created Decision Trees”.

The Decision Tree algorithm has a major disadvantage in that it causes over-fitting. This problem can be limited by implementing the Random Forest Regression in place of the Decision Tree Regression. Additionally, the Random Forest algorithm is very fast and more robust than other regression models.

Radar chart

Description automatically generated with low confidence

The diagram above shows the structure of a Random Forest. You can notice that the trees run in parallel with no interaction amongst them. A Random Forest operates by constructing several decision trees during training time and outputting the mean of the classes as the prediction of all the trees. To get a better understanding of the Random Forest algorithm, Let’s walk through the steps:

1. Pick at random k data points from the training set.
2. Build a decision tree associated with these k data points and randomly sample the features in the feature space.
3. Choose the N number of trees you want to build and repeat steps1 and 2.
4. For a new data point, each one of your N-tree predicts the value of y for the data point in question and assigns the new data point to the average across all the predicted y values.

## Support Vector Machine Regressor

Support Vector Regression (SVR) uses the same principles as SVM, but for regression problems.

**The Idea Behind Support Vector Regression**

The problem of regression is to find a function that approximates mapping from an input domain to real numbers based on a training sample. So, let’s now dive deep and understand how SVR works.

The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only a few minor differences. First of all, because the output is a real number it becomes very difficult to predict the information at hand, which has infinite possibilities. In the case of regression, a margin of tolerance (epsilon) is set in approximation to the SVM which would have already been requested from the problem. But besides the fact, there is also a more complicated reason, the algorithm is more complicated therefore to be taken into consideration. However, the main idea is always the same: to minimize error, individualizing the hyperplane which maximizes the margin. Keep in mind that part of the error is tolerated.

Chart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generated

Linear SVR

A picture containing text, watch, gauge

Description automatically generated

Let’s understand the above mathematical concept in simple terms

Chart, line chart

Description automatically generated

Consider these two red lines as the decision boundary and the green line as the hyperplane. Our objective, when we are moving on with SVR, is to consider the points that are within the decision boundary line, our best fit line is the hyperplane that has a maximum number of points.

The first thing we’ll understand is what is decision boundary is (the danger red line above!). consider this line as being at any distance, say ‘a’, from the hyperplane. So, these are the lines that we draw at distance ‘+a’ and ‘-a’ from the hyperplane. This ‘a’ in the text is referred to be as epsilon.

Assuming that the equation of the hyperplane is as follows:

**Y = wx+b (equation of hyperplane)**

Then the equations of decision boundary become:

**wx+b = +a**

**wx+b = -a**

Thus, any hyperplane that satisfies our SVR should satisfy.

**-a < y – wx+b < +a**

Our main aim here is to decide a decision boundary at ‘a’ distance from the original hyperplane such that data points closest to the hyperplane or the support vectors are within that boundary line.

Hence, we are going to take only those points that are within the decision boundary and have the least error rate, or within the Margin of Tolerance. This gives us a better fitting model.

## KNN Regressor

Out of all the machine learning algorithms I have come across, the KNN algorithm has easily been the simplest to pick up. Despite its simplicity, it has proven to be incredibly effective at certain tasks.

And even better? It can be used for both classification and regression problems! The KNN algorithm is by far more popularly used for classification problems, however. I have seldom seen KNN being implemented on any regression task. My aim here is to illustrate and emphasize how KNN can be equally effective when the target variable is continuous.

**How does the KNN algorithm work?**

As we saw above, KNN can be used for both classification and regression problems. The KNN algorithm uses ‘feature similarity’ to predict the values of any new data points. This means that the new point is assigned a value based on how closely it resembles the points in the training set. Any guesses on how the final value will be calculated? The average of the values is taken to be the final prediction.

Below is a stepwise explanation of the algorithm:

1. First, the distance between the new point and each training point is calculated.
2. The closest k data points are selected (based on the distance).
3. The average of these data points is the final prediction for the new point.

