Signitive TECHNOLOGIES

Signify the Learning

Certification Training Courses for Graduates and Professionals

www.signitivetech.com

Classification
Part 1

Parametric vs Nonparametric

Parametric v/s Non Parametric ML Algos

Machine Learning Function:

- Machine learning can be summarized as learning a function (f) that maps input variables (X) to output variables (Y).

$$Y = f(x)$$

- An algorithm learns this target mapping function from training data.
- The form of the function is unknown, so our job as machine learning practitioners is to evaluate different machine learning algorithms and see which is better at approximating the underlying function.
- Different algorithms make different assumptions or biases about the form of the function and how it can be learned.
- There are two types of Machine Learning Algorithms:
    - Parametric
    - Non Parametric

Parametric v/s Non Parametric ML Algos

Parametric Machine Learning Algorithms:

- Assumptions can greatly simplify the learning process, but can also limit what can be learned. Algorithms that simplify the function to a known form are called parametric machine learning algorithms.
- *A learning model that summarizes data with a set of parameters of fixed size (independent of the number of training examples) is called a parametric model. No matter how much data you throw at a parametric model, it won’t change its mind about how many parameters it needs.*
- The algorithms involve two steps:
  1. Select a form for the function.
  2. Learn the coefficients for the function from the training data.

Parametric v/s Non Parametric ML Algos

- Functional form for the mapping function is a line, in linear regression:

$$b0 + b1*x1 + b2*x2 = 0$$

- Where b0, b1 and b2 are the coefficients of the line that control the intercept and slope, and x1 and x2 are two input variables.
- Assuming the functional form of a line greatly simplifies the learning process. Now, all we need to do is estimate the coefficients of the line equation and we have a predictive model for the problem.
- Often the assumed functional form is a linear combination of the input variables and as such parametric machine learning algorithms are often also called "*linear machine learning algorithms*".
- The problem is, the actual unknown underlying function may not be a linear function like a line. It could be almost a line and require some minor transformation of the input data to work right. Or it could be nothing like a line in which case the assumption is wrong and the approach will produce poor results.

Parametric v/s Non Parametric ML Algos

Some more examples of parametric machine learning algorithms include:

➔ Logistic Regression
➔ Naive Bayes
➔ Simple Neural Networks

Benefits of Parametric Machine Learning Algorithms:

- **Simpler**: These methods are easier to understand and interpret results.
- **Speed**: Parametric models are very fast to learn from data.
- **Less Data**: They do not require as much training data and can work well even if the fit to the data is not perfect.

Limitations of Parametric Machine Learning Algorithms:

- **Limited Complexity**: The methods are more suited to simpler problems.
- **Poor Fit**: In practice the methods are unlikely to match the underlying mapping function.

Parametric v/s Non Parametric ML Algos

Non Parametric Machine Learning Algorithms:

- Algorithms that do not make strong assumptions about the form of the mapping function are called nonparametric machine learning algorithms. By not making assumptions, they are free to learn any functional form from the training data.
- Nonparametric methods are good when you have a lot of data and no prior knowledge, and when you don't want to worry too much about choosing just the right features.
- Nonparametric methods seek to best fit the training data in constructing the mapping function, while maintaining some ability to generalize to unseen data. So, they are able to fit a large number of functional forms.
- The KNN algorithm that makes predictions based on the k most similar training patterns for a new data instance. The method does not assume anything about the form of the mapping function other than patterns that are close are likely have a similar output variable.

Parametric v/s Non Parametric ML Algos

Some more examples of nonparametric machine learning algorithms are:

➔ K-Nearest Neighbors
➔ Decision Trees like CART
➔ Support Vector Machines

Benefits of Nonparametric Machine Learning Algorithms:

- **Flexibility**: Capable of fitting a large number of functional forms.
- **Power**: No assumptions (weak assumptions) about the underlying function.
- **Performance**: Can result in higher performance models for prediction.

Limitations of Nonparametric Machine Learning Algorithms:

- **More data**: Require a lot more train data to estimate the mapping function.
- **Slower**: Slower to train as they often have far more parameters to train.
- **Overfitting**: More of a risk to overfit the training data and it is harder to explain why specific predictions are made.

K Nearest Neighbors

K Nearest Neighbors

KNN Model Representation:

- KNN uses all of the data for training while classifying a new data point or instance. KNN is a non-parametric learning algorithm, which means that it doesn't assume anything about the underlying data.
- KNN has no model other than storing the entire dataset, so there is no learning required.
- Because the entire training dataset is stored, you may want to think carefully about the consistency of your training data. It might be a good idea to curate it, update it often as new data becomes available and remove erroneous and outlier data.
- The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

K Nearest Neighbors

Making Predictions with KNN:

- KNN makes predictions using the training dataset directly.
- Predictions are made for a new instance (x) by searching through the entire training set for the K most similar instances (the neighbors) and summarizing the output variable for those K instances. For regression this might be the mean output variable, in classification this might be the mode (or most common) class value.
- To determine which of the K instances in the training dataset are most similar to a new input a distance measure is used. For real-valued input variables, the most popular distance measure is Euclidean distance.
- Popular distance measures:
  - Euclidean Distance
  - Manhattan Distance
  - Minkowski Distance

K Nearest Neighbors

Euclidean Distance:

$$d(x, y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Manhattan Distance:

$$d(x, y) = \sum_{i=1}^{m} | x_i - y_i |$$

Minkowski Distance:

$$d(x, y) = \left[\sum_{i=1}^{m} | x_i - y_i |^p \right]^{1/p}$$

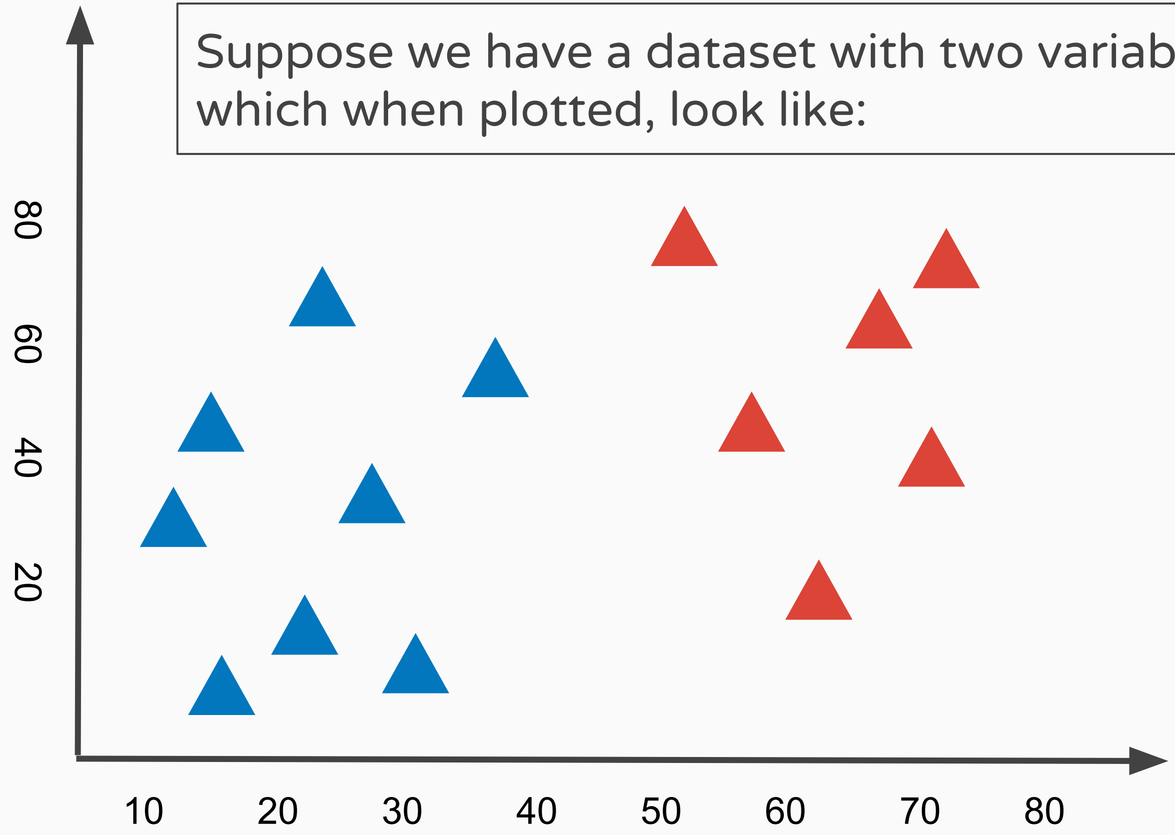
When p = 1: Distance Metric: Manhattan and When p = 2: Distance Metric: Euclidean

K Nearest Neighbors

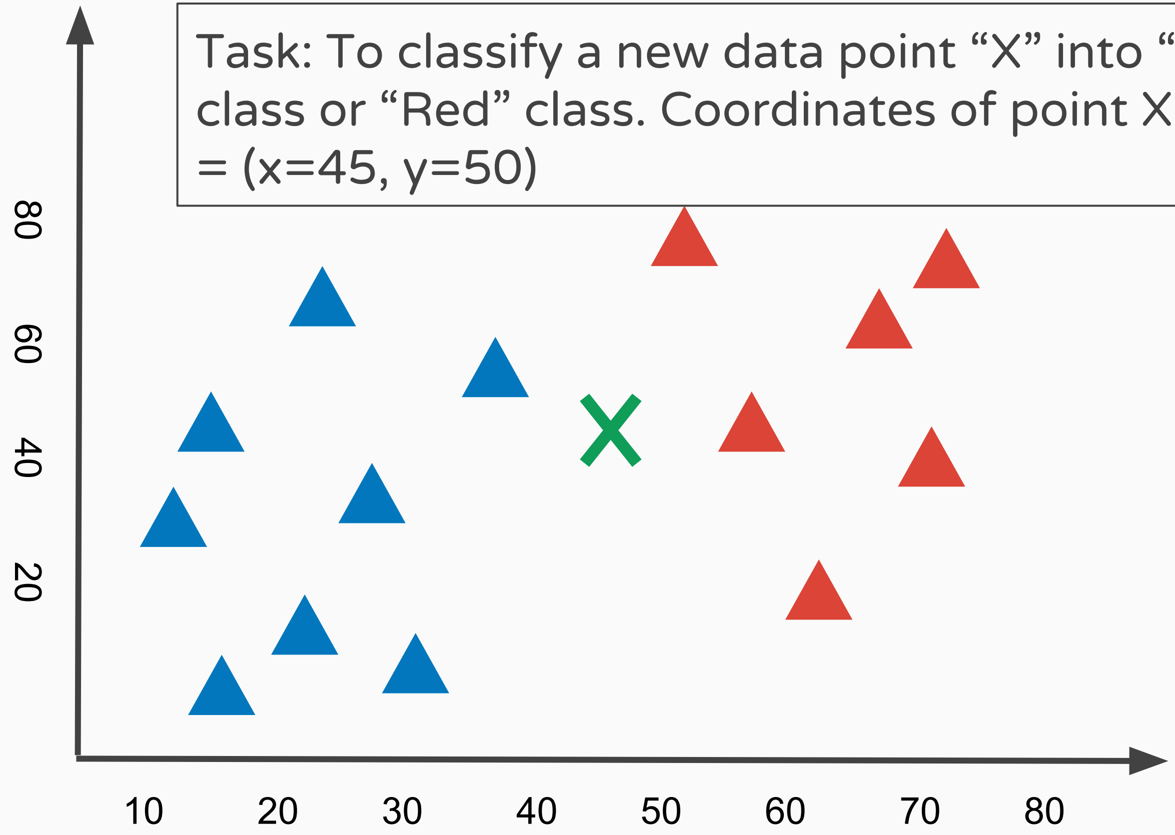
- We can choose the best distance metric based on the properties of your data. If we are unsure, we can experiment with different distance metrics and different values of K together and see which mix results in the most accurate models.
- Euclidean is a good distance measure to use if the input variables are similar in type (e.g. all measured widths and heights). Manhattan distance is a good measure to use if the input variables are not similar in type (such as age, gender, height, etc.).
- The value for K can be found by algorithm tuning. It is a good idea to try many different values for K (e.g. values from 1 to 21) and see what works best for your problem.

