**Churn Prediction Using Decision Tree Algorithm**

Project link

<https://colab.research.google.com/drive/1E9yC6VMnell74zEZlPNxBaFoClAKDxmu#scrollTo=_bAjRQdY7PJ4>

Git Hib : <https://github.com/samarthmaiya/ML/tree/master/uts-mc>’

Introduction

This document shows algorithm used to predict customer churn. Focus is to predict customer churn behaviour based on previous customer data. The raw data set contains 7043 rows (customers) and 21 columns (features). The “Churn” column is target. Data is available in <https://www.kaggle.com/blastchar/telco-customer-churn>.

Algorithm

1. Data Preparation

Zipped csv data is kept at git hub location, using ‘wget’ package downloaded to current workspace. ‘zipfile’ package used to un-zip the package to get csv data file.

1. Data pre-process

Before actual implementation exploratory data analysis done on the data set.

As a part of pre-processing implemented categorical string data to one hot encoded

numeric data.

* Dictionary preparation is done for each column

for i in datafram[coloumnname].unique(): // identified unique data from each coloumn

   dicdata.append((i,index)) //appended with unique data

* Dropped data row which are empty .
* Fed this dictionary to onehotencoder class:, return column data frame with encoded numeric data. Thus converted from categorical string to numeric data frame.

for k,v in encodingvalue:   // each value in datafram if the parma match with previously encoded value

      if(i == k):

   retundata =  v // append numeric value based on match found

      return retundata;

1. Data split

split the data for train and validation set based on user input. Method ‘train\_test\_split’ takes 2 parameter dataframe and split size (default split size is 10%).This method split the entire data frame to train data frame and test data frame. As test data frame extracted from the total data set helps to validate the prepared model accuracy.

def train\_test\_split(dataframe,test\_size=0.1):

    totalsize = len(dataframe)

     testdatasize = int(test\_size \* totalsize) //if total size is

7043, 10% of 7043 is 704 data point used for testing prepared

model

1. Node definition

Defined singe node with its feature. Later on used this node recursively to construct the tree

        self.feature = feature //feature information

        self.threshold = threshold //threshold value of each node

        self.data\_left = data\_left // left reference to the node

        self.data\_right = data\_right //right reference to node

        self.gain = gain // gain information

        self.value = value // associated value

1. Entropy

If we have n different value , entropy is calculated using formula where

* n is continuous value to calculate entropy
* Pi is the probability of occurrence of ith value

\_dict = {x:coll.count(x)/len(coll) for x in coll} // identified coloumn frequency average

\_prob = np.array(list(\_dict.values())) // calculated probability

    return - \_prob.dot(np.log2(\_prob)) // applied above mentioned formulae to calculate entropy

1. Information Gain and Build

Is how important a given attribute of the feature vectors. Formula used to calculate information gain is entropy(parent) – [average entropy(children)].

Recursively identified the best split of the data set. Based on threshold value segregated parent ,left and right child node calculated information gain for each node if the gain is grater than previous , new node consider for splitting. Best split method hold the information on left node, right node, threshold ,gain and feature index where it is split.

        for f\_idx in range(n\_cols): // for each of coloumn index

            X\_curr = X[:, f\_idx]

            for currentth in np.unique(X\_curr):

// identified current unique path

                p= np.concatenate((X, y.reshape(1, -1).T), axis=1)

//with respect to target variable best split calculated

                for row in p:

                  if (row[f\_idx] <= currentth):

                    leftdata.append(row)

// left node array preparation

                for row in p :

                  if (row[f\_idx] > currentth):

                    rightdata.append(row)

//right node array preparation

                    gain = self.\_info\_gain(y, y\_left, y\_right)

// left and right node gain calculation with

respect to target

                    if gain > best\_info\_gain:

//if gain is more than previous hold this information in Node object

During build face, best split is identified recursively and gathered node information on split. This split is controlled by max depth and min sample to split. Here default value considered min sample to split is 2.This can be tuned during hyper parameter tuning step.

1. Model Evaluation

Data split is done for entire data to train and test .Test data set considered for evaluating the model accuracy. Accuracy is calculated for the model by calculating mean value for actual and predicted data . Function is defined by name ‘accuracy’ to calculate prepared model accuracy.

def accuracy(actual,predicted):

     yhat=actual // actual labelled target

     y=predicted //predicted target

     acc=np.mean(y==yhat) //average value

     return acc

1. Experiment Design and Evaluation

Trained model with 6000 data points , it took almost 1hour 33m 6 s to train the model.

Run for multiple hyper parameter , finally for min\_samples\_split=2 and max\_depth=3

Ended up in good test score.

Accuracy of 0. 7985714285714286 for 700 data set obtained during prediction phase

1. Evaluation Results

|  |  |  |
| --- | --- | --- |
| sl no | parameter | score |
| 1 | min\_samples\_split=2 and max\_depth=3 | 0.7985714285714286 |
| 2 | min\_samples\_split=2 and max\_depth=4 | 0.7885714285714286 |
| 3 | min\_samples\_split=2 and max\_depth=5 | 0.7871428571428571 |

1. Conclusion

Churn data set has multiple feature , tree based algorithm is best suited option to predict. Also tree based algorithm has provision to tune using max depth and min sample split hyper para meter help algorithm for better accuracy.

Improvement area with respect to this algorithm is to introduce multiple hyper parameter such as ‘min\_samples\_leaf’ ,’ min\_weight\_fraction\_leaf’,’ max\_features’,’ random\_state’, through this we can improve test accuracy.

Reduction in time taken to train algorithm is another area of improvement in the current algorithm.