**Methods of the calculating distance between points**

The first step is to calculate the distance between the new point and each training point. There are various methods for calculating this distance, of which the most known methods are – Euclidian, Manhattan (for continuous), and Hamming distance (for categorical).

1. **Euclidean Distance:** Euclidean distance is calculated as the square root of the sum of the squared difference between a new point (x) and an existing point (y).
2. **Manhattan Distance:** This is the distance between real vectors using the sum of their absolute difference.

Text

Description automatically generated

1. **Hamming Distance:** It is used for categorical variables. If the value (x) and the value (y) are the same, the distance D will be equal to 0. Otherwise, D=1.

Text, letter

Description automatically generated

Once the distance of a new observation from the points in our training set has been measured, the next step is to pick the closest points. The number of points to be considered is defined by the value of k.

## Practical Implementation

The first step before we get coding is to understand the problem we are trying to solve and the available data. In this case study, we will work with publicly available **Building Energy** data from New York City.

Data: https://github.com/fenago/MLBook/blob/main/Chapter%205/Code/Dataset/Energy\_and\_Water\_Data.csv

General Description of the data

https://github.com/fenago/MLBook/blob/main/Chapter%205/Code/Dataset/nyc\_benchmarking\_disclosure\_data\_definitions\_2017.pdf

**Step1**: Importing data

First, we can load in the data as Pandas Data frame and take a look:

Graphical user interface, text, application, email

Description automatically generated

Table

Description automatically generated

This is a subset of the full data which contains 60 columns. Already, we can see a couple of issues: first, we know that we want to predict the **Energy Star score** but we don’t know what any of the columns mean. While this isn’t necessarily an issue --- we can often make an accurate model without any knowledge of the variables --- we want to focus on interpretability, and it might be important to understand at least some of the columns.

We should at least understand the **Energy Star score**, which is described as:

A 1-to-100 percentile ranking based on self-reported energy usage for the reporting year. The **Energy Star score** is a relative measure used for comparing the energy efficiency of buildings.

That clears up the first problem, but the second issue is that missing values are encoded as “Not Available”. This is a “string” in Python which means that even the columns with numbers will be stored as “object” data types because Pandas converts a column with any string into a column of all strings. We can see the data types of the columns using the following command



Text, letter

Description automatically generated

Sure enough, some of the columns that contain numbers (such as ft2), are stored as objects. We can’t do numerical analysis on strings, so these will have to be converted to a number (especially float) data type!

Here’s a little Python code that replaces all the “Not Available” entries with not a number (np. nan), which can be interpreted as numbers, and then convert the relevant columns to the float data type.

Text

Description automatically generated

Once the correct columns are numbers, we can start to investigate

**Step2:** Missing Values

In addition to incorrect data types, another common problem when dealing with real-world data is missing values. These can arise for many reasons and have to be either filled in or removed before we train a machine learning model.

First, let’s get a sense of how many missing values are in each column

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Table

Description automatically generated

While we always want to be careful about removing information, if a column has a high percentage of missing values, then it probably will not be useful to our model.

What columns to retain may be a little arbitrary, but for this case study; we will remove any columns with more than 50% missing values. In general, be careful about dropping any information because even if it is not there for all the observations, it may still be useful for predicting the target value.

Graphical user interface, text, application

Description automatically generated

A picture containing chart

Description automatically generated

**Step3:** Imputing Missing Values

While we dropped the columns with more than 50% missing values when we cleaned data, there are still quite a few missing observations. Machine learning models cannot deal with any absent values, so we have to fill them in, a process known as imputation.

Every value that is Nan represents a missing observation. Where there are several ways to fill in missing data, we will use a relatively simple method, median imputation for numeric variables, and most frequent (mode) imputation for categorical variables. This replaces all the missing values in columns with the median and mode values of the column.

First, we need to subset the categorical and numerical variables for the imputation process

Text

Description automatically generated

In the following code, we create a Scikit-Learn Imputer object with the strategy set to “median” (numeric variable) and “most\_frequent” (categorical variables). We then train this object on the numeric and category data (using imputer.fit) and use it to fill in the missing values in both the numeric and category data (using imputer.transform). This means missing values in the numeric and category data are filled in with the corresponding median and mode values.