K Nearest Neighbors

Suppose we have a dataset with two variables, which when plotted, look like:

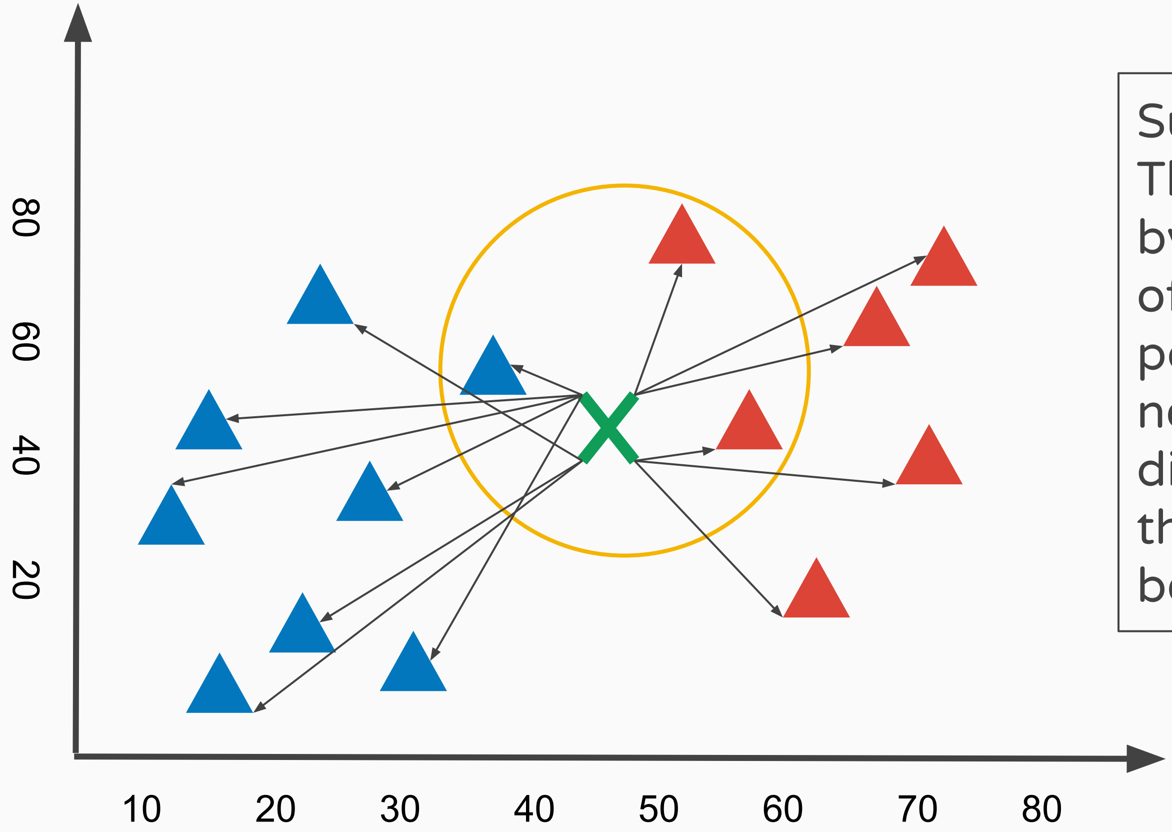


K Nearest Neighbors



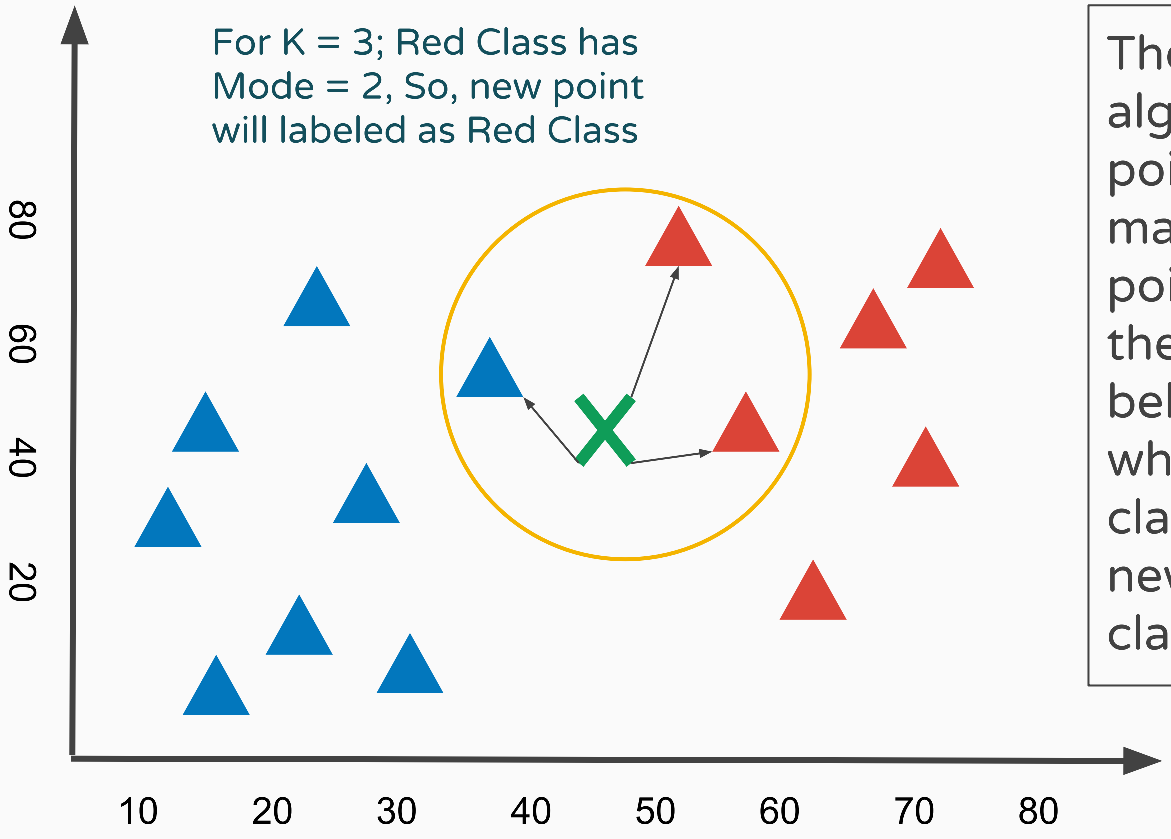
Task: To classify a new data point “X” into “Blue” class or “Red” class. Coordinates of point X are: X = (x=45, y=50)

K Nearest Neighbors



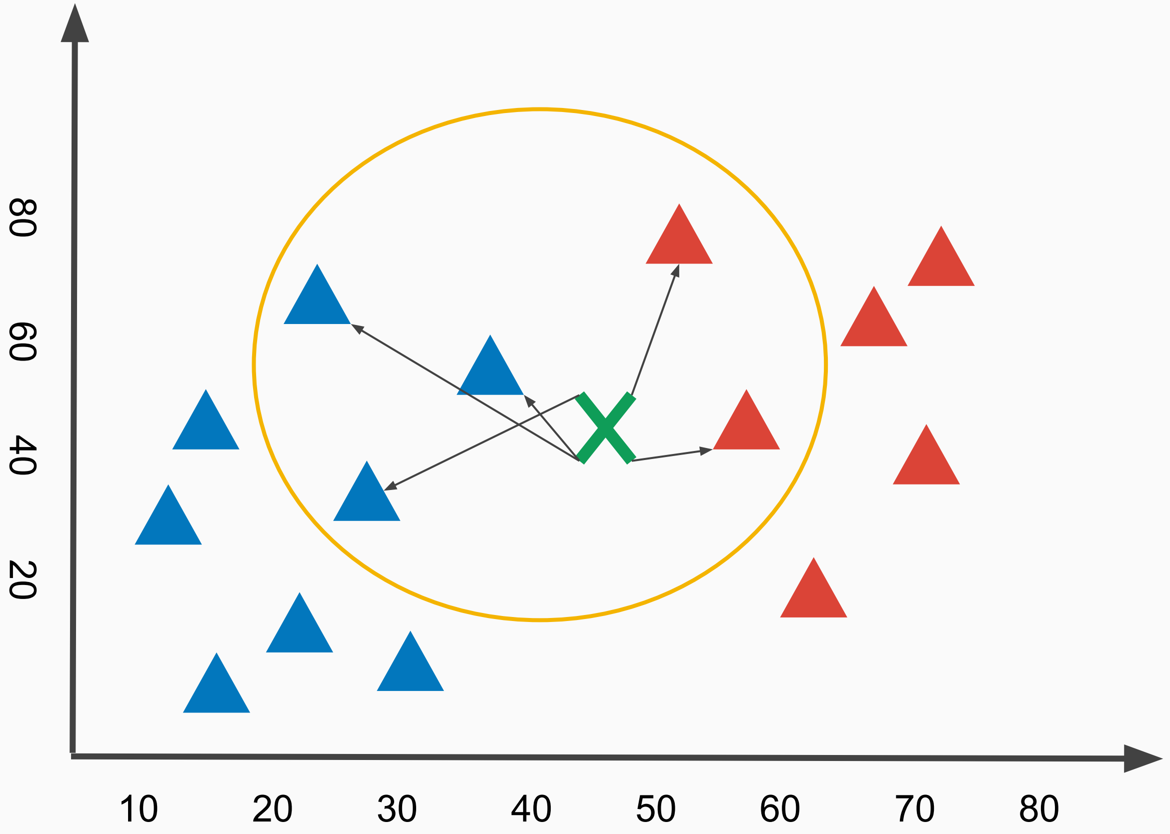
Suppose the value of K = 3. The KNN algorithm starts by calculating the distance of point X from all the points. It then finds the 3 nearest points with least distance to point X.The three nearest points have been encircled.

K Nearest Neighbors


For K = 3; Red Class has Mode = 2, So, new point will labeled as Red Class
80
60
40
20
10
20
30
40
50
60
70
80

The final step of the KNN algorithm is to assign new point to the class to which majority of the three nearest points belong. The two of the three nearest points belong to the class "Red" while one belongs to the class "Blue". Therefore the new data point will be classified as "Red".

K Nearest Neighbors



For K = 5; Blue Class has Mode = 3, So, new point will labeled as Blue Class

K Nearest Neighbors

- Suppose we have height, weight and T-shirt size of some customers and we need to predict the T-shirt size of a new customer given only height and weight information we have

Height(Cm)	Weight(Kg)	T-Shirt Size
158	58	M
158	59	M
158	63	M
160	59	M
160	60	M
163	60	M
163	61	M
160	64	L
163	64	L
165	61	L
165	62	L
165	65	L
168	62	L
168	66	L
170	64	L

K Nearest Neighbors

Step 1: Calculate Similarity based on distance function:

- Using Euclidean or Manhattan distance: Both are used for continuous data.
- The idea to use distance measure is to find the distance (similarity) between new sample and training cases and then finds the k-closest customers to new customer in terms of height and weight.

Step 2: Find K-Nearest Neighbors:

- **Let k be 5.** Then the algorithm searches for the 5 customers closest to Sameer, i.e. the most similar to Sameer in terms of attributes, and see what categories those 5 customers were in.

K Nearest Neighbors

- New customer Sameer has height 161 cm and weight 61 kg.
- Euclidean Distance between first observation and new observation is:

Euclidean Distance:

$$d(x, y) = \sqrt{(161 - 158)^2 + (61 - 58)^2}$$

$$d(x, y) = \mathrm{sqrt}(3^2 + 3^2) = \mathrm{sqrt}(9 + 9) = \mathrm{sqrt}(18) = 4.25$$

K Nearest Neighbors

- All Euclidean Distances between all observations and new observation.

Height(Cm)	Weight(Kg)	T-Shirt Size	Distance
158	58	M	4.2
158	59	M	3.6
158	63	M	3.6
160	59	M	2.2
160	60	M	1.4
163	60	M	2.2
163	61	M	2.0
160	64	L	3.2
163	64	L	3.6
165	61	L	4.0
165	62	L	4.1
165	65	L	5.7
168	62	L	7.1
168	66	L	8.6
170	64	L	9.4

K Nearest Neighbors

- KNN Algorithm will find nearest 5 neighbors from Sameer having least Euclidean Distances. It seems M has 4 values and L have 1 value, so Sameer will fit into Medium T-Shirt Size.

Height(Cm)	Weight(Kg)	T-Shirt Size	Distance
158	58	M	4.2
158	59	M	3.6
158	63	M	3.6
160	59	M	2.2 (3)
160	60	M	1.4 (1)
163	60	M	2.2 (3)
163	61	M	2.0 (2)
160	64	L	3.2 (4)
163	64	L	3.6
165	61	L	4.0
165	62	L	4.1
165	65	L	5.7
168	62	L	7.1
168	66	L	8.6
170	64	L	9.4

K Nearest Neighbors

KNN for Regression:

- When KNN is used for regression problems the prediction is based on the mean or the median of the K-most similar instances.

KNN for Classification:

- When KNN is used for classification, the output can be calculated as the class with the highest frequency from the K-most similar instances. Each instance in essence votes for their class and the class with the most votes is taken as the prediction.
- If you are using K and you have an even number of classes, it is a good idea to choose a K value with an odd number to avoid a tie. And the inverse, use an even number for K when you have an odd number of classes.
- Ties can be broken consistently by expanding K by 1 and looking at the class of the next most similar instance in the training dataset.

K Nearest Neighbors

Choosing the right value for K:

- As we decrease the value of K to 1, our predictions become less stable.
- Inversely, as we increase the value of K, our predictions become more stable due to majority voting / averaging, and thus, more likely to make more accurate predictions (up to a certain point). Eventually, we begin to witness an increasing number of errors.
- For small values of **k**, the classification becomes locally sensitive, i.e. training samples similar to the test sample heavily dominate its classification. The decision boundary, or rather decision surface becomes quite jittery.
- For large values of **k**, the local sensitivity of the classification decreases, i.e. training samples less-like the test sample also affect its classification. The decision surface becomes smoother.

K Nearest Neighbors

Choosing the right value for K:

If we have EVEN Number of Classes, Use ODD value for K

If we have ODD Number of Classes, Use EVEN value for K

If there is TIE i.e. suppose for K = 6 we get 3 Points for Class A and 3 Points for Class B, KNN will randomly choose the Class in such a TIE Case. So be careful while choosing the K VALUE

K Nearest Neighbors

Lazy Learning:

- No learning of the model is required and all of the work happens at the time a prediction is requested. As such, KNN is often referred to as a lazy learning algorithm.
- KNN does not use the training data points to do any generalization. In other words, there is no explicit training phase or it is very minimal.
- This also means that the training phase is pretty fast. Lack of generalization means that KNN keeps all the training data.
- To be more exact, all (or most) the training data is needed during the testing phase.

K Nearest Neighbors

Best Prepare Data for KNN:

- **Rescale Data**: KNN performs much better if all of the data has the same scale. Normalizing your data to the range [0, 1] is a good idea. It may also be a good idea to standardize your data if it has a Gaussian distribution.
- **Address Missing Data**: Missing data will mean that the distance between samples can not be calculated. These samples could be excluded or the missing values could be imputed.
- **Lower Dimensionality**: KNN is suited for lower dimensional data. You can try it on high dimensional data (hundreds or thousands of input variables) but be aware that it may not perform as well as other techniques. KNN can benefit from feature selection that reduces the dimensionality of the input feature space.

K Nearest Neighbors

Pros and Cons of KNN:

Pros:

- No assumptions about data. So, useful for nonlinear data
- Simple algorithm to explain, understand and interpret
- High accuracy (relatively): It is pretty high but not competitive in comparison to better supervised learning models
- Versatile: Because It is useful for classification or regression

Cons:

- Computationally expensive because the algorithm stores all training data
- High memory requirement
- Stores all (or almost all) of the training data
- Prediction stage might be slow (with big N)
- Sensitive to irrelevant features and the scale of the data

K Nearest Neighbors

Steps to Perform KNN using Scikit-Learn:

- Import required libraries
- Load the Dataset
- Do the Data preprocessing
- Separate features and target
- Feature Scaling using Standard Scalar
- Split the data into train and test part

Note: KNN calculates the distance between two points by the Euclidean distance. If one of the features has a broad range of values, the distance will be governed by this particular feature. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance.

