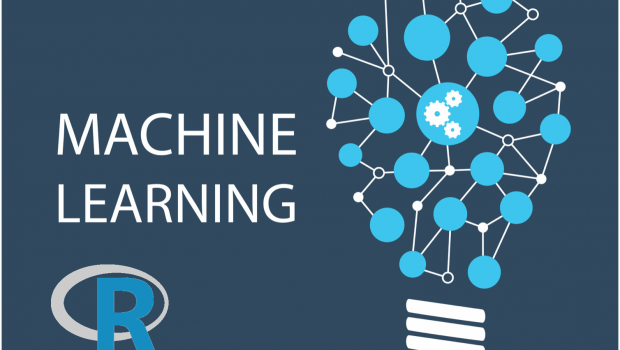
**A Guide to Machine Learning in R for Beginners: Logistic Regression**

This is part 5 of my beginner’s series on Machine Learning in R



**Introduction**

Imagine you are working as a data scientist for an e-commerce company. One of the company’s task is to send out **e-mail**offers to customers with a proposal to buy certain products. Your job as a data scientist is to determine whether the contacted person will buy the product or not. All you have is a sample of customers that were contacted recently, their age and a variable whether or not they took action.

So how do we do that? The only way that appears is to contact every person on the list and ask them whether they will buy the product or not. Although this appears to be the only solution, it isn’t the best one.

So as a Data Scientist, you apply your knowledge of Machine Learning to the problem. Clearly, the [Linear Regression](https://blog.usejournal.com/guide-to-machine-learning-in-r-for-beginners-part-4-6bacf6a82ce8) algorithm will not work here since it only works for problems with a continuous outcome variable. On the other hand, the problem at hand is categorical i.e whether customers will buy a product( =1) or not( =0).

So, Instead of trying to predict exactly whether the people will buy a product or not, you calculate the probability or a likelihood of the person saying yes. Basically you try to fit in probabilities between 0 and 1, which are the two possible outcomes. You also decide a cut off value/threshold and then conclude that people with a probability higher than the threshold will buy the product and vice versa.

And how does it make the work of the company, easier?

Since it gives the probability of people who are more likely to buy a product, it enables the company, to focus only on the customers who are most likely to say **Yes**.

This the **basic intuition** behind Logistic Regression. Now let us get to know the math behind it.

**What is Logistic Regression**

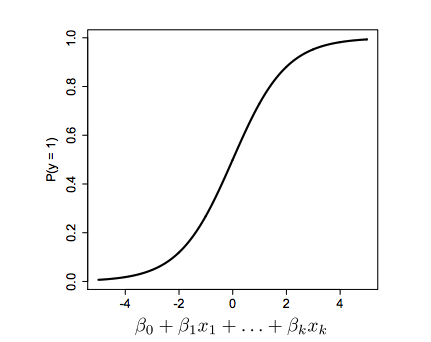
It’s an extension of [linear regression](https://blog.usejournal.com/guide-to-machine-learning-in-r-for-beginners-part-4-6bacf6a82ce8) where the dependent variable is categorical and not continuous. It predicts the probability of the outcome variable.

Logistic regression can be binomial or multinomial. In the binomial or binary logistic regression, the outcome can have only two possible types of values (e.g. “Yes” or “No”, “Success” or “Failure”). Multinomial logistic refers to cases where the outcome can have three or more possible types of values (e.g., “good” vs. “very good” vs. “best” ). Generally, the outcome is coded as “0″ and “1″ in binary logistic regression.

**Representation of Logistic regression**

1. **Logistic Response Function**

https://miro.medium.com/max/623/1*XJlpZH9gB5Kf-loQluPNpQ.png



* Positive values are predictive of class 1
* Negative values are predictive of class 0

The coefficients, or β values, are selected to maximize the likelihood of predicting a high probability for observations actually belonging to class 1 and predicting a low probability for observations actually belonging to class 0. The output of this function is always between **0 and 1**

2. **Odds Ratio**

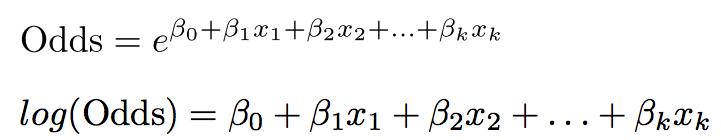
Odds : P(y=1)/P(y=0)

The odds ratio for a variable in logistic regression represents how the odds change with a 1 unit increase in that variable holding all other variables constant.

Odds > 1 if y = 1 is more likely

Odds < 1 if y = 0 is more likely

3. **The Logit**



This is called the “Logit” and looks like linear regression

The bigger the Logit is, the bigger is P(y = 1).

**Baseline Model:**

The baseline model in case of Logistic Regression is to predict the most frequent outcome as the outcome for all data points. (In the case of [Linear regression](https://blog.usejournal.com/guide-to-machine-learning-in-r-for-beginners-part-4-6bacf6a82ce8), the baseline model predicts the average of all data points as the outcome)

Logistic Regression tries to –

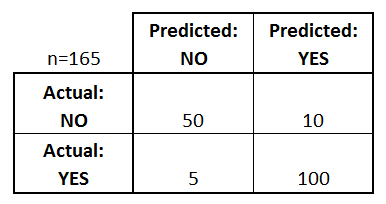
* **Model** the probability of an event occurring depending on the values of the independent variables
* **Estimate** the probability that an event occurs vs the probability that the event does not occur
* **Predict** the effect of a series of variables on a binary response variable
* **Classify** observations by estimating the probability that an observation is in a particular category or not.

The output of a Logistic regression model is a probability. We can select a **threshold** value. If the probability is greater than this threshold value, the event is predicted to happen otherwise it is predicted not to happen.

A **confusion or classification matrix** compares the actual outcomes to the predicted outcomes. The rows are labelled with actual outcomes while the columns are labelled with predicted outcomes.

**Describing the Performance of a Logistic model**

A confusion matrix is a table that is often used to **describe the performance of a classification model** (or “classifier”) on a set of test data for which the true values are known.Let us look at some of the important terms of confusion matrix.



confusion matrix whether employees will leave a company or not

**The Confusion Matrix tells us the following:**

* There are two possible predicted classes: **“yes”**and**“no”**. If we were predicting that employees would leave an organisation, for example, “yes” would mean they will, and “no” would mean they won’t leave the organisation.
* The classifier made a total of 165 predictions (e.g., 165 employees were being studied).
* Out of those 165 cases, the classifier predicted “yes” 110 times, and “no” 55 times.
* In reality, 105 employees in the sample leave the organisation, and 60 do not.

**Basic terms related to Confusion matrix**

* **True positives (TP):** These are cases in which we predicted yes (employees will leave the organisation), and employees actually leave i.e 100
* **True negatives (TN):** We predicted no(employees will not leave the organisation) and they don’t leave i.e 50
* **False positives (FP):** We predicted yes they ***will*** leave, but they don’t leave. (Also known as a “Type I error.”) i.e 10
* **False negatives (FN):** We predicted no they will ***not*** leave, but they actually leave (Also known as a “Type II error.”) i.e 5

**Accuracy :** (TP+TN)/Total . Describes overall, how often the classifier correct. i.e 100+50/165

**Measures of Accuracy**

**Sensitivity and specificity are statistical measures of the performance of a binary classification test:**

**Sensitivity/Recall** = TP/(TP + FN). When it’s actually yes, how often does it predict yes? i.e 100/(100+5)

**Specificity** = TN/(TN + FP) .When it’s actually no, how often does it predict no?? i.e 50/(50+10)

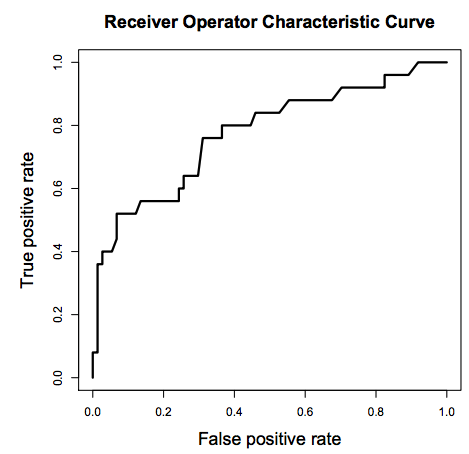
**Precision** = TP/predicted yes. When it predicts yes, how often is it correct?100/(10+100)

**Evaluation metrics for a Classification model’s performance.**

**ROC curve**

A ROC(Receiver Operator Characteristic Curve) can help in deciding the best threshold value. It is generated by plotting the True Positive Rate (y-axis) against the False Positive Rate (x-axis) as you vary the threshold for assigning observations to a given class.ROC curve will always end at (1,1). The threshold at this point will be 0. This means that we will always classify these observations falling into class 1(Specificity will be 0. False-positive rate is 1)

One should select the best threshold for the trade-off you want to make. According to the criticality of the business, **we need to compare the cost of failing to detect positives vs cost of raising false alarms.**



ROC curve

**High Threshold** :

* High specificity
* Low sensitivity

**Low Threshold**

* Low specificity
* High sensitivity

The area under ROC is called **Area Under the Curve(AUC)**. AUC gives the rate of successful classification by the logistic model. To get a more in-depth idea of what a ROC-AUC curve is and how is it calculated, here is a link to the article I wrote on the same topic.