Import the Scikit-Learn package before we start the imputation process



Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

We concatenated the data and saved it as modified data set.

Now let’s check the presence of any null values in the data

Graphical user interface

Description automatically generated with medium confidence

We can see there are no null values present in the data set.

**Note:** Theoretically, 25 to 30% is the maximum missing values are allowed, beyond which we might want to drop the variable from analysis. Practically this varies. At times we get variables with ~50% of missing values but still, the customer insists to have it for analyzing. In those cases, we might want to treat them accordingly.

**Step4:** Outliers

At this point, we may also want to remove outliers. When we remove outliers, we want to be careful that we are not throwing away measurements just because they look strange. They may be the result of actual phenomena that we should further investigate. When removing outliers, I try to be as conservative as possible, using the definition of an extreme outlier:

On the low end, an extreme outlier is below **Q1 – 3 \* IQR**

On the high end, an extreme outlier is above **Q3 + 3 \* IQR**

These can be due to typos in data entry, mistakes in units, or they could be legitimate but extreme values.

Now that the tedious --- but necessary --- step of data cleaning is complete, we can move on to exploring our data!

**Step4:** Exploratory Data Analysis

In short, the goal of EDA is to learn what our data can tell us. It generally starts out with a high-level overview and then narrows into specific areas as we find interesting parts of the data. The findings may be interesting, or they can be used to inform our modeling choices, such as by helping us to decide which features to use.

Univariate analysis (Single variable plots)

A single variable (called univariate) plot shows the distribution of a single variable such as in a histogram.

The goal is to predict the Energy Star score (Renamed to score in our data) so a reasonable place to start is examining the distribution of this variable. A histogram is a simple yet effective way to visualize the distribution of a single variable and is easy to make using matplotlib.

Text

Description automatically generated

Chart, histogram

Description automatically generated

This looks quite suspicious! The Energy Star score is a percentile rank, which means we would expect to see a uniform distribution, with each score assigned to the same number of buildings. However, a disproportionate number of buildings have either highest, 100, or lowest, 1, score (higher is better for the Energy Star score).

If we go back to the definition of the score, we see that it is based on “self-reported energy usage” which might explain the very high scores. Asking building owners to report their own energy usage is like asking students to report their own scores on a test! As a result, this probably is not the most objective measure of a building’s energy efficiency.

If we had an unlimited amount of time, we might want to investigate why so many buildings have very high and very low scores which we could do by selecting these buildings and seeing what they have in common. However, our objective is only to predict the score and not to devise a better method of scoring buildings! We can make a note in our report that the scores have a suspect distribution, but our main focus is on predicting the score.

Let’s look at the distribution of ‘site EUI’ variable

Text

Description automatically generated with medium confidence

Chart

Description automatically generated

Well this shows us we have another problem: outliers! The graph is incredibly skewed because of the presence of a few buildings with very high scores. It looks like we will have to take a slight detour to deal with the outliers. Let's look at the stats for this feature.

Text

Description automatically generated



Text

Description automatically generated

Wow! One building is clearly far above the rest

Timeline

Description automatically generated

It might be worthwhile for someone to follow up with this building owner! However, that is not our problem and we only need to figure out how to handle this information. Outliers can occur for many reasons: typos, malfunctions in measuring devices, incorrect units, or they can be legitimate but extreme values. Outliers can throw off a model because they are not indicative of the actual distribution of data.

**Step5:** Removing Outliers

Text, letter

Description automatically generated

Now, let’s check the distribution of ‘site EUI’ variable

Chart, histogram

Description automatically generated

Text

Description automatically generated

This plot looks a little less suspicious and is close to normally distributed with a long tail on the right side (it has a positive skew)

**Bivariate Analysis**

In order to look at the effect of categorical variables on the score, we can make a density plot colored by the value of the categorical variable. Density plots also show the distribution of a single variable and can be thought of as a smoothed histogram. If we color the density curves by a categorical variable, this will show us how the distribution changes based on the class.