K Nearest Neighbors

Steps to Perform KNN using Scikit-Learn:

- Train the Algorithm: Import the KNeighborsClassifier class, instantiate it with one parameter i.e n_neighbours (This is basically the value for K), and call the fit() method along with our training data.

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors = 5)
model.fit(X_train, Y_train)
```

- Make the Predictions: To make predictions, use testing data.

```
y_predicted = model.predict(x_test)
```

K Nearest Neighbors

Steps to Perform KNN using Scikit-Learn:

- Check the Accuracy: Check Model Accuracy

```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_predicted)
```

- Create Confusion Matrix: 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.

```
from sklearn.metrics import confusion_matrix

confusion_matrix = confusion_matrix(y_test, y_predicted
```

K Nearest Neighbors

Steps to Perform KNN using Scikit-Learn:

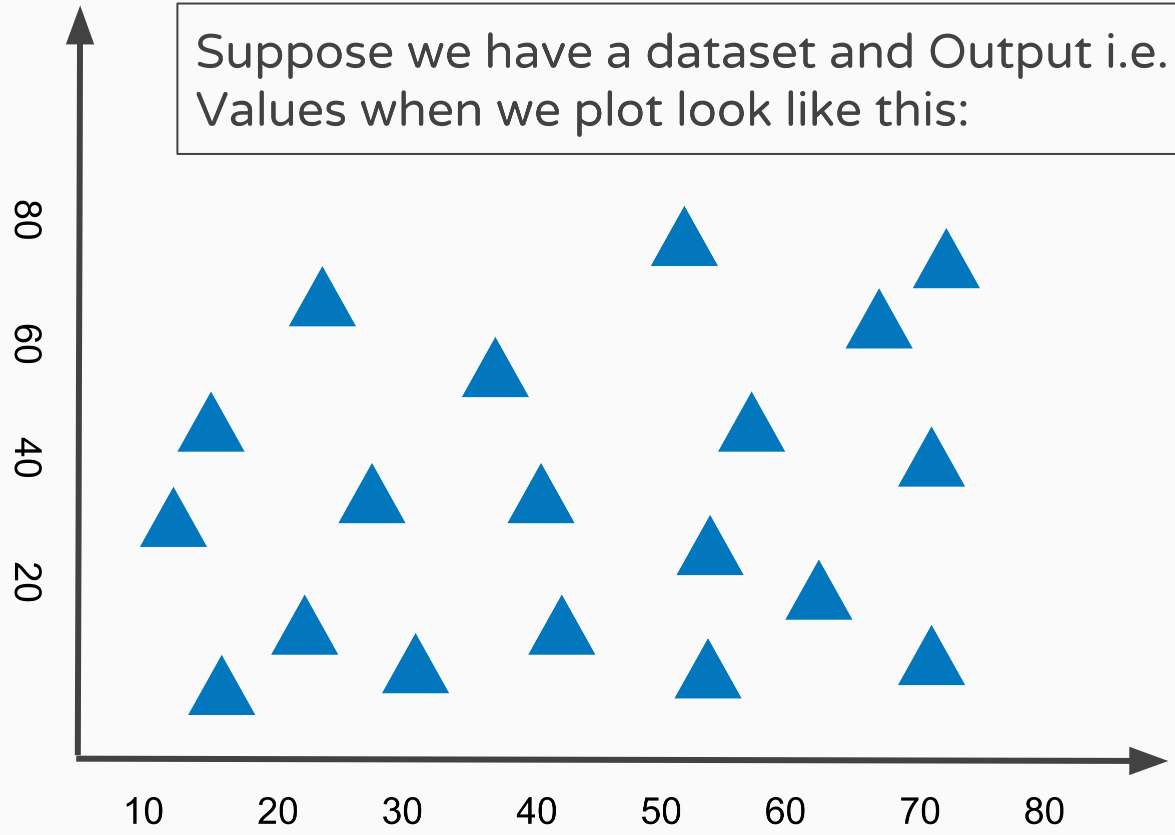
- Compute Precision, Recall, F-Score and Support: Classification report is used to evaluate a model’s predictive power.

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_predicted))
```

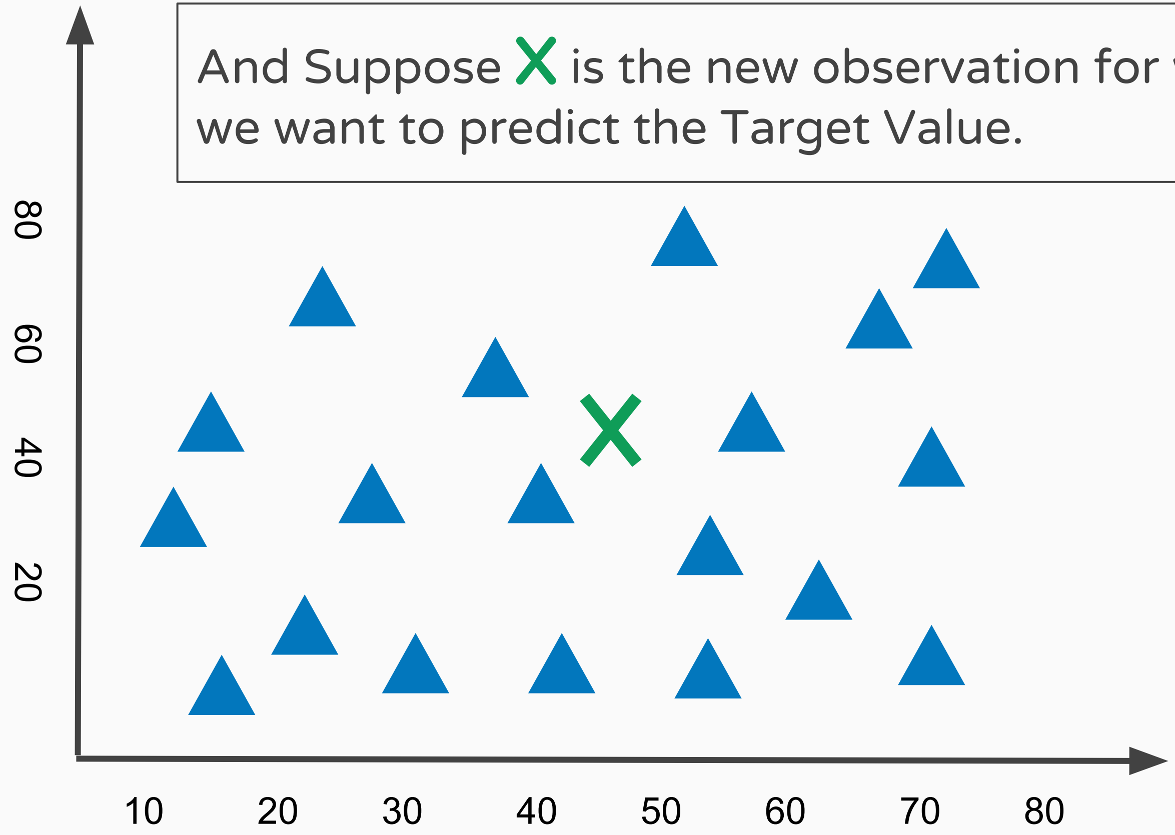
K Nearest Neighbors: Regression

Suppose we have a dataset and Output i.e. Target Values when we plot look like this:



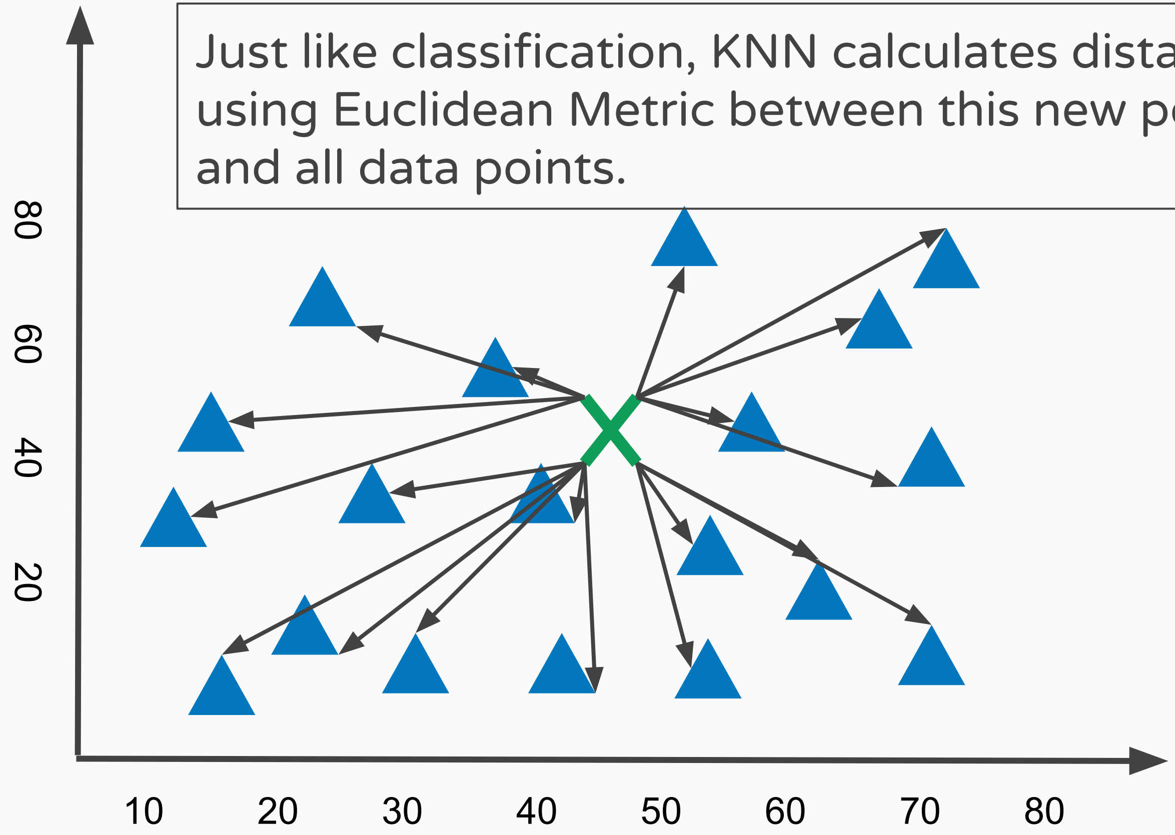
K Nearest Neighbors: Regression

And Suppose X is the new observation for which we want to predict the Target Value.



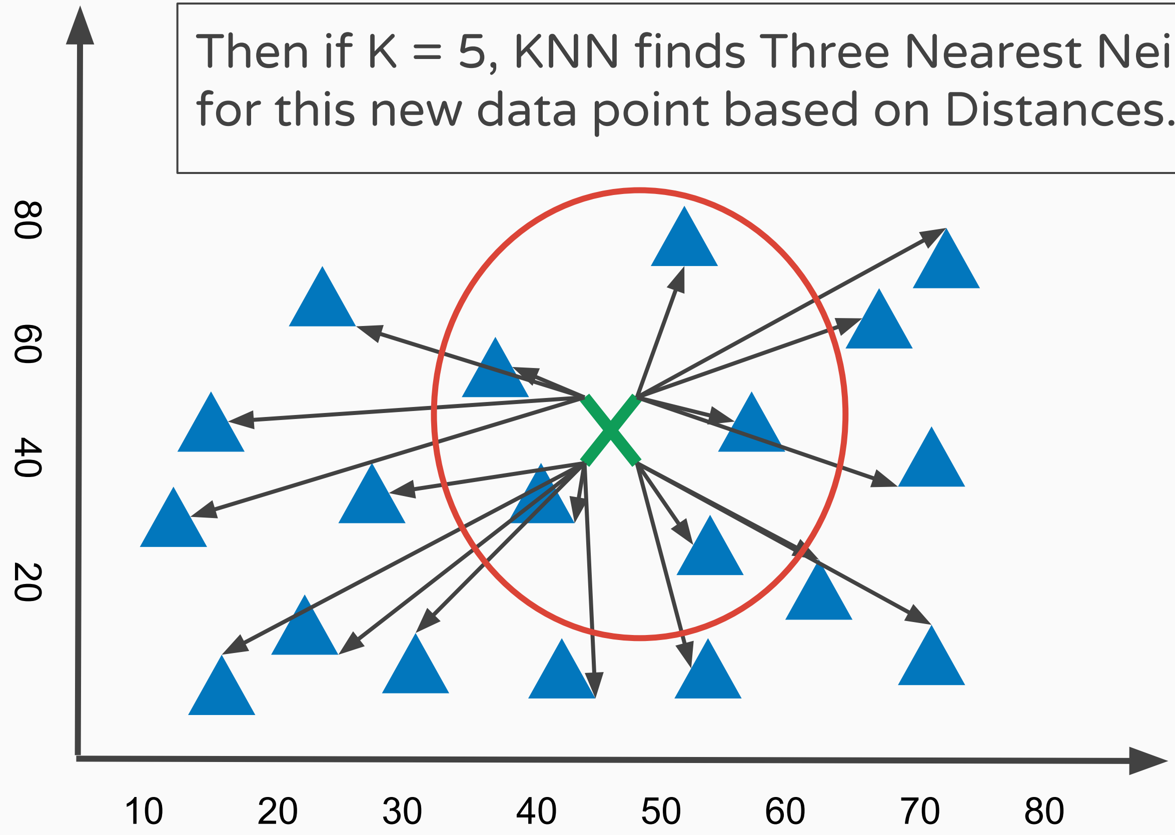
K Nearest Neighbors: Regression

Just like classification, KNN calculates distance using Euclidean Metric between this new point and all data points.