**[Understanding the ROC and AUC metrics.](https://towardsdatascience.com/understanding-the-roc-and-auc-curves-a05b68550b69" \t "_blank)**

[Taking the confusion out of classification metrics](https://towardsdatascience.com/understanding-the-roc-and-auc-curves-a05b68550b69" \t "_blank)

[towardsdatascience.com](https://towardsdatascience.com/understanding-the-roc-and-auc-curves-a05b68550b69" \t "_blank)

**Logistic Regression in R**

In this article, we’ll be working with the [Framingham Dataset](https://biolincc.nhlbi.nih.gov/media/teachingstudies/FHS_Teaching_Longitudinal_Data_Documentation.pdf?link_time=2019-12-26_22:56:26.929689). This data comes from the [BioLINCC website](https://www.nhlbi.nih.gov/" \t "_blank). The objective of the dataset is to assess health care quality. Hence, 131 diabetic patients were randomly selected between the ages of 35 and 55. This case study has been adapted from the **Framingham Heart Study exercise** in the Analytics Edge course on Edx.

We’ll be using the dataset quality.csv to build a logistic regression model in R to predict the ***quality of care*** in a hospital. **PoorCare** is the outcome or dependent variable and is **equal to 1 if the patient had poor care, and equal to 0 if the patient had good care.**

Download this file from [here](https://github.com/parulnith/A-guide-to-Machine-Learning-in-R/blob/master/Part%205%20Logistic%20regression%20dataset/quality.csv)to follow along. An R script file with all of the commands used in this lecture can also be downloaded from my [Github repository.](https://github.com/parulnith/A-guide-to-Machine-Learning-in-R/blob/master/Part%205%20Logistic%20regression%20dataset/LogisticRegression.R" \t "_blank)

**Working**

* Reading data into R console:

> quality = read.csv('quality.csv')

* Analyzing the quality dataset

> **str(quality)**'data.frame': 131 obs. of 14 variables:  
 $ MemberID : int 1 2 3 4 5 6 7 8 9 10 ...  
 $ InpatientDays : int 0 1 0 0 8 2 16 2 2 4 ...  
 $ ERVisits : int 0 1 0 1 2 0 1 0 1 2 ...  
 $ OfficeVisits : int 18 6 5 19 19 9 8 8 4 0 ...  
 $ Narcotics : int 1 1 3 0 3 2 1 0 3 2 ...  
 $ DaysSinceLastERVisit: num 731 411 731 158 449 ...  
 $ Pain : int 10 0 10 34 10 6 4 5 5 2 ...  
 $ TotalVisits : int 18 8 5 20 29 11 25 10 7 6 ...  
 $ ProviderCount : int 21 27 16 14 24 40 19 11 28 21 ...  
 $ MedicalClaims : int 93 19 27 59 51 53 40 28 20 17 ...  
 $ ClaimLines : int 222 115 148 242 204 156 261 87 98 66 ...  
 $ StartedOnCombination: logi FALSE FALSE FALSE FALSE FALSE FALSE ...  
 $ AcuteDrugGapSmall : int 0 1 5 0 0 4 0 0 0 0 ...  
 $ PoorCare : int 0 0 0 0 0 1 0 0 1 0 ...

We have 131 observations, one for each of the patients in our data set, and 14 different variables. The 12 variables from InpatientDays to AcuteDrugGapSmall are the independent variables while PoorCare is the dependent/outcome variable.

**Creating a Baseline Model**

We know good care is more common than poor care. Hence, in this case, we would predict that all patients are receiving good care. By doing this. we would get 98/131 observations correct and an accuracy of 75%. So our **baseline model has an accuracy of 75%**. This is what we’ll try to beat with our logistic regression model.

> table(quality$PoorCare)  
0 1   
98 33> 98/131  
[1] 0.7480916

**Splitting Training & Testing Data**

Since we have only one data set, we want to randomly split our data set into a training set and testing set. The testing set is essential to validate our results. For splitting the data we will use the caTools Package. To make sure that we all get the same split, we’ll set our seed. This initializes the random number generator. The package contains sample.split command to split the data with a split ratio of 0.75. This means we’ll put 75% of the data in the training set, which we’ll use to build the model, and 25% of the data in the testing  
set to test our model.

# Install and load caTools package > install.packages("caTools")  
> library(caTools)# Randomly split data> set.seed(88)  
> split = sample.split(quality$PoorCare, SplitRatio = 0.75)  
> split  
[1] TRUE TRUE TRUE TRUE FALSE TRUE FALSE TRUE FALSE FALSE TRUE FALSE TRUE TRUE  
 [15] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE TRUE TRUE TRUE  
 [29] TRUE FALSE FALSE FALSE FALSE TRUE TRUE TRUE FALSE TRUE TRUE TRUE FALSE FALSE  
 [43] TRUE TRUE FALSE TRUE FALSE TRUE FALSE TRUE TRUE FALSE FALSE TRUE TRUE TRUE  
 [57] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE TRUE TRUE TRUE TRUE  
 [71] FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE  
 [85] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE TRUE TRUE TRUE TRUE  
 [99] TRUE TRUE TRUE TRUE TRUE FALSE TRUE TRUE TRUE FALSE TRUE FALSE FALSE TRUE  
[113] TRUE FALSE TRUE TRUE TRUE FALSE TRUE TRUE FALSE TRUE TRUE FALSE TRUE TRUE  
[127] FALSE TRUE TRUE TRUE FALSE

There is a TRUE or FALSE value for each of our observations.TRUE means that we should put that observation in the training set, and FALSE means that we should put that observation in the testing set. So now let’s create our training and testing sets using the subset function.

**Creating Training & Testing Sets**

We will use the subset function to create the sets.The training set will be called qualityTrain and testing set qualityTest.

# Create training and testing sets> qualityTrain = subset(quality, split == TRUE)  
> qualityTest = subset(quality, split == FALSE)> nrow(qualityTrain)  
[1] 99  
> nrow(qualityTest)  
[1] 32

There are 99 training samples and 32 testing samples.

**Logistic Regression Model**

Now, we are ready to build a logistic regression model using OfficeVisits and Narcotics as independent variables. We’ll call our model QualityLog and use the “glm” function or “generalized linear model” to build  
our logistic regression model. Since we are building the model on training data, we use qualityTrain .The family argument tells the glm function to build a logistic regression model.

# Logistic Regression Model> QualityLog = glm(PoorCare ~ OfficeVisits + Narcotics,data=qualityTrain, family=binomial)> summary(QualityLog)Call:  
glm(formula = PoorCare ~ OfficeVisits + Narcotics, family = binomial,   
 data = qualityTrain)Deviance Residuals:   
 Min 1Q Median 3Q Max   
-2.06303 -0.63155 -0.50503 -0.09689 2.16686Coefficients:  
 Estimate Std. Error z value Pr(>|z|)   
(Intercept) -2.64613 0.52357 -5.054 4.33e-07 \*\*\*  
OfficeVisits 0.08212 0.03055 2.688 0.00718 \*\*   
Narcotics 0.07630 0.03205 2.381 0.01728 \*   
---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1(Dispersion parameter for binomial family taken to be 1)Null deviance: 111.888 on 98 degrees of freedom  
Residual deviance: 89.127 on 96 degrees of freedom  
AIC: 95.127Number of Fisher Scoring iterations: 4

* The **coefficients** table gives the estimated values for the coefficients, or the betas, for our logistic regression model. We see here that the coefficients for OfficeVisitsand Narcotics are both positive, which means that higher values in these two variables are indicative of poor care as we suspected from looking at the data.
* We also see that both of these variables have at least one **star**, meaning that they’re significant in our model.
* **AIC value:**This is a measure of the quality of the model and is like Adjusted R-squared in that it accounts for the number of variables used compared to the number of observations. A low AIC is desirable.