The first plot we will make shows the distribution of scores by the property type. In order to not clutter the plot, we will limit the graph to building types that have more than 100 observations in the dataset.

Text

Description automatically generated with medium confidence



Text, letter

Description automatically generated

Chart, line chart

Description automatically generated

This graph tells us that we should include the property type because this information can be useful for determining the score.

To examine another categorical variable, borough, we can make the same graph, but this time colored by the borough.

Text

Description automatically generated

Text

Description automatically generated

Chart, line chart

Description automatically generated

The borough of the building does not seem to make a significant difference in the distribution of the score as does the building type. Nonetheless, it might make sense to include the borough as a categorical variable.

Correlations between Features and Target

Text

Description automatically generated

Top 15

Text, letter

Description automatically generated

Bottom 15

Text, letter

Description automatically generated

There are several strong negative correlations between the features and the target. The most negative correlations with the score are the different categories of Energy Use Intensity (EUI), Site EUI (kBtu/ft²), and Weather Normalized Site EUI (kBtu/ft²) (these vary slightly in how they are calculated). The EUI is the amount of energy used by a building divided by the square footage of the buildings and is meant to be a measure of the efficiency of a building with a lower score being better. Intuitively, these correlations then make sense: as the EUI increases, the Energy Star Score tends to decrease.

To account for possible non-linear relationships, we can take the square root and natural log transformations of the features and then calculate the correlation coefficients with the score. Try to capture any possible relationships between the borough and building type.

In the following code, we take the log and square root transformations of the numerical variables, one-hot encode the two selected categorical variables (building type and borough), calculate the correlations between all of the features and the score, and display the top 15 most positive and top 15 most negative correlations. This is a lot, but with pandas, it is straightforward to do each step!

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, email

Description automatically generated

After transforming the features, the strongest relationships are still those related to Energy Use Intensity (EUI). The log and square root transformations do not seem they have resulted in any stronger relationships. There are no strong positive linear relationships although we do see that a building type of office (Largest Property Use Type\_Office) is slightly positively correlated with the score. This variable is a one-hot encoded representation of the categorical variables for building type.

We can use these correlations to perform feature selection (coming up in future chapters). Right now, let's graph the most significant correlation (in terms of absolute value) in the dataset which is Site EUI (kBtu/ft^2). We can color the graph by the building type to show how that affects the relationship.

To visualize the relationship between two variables, we use a scatter plot. We can also include additional variables using aspects such as the color of the markers or the size of the markers. Here we will plot two numeric variables against one another and use color to represent a third categorical variable.

Chart, scatter chart

Description automatically generated

There is a clear negative relationship between the Site EUI and the score. The relationship is not perfectly linear (it looks with a correlation coefficient of -0.7, but it does look like this feature will be important for predicting the score of a building.

Multivariate Analysis

As a final exercise for exploratory data analysis, we can make a pairs plot between several different variables. The Pairs Plot is a great way to examine many variables at once as it shows scatterplots between pairs of variables and histograms of single variables on the diagonal.

Using the seaborn PairGrid function, we can map different plots onto the three aspects of the grid. The upper triangle will have scatterplots, the diagonal will show histograms, and the lower triangle will show both the correlation coefficient between two variables and a 2-D kernel density estimate of the two variables.

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated

A picture containing Teams

Description automatically generated

We can see all the three variables have a negative correlation with our target variable “Score”.

We also use a heat map to visualize the correlation between the continuous variables

A picture containing diagram

Description automatically generated

Chart, treemap chart

Description automatically generated

Let’s understand the distribution of energy score by grouping the data with categorical variables using a facet grid from the seaborn package.

Text

Description automatically generated



We filter the data set based on the building types and boroughs

Text

Description automatically generated

Table

Description automatically generated

Text

Description automatically generated

A picture containing graphical user interface

Description automatically generated

By looking at the graph, most of the multifamily buildings have high energy scores in Manhattan, Brooklynn, Bronx boroughs.