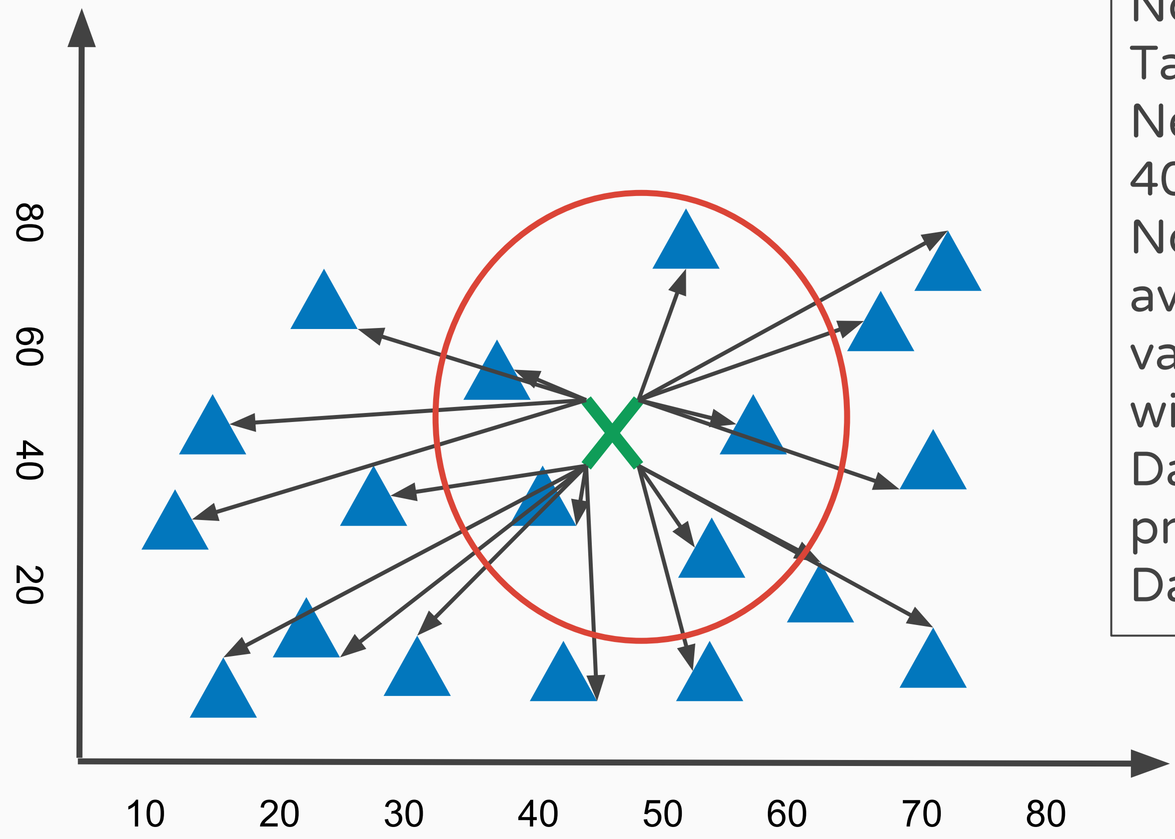


K Nearest Neighbors: Regression

Then if K = 5, KNN finds Three Nearest Neighbors for this new data point based on Distances.

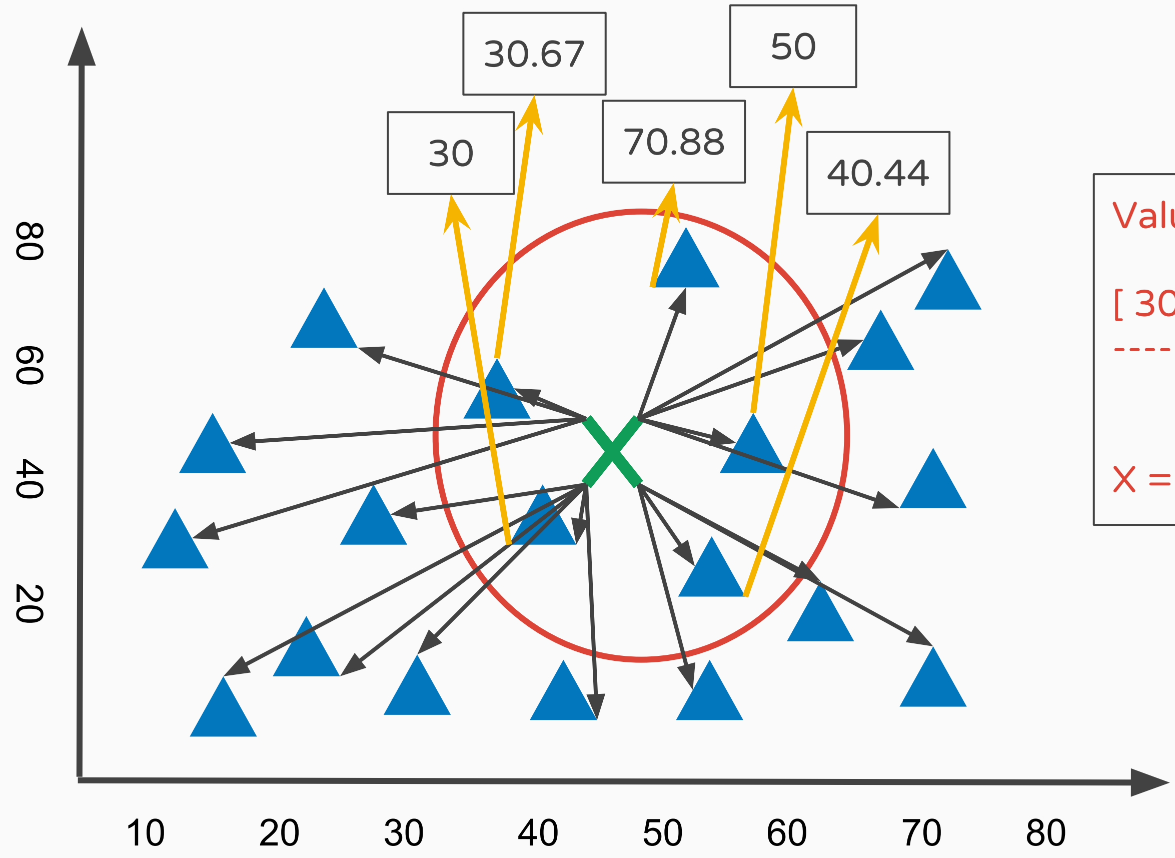


K Nearest Neighbors: Regression



Now Suppose values or Target for these 5 Nearest Neighbors are 30, 30.67, 40.44, 50 and 70.88
Now KNN will calculate average or mean of these values and respective mean will be assigned to New Data Point which will be a predicted value of our New Data Point.

K Nearest Neighbors: Regression



Value_for_New_Data_Point X:

$$\frac{[\ 30 + 30.67 + 70.88 + 50 + 40.44\ ]}{5}$$

X = 32.26

K Nearest Neighbors: Regression

Steps to Perform KNN Regression using Scikit-Learn:

- Import required libraries
- Load the Dataset
- Check the relationship between Independent and Dependent Variables
- Do the Data preprocessing
- Separate features and target
- Feature Scaling using Standard Scalar
- Split the data into train and test part

Note: KNN calculates the distance between two points by the Euclidean distance. If one of the features has a broad range of values, the distance will be governed by this particular feature. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance.

K Nearest Neighbors: Regression

Steps to Perform KNN Regression using Scikit-Learn:

- Train the Algorithm: Import the KNeighborsRegressor class, instantiate it with one parameter i.e n_neighbours (This is basically the value for K), and call the fit() method along with our training data.

```
from sklearn.neighbors import KNeighborsRegressor
model = KNeighborsRegressor(n_neighbors = 5)
model.fit(X_train, Y_train)
```

- Make the Predictions: To make predictions, use testing data.

```
y_predicted = model.predict(x_test)
```

- Evaluate the Model using MSE and R2 Score Value

Evaluation Metrics

Evaluation Metrics

Confusion Matrix:

- A confusion matrix is a technique for summarizing the performance of a classification algorithm.
- Calculating a confusion matrix can give you a better idea of what your classification model is getting right and what types of errors it is making.
- A confusion matrix is a summary of prediction results on a classification problem.
- The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix.
- **The confusion matrix shows the ways in which your classification model is confused when it makes predictions.**
- It gives you insight not only into the errors being made by your classifier but more importantly the types of errors that are being made.

Evaluation Metrics

Confusion Matrix:

Total = TP + TN + FP + FN	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	True Positive TP	False Negative FN
Class 2 Actual	False Positive FP	True Negative TN

Evaluation Metrics

Here,

- Class 1 : Positive
- Class 2 : Negative

Definition of the Terms:

- Positive (P) : Observation is positive.
- Negative (N) : Observation is not positive.
- True Positive (TP) : Observation is positive, and is predicted to be positive.
- False Negative (FN) : Observation is positive, but is predicted negative.
- True Negative (TN) : Observation is negative, and is predicted to be negative.
- False Positive (FP) : Observation is negative, but is predicted positive.

Evaluation Metrics

Classification Rate/Accuracy:

It shows: Overall, how often is the classifier correct?

Classification Rate or Accuracy is given by the relation:

$$\text{Accuracy} = \frac{TP + TN}{\text{Total}}$$

Misclassification Rate/Error Rate:

It shows: Overall, how often is the classifier wrong?

Misclassification Rate or Error Rate is given by the relation:

$$\text{Error Rate} = \frac{FP + FN}{\text{Total}}$$

Evaluation Metrics

Recall / True Positive Rate:

High Recall indicates the class is correctly recognized (small number of FN).

$$\text{Recall} = \frac{TP}{TP + FN \text{ (Actual Positive)}}$$

Recall / True Negative Rate:

High Recall indicates the class is correctly recognized (small number of FP).

$$\text{Recall} = \frac{TN}{TN + FP \text{ (Actual Negative)}}$$

Evaluation Metrics

False Positive Rate:

$$\text{False Positive Rate} = \frac{FP}{FP + TN \text{ (Actual Negative)}}$$

False Negative Rate:

$$\text{False Negative Rate} = \frac{FN}{TP + FN \text{ (Actual Positive)}}$$

Evaluation Metrics

Precision:

High Precision indicates an example labeled as positive is indeed positive (small number of FP) or labeled as negative is indeed negative (small number of FN)

$$\text{Precision} = \frac{TP}{TP + FP \text{ (Predicted Positive)}}$$

$$\text{Precision} = \frac{TN}{TN + FN \text{ (Predicted Negative)}}$$

Evaluation Metrics

F-Measure/F-Score:

Since we have two measures (Precision and Recall) it helps to have a measurement that represents both of them. We calculate an F-measure which uses Harmonic Mean in place of Arithmetic Mean as it punishes the extreme values more.

The F-Measure will always be nearer to the smaller value of Precision or Recall.

$$\text{F Score} = \frac{2 * \text{Recall} * \text{Precision}}{\text{Recall} + \text{Precision}}$$

Support:

The support is the number of occurrences of each class in y_test.

Evaluation Metrics

- Let us assume there is a medical test which accurately predict whether a person have certain disease or not. And test has been conducted on people those have disease and those do not have disease in reality.
- There are two possible predicted classes: "yes" and "no". If we were predicting the presence of a disease, for example, "yes" would mean they have the disease, and "no" would mean they don't have the disease.
- The classifier made a total of 165 predictions i.e. 165 patients were being tested for the presence of that disease.
- Out of those 165 cases, the classifier predicted "yes" 110 times, and "no" 55 times.
- In reality, 105 patients in the sample have the disease, and 60 patients do not have the disease.

Evaluation Metrics

- The final report is being displayed in the form of confusion matrix as follows:

Total N = 165	Predicted NO	Predicted YES	
Actual NO	50	10	Total Actual No = 60
Actual YES	5	100	Total Actual Yes = 105
	Total Predicted No = 55	Total Predicted Yes = 110	

Evaluation Metrics

- Accuracy / Classification Rate: 50 + 100 / 165 = 0.91
- Misclassification Rate / Error: 10 + 5 / 165 = 0.09
- Recall (Class Yes): 100/105 = 0.95
- Recall (Class No): 50/60 = 0.83
- Precision (Class Yes): 100/110 = 0.90
- Precision (Class No): 50/55 = 0.90
- F-Score (Class Yes): (2*0.95*0.90) / (0.95+0.90) = 1.71/1.85 = 0.92
- F-Score (Class No): (2*0.83*0.90) / (0.83+0.90) = 1.494/1.73 = 0.86
- Support (Class Yes): Total Yes Predicted = 110
- Support (Class No): Total No Predicted = 55

AUC - ROC Curve

AUC - ROC Curve

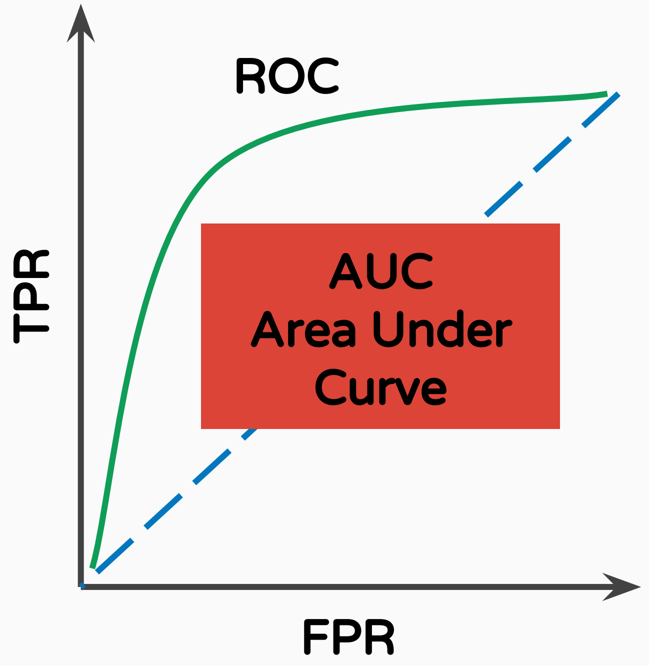
- In Machine Learning, performance measurement is an essential task.
- When we need to check or visualize the performance of the classification problem, we use AUC (**Area Under The Curve**) and ROC (**Receiver Operating Characteristics**) curve. It is one of the most important evaluation metrics for checking any classification model's performance. It is also written as AUROC (**Area Under the Receiver Operating Characteristics**)

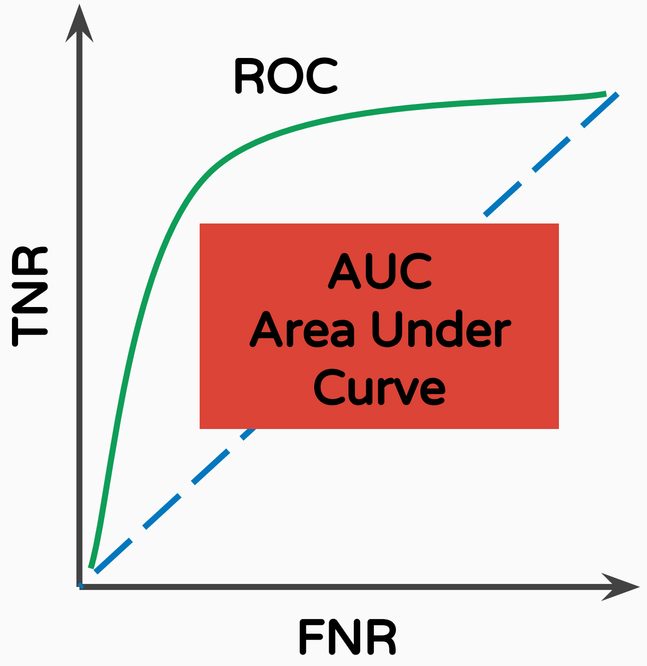
What is AUC - ROC Curve?