**Making predictions on the Training set**

Let us call it predictTrain and use the predict function to make predictions using the model QualityLog. We will also use an argument called type=” response” which gives us the probabilities. We should always predict on the unseen observations but here we want to get the value of the threshold , hence the predictions on the train set.

> predictTrain = predict(QualityLog, type="response")  
>   
> summary(predictTrain)  
 Min. 1st Qu. Median Mean 3rd Qu. Max.   
0.06623 0.11912 0.15967 0.25253 0.26765 0.98456   
>  
> tapply(predictTrain, qualityTrain$PoorCare, mean) 0 1   
0.1894512 0.4392246

the tapply function computes the average prediction for each of the true outcomes.

We find that for all of the **true poor care** cases, we predict an average probability of about **0.44**. And for all of the **true good care** cases, we predict an average probability of about **0.19**.  
This is good because it looks like we’re predicting a higher probability of the actual poor care cases.

**Thresholding**

We can convert the probabilities to predictions using what’s called a threshold value,**t**. If the probability of poor care is greater than this threshold value, **t**, we predict poor quality care. But if the probability of poor care is less than the threshold value, **t**, then we predict good quality care.

**How to select the value for t:**

The threshold value, **t**, is often selected based on which errors are better. This would imply that **t** would be best for no errors but it’s rare to have a model that predicts perfectly,

There are two types of errors that this model can make:  
1. where the model predicts 1, or poor care, but the actual outcome is 0,

2. Where the model predicts 0,or good care, but the actual outcome is 1.

**Confusion matrix**

To make this discussion a little more quantitative, we use what’s called a confusion matrix or classification matrix.

# Confusion matrix for threshold of 0.5  
> table(qualityTrain$PoorCare, predictTrain > 0.5) FALSE TRUE  
 0 70 4  
 1 15 10# Sensitivity  
> 10/25  
[1] 0.4# Specificity   
> 70/74  
[1] 0.9459459# Confusion matrix for threshold of 0.7  
> table(qualityTrain$PoorCare, predictTrain > 0.7)  
 FALSE TRUE  
 0 73 1  
 1 17 8# Sensitivity   
> 8/25  
[1] 0.32# Specificity  
> 73/74  
[1] 0.9864865# Confusion matrix for threshold of 0.2  
> table(qualityTrain$PoorCare, predictTrain > 0.2)  
 FALSE TRUE  
 0 54 20  
 1 9 16# Sensitivity   
> 16/25  
[1] 0.64# Specificity  
> 54/74  
[1] 0.7297297

We see that by increasing the threshold value, the model’s sensitivity decreases and specificity increases while the reverse happens if the threshold value is decreased. So how to choose the **optimum threshold value.**Picking a good threshold value is often challenging. A Receiver Operator Characteristic curve, or **ROC curve**, can help us decide which value of the threshold is best.

**Install and load ROCR package**

# Install and load ROCR package> install.packages("ROCR")  
> library(ROCR)

**Recall that we made predictions on our training set and called them predictTrain. We’ll use these predictions to create our ROC curve.**

> ROCRpred = prediction(predictTrain, qualityTrain$PoorCare)

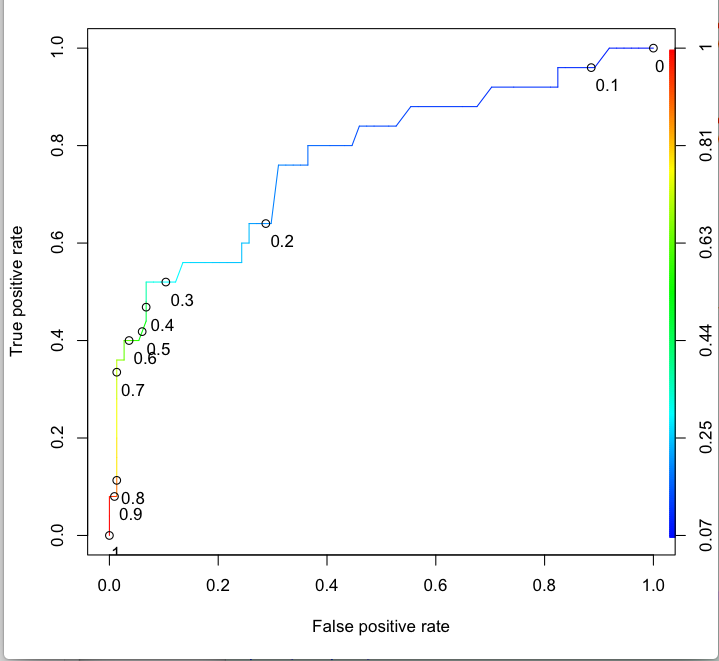
The first is the predictions we made with our model, which we called predictTrain.The second argument is the true outcomes of our data points,  
which in our case, is qualityTrain$PoorCare.

We now use the performance function which defines what we’d like to plot  
on the x and y-axes of our ROC curve.

# Performance function  
> ROCRperf = performance(ROCRpred, "tpr", "fpr")

This function takes as arguments the output of the prediction function,  
and then what we want on the x and y-axes.  
  
Now, we just need to plot the output of the performance function.

# Plot ROC curve  
> plot(ROCRperf)# Add colors  
> plot(ROCRperf, colorize=TRUE)# Add threshold labels > plot(ROCRperf, colorize=TRUE, print.cutoffs.at=seq(0,1,by=0.1), text.adj=c(-0.2,1.7))



The sensitivity, or true positive rate of the model, is shown on the y-axis. while the false positive rate, or 1 minus the specificity, is given on the x-axis. The line shows how these two outcome measures vary with different threshold values.

The ROC curve always starts at the point (0, 0) i.e threshold of value 1. This means at this threshold we will not catch any poor care cases(sensitivity of 0) but will correctly label all the good care cases(FP = 0)

The ROC curve always ends at the point (1,1) i.e threshold of value 0. This means at this threshold we will catch all the poor care cases(sensitivity of 1) but will incorrectly label of all the good care case as poor cases(FP = 1)

The threshold decreases as you move from (0,0) to (1,1). At the point (0, 0.4), we’re correctly labeling about 40% of the poor care cases with a very small false positive rate. On the other hand, at the point (0.6, 0.9), we’re correctly labeling about 90% of the poor care cases, but have a false positive rate of 60%. In the middle, around (0.3, 0.8), we’re correctly labeling about 80% of the poor care cases, with a 30% false-positive rate.

The ROC curve captures all thresholds simultaneously. The higher the threshold, or closer to (0, 0), the higher the specificity and the lower the sensitivity. The lower the threshold, or closer to (1,1), the higher the sensitivity and lower the specificity.

**So which threshold value one should pick**?

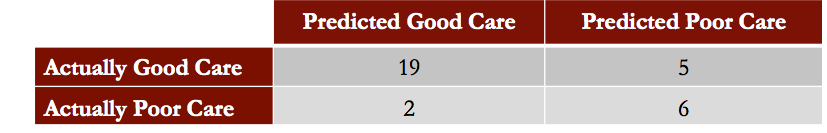
One should select the best threshold for the trade-off one wants to make. If you’re more concerned with having a high specificity or low false-positive rate, pick the threshold that maximizes the true positive rate while keeping the false positive rate really low. A threshold around (0.1, 0.5) on this ROC curve looks like a good choice in this case. On the other hand, if one is more concerned with having a high sensitivity or high true positive rate, one should pick a threshold that minimizes the false positive rate

**Prediction on Test Set**

In this particular example, we used a threshold value of 0.3 and we obtain the following confusion matrix.

> predictTest = predict(QualityLog, type = "response", newdata = qualityTest)> table(qualityTest$PoorCare,predictTest >= 0.3)  
   
 FALSE TRUE  
 0 19 5  
 1 2 6# Accuracy> (19+6)/32  
[1] 0.78125

There is a total of 32 cases in test Set, out of which 24 of them are actually good care, and 8 of them are actually poor care.



