Classification: Decision Tree (CART)

1 Introduction

An implementation of a classifier using the Decision Tree algorithm. The entire implementation of the decision trees algorithm is done native python and void of any data wrapper or data-mining tool. The implemented algorithm is tested on a dataset, for which accuracy metrics are evaluated.

2 Data

The dataset used is shortened version of car evaluation data from the University of California Irvine (UCI) machine database available on Kaggle (Bohanec, 1997; Darlington, 2018). The dataset was originally developed for the presentation of DEX (Decision Expert) representing an expert system for multi-attribute decision making using modelled decision representation with trees (Bohanec & Rajkovič, 1988, 1990).

2.1 Data Structure

The dataset is a multivariate dataset with seven attributes. The attributes and their unique values are described in the table below.

Attribute	Description	Type	Values
Buying Price	Relative purchase cost of vehicle.	Categorical	vhigh, high, med, low
Maintenance Cost	Relative maintenance cost of car	Categorical	vhigh, high, med, low
Number of Doors	Number of doors on vehicle	Categorical	2, 3, 4, 5more.
Number of Persons	Number of passengers including driver	Categorical	2, 4, more.
Lug Boot	Size of the vehicle's boot / trunk	Categorical	small, med, big
Safety	Degree of vehicle safety	Categorical	low, med, high.
Decision	Decision to purchase	Categorical	unacc, acc, good, vgood

The dataset consists of 1728 records on vehicles collated. The multi-attribute nature of the dataset makes it suitable for the classification task. More so, the categorical nature of all the attribute is a reason for the selection of this dataset for use by the decision tree algorithm.

For the purpose of this report, six of the attributes in each record is set to features for the classification algorithm. Subsequently, maintenance cost, buying price, number of doors, number of persons, lug boot, and safety attributes are used as classification features. While the Decision attribute is used as a label for training and test the decision tree model.

2.2 Data Collection

The data is downloaded through a link to the UCI database available on Kaggle. The downloaded data when unzipped contains a Comma Separated Value (CSV) file containing all the data records respectively. However, the column heads are missing. In the data cleaning Section, column heads retrieved from the data description are placed into the data.

2.3 Data Pre-processing

The Pandas framework is used for data cleaning. The framework uses unique data objects called DataFrame and Series. The data cleaning task is described in the steps below which can be found in the “exercise.ipynb” project file.

2.3.1 Inspection

The downloaded dataset was parsed into a pandas DataFrame. The dataset was then inspected for completeness using info() and describe() function. It was observed that the number of records for each attribute were 1728 correspondingly.

2.3.2 Null Values

Next, the dataset is inspected for null values using the isnull().sum() member function of the DataFrame object. No null objects are found for the dataset.

2.3.3 Separation / Conversion of Data to Features and Labels

Since, the algorithm is implemented in native python with no data wrappers, the DafaFrame object is converted to dictionary of all the attributes as keys and a list of their values. The data is then randomly separated into training data and testing data. 80 percent of the data is used for training while the rest is used for testing. The data is then separated into six features and “Decision” label.

3 Algorithm

Decision Tree is a supervised learning technique that represents the decision-making process graphically. This predictive modelling technique is applied in statistics, data mining and machine learning. The tree designed in this project is a CART (Classification and Regression Tree) which allows its leaves to evaluate discrete and continuous data.

3.1 Information Gain

Several mathematical and computation techniques are used in the development of the algorithm: information gain, node purity, entropy and logarithm. Information gain is refers to the amount of useful information derived by diving data by an attribute given may have already been divided by a previous attribute (Gopalan, 2020).

Mathematically,

$$gain \ = \ \{previsous\ gain\} \ - \sum_{value\ =\ i}^{unique\ values} f(i) \ * \ P(i)$$

where:

P(i) is probability of value

F(i) is entropy of value

3.1.1 Entropy

Information entropy is an evaluation of the purity obtained if a dataset is split by the unique values of a feature. The purity obtained is evaluated by observing how the labels of each record within the split is distinguished from other labels of other unique values. A pure split would divide data perfectly using a feature to obtain well differentiated label values.

Mathematically,

$$entropy \ = \sum_{label\ values\ =\ i}^{unique\ label\ values} - n_i \ \log_2 n_i$$

where: **n** is a fraction of the number of records with that label value divided by the total number of records in data identifying a feature value.