- AUC - ROC curve is a performance measurement for classification problem at various thresholds settings.
- ROC is a probability curve and AUC represents degree or measure of separability.
- It tells how much model is capable of distinguishing between classes.

AUC - ROC Curve

The ROC curve is plotted with TPR (True Positive Rate) against the FPR (False Positive Rate) where TPR is on y-axis and FPR is on the x-axis. As well as with TNR (True Negative Rate) against the FNR (False Negative Rate) where TNR is on y-axis and FNR is on the x-axis.





AUC - ROC Curve

How to speculate the performance of the model?

- An excellent model has AUC near to the 1 which means it has good measure of separability.
- A poor model has AUC near to the 0 which means it has worst measure of separability. In fact it means it is reciprocating the result. It is predicting 0s as 1s and 1s as 0s.
- And when AUC is 0.5, it means model has no class separation capacity whatsoever.

How to use AUC ROC curve for multi-class model?

- In multi-class model, we can plot N number of AUC ROC Curves for N number classes using One vs ALL methodology.
- So for Example, If you have **three** classes named **X, Y** and **Z**, you will have one ROC for X classified against Y and Z, another ROC for Y classified against X and Z, and a third one of Z classified against Y and X.

Cross Validation

Cross Validation

- There is always a need to validate the stability of your machine learning model. We need some kind of assurance that our model has got most of the patterns from the data correct, and its not picking up too much on the noise, or in other words its low on bias and variance.
- This process of deciding whether the numerical results quantifying hypothesized relationships between variables, are acceptable as descriptions of the data, is known as validation.
- Generally, an error estimation for the model is made after training, better known as evaluation of residuals. In this process, a numerical estimate of the difference in predicted and original responses is done, also called the training error.
- However, this only gives us an idea about how well our model does on data used to train it. Now its possible that the model is underfitting or overfitting the data.

Cross Validation

- Cross-validation is a statistical method used to estimate the skill of machine learning models.
- It is commonly used in applied machine learning to compare and select a model for a given predictive modeling problem because it is easy to understand, easy to implement, and results in skill estimates that generally have a lower bias than other methods.
- Cross Validation is a technique which involves reserving a particular sample of a dataset on which we do not train the model. Later, we test our model on this sample before finalizing it. Here are the steps involved in cross validation:

1. We reserve a sample data set.
2. We train the model using the remaining part of the dataset (training set).
3. We Use the reserve sample (test or validation set) for testing the model.

Cross Validation

K Fold Cross Validation:

- In K Fold cross validation, the data is divided into k subsets.
- Now this method is repeated k times, such that each time, one of the k subsets is used as the test set/validation set and the other k-1 subsets are put together to form a training set.
- The error estimation is averaged over all k trials to get total effectiveness of our model. As we can see, every data point gets to be in a validation set exactly once, and gets to be in a training set k-1 times.
- This significantly reduces bias as we are using most of the data for fitting, and also significantly reduces variance as most of the data is also being used in validation set.
- Interchanging the training and test sets also adds to the effectiveness of this method. As a general rule and empirical evidence, K = 5 or 10 is generally preferred, but nothing’s fixed and it can take any value.

Cross Validation

The general procedure is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups.
3. For each unique group:
    1. Take the group as a hold out or test data set.
    2. Take the remaining groups as a training data set.
    3. Fit a model on the training set and evaluate it on the test set.
    4. Retain the evaluation score and discard the model.
4. Summarize the skill of the model using model evaluation scores.

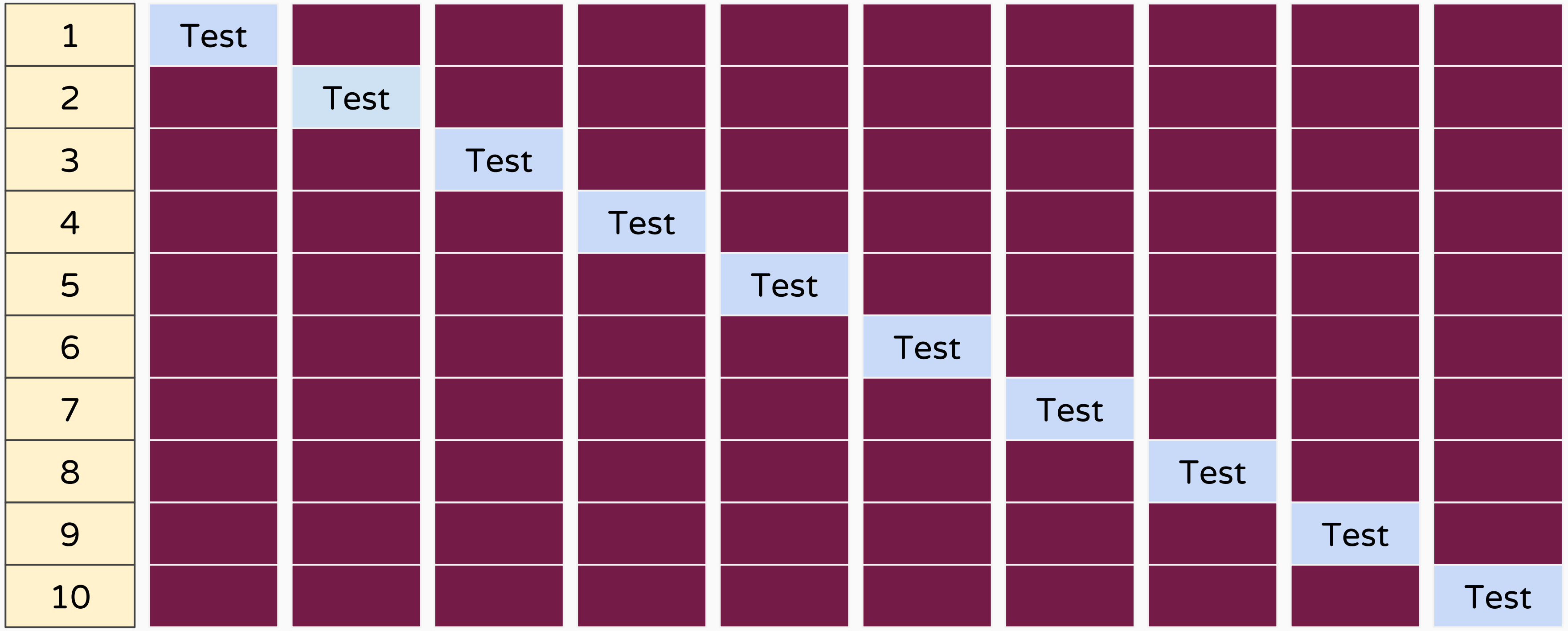
Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model k-1 times.

Cross Validation

- The results of a k-fold cross-validation run are often summarized with the mean of the model skill scores. It is also good practice to include a measure of the variance of the skill scores, such as the standard deviation.
- A poorly chosen value for k may result in a mis-representative idea of the skill of the model, such as a score with a high variance (that may change a lot based on the data used to fit the model), or a high bias, (such as an overestimate of the skill of the model).
- **k=10**: The value for k is fixed to 10, a value that has been found through experimentation to generally result in a model skill estimate with low bias a modest variance.
- The choice of k is usually 5 or 10, but there is no formal rule. As k gets larger, the difference in size between the training set and the resampling subsets gets smaller. As this difference decreases, the bias of the technique becomes smaller.

Cross Validation

Below is the visualization of a k-fold cross validation when k = 10



Cross Validation

Leave One Out Cross Validation (LOOCV):

- In this approach, we reserve only one data point from the available dataset, and train the model on the rest of the data. This process iterates for each data point.
- We make use of all data points, hence the bias will be low
- We repeat the cross validation process n times (where n is number of data points) which results in a higher execution time.
- This approach leads to higher variation in testing model effectiveness because we test against one data point. So, our estimation gets highly influenced by the data point. If the data point turns out to be an outlier, it can lead to a higher variation.

Cross Validation

Leave P Out Cross Validation (LPOCV):

- This approach leaves p data points out of training data, i.e. if there are n data points in the original sample then, n-p samples are used to train the model and p points are used as the validation set.
- This is repeated for all combinations in which original sample can be separated this way, and then the error is averaged for all trials, to give overall effectiveness.
- This method is exhaustive in the sense that it needs to train and validate the model for all possible combinations, and for moderately large p, it can become computationally infeasible.
- Exhaustive Methods compute all possible ways the data can be split into training and test sets.

Cross Validation

Stratified k-Fold Cross Validation:

- In some cases, there may be a large imbalance in the response variables. For example, in dataset concerning price of houses, there might be large number of houses having high price. Or in case of classification, there might be several times more negative samples than positive samples.
- For such problems, a slight variation in the K Fold cross validation technique is made, such that each fold contains approximately the same percentage of samples of each target class as the complete set, or in case of prediction problems, the mean response value is approximately equal in all the folds. This variation is also known as Stratified K Fold.
- This explained validation technique is also referred to as Non-exhaustive cross validation method. This does not compute all ways of splitting the original sample, i.e. you just have to decide how many subsets need to be made.

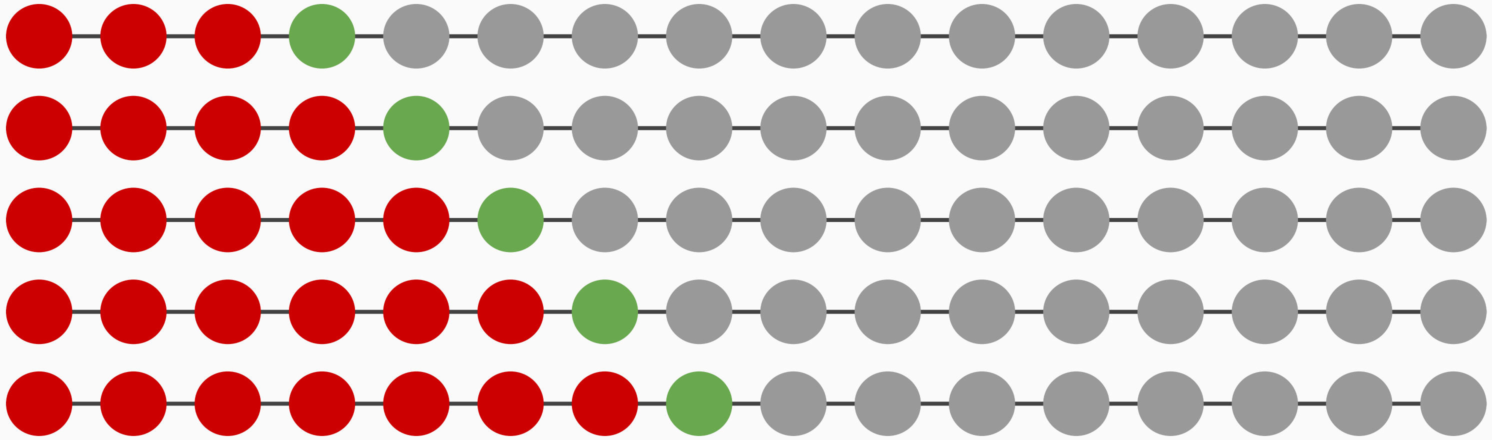
Cross Validation

Cross Validation for Time Series:

- Splitting a time-series dataset randomly does not work because the time section of your data will be messed up.
- Folds for time series cross validation are created in a forward chaining.
- Suppose we have a time series for yearly consumer demand for a product during a period of *n* years. The folds would be created like:

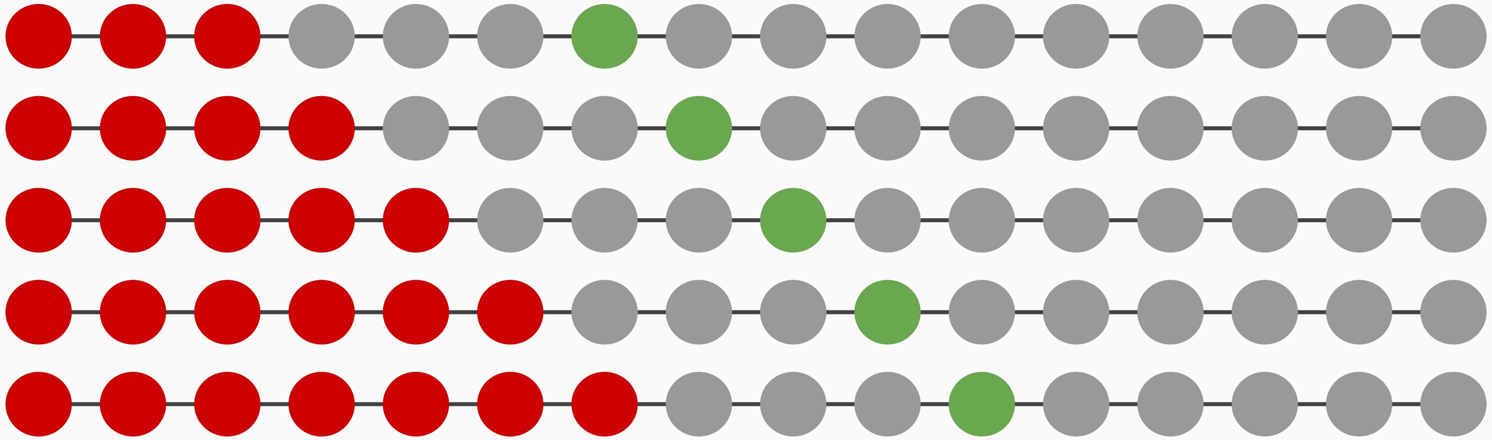
fold 1:	training [1],	test [2]
fold 2:	training [1 2],	test [3]
fold 3:	training [1 2 3],	test [4]
fold 4:	training [1 2 3 4],	test [5]
fold 5:	training [1 2 3 4 5],	test [6]
……………		
fold n:	training [1 2 3 ….. n-1],	test [n]

Cross Validation



- We progressively select a new train and test set. We start with a train set which has a minimum number of observations needed for fitting the model. Progressively, we change our train and test sets with each fold.
- In most cases, 1 step forecasts might not be very important. In such instances, the forecast origin can be shifted to allow for multi-step errors to be used.

