**1.Summary for Decision Tree from Scratch**

**1. Introduction**

A decision tree is series of it-then statements which is when applied to a record in a data set, results in the classification of that record. It is a family of supervised learning algorithm which is used to solve both the classification and regressions problems in a tree representation where leaf node holds a class label and attributes are represented on the internal node of the tree.

**2. Decision Tree classifier**

It is started from the root of the tree for predicting a class label for a record in decision trees. The record’s attribute value is compared with the root attribute. Based on such comparison, we follow the branch corresponding to that value and jump onto the next value. After the comparison with the root attribute, we compare with the internal nodes of the tree until we get the predicted class value of a leaf node.

**3. Process of splitting nodes**

**Entropy** measures the purity of the split. The higher the entropy, the harder it is to draw any conclusions from that information. Information gain calculates the reduction in entropy The entropy is said to be zero when either the positive or negative example is 0. The entropy is 1 if the positive example is equal to the negative example. After calculating the entropy of all the attributes and their gain, we will choose the greatest gain as it gives the most information then consider it as a root node.

**The Gini** index is the expected error rate.

**ID3** stands for Iterative Dichotomiser 3 is a classification algorithm that helps in building a decision tree by selecting the best attribute that yields maximum Information Gain (IG) or minimum entropy.

**4. Cross-validation**

The strategy of reserving a specific sample of the dataset on which we do not train the model. Later, we test the model on the same sample before finalizing the model.

**5. Overfitting**

The model is considered to be overfitting, when the accuracy of the prediction for the model goes down with the an increased in test set error. It also happens when it builds many branches due to outliers and irregularities in data.

**Pre-Pruning** is one of the approach which prevent the generation of non-significant branches and avoid overfitting. It stops the node to split further if its goodness measure is below a threshold value.

**Post-Pruning** is another approach to avoid overfitting. Cross-validation data is used to check the effect pruning (performed when the tree shows the overfitting problem) and it will test whether expanding a node will make an improvement or not.

**2. Summary for K-Nearest Neighbor (KNN) from scratch**

K Nearest Neighbor is a simple algorithm that uses the whole dataset in its training dataset. It looks for the k-most similar across the entire testing dataset whenever we make prediction for an unknown data instance and returns the most similar instances as the predictions.

KNN works by finding the distance between the query and all examples in the data. The neighbor class which is closest to the query is selected as K then voted for the most frequent label in case of classification or averages the labels in the case of regression.

The number of nearest neighboring points which vote for a new class of test data is represented by the K in KNN.

The major steps are listed below while implementing the KNN algorithm using python.

**1. Data Handling**

Here, the iris dataset is provided and data handling is carried out as the first step. The dataset iris is open using open function and using a reader function the data inside the iris dataset is read available under the csv module.

To find the prediction and evaluate the accuracy of the model, we need to change the categorical data into a numerical form and separate it into the train set and test set using the handle Dataset.

**2. Distance Calculation**

Euclidean distance is used to calculate the distance which is known as the square root of the total of the square difference between the two numerical arrays.

**3. Finding K nearest Point**

**getKNeighbors** function is used to return K nearest neighbors from the training set for a given test instance.

**4. Predict the class**

**getResponse** function is used to vote the class attribute and the majority vote is taken as prediction.

**5. Check the accuracy**

**getAccuracy** function is created to measure the accuracy of the majority vote by measuring the percentage of all predictions from all the predictions made.

**KNN using Scikit-learn**

The given iris data contains the column as target\_class which contains string and is not recognized by machine. Using the **LabelEncoder** for target\_class, we encoded as 0 for 1 string data and 0 for another string data.

It is not good approach to test and train the same data so using the **train\_test\_split** provided by sklearn, we split the data and **test\_size** will split the certain percentage to both train and test data. The parameter, **Random\_state** helps to give the same result for every count of run.

The **fit method** is used to train the X\_train and Y\_train and **predict method** is used to perform testing on X\_test.