**Conclusion**

The model can accurately identify patients receiving low-quality care with test set accuracy being equal to 78% which is greater than our baseline model.

In practice, the probabilities returned by the logistic regression model can be used to prioritize patients for intervention.

This was all about Logistic Regression in R. We studied the intuition and math behind it and also how Logistic regression makes it very easy to solve a problem with the categorical outcome variable.

Click here Guide to Machine Learning(in R) for Beginners: Linear Regression.

Click [here](https://medium.com/analytics-vidhya/a-guide-to-machine-learning-in-r-for-beginners-decision-trees-c24dfd490abb) Guide to Machine Learning(in R) for Beginners: Decision Trees

**Understanding Decision Trees**

**My notes on Decision Trees from the course —** [**Analytics Edge**](https://www.edx.org/course/the-analytics-edge)



Photo by [Fabrice Villard](https://unsplash.com/@fabulu75?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com?utm_source=medium&utm_medium=referral)

**Introduction**

In his book, [Data Science from Scratch](https://www.amazon.com/Data-Science-Scratch-Principles-Python/dp/149190142X), Joel Grus has used a very interesting example to make his readers understand the concept of Decision Trees. Since the example is too perfect, I shall quote the same. He says — As children, how many of you remember playing the game of *twenty questions*? In this game, one child would think of an animal or a place or a famous personality, etc. Others would ask questions to guess it. The game would go something like this :

“I am thinking of an animal.”

“Does it have more than five legs?”

“No”

“Is it delicious?”

“No”

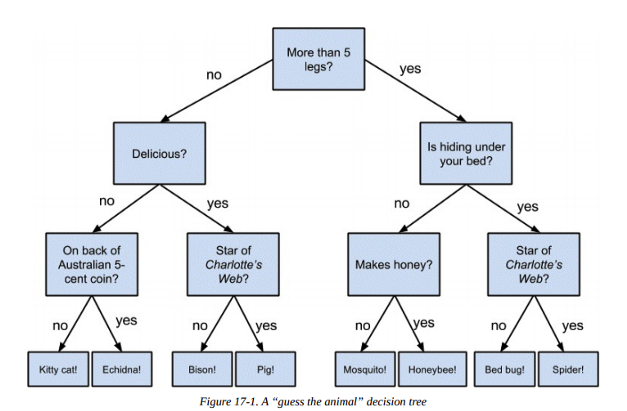
“Does it appear on the back of the Australian 5 cent coin?”

“Yes”

“Is it an echidna?”

“Yes, it is!”

Now let’s create a little elaborate graph for the “Guess the Animal “ game we just played.



Source: [Data Science from Scratch: First Principles with Python](https://www.amazon.in/Data-Science-Scratch-Joel-Grus/dp/1492041130)

This is exactly how we would create a Decision Tree for any Data Science Problem also. Now let us study in detail the math behind it.

The following article is primarily the notes I made while taking the course titled [Analytics Edge](https://www.edx.org/course/the-analytics-edge) on Edx.

**What is a Decision Tree?**

A Decision Tree is a supervised learning predictive model that uses a set of binary rules to calculate a target value. It is used for either **classification (categorical target variable)** or **regression (continuous target variable)**. Hence, it is also known as **CART** (**Classification & Regression Trees)**. Some real-life applications of Decision Trees include:

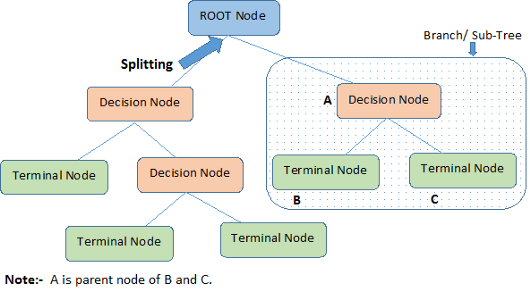
* Credit scoring models in which the criteria that cause an applicant to be rejected need to be clearly documented and free from bias
* Marketing studies of customer behavior such as satisfaction or churn, which will be shared with management or advertising agencies
* Diagnosis of medical conditions based on laboratory measurements, symptoms, or the rate of disease progression

**Structure of a Decision Tree**

Decision trees have three main parts:

* **Root Node:** The node that performs the first split. In the above “Guess the Animal” example, the root node would be the question. lives in water.
* **Terminal Nodes/Leaves:** Nodes that predict the outcome. Likewise, for the example above, terminal nodes would be bull ,cow, Lion, Tiger etc
* **Branches:** arrows connecting nodes, showing the flow from question to answer.

The root node is the starting point of the tree, and both root and terminal nodes contain questions or criteria to be answered. Each node typically has two or more nodes extending from it. For example, if the question in the first node requires a “yes” or “no” answer, there will be one leaf node for a “yes” response and another node for “no.”



PC: Analytics Vidhya

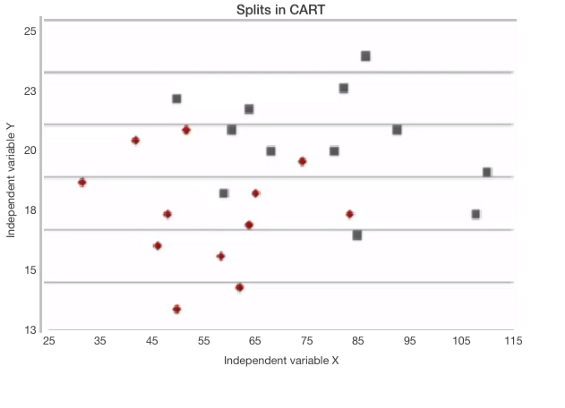
**The Algorithm behind Decision Trees.**

The algorithm of the decision tree models works by repeatedly partitioning the data into multiple sub-spaces so that the outcomes in each final sub-space is as homogeneous as possible. This approach is technically called *recursive partitioning*. The produced result consists of a set of rules used for predicting the outcome variable, which can be either:

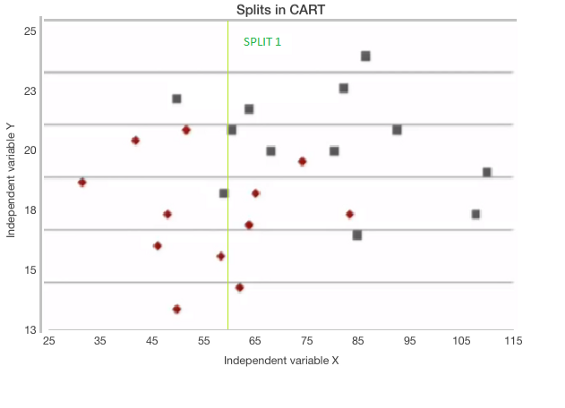
* a continuous variable, for regression trees
* a categorical variable, for classification trees

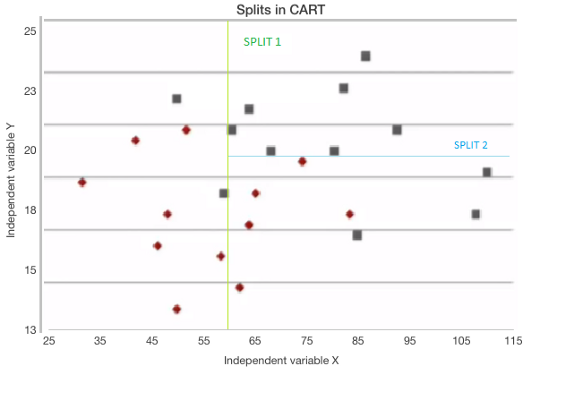
The decision rules generated by the CART (Classification & Regression Trees) predictive model are generally visualized as a binary tree.

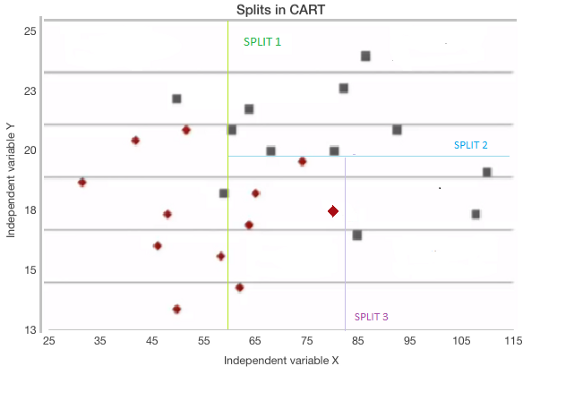
Let’s look at an example to understand it better. The plot below shows sample data for two independent variables, **x** and **y**, and each data point is colored by the outcome variable, **red** or **grey**.