Cross Validation



- Here are giving 3 steps forecasting

Hyperparameter Optimization

Hyperparameter Optimization

- Machine learning involves predicting and classifying data and to do so, we employ various machine learning models according to the dataset. Machine learning models are parameterized so that their behavior can be tuned for a given problem. These models can have many parameters and finding the best combination of parameters can be treated as a search problem.

What is a parameter in a machine learning learning model?
A model parameter is a configuration variable that is internal to the model and whose value can be estimated from the given data.

- They are required by the model when making predictions.
- Their values define the skill of the model on your problem.
- They are estimated or learned from data.
- They are often not set manually by the practitioner.
- They are often saved as part of the learned model.

Hyperparameter Optimization

- So our main take away from the above points should be parameters are crucial to machine learning algorithms. Also, they are the part of the model that is learned from historical training data.
- In machine learning, the specific model you are using is the function and requires parameters in order to make a prediction on new data. Whether a model has a fixed or variable number of parameters determines whether it may be referred to as *"parametric"* or *"nonparametric"*.

Some examples of model parameters include:

- The weights in an artificial neural network.
- The support vectors in a support vector machine.
- The coefficients in a linear regression or logistic regression.

Hyperparameter Optimization

What is a hyperparameter in a machine learning learning model?

A model hyperparameter is a configuration that is external to the model and whose value cannot be estimated from data.

- They are often used in processes to help estimate model parameters.
- They are often specified by the practitioner.
- They can often be set using heuristics.
- They are often tuned for a given predictive modeling problem.

➔ We cannot know the best value for a model hyperparameter on a given problem. We may use rules of thumb, copy values used on other issues, or search for the best value by trial and error.

➔ When a machine learning algorithm is tuned for a specific problem then essentially we are tuning the hyperparameters of the model to discover the parameters of the model that result in the most skillful predictions.

Hyperparameter Optimization

Some examples of model hyperparameters include:

- The learning rate for training a neural network.
- The C and sigma hyperparameters for support vector machines.
- The k in k-nearest neighbors.

Importance of the right set of hyperparameters in machine learning model:

- The best way to think about hyperparameters is like the settings of an algorithm that can be adjusted to optimize performance.
- In a true machine learning fashion, we’ll ideally ask the machine to perform the exploration and select the optimal model architecture automatically. Choosing the right set of values is typically known as “*Hyperparameter optimization*” or “*Hyperparameter tuning*”.

Two simple strategies to optimize/tune the hyperparameters:

1. Grid Search
2. Random Search.

Hyperparameter Optimization

Grid searching of hyperparameters:

- Grid search is an approach to hyperparameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid.

Let's consider the following example:

- Suppose, a machine learning model X takes hyperparameters a_1, a_2 and a_3.
- In *grid searching*, we first define the range of values for each of the hyperparameters a_1, a_2 and a_3.
- Now the *grid search* technique will construct many versions of X with all the possible combinations of hyperparameter (a_1, a_2 and a_3) values that we defined in the first place.
- This range of hyperparameter values is referred to as the ***grid***.

Hyperparameter Optimization

Grid searching of hyperparameters:
Suppose, we defined the grid as:

a_1 = [0,1,2,3,4,5]

a_2 = [10,20,30,40,5,60]

a_3 = [105,105,110,115,120,125]

- Now, *grid search* will begin its process of constructing several versions of X with the grid that we just defined.
- It will start with the combination of [0,10,105], and it will end with [5,60,125].
- It will go through all the intermediate combinations between these two which makes *grid search computationally very expensive.*

Hyperparameter Optimization

Random searching of hyperparameters:

- Random search differs from a grid search. In Grid Search we longer provide a discrete set of values to explore for each hyperparameter and in Random Search we provide a statistical distribution for each hyperparameter from which values may be randomly sampled.
- Distribution is essentially meant an arrangement of values of a variable showing their observed or theoretical frequency of occurrence.
- Sampling is the process of choosing a representative sample from a target population and collecting data from that sample in order to understand something about the population as a whole.
- We’ll define a sampling distribution for each hyperparameter. We can also define how many iterations we’d like to build when searching for the optimal model. For each iteration, the hyperparameter values of the model will be set by sampling the defined distributions.

Logistic Regression

Logistic Regression

- Logistic regression is useful for situations where there could be an ability to predict the presence or absence of a characteristic or outcome, based on values of a set of predictor variables.
- It is similar to a linear regression model but is suited to models where the dependent variable is dichotomous. It’s coefficients can be used to estimate odd ratios for each of the independent variables in the model.
- Logistic Regression on the other hand is used to ascertain the probability of an event, this event is captured in binary format, i.e. 0 or 1.
- With logistic regression, multi-class classification is possible, not just binary. But logistic regression is mostly used in binary classification.
- Linear Regression or least square regression estimates the coefficients of the linear equation, involving one or more independent variables, that best predict the value of the dependent variable.
- Linear regression is continuous while logistic regression is discrete.

Logistic Regression

How They Are Distinctively Related?

- Logistic regression estimates the probability of an outcome. Events are coded as binary variables with a value of 1 representing the occurrence of a target outcome, and a value of 0 representing its absence.
- Least Square Regression can also model binary variables using linear probability models. Least Square Regression may give predicted values beyond the range (0,1), but the analysis may still be useful for classification and hypothesis testing.
- Logistic regression models estimate probabilities of events as functions of independent variables.

Logistic Regression

How They Are Distinctively Related?

- Least Square Regression models the relationship between a dependent variable and a collection of independent variables. The value of a dependent variable is defined as a linear combination of the independent variables plus an error term ϵ.
- Logistic regression should be used to model binary dependent variables. The structure of the logistic regression model is designed for binary outcomes.
- Least Square regression is not built for binary classification, as logistic regression performs a better job at classifying data points and has a better logarithmic loss function as opposed to least squares regression.

Logistic Regression

What is Logistic Regression?

- It’s a classification algorithm, that is used where the response variable is *categorical*. The idea of Logistic Regression is to find a **relationship between features and probability of particular outcome**.
- This type of a problem is referred to as **Binomial Logistic Regression**, where the response variable has two values 0 and 1 or True and False etc.
- **Multinomial Logistic Regression** deals with situations where the response variable can have three or more possible values.

Logistic Regression

Why Logistic, not Linear?

- With binary classification, let ***'x'*** be some feature and ***'y'*** be the output which can be either 0 or 1.
- The probability that the output is 1 given its input can be represented as:

$$P(y = 1 \mid x)$$

- If we predict the probability via linear regression, we can state it as:

$$P(X) = \beta_0 + \beta_1 * X \quad \text{where, } P(X) = P(y = 1 \mid x)$$

- Linear regression model can generate the ***predicted probability*** as any number ranging from negative to positive infinity, whereas probability of an outcome can only lie between $0< P(x)<1$.
- Also, Linear regression has a considerable effect on outliers.
- To avoid this problem, ***log-odds*** function or ***logit*** function is used.

Logistic Regression

Logit Function:

Logistic Regression can be represented as:

$$\log\left(\frac{P(X)}{1 - P(X)}\right) = \beta_0 + \beta_1 * X$$

- The left hand side is called the **logit** or log-odds function.
- And **P(x)/(1 - P(x))** is called odds.
- The *odds* signifies the ratio of probability of success to probability of failure. Therefore, in Logistic Regression, linear combination of inputs are mapped to the log(odds), the output lies between 0 and 1

Logistic Regression

Logit Function:

If we take an inverse of the Logit Function, we get:

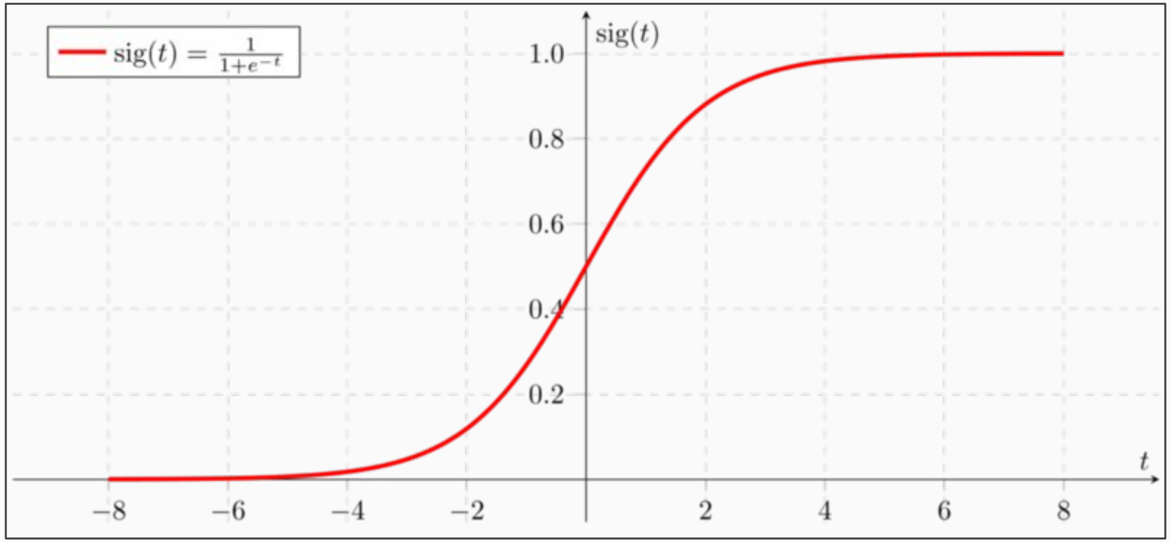
$$P(X) = \frac{e^{\beta_0 + \beta_1 * X}}{1 + e^{\beta_0 + \beta_1 * X}}$$

Above equation can also be written in simplified form as:

$$P(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 * X)}}$$

Logistic Regression

This function is also called as Sigmoid Function which gives S shaped curve when plotted on the graph. It always gives a value of probability ranging from: $0 < p < 1$ If the value goes to infinity, Y(predicted) will become 1 and if the value goes to negative infinity, Y(predicted) will become 0.



Logistic Regression

Where they align together:

- **Linear regression uses the general linear** equation:

$$Y = \beta_0 + \sum_{i=1}^{N} (\beta_i * X_i) + \epsilon$$

- where Y is a continuous dependent variable and independent variables X_i are usually continuous or other discrete domains. ϵ is a term for the variance that is not explained by the model and is usually just called “error”. Individual dependent values denoted by Y can be solved by this equation.

Logistic Regression

Where they align together:

- **Logistic regression is another generalized linear model** (GLM) procedure using the same basic formula, but instead of the continuous Y, it is regressing for the probability of a categorical outcome. In simplest form, this means that we're considering just one outcome variable and two states of that variable- either 0 or 1. The equation for the probability of Y=1 looks like this:

$$P(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 * X)}}$$

- Independent variables Xi can be continuous or binary. The regression coefficients bi can be exponentiated to give the change in odds of Y per change in Xi.

Logistic Regression

Error Equations:
For Linear Regression: Error Function is Mean Squared Error MSE

$$MSE = 1/N \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

For Logistic Regression: Error Function is Binary Cross Entropy BCE

$$BCE = -1/N \sum_{i=1}^{N} y_i * \log(\hat{y}_i) + (1 - y_i) * \log(1 - \hat{y}_i)$$

Logistic Regression

Assumptions in Logistic Regression:

- **Binary Output Variable**: Logistic regression is intended for binary (two-class) classification problems. It will predict the probability of an instance belonging to the default class, which can be snapped into a 0 or 1
- **Remove Noise**: Logistic regression assumes no error in the output variable (y), consider removing outliers and possibly misclassified instances from your training data.
- **Gaussian Distribution**: Logistic regression is a linear algorithm (with a non-linear transform on output). It does assume a linear relationship between the input variables with the output. Data transforms of your input variables that better expose this linear relationship can result in a more accurate model. For example, you can use log, root, Box-Cox and other univariate transforms to better expose this relationship.

Logistic Regression

Assumptions in Logistic Regression:

- **Remove Correlated Inputs**: Like linear regression, the model can overfit if you have multiple highly-correlated inputs. Consider calculating the pairwise correlations between all inputs and removing highly correlated inputs.
- **Fail to Converge**: It is possible for the expected likelihood estimation process that learns the coefficients to fail to converge. This can happen if there are many highly correlated inputs in your data or the data is very sparse (e.g. lots of zeros in your input data).

Logistic Regression

Assumptions in Logistic Regression:

- **Linear Relationship:**
    - **Linear regression:** needs a linear relationship between the dependent and independent variables.
    - **Logistic regression:** does not need a linear relationship between the dependent and independent variables.
- **Distribution of Residuality:**
    - **Linear regression:** requires error term to be normally distributed.
    - **Logistic regression:** does not require error term to be normally distributed.

Logistic Regression

Multi-class Logistic Regression:

- The basic intuition behind Multi-class and binary Logistic regression is same. However, for multi-class problem we follow a **one v/s all approach**.

- ***For example:*** If we have to predict whether the weather is sunny, rainy, or windy, we are dealing with a Multi-class problem. We turn this problem into three binary classification problem i.e whether it is sunny or not, whether it is rainy or not and whether it is windy or not.
- We run all three classifications **independently** on input. The classification for which the value of probability is maximum relative to others, is the solution.

Logistic Regression

Steps to Perform Logistic Regression using Scikit-Learn:

- Import required libraries
- Load the Dataset
- Do the Data preprocessing
- Separate features and target
- Feature Scaling using Standard Scalar
- Split the data into train and test part

Logistic Regression

Steps to Perform LogisticRegression using Scikit-Learn:

- Train the Algorithm: Import the LogisticRegression class, instantiate it, and call the fit() method along with our training data.