The Majority of the office and hotel properties are in Manhattan. Office buildings have high energy scores, and hotel buildings have low energy scores.

**Step6:** Feature Engineering and Feature Selection

Graphical user interface, text, application, email

Description automatically generated



At this point, we have 11319 observations (buildings) with 109 different features (one column is the score). Not all these features are likely to be important for predicting the score, and several of these features are also redundant because they are highly correlated. We will deal with this second issue below.

**Step7:** Remove Collinear Features

Highly collinear features have a significant correlation coefficient between them. For example, in our dataset, the Site EUI and Weather Norm EUI are highly correlated because they are just slightly different means of calculating the energy use intensity.

Text

Description automatically generated

Chart, scatter chart

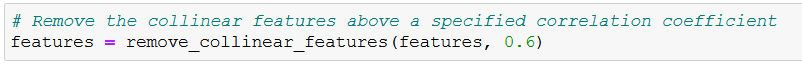
Description automatically generated

Text

Description automatically generated

Text

Description automatically generated



We have removed all the features whose multicollinearity is more than 0.6.

**Step8:** Splitting the dataset

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

**Step9:** Establish a Baseline

It's important to establish a naive baseline before we are beginning making machine learning models. If the models we build cannot outperform a naive guess, then we might have to admit that machine learning is not suited for this problem. This could be because we are not using the right models because we need more data, or because there is a simpler solution that does not require machine learning. Establishing a baseline is crucial so we do not end up building a machine learning model only to realize we can't solve the problem.

For a regression task, a good naive baseline is to predict the median value of the target on the training set for all examples on the test set. This is simple to implement and sets a relatively low bar for our models: if they cannot do better than guessing the median value, then we will need to rethink our approach.

Metric: Mean Absolute Error

There are a number of metrics used in machine learning tasks and it can be difficult to know which one to choose. Most of the time it will depend on the particular problem and if you have a specific goal to optimize for. I like Andrew Ng's advice to use a single real-value performance metric in order to compare models because it simplifies the evaluation process. Rather than calculating multiple metrics and trying to determine how important each one is, we should use a single number. In this case, because we doing regression, the mean absolute error is an appropriate metric. This is also interpretable because it represents the average amount our estimate is off by in the same units as the target value.

The function below calculates the mean absolute error between true values and predictions.

A picture containing text

Description automatically generated

Text

Description automatically generated

**Step10:** Save, no\_scores, training, testing data

Text, letter

Description automatically generated

Reloading the files which were saved above

Text

Description automatically generated

**Step11:** Imputing the Missing Values

Text

Description automatically generated

**Step12:** Feature Scaling

Graphical user interface, text

Description automatically generated

**Step13:** Fitting and Evaluating the performance

We will compare different machine learning models using the great Scikit-learn library:

Graphical user interface, text, application, email

Description automatically generated

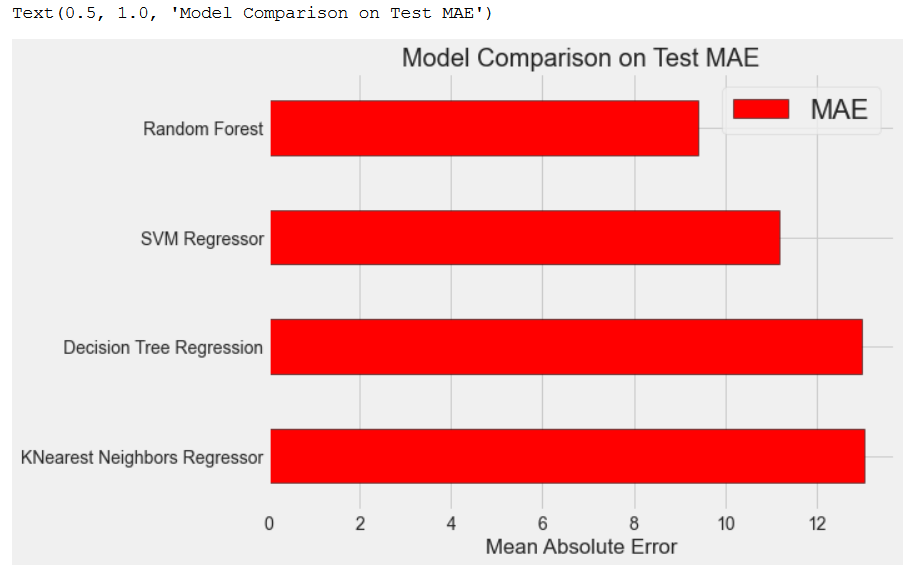
Text

Description automatically generated

Comparing the results visually

Graphical user interface, text, application

Description automatically generated with medium confidence



The best model among all the above models is Random Forest Regressor.

Let’s check the top 10 feature importance of the variables.

Table

Description automatically generated

“Site EUI (kBtu/ft²)” is contributing 59% of the total variance explained by the model.

**SUMMARY**

In this chapter, we studied what is nonparametric method, different algorithms like Decision Tree, Random Forest, Support Vector Machines and KNN Regressor. We have also seen the implementation of all the algorithms.

## Assessment

**Choose the appropriate option**

1. **A decision tree can be used to build models for \_\_\_\_\_\_\_\_\_\_\_**
   1. Classification
   2. Regression
   3. Both of the above
   4. None of the above
2. **A decision tree is built in \_\_\_\_\_\_\_\_\_\_\_\_\_ fashion**
   1. Top-down
   2. Bottom-up
   3. Both of the above
   4. None of these
3. **Can the Random Forest algorithm be used both for Continuous and Categorical independent Variables?**
   1. Only categorical
   2. Only continuous
   3. Both A & B
   4. None of the Above
4. **Random Forest perform bootstrapping on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**
   1. Observations
   2. Variables
   3. Both of the Above
   4. None of the Above
5. **What do you mean by generalization error in terms of the SVM?**
   1. How far the hyperplane is from the support vectors
   2. How accurately the SVM can predict outcomes for unseen data
   3. The threshold amount of error in an SVM
   4. None of the above

**Fill in the spaces with appropriate answers**

1. The SVM allows very low error in regression such margin is known as \_\_\_\_\_\_\_\_\_\_\_\_ margin.
2. A data segment can be said to be pure of all the data instances that belong to the \_\_\_\_\_\_\_\_\_\_\_\_\_.
3. Entropy value of \_\_\_\_\_\_\_\_\_\_\_\_\_\_ represents that data sample is pure or homogenous.
4. The \_\_\_\_\_\_\_\_\_\_\_ value of K should be preferred over \_\_\_\_\_\_\_ values to ensure that there are no ties in the voting.
5. For KNN regression, the predicted value is given by \_\_\_\_\_\_\_\_\_\_\_\_ of the values of its k nearest neighbours.

**True or False**

1. Support vectors are the data points that lie closest to the decision surface.
   1. True
   2. False
2. SVM’s are more effective when the data is noisy and contains overlapping points.
   1. True
   2. False
3. Is Features Scaling required for the KNN algorithm?
   1. True
   2. False
4. A fully grown decision tree can lead the model to overfit.
   1. True
   2. False
5. Random Forest is a boosting algorithm
   1. True
   2. False

## Programming Assessment

Using the data in the below URL, Perform the following tasks

https://github.com/fenago/MLBook/blob/main/Chapter%204/Code/datasets/mtcars.csv

1. Import the data
2. Perform Data Cleaning
3. Perform EDA
4. Fit Decision Tree, KNN, SVR, and Random Forest Regressor models.
5. Evaluate the models.

## Assessment Solutions

**Choose the appropriate options**

1. C
2. A
3. C
4. C
5. B

**Fill in the spaces with appropriate answers**

1. Hard margin
2. Single Class
3. 0
4. Odd, Even
5. The average

**True or False**

1. True
2. False
3. True
4. True
5. False