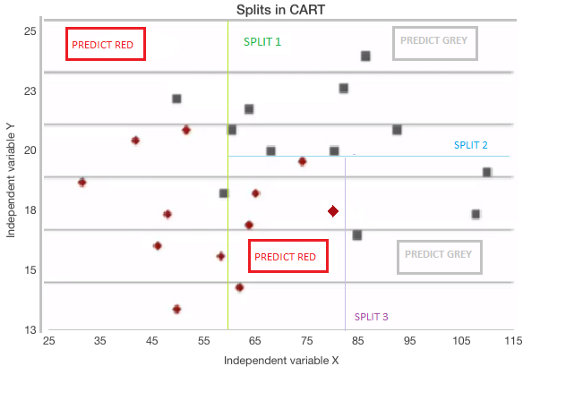
CART tries to split this data into subsets so that each subset is as pure or **homogeneous** as possible. The first three splits that CART would create are shown here.



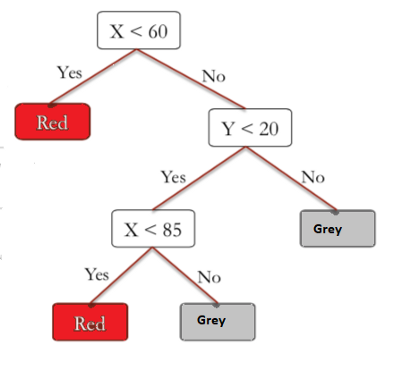




If a new observation fell into any of the subsets, it would now be decided by the majority of the observations in that particular subset.



Let us now see how a Decision Tree algorithm generates a TREE. The tree for the splits we just generated is shown below.



* The first split tests whether the variable x is less than 60. If yes, the model says to predict red, and if no, the model moves on to the next split.
* Then, the second split checks whether or not the variable y is less than 20.  
  If no, the model says to predict gray, but if yes, the model moves on to the next split.

The third split checks whether or not the variable x is less than 85. If yes, then the model says to predict red, and if no, the model says to predict grey.

**Advantages of Decision Trees**

* It is quite interpretable and easy to understand.
* It can also be used to identify the most significant variables in your data-set

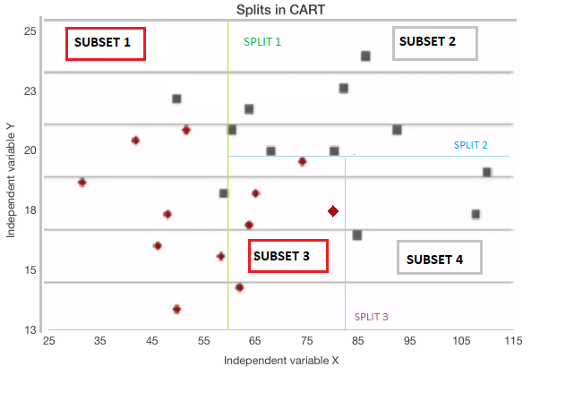
**Predictions from Decision Trees**

In the above example, we discussed C**lassification trees,** i.e., when the output is a factor/category: red or gray. Trees can also be used for regression where the output at each leaf of the tree is no longer a category but a number. They are called **Regression Trees**.

***Classification Trees:***

With Classification Trees, we report the average outcome at each leaf of our tree. However, Instead of just taking the majority outcome to be the prediction, we can compute the percentage of data in a subset of each type of outcome.

Let us understand it through the same example that we used above.



The above dataset has been split into four subsets.

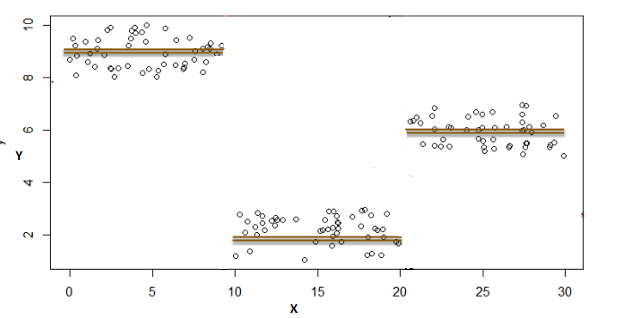
**Predictions for Subset 1:**

* Red data = 7, Grey data = 2
* % of Red data = 7/(7+2) ~ **78%** and % of Grey data ~**22%.** This means 78% of the data is Red.
* Now just like in L[ogistic Regression](https://medium.com/analytics-vidhya/a-guide-to-machine-learning-in-r-for-beginners-part-5-4c00f2366b90), we can use a threshold value to obtain our prediction.
* A Threshold of 0.5/50% corresponds to picking the most frequent outcome, which would be **Red.**
* But if we increase that threshold to 0.9/90%, we would predict **Grey**

***Regression Trees:***

To predict the outcome in such cases, since we have continuous output variables, we report the **average values** at that leaf*.* For example, if we had the values 3, 4, and 5 at one of the leaves, we will take the average, i.e., 4.

Let us see it graphically.

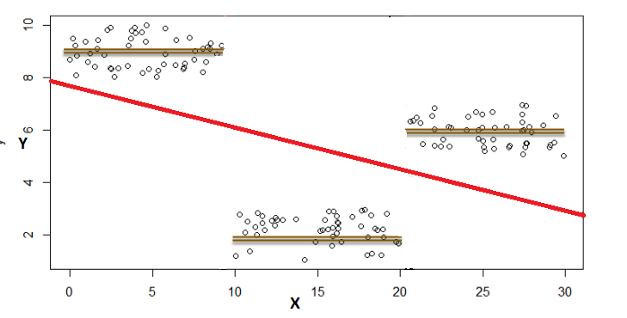


In the above graph:

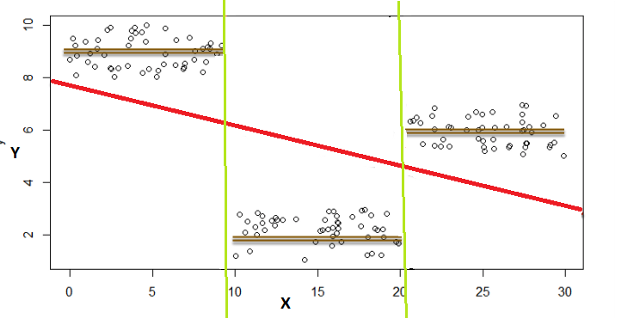
y = Outcome/target variable i.e variable we are trying to predict

x = Independent variable

Firstly, Let’s fit a **linear regression** to this data set. By doing so, we obtain a line.



As is quite evident, [linear regression](https://blog.usejournal.com/guide-to-machine-learning-in-r-for-beginners-part-4-6bacf6a82ce8) does not do very well on this data set.



However, we can notice a very interesting feature. The data lies in three different groups. If we draw lines here, we see x is less than 10, between 10 and 20, or greater than 20.

We recall that Decision Trees can fit in this kind of problem easily. So if splits are at:

* x ≤10 |output would be the average of those values.
* 10 < x ≤ 20 |output would be the average of those values.
* 20< x≤ 30 |output would be the average of those values.

**Measures Used for Split**

There are different ways to control how many splits are generated.

1. **Gini Index:** It is the measure of inequality of distribution. It says if we select two items from a population at random, then they must be of the same class and the probability for this is 1 if the population is pure.

* It works with the categorical target variable “Success” or “Failure.”
* It performs only Binary splits.
* Lower the value of Gini, the higher the homogeneity.
* CART uses the Gini method to create binary splits.

**The process to calculate Gini Measure:**

https://miro.medium.com/max/186/1*1R9K3c1I1XbGYWd9No845Q.png

where **P(j)** is the Probability of Class j

2. **Entropy**: Entropy is a way to measure impurity.