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, Y_train)
```

- Make the Predictions: To make predictions, use testing data.

```
y_predicted = model.predict(x_test)
```

Logistic Regression

Steps to Perform LogisticRegression using Scikit-Learn:

- Evaluate the Model by Drawing Confusion Matrix, Accuracy and Misclassification Rate

- Check the Probability of Class or Group for each value:

```
model.predict_proba(x_test)
```

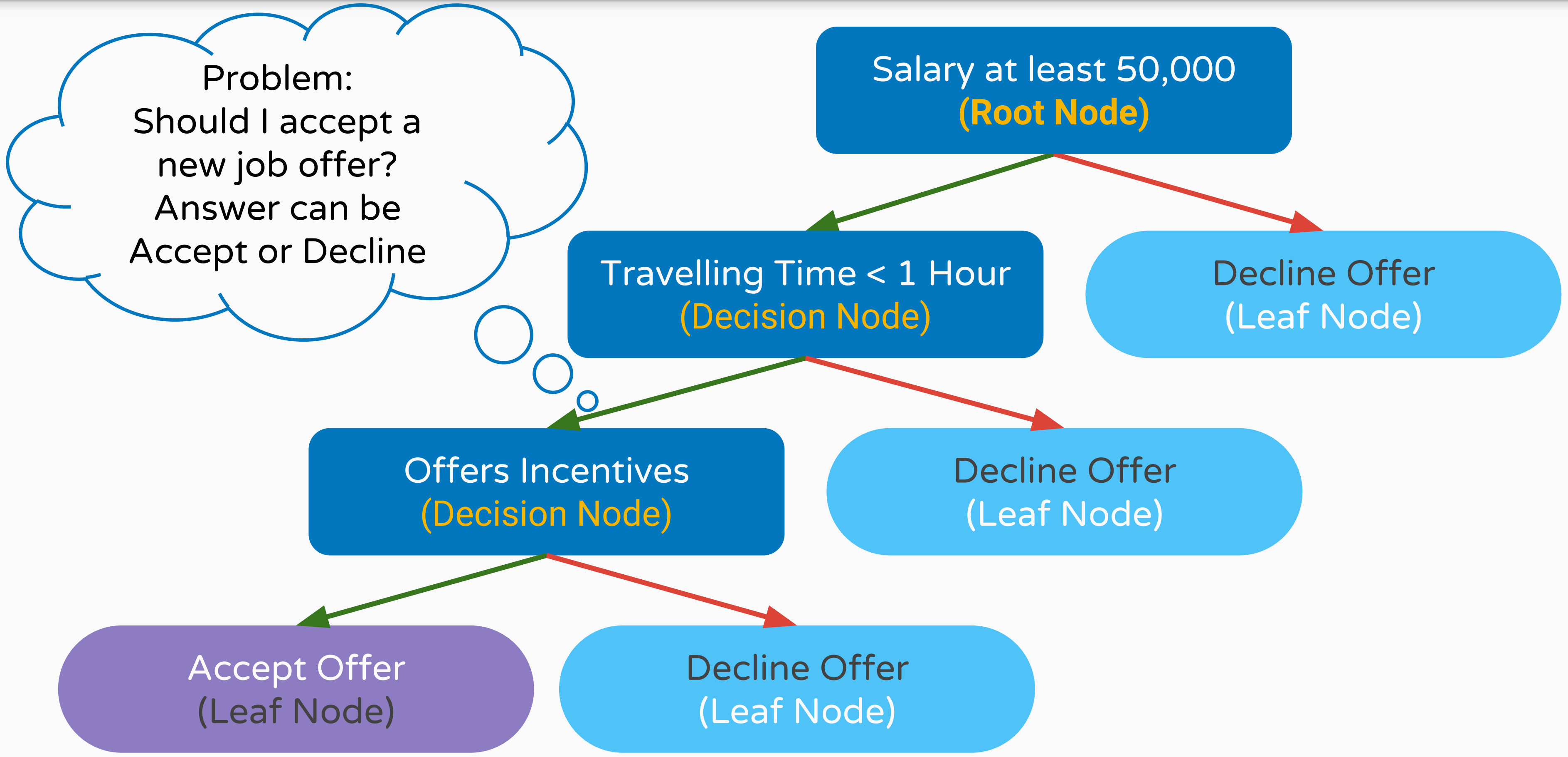
This returns the probability estimates for all classes ordered by the label of classes i.e. it is used to find the predicted probability of each class. Higher the value of probability for a class, the value will belong to that particular class.

Decision Tree

Decision Tree

- A decision tree is one of most frequently and widely used supervised machine learning algorithms that can perform both regression and classification tasks.
- For each attribute in the dataset, the decision tree algorithm forms a node, where the most important attribute is placed at the root node. For evaluation we start at the root node and work our way down the tree by following the corresponding node that meets our condition or "decision". This process continues until a leaf node is reached, which contains the prediction or the outcome of the decision tree.
- The general motive of using Decision Tree is to create a training model which can use to predict class or value of target variables by **learning decision rules** inferred from prior data (training data).
- The decision tree algorithm tries to solve the problem, by using tree representation. Each **internal node** of the tree corresponds to an attribute, and each **leaf node** corresponds to a class label.

Decision Tree

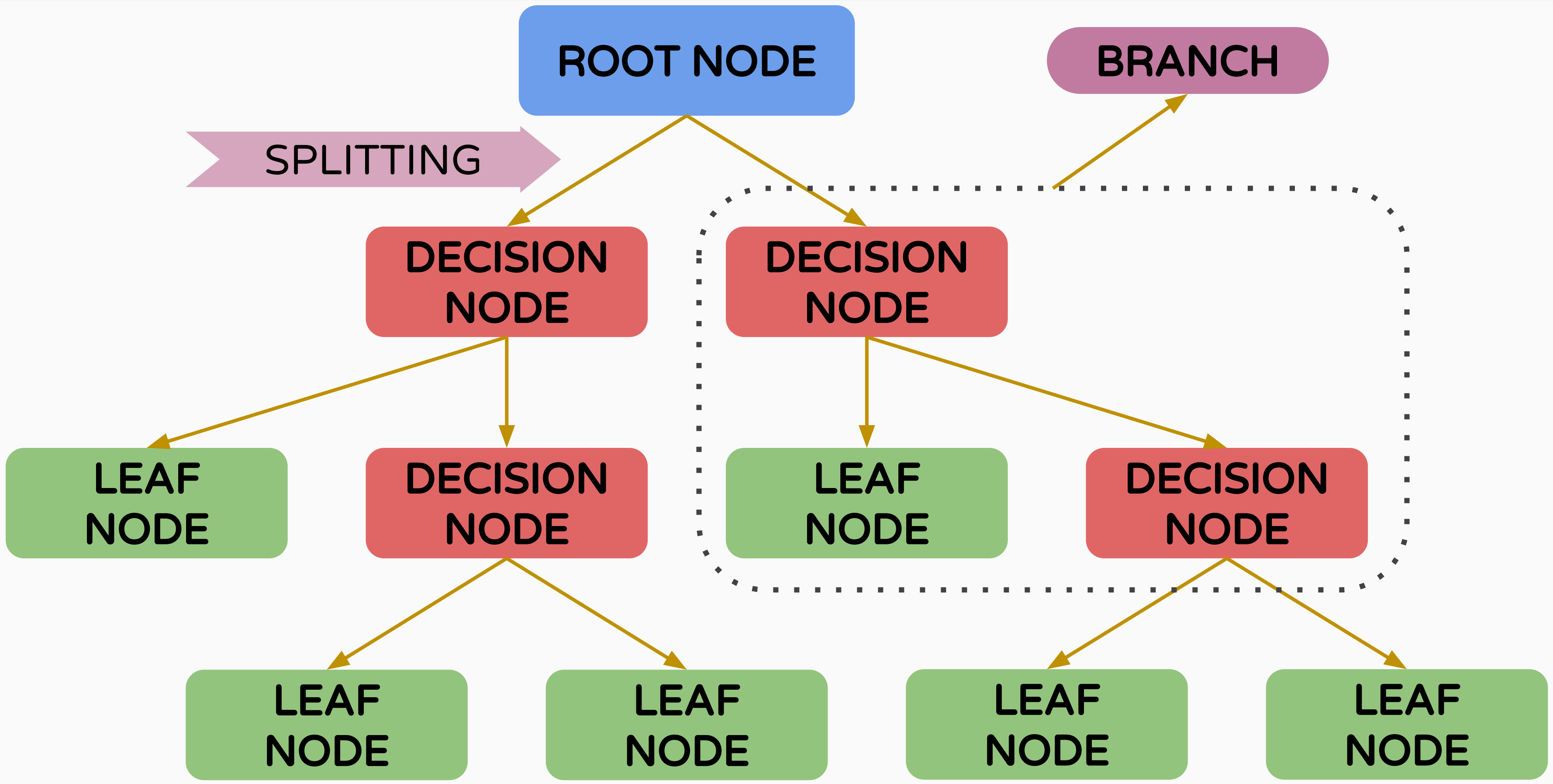


Decision Tree

Components of Decision Tree:

- **Root Node :** The top most node is called Root Node. It implies the best predictor (independent variable).
- **Decision / Internal Node :** The nodes in which predictors (independent variables) are tested and each branch represents an outcome of the test.
- **Leaf / Terminal Node :** It holds a class label (category) - Yes or No (Final Classification Outcome) or value (continuous in Regression Tree).
- **Splitting:** This is a process of dividing a node into two or more sub-nodes.
- **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. The opposite of pruning is splitting.
- **Branch:** A subsection of an entire tree is called a branch.
- **Parent Node and Child Node:** A node, which is divided into sub-nodes is called a Parent Node of the sub-nodes; whereas the sub-nodes are called the Child of the parent node.

Decision Tree



Decision Tree

Assumptions while creating Decision Tree:

- At the beginning, the whole training set is considered as the **root.**
- Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
- Records are distributed recursively on the basis of attribute values.
- Order to placing attributes as root or internal node of the tree is done by using some statistical approach.

Decision Tree

- The primary challenge in the decision tree implementation is to identify which attributes do we need to consider as the root node and each level. Handling this is known as attributes selection.

The popular attribute selection measures:

- Information gain
- Gini index

- **Attributes Selection:** If dataset consists of **“n”** attributes then deciding which attribute to place at the root or at different levels of the tree as internal nodes is called as Attributes Selection.

➔ The attribute with a HIGH VALUE of Information Gain is placed at the root when Information Gain selection is selected.

➔ The attribute with a LOW VALUE of Gini Index is placed at the root when Gini Index selection is selected.

Decision Tree

Consider the following Dataset:

A	B	C	D	E
4.8	3.4	1.9	0.2	Positive
5.0	3.0	1.6	0.2	Positive
5.0	3.4	1.6	0.4	Positive
5.2	3.5	1.5	0.2	Positive
5.2	3.4	1.4	0.2	Positive
4.7	3.2	1.6	0.2	Positive
4.8	3.1	1.6	0.2	Positive
5.4	3.4	1.5	0.4	Positive
7.0	3.2	4.7	1.4	Negative
6.4	3.2	4.5	1.5	Negative
6.9	3.1	4.9	1.5	Negative
5.5	2.3	4.0	1.3	Negative
6.5	2.8	4.6	1.5	Negative
5.7	2.8	4.5	1.3	Negative
6.3	3.3	4.7	1.6	Negative
4.9	2.4	3.3	1.0	Negative

Decision Tree

Let’s solve above Dataset using Decision Tree Algo.
First we will have to create a Discretized Continuous values i.e. convert continuous values of features to Categorical as follows:

A	B	C	D
>= 5.0	>= 3.0	>= 4.2	>= 1.4
< 5.0	< 3.0	< 4.2	< 1.4

We will select Entropy or Information Gain as Attribute Selection Measure.
Steps to calculate Information Gain of each attribute.

1. Calculate entropy of Target.
2. Calculate entropy for every attribute A, B, C, D.

   IG of each attribute = Entropy of Target - Entropy of every attribute.

Decision Tree

Formula for calculating Entropy:

$$\text{Entropy}(t) = - \sum p(t) * \log_2 p(t)$$

The entropy of Target: We have 8 records with negative class and 8 records with positive class.
Calculate entropy using formula:

$$\begin{aligned} E(\text{Target}) &= -1*[(p(+ve)*\log_2(p(+ve)) + (p(-ve)*\log_2(p(-ve))] \\ &= -1*((8/16)*\log_2(8/16)) + (8/16) * \log_2(8/16)) \\ &= 1 \end{aligned}$$

Decision Tree

Information gain for Attribute A:

Attribute A has value >= 5 for 12 records out of 16 and < 5 for 4 records.

- Probability(A) for A >= 5 & class == positive: 5/12
- Probability(A) for A >= 5 & class == negative: 7/12
  - Entropy(5,7) = -1 * ((5/12)*$\log_2$(5/12) + (7/12)*$\log_2$(7/12)) = 0.9799
- Probability(A) for A < 5 & class == positive: 3/4
- Probability(A) for A < 5 & class == negative: 1/4
  - Entropy(3,1) = -1 * ((3/4)*$\log_2$(3/4) + (1/4)*$\log_2$(1/4)) = 0.81128

Weighted_Entropy(A) = P(A >= 5) * E(5,7) + P(A < 5) * E(3,1)
= (12/16) * 0.9799 + (4/16) * 0.81128 = 0.937745

Information_Gain(A) = E(Target) - WE(A) = 1 - 0.937745 = 0.062255

Decision Tree

Information gain for Attribute B:

Attribute B has value >= 3 for 12 records out of 16 and < 3 for 4 records.

- Probability(B) for B >= 3 & class == positive: 8/12
- Probability(B) for B >= 3 & class == negative: 4/12
  - Entropy(8,4) = -1 * ((8/12)*$\log_2$(8/12) + (4/12)*$\log_2$(4/12)) = 0.39054
- Probability(B) for B < 3 & class == positive: 0/4
- Probability(B) for B < 3 & class == negative: 4/4
  - Entropy(0,4) = -1 * ((0/4)*$\log_2$(0/4) + (4/4)*$\log_2$(4/4)) = 0

Weighted_Entropy(B) = P(B >= 3) * E(8,4) + P(B < 3) * E(0,4)
= (12/16) * 0.39054 + (4/16) * 0 = 0.292905

Information_Gain(B) = E(Target) - WE(B) = 1 - 0.292905 = 0.7070795

Decision Tree

Information gain for Attribute C:

Attribute C has value >= 4.2 for 6 records out of 16 and < 4.2 for 10 records.

- Probability(C) for C >= 4.2 & class == positive: 0/6
- Probability(C) for C >= 4.2 & class == negative: 6/6
    - Entropy(0,6) = -1 * ((0/6)*$\log_2$(0/6) + (6/6)*$\log_2$(6/6)) = 0
- Probability(C) for C < 4.2 & class == positive: 8/10
- Probability(C) for C < 4.2 & class == negative: 2/10
    - Entropy(8,2) = -1 * ((8/10)*$\log_2$(8/10) + (2/10)*$\log_2$(2/10)) = 0.72193