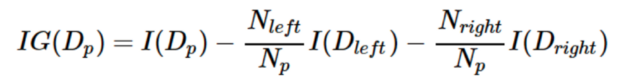
Less impure nodes require less information to describe them, and more impure nodes require more information. If the sample is completely homogeneous, then the entropy is zero, and if the sample is an equally divided one, it has an entropy of one.

https://miro.medium.com/max/210/1*x-doVbNsyB8dP5HWxd3-Og.png

3. **Information Gain:** Information Gain is simply a mathematical way to capture the amount of information one gains(or reduction in randomness) by picking a particular attribute.

In a decision algorithm, we start at the tree root and split the data on the feature that results in the largest **information gain (IG)**. In other words, IG tells us how important a given attribute is.

The **Information Gain (IG)** can be defined as follows:



Where ***I*** could be **entropy or Gini index, D** (p), D(Left), and D(Right) are the dataset of the parent, left, and right child node.

*In R, a parameter that controls this is* ***minbucket.*** *The smaller it is, the more splits will be generated However, If it is too small, overfitting will occur. And, if it is too large, model will be too simple and accuracy will be poor*

**Decision Trees in R**

We will be working on the famous Boston housing dataset. This data comes from a paper, “**Hedonic Housing Prices and the Demand for Clean Air,**” exploring the relationship between prices and clean air in the late 1970s. We will explore the boston.csv data set with the aid of trees. Here we are interested in building a model initially of how **prices vary by location across a region.**

**Dataset**

We will explore the boston.csv data set with the aid of trees. Download this file from [here](https://github.com/parulnith/A-guide-to-Machine-Learning-in-R/tree/master/Decision%20Trees) to follow along. Each entry of the dataset corresponds to a census tract. As a result, there are multiple census tracts :

**LON and LAT** are the longitude and latitude of the center of the census tract.  
**MEDV** is the median value of owner-occupied homes, measured  
in thousands of dollars.  
**CRIM** is the per capita crime rate.  
**ZN** is related to how much of the land is zoned for large residential properties.  
**INDUS** is the proportion of the area used for industry.  
**CHAS** is 1 if a census tract is next to the Charles River else 0  
**NOX** is the concentration of nitrous oxides in the air, a measure of air pollution.  
**RM** is the average number of rooms per dwelling.  
**AGE** is the proportion of owner-occupied units built before 1940.  
**DIS** is a measure of how far the tract is from centres of employment in Boston.  
**RAD** is a measure of closeness to important highways.  
**TAX** is the property tax per $10,000 of value.  
**PTRATIO** is the pupil to teacher ratio by town.

here **MEDV** is the **output /target** variable, i.e., the price of the house to be predicted. Since the output variable is continuous, it is an example of a regression tree.

**Working**

**1. Analyzing the Data**

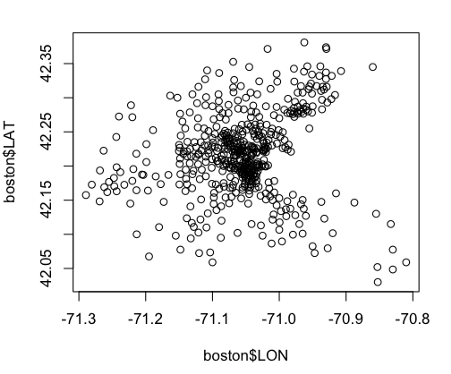
* Loading data into R console:

> boston = read.csv('boston.csv')> **str(boston)**'data.frame': 506 obs. of 16 variables:  
 $ TOWN : Factor w/ 92 levels "Arlington","Ashland",..: 54 77 77 46 46 46 69 69 69 69 ...  
 $ TRACT : int 2011 2021 2022 2031 2032 2033 2041 2042 2043 2044 ...  
 $ LON : num -71 -71 -70.9 -70.9 -70.9 ...  
 $ LAT : num 42.3 42.3 42.3 42.3 42.3 ...  
 $ MEDV : num 24 21.6 34.7 33.4 36.2 28.7 22.9 22.1 16.5 18.9 ...  
 $ CRIM : num 0.00632 0.02731 0.02729 0.03237 0.06905 ...  
 $ ZN : num 18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...  
 $ INDUS : num 2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...  
 $ CHAS : int 0 0 0 0 0 0 0 0 0 0 ...  
 $ NOX : num 0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...  
 $ RM : num 6.58 6.42 7.18 7 7.15 ...  
 $ AGE : num 65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...  
 $ DIS : num 4.09 4.97 4.97 6.06 6.06 ...  
 $ RAD : int 1 2 2 3 3 3 5 5 5 5 ...  
 $ TAX : int 296 242 242 222 222 222 311 311 311 311 ...  
 $ PTRATIO: num 15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...

There are 506 observations corresponding to 506 census tracts in the Greater Boston area. We are interested in building a model of how prices vary by location across a region. So, let’s first see how the points are laid out.  
Using the plot commands, we can plot the latitude and longitude  
of each of our census tracts.

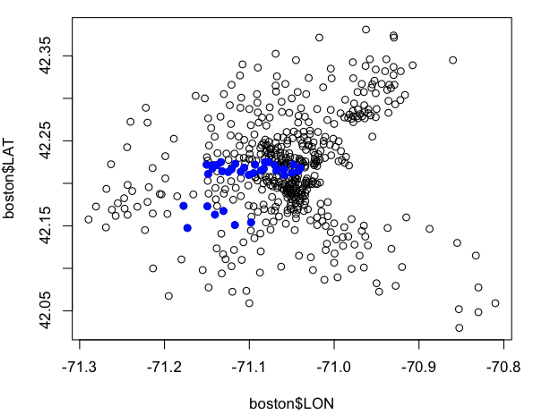
Let’s first see how the points are laid out using the plot commands.

# Plot observations  
> plot(boston$LON, boston$LAT)



The dense central core of points  
corresponds to Boston city and other urban cities.  
Since we also have the Charles river attribute(**CHAS**), we also want to show all the points that lie along the Charles River in blue color.  
  
We can do this by the points command.

# Tracts alongside the Charles River> points(boston$LON[boston$CHAS==1], boston$LAT[boston$CHAS==1], col="blue", pch=19)



Now we have plotted the tracks in Boston along Charles River.

What other things can we do?

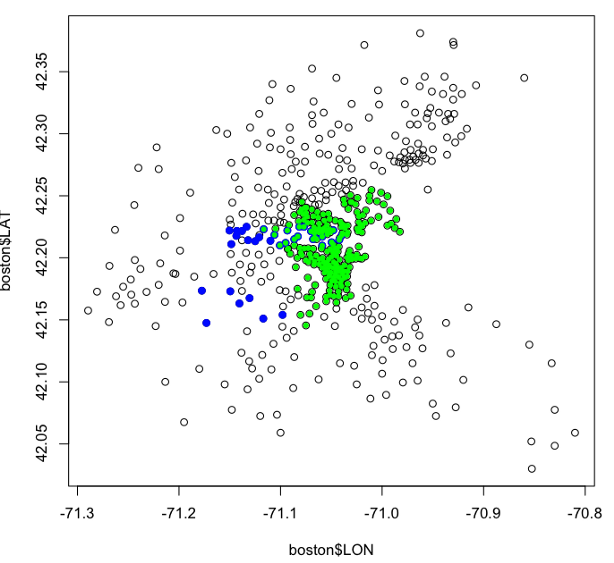
Well, this data set was originally constructed to investigate questions about how air pollution affects prices.  
The air pollution variable in the data is **NOX.** Let’s have a look at the distribution of NOX.

# Plot pollution/NOX > summary(boston$NOX) Min. 1st Qu. Median Mean 3rd Qu. Max.   
 0.3850 0.4490 0.5380 0.5547 0.6240 0.8710

The minimum value is 0.385, and the maximum value is 0.87. The median  
and the mean are about 0.53, 0.55. So, let’s use the value of 0.55,  
as it is the centermost value.

Let’s look at the tracts that have above-average pollution.

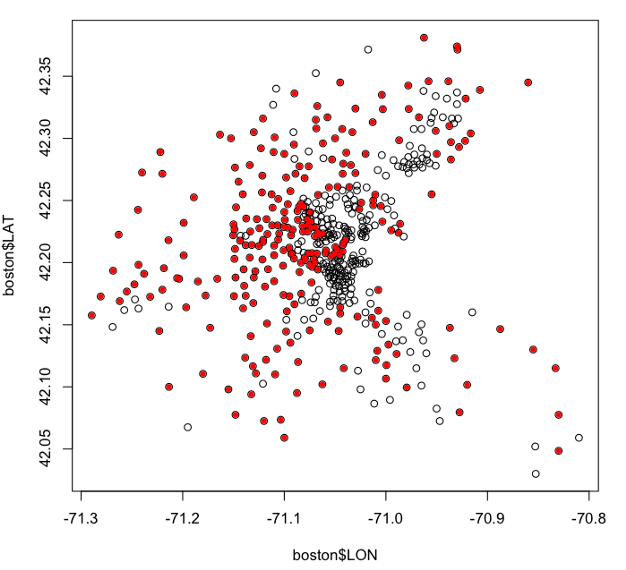
> points(boston$LON[boston$NOX>=0.55], boston$LAT[boston$NOX>=0.55], col="green", pch=20)



All the points that have got above-average pollution are colored green.  
Now it kind of makes sense since the area most densely polluted is the one that is also most densely populated.

Now let us look at how the **prices** vary over the area as well. We can do this with the help of the MEDV variable using the same methodology as done when plotting the pollution.

# Plot prices  
> plot(boston$LON, boston$LAT)> summary(boston$MEDV)  
 Min. 1st Qu. Median Mean 3rd Qu. Max.   
 5.00 17.02 21.20 22.53 25.00 50.00> points(boston$LON[boston$MEDV>=21.2], boston$LAT[boston$MEDV>=21.2], col="red", pch=20)



above median values graph

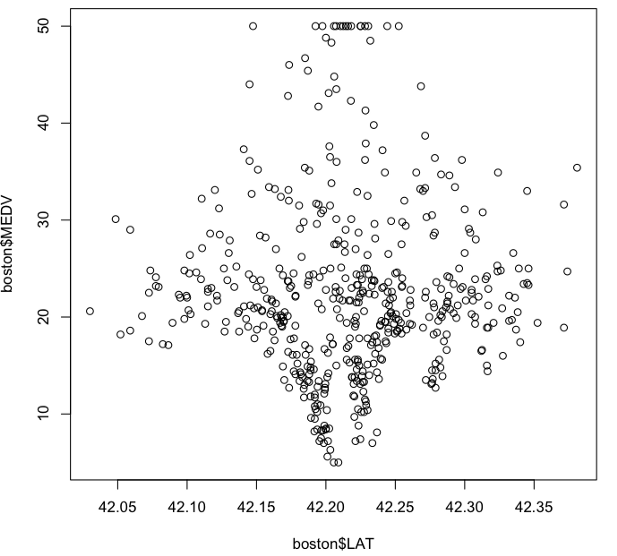
So what we see now are all the census tracts with above-average housing prices in red.  
However, the census tracts of above-average and below-average are mixed in between each other.  
But there are some patterns.  
For example, look at that dense black bit in the middle. That corresponds to most of the city of Boston, especially the southern parts of the city.  
So there’s definitely some structure to it, but it’s certainly not simple in relation to latitude and longitude, at least.

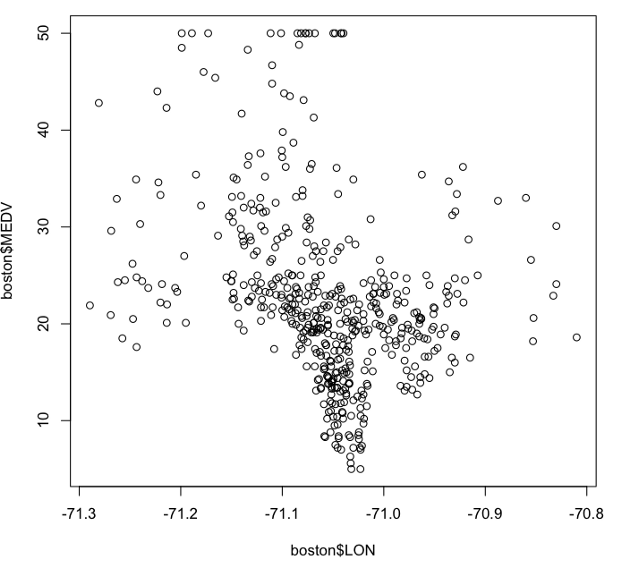
**2. Applying Linear Regression to the problem**

Since this is a regression problem as the target value to be predicted is continuous(house price), it is but natural that we look up to the Linear Regression algorithm to solve the problem. We saw in the last graph that the house prices  
were distributed over the area in an interesting way, certainly not the kind of linear way, and we feel Linear Regression is not going to work very well here. Let's back up our intuition with facts.

Here we are plotting the relationship between **latitude and house  
prices** and the **longitude and the house prices**, which look pretty nonlinear.

# Linear Regression using LAT and LON  
> plot(boston$LAT, boston$MEDV)  
> plot(boston$LON, boston$MEDV)





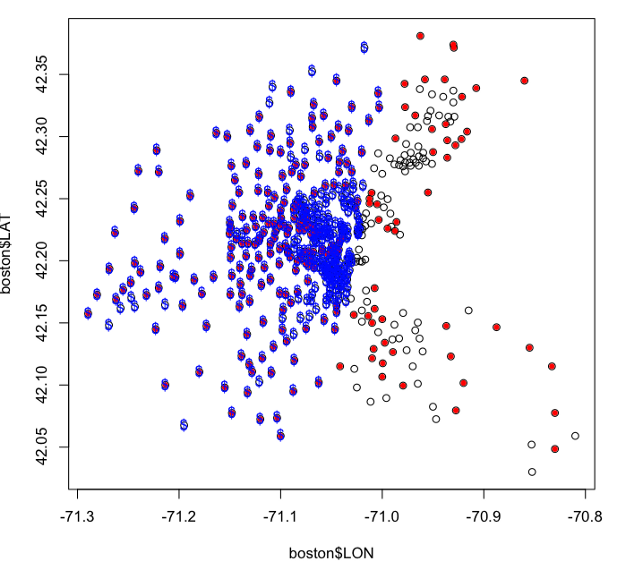
**Linear Regression Model**

> latlonlm = lm(MEDV ~ LAT + LON, data=boston)  
> summary(latlonlm)Call:  
lm(formula = MEDV ~ LAT + LON, data = boston)Residuals:  
 Min 1Q Median 3Q Max   
-16.460 -5.590 -1.299 3.695 28.129Coefficients:  
 Estimate Std. Error t value Pr(>|t|)   
(Intercept) -3178.472 484.937 -6.554 1.39e-10 \*\*\*  
LAT 8.046 6.327 1.272 0.204   
LON -40.268 5.184 -7.768 4.50e-14 \*\*\*  
---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1Residual standard error: 8.693 on 503 degrees of freedom  
Multiple R-squared: 0.1072, Adjusted R-squared: 0.1036   
F-statistic: 30.19 on 2 and 503 DF, p-value: 4.159e-13

* R-squared is around 0.1, which is not great.
* The latitude is not significant, which means the north-south location differences aren’t going to be really used at all. This also seems unlikely.
* Longitude is significant but negative, which means that house prices decrease linearly as we go towards the east, which is also unlikely.

Let’s see how this linear regression model looks on a plot. So we shall plot the census tracts again and then plot the above-median house prices with bright red dots. The red dots will tell us the actual positions in Boston where houses are costly. We shall then test the same fact with Linear Regression predictions using the blue $ sign,

# Visualize regression output  
> plot(boston$LON, boston$LAT)  
> points(boston$LON[boston$MEDV>=21.2], boston$LAT[boston$MEDV>=21.2], col="red", pch=20)> latlonlm$fitted.values  
> points(boston$LON[latlonlm$fitted.values >= 21.2], boston$LAT[latlonlm$fitted.values >= 21.2], col="blue", pch="$")



The linear regression model has plotted a dollar sign every time it thinks the census tract is above the median value. It’s almost a sharp line that the linear regression defines. Also, the shape is almost vertical since the latitude variable was not very significant in the regression. The blue $ and the red dots do not overlap, especially in the east.

It turns out; the linear regression model isn’t really doing a good job. And it has completely ignored everything to the right side of the picture. So that’s interesting and pretty wrong.