Weighted_Entropy(C) = P(C >= 4.2) * E(0,6) + P(C < 4.2) * E(8,2)
= (6/16) * 0 + (10/16) * 0.72193 = 0.4512

Information_Gain(C) = E(Target) - WE(C) = 1 - 0.4512 = 0.5488

Decision Tree

Information gain for Attribute D:

Attribute D has value >= 1.4 for 5 records out of 16 and < 1.4 for 11 records.

- Probability(D) for D >= 1.4 & class == positive: 0/5
- Probability(D) for D >= 1.4 & class == negative: 5/5
  - Entropy(0,5) = -1 * ((0/5)*$\log_2$(0/5) + (5/5)*$\log_2$(5/5)) = 0
- Probability(D) for D < 1.4 & class == positive: 8/11
- Probability(D) for D < 14 & class == negative: 3/11
  - Entropy(8,3) = -1 * ((8/11)*$\log_2$(8/11) + (3/11)*$\log_2$(3/11)) = 0.84532

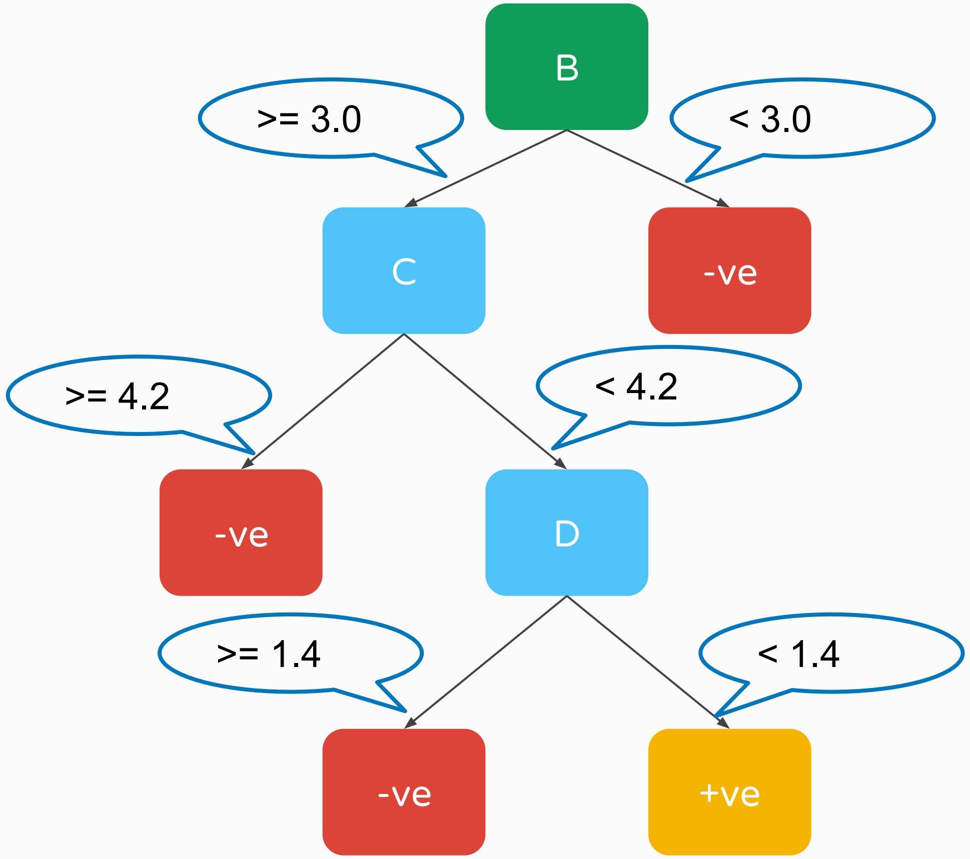
Weighted_Entropy(D) = P(D >= 1.4) * E(0,5) + P(D < 1.4) * E(8,3)
= 5/16 * 0 + (11/16) * 0.84532 = 0.5811575

Information_Gain(D) = E(Target) - WE(D) = 1 - 0.5811575 = 0.41189

Decision Tree

We will select Attribute at Root Node for which Information Gain has Highest Value.
Here Attribute B has Highest Value, so Decision tree will start with Attribute B

Attribute	Information Gain
A	0.062255
B	0.707095
C	0.5488
D	0.41189



Decision Tree

Now we will select Gini Index as Attribute Selection Measure.
Steps to calculate Gini Index:

1. Calculate Gini Index of Each Attribute
2. Select Attribute for Decision Tree for which Gini Index has Lowest Value

Formula for calculating Gini Index:

$$\mathrm{Gini}(t) = 1 - \sum (p(t))^2$$

Decision Tree

Gini Index for Attribute A:

Attribute A has value >= 5 for 12 records out of 16 and < 5 for 4 records.

- Probability(A) for A >= 5 & class == positive: 5/12
- Probability(A) for A >= 5 & class == negative: 7/12
    - $gini(5,7) = 1 - ((5/12)^2 + (7/12)^2) = 0.4860$
- Probability(D) for A < 5 & class == positive: 3/4
- Probability(A) for A < 5 & class == negative: 1/4
    - $gini(3,1) = 1 - ((3/4)^2 + (1/4)^2) = 0.375$

Gini(A) = P(A >= 5) * gini(5,7) + P(A < 5) * gini(3,1)

Gini(A) = (12/16)*(0.486) + (4/16)*(0.375) = 0.45825

Decision Tree

Gini Index for Attribute B:

Attribute B has value >= 3 for 12 records out of 16 and < 3 for 4 records.

- Probability(B) for B >= 3 & class == positive: 8/12
- Probability(B) for B >= 3 & class == negative: 4/12
    - gini(8,4) = 1- $((8/12)^2 + (4/12)^2)$ = 0.446
- Probability(B) for B < 3 & class == positive: 0/4
- Probability(B) for B < 3 & class == negative: 4/4
    - gini(0,4) = 1- $((0/4)^2 + (4/4)^2)$ = 0

Gini(B) = P(B >= 3) * gini(8,4) + P(B < 3) * gini(0,4)

Gini(B) = (12/16)*(0.446) + (4/16)*(0) = 0.3345

Decision Tree

Gini Index for Attribute C:

Attribute C has value >= 4.2 for 6 records out of 16 and < 4.2 for 10 records.

- Probability(C) for C >= 4.2 & class == positive: 0/6
- Probability(C) for C >= 4.2 & class == negative: 6/6
    - gini(0,6) = 1- $((0/8)^2 + (6/6)^2) = 0$
- Probability(C) for C < 4.2& class == positive: 8/10
- Probability(C) for C < 4.2 & class == negative: 2/10
    - gini(8,2) = 1- $((8/10)^2 + (2/10)^2) = 0.32$

Gini(C) = P(C >= 4.2) * gini(0,6) + P(C < 4.2) * gini(8,2)

Gini(C) = (6/16)*(0) + (10/16)*(0.32) = 0.2

Decision Tree

Gini Index for Attribute D:

Attribute D has value >= 1.4 for 5 records out of 16 and < 1.4 for 11 records.

- Probability(D) for D >= 1.4 & class == positive: 0/5
- Probability(D) for D >= 1.4 & class == negative: 5/5
  - gini(0,5) = 1- $((0/5)^2 + (5/5)^2)$ = 0
- Probability(D) for D < 1.4 & class == positive: 8/11
- Probability(D) for D < 1.4 & class == negative: 3/11
  - gini(8,3) = 1- $((8/11)^2 + (3/11)^2)$ = 0.397

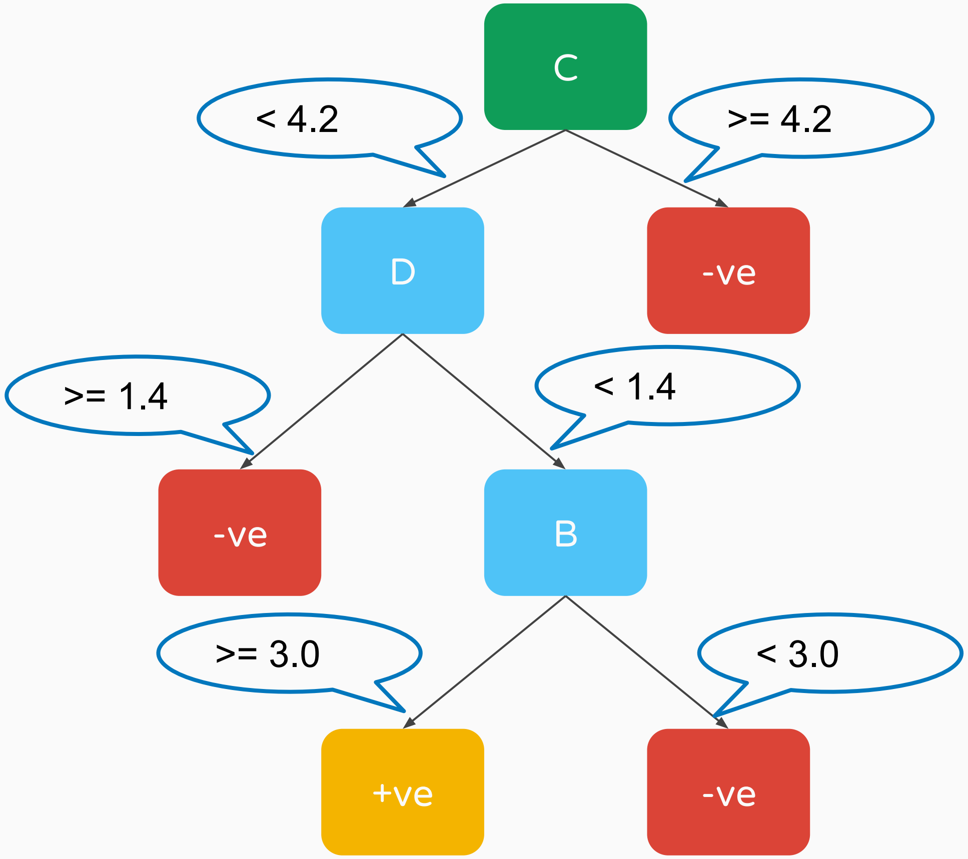
Gini(D) = P(D >= 1.4) * gini() + P(B < 1.4) * gini(8,3)

Gini(D) = (5/16)*(0) + (11/16)*(0.397) = 0.273

Decision Tree

We will select Attribute at Root Node for which Gini Index has Lowest Value.
Here Attribute C has Lowest Value, so Decision tree will start with Attribute C

Attribute	Gini Index
A	0.45825
B	0.3345
C	0.2
D	0.273



Decision Tree

➔ Gini Index: Important Points:

1. ZERO Gini Index implies perfect classification.
2. (1 - (1/ No. of classes) implies worst classification.
3. For binary dependent variable, max gini index value can be 0.5
4. Gini Index favors larger partitions.

➔ Information Gain: Important Points:

1. It favors partitions that have small counts but many distinct values.
2. Smaller value of Entropy signifies a good classification.

Decision Tree

- Overfitting: The model is having an issue of overfitting is considered when the algorithm continues to go deeper and deeper in the to reduce the training set error but results with an increased test set error i.e. Accuracy of prediction for our model goes down. It generally happens when it builds many branches due to outliers and irregularities in data.

- Two approaches which we can use to avoid overfitting are:
  - Pre-Pruning
  - Post-Pruning

➔ **Pre-Pruning:** In pre-pruning, it stops the tree construction bit early. It is preferred not to split a node if its goodness measure is below a threshold value. But it’s difficult to choose an appropriate stopping point.

➔ **Post-Pruning:** In post-pruning first, it goes deeper and deeper in the tree to build a complete tree. If the tree shows the overfitting problem then pruning is done as a post-pruning step.

Decision Tree

Steps to Perform Decision Tree Classification using Scikit-Learn:

- Import required libraries
- Load the Dataset
- Do the Data preprocessing
- Separate features and target
- Feature Scaling using Standard Scalar
- Split the data into train and test part

Decision Tree

Steps to Perform Decision Tree Classification using Scikit-Learn:

- Train the Algorithm: Import the DecisionTreeClassifier class, instantiate it, and call the fit() method along with our training data.

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(X_train, Y_train)
```

- Make the Predictions: To make predictions, use testing data.

```
y_predicted = model.predict(x_test)
```

- Evaluate the Model by calculating Confusion Matrix, Accuracy Score and Misclassification Rate

Decision Tree

Steps to Perform Decision Tree Regression using Scikit-Learn:

- Import required libraries
- Load the Dataset
- Check the relationship between Independent and Dependent Variables
- Do the Data preprocessing
- Separate features and target
- Feature Scaling using Standard Scalar
- Split the data into train and test part

Decision Tree

Steps to Perform Decision Tree Regression using Scikit-Learn:

- Train the Algorithm: Import the DecisionTreeRegressor class, instantiate it, and call the fit() method along with our training data.

```
from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor()
model.fit(X_train, Y_train)
```

- Make the Predictions: To make predictions, use testing data.

```
y_predicted = model.predict(x_test)
```

- Evaluate the Model by calculating R2 Score and Mean Squared Error

Thanks!

Signitive Technologies

Sneh Nagar, Behind ICICI Bank,
Chhatrapati Square, Nagpur 15

Landmark:
Bharat Petrol Pump
Chhatrapati Square

Contact: 9011033776
www.signitivetech.com

“Keep Learning, Happy Learning”

Best Luck!

Have a Happy Future