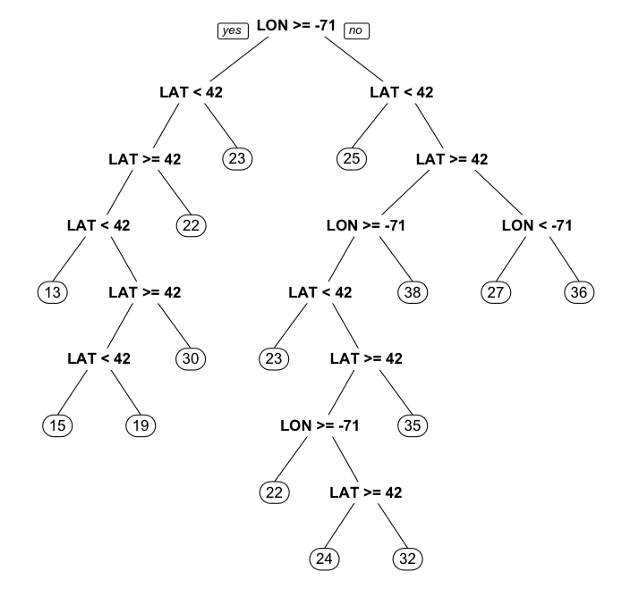
**3. Applying Regression Trees to the problem**

We’ll first load the rpart library and also install and load the rpart plotting library.

# Load CART packages  
> library(rpart)# install rpart package  
> install.packages("rpart.plot")  
> library(rpart.plot)

We will build a regression tree in the same way we would build a classification tree, using the part command. We would be predicting MEDV as a function of latitude and longitude, using the boston dataset.

# CART model  
> latlontree = rpart(MEDV ~ LAT + LON, data=boston)# Plot the tree using prp command defined in rpart.plot package  
> prp(latlontree)



Regression Tree

The **leaves** of the tree are important.

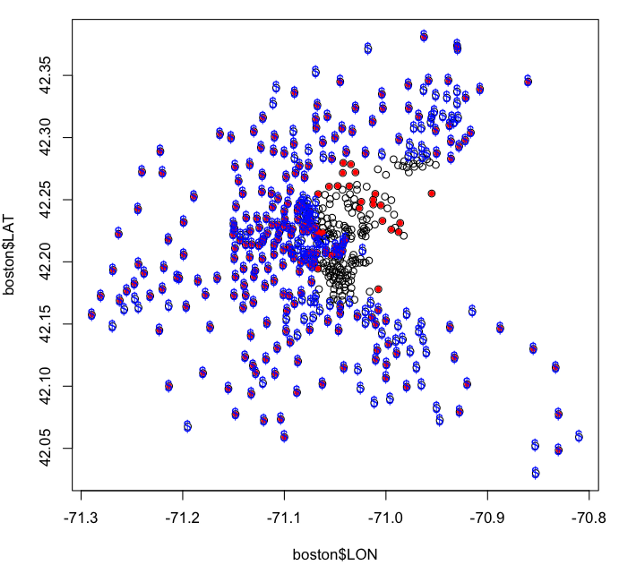
* In a **classification tree**, the leaves would be the classification that we assign.
* In **regression trees**, we instead predict the number. That number here is the average of the median house prices in that bucket.

Now, let us visualize the output. We’ll again plot the points with above-median prices just like in Linear Regression.

# Visualize output> plot(boston$LON, boston$LAT)  
>points(boston$LON[boston$MEDV>=21.2],boston$LAT[boston$MEDV>=21.2], col="red", pch=20)

The above plot is of actual known prices. It is the same plot that we observed with red dots. We want to predict what the tree thinks is above the median house price. So we’ll name those values as fitted values obtained from using the predict command on the tree we just built.

> fittedvalues = predict(latlontree)  
>points(boston$LON[fittedvalues>21.2],boston$LAT[fittedvalues>=21.2], col="blue", pch="$")



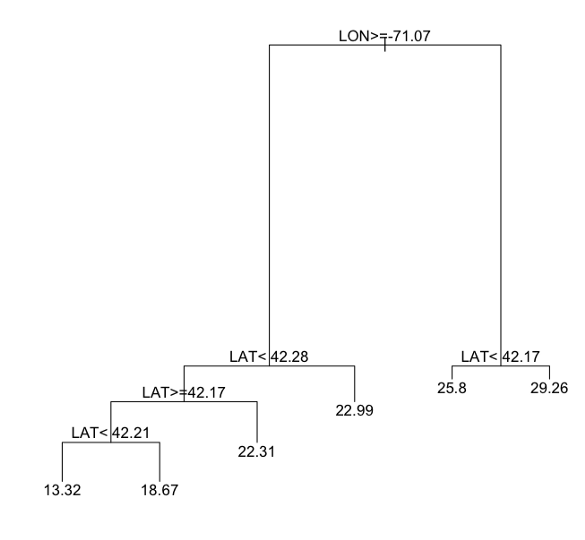
The fitted values are greater than 21.2, the color is blue, and the character is a $ to signify Price

The Regression tree has done a much better job and has kind of overlapped the red dots. It has left the low-value area in Boston out and has correctly managed to classify some of those points  
in the bottom right and top right.  
`

But the tree obtained was very complicated and was **overfitted**.  
How to avoid overfitting? By changing the minbucket size.  
So let’s build a new tree using the rpart command again.

**Simplifying Tree by increasing minbucket**

> latlontree = rpart(MEDV ~ LAT + LON, data=boston, minbucket=50)  
> plot(latlontree)  
> text(latlontree)



tree with minbucket = 50

Here we have far fewer splits, and it’s far more interpretable.

We have seen that regression trees can do what we would never expect linear regression to do. Now let’s see how the regression trees will help us build a predictive model and predict house prices?

**4. Prediction with Regression Trees**

We’re going to try to predict house prices using all the variables we have available to us.

**Steps:**

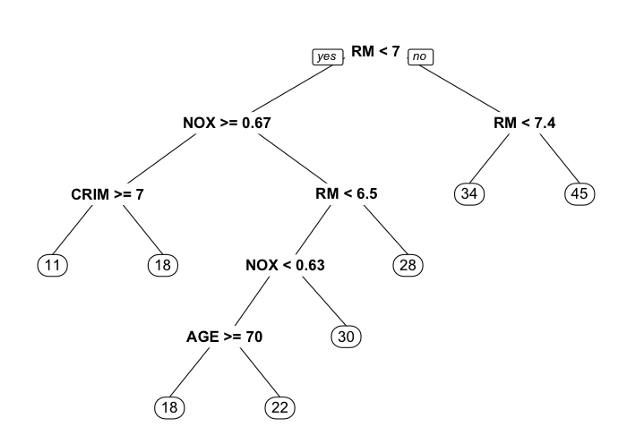
* Split the data into the **Training** and **Testing** set using caTools library. We shall then set the seed, so our results are reproducible.

# Split the data  
> library(caTools)  
> set.seed(123)  
> split = sample.split(boston$MEDV, SplitRatio = 0.7)  
> train = subset(boston, split==TRUE)  
> test = subset(boston, split==FALSE)

Our training data is a subset of the Boston data where the split is TRUE.  
And the testing data is the subset of the Boston data where the split is FALSE

* **Making a Regression Tree Model**

# Create a CART model  
> tree = rpart(MEDV ~ LAT + LON + CRIM + ZN + INDUS + CHAS + NOX + RM + AGE + DIS + RAD + TAX + PTRATIO, data=train)  
> prp(tree)



**Results:**

* Latitude and Longitude aren’t significant
* **The rooms are the most important split.**
* Pollution appears in there twice, so it’s, in some sense, nonlinear on the amount of pollution, i.e., if it’s greater than a certain amount or less than a certain amount, it does different things.
* Very nonlinear on the number of rooms.

**Regression Tree Predictions**

> tree.pred = predict(tree, newdata=test)  
> tree.sse = sum((tree.pred - test$MEDV)^2)  
> tree.sse4328.988

**Conclusion**

Even though the Decision Trees appears to be working very well in certain conditions, it comes with its own perils. The model has very high chances of **“over-fitting.”** Infact it is the key challenge in the case of Decision Trees. If no limit is set, it will end up putting each observation into a leaf node in the worst case.

Some techniques help improve the performance of **Decision Trees,** but wewill learn about them but in the next article. We will learn to improve the results of our Decision Trees using the **“cp” parameter.** Also, we will try and implement Cross-Validation**,** a technique to avoid overfitting in predictive models. Finally, we will dive into **Ensembling,** i.e., combining the results of multiple models to solve a given prediction or classification problem.

So stay tuned for the next part.