3.2 Algorithm and Pseudo Code

```
func fit(data):

    if node expored or data is empty
        then return most occuring value in label column

    else
        then search for node (feature) with highest gain (i.e highest and non negative gain)
        identify the feature

        data_items = spit_data_by_feature(feature)

        create result_map with keys: feature_values and values: empty list

        for data_item in data_items
            value = fit(data_item)
            append value to result_map[feature_value]

        if node_level is highest:
            set tree_representation to result_map

        return result_map
```

```
    def recursive_fit(self, data, level, tested = [], prev_gain = 1):
        """
            Performs recursive fit over training data using information gain theory
            arguments:
        if level < 0: # if data is empty or depth is exceeded
            return max(data[self.label], key = data[self.label].count) # return most
occuring label value
        if len(tested) == len(self.features): #if all features have been tested
            return max(data[self.label], key = data[self.label].count)

      bestfeature_value_gain_list = [None, None, -math.inf]

        for feature in self.get_features(data):
            feature_val_gain = None
            if self.discrete_check(feature):
                if not f in tested:
                    feature_val_gain = self.compute_gain(feature, prev_gain)
                else:
                    continue
            else:
                feature_val_gain = self.n_search(feature, prev_gain)
            if bestfeature_value_gain_list[2] <= feature_val_gain[2]:
                bestfeature_value_gain_list = feature_val_gain

      # last leave reached
      if not bestfeature_value_gain_list[0]: # if no feature is selected.
            return max(data[self.label], key = data[self.label].count) # return most
frequently occurring label prediction

        result_store = [bestfeature_value_gain_list, {i:[] for i in
set(data[bestfeature_value_gain_list [0]])}] # node result is prepared and stored

        tested.append(store[0])
        t = self.split_data(data, store[0], store[1])
        for i in t: # splits data into for leaves to begin decision selection again.
            l = self.recursive_fit(t[i], level -1, tested.copy(), store[2])
            result_store[1][i].append(l)
        if level == self.count: # at top most node save result
            self.representation = result_store
        return result_store
```

(Mitchel & Balcan, 2015; Strobl et al., 2009; Mitchell, 2006; Bohanec & Rajkovič, 1988)

4 Testing

The unit test is prepared for all functions used in implementing the algorithm. Thus, the Pytest framework is used to implement these tests. The implementation passed all prepared unit tests module (Krekel, 2004).

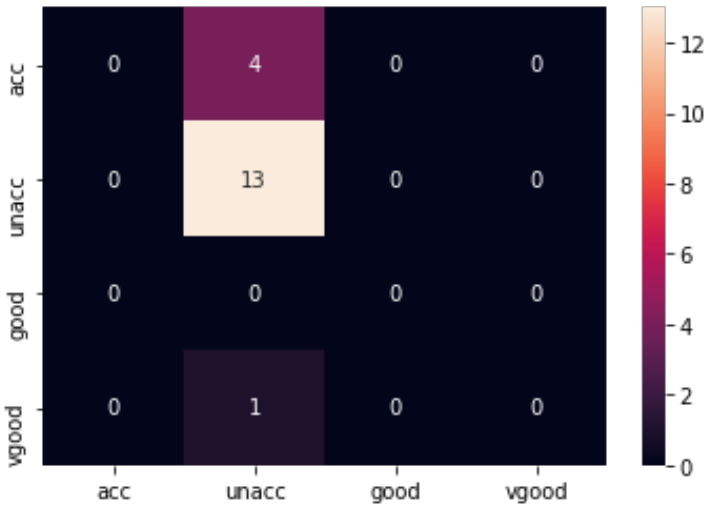
5 Performance Results

5.1 Accuracy

Tree Depth	Train (%) - Test (%) Ratio	Accuracy (%)
6	95:5	72.22
6	90:10	65.52
6	85:15	69.23
5	95:5	65.52
5	90:10	64.74
4	95:5	65.52

5.2 Confusion Matrix

A confusion matrix plot of the first experiment is shown below:



(Bhandari, 2020)

6 Conclusion

The following can be concluded from the result on the algorithm implemented:

- Decision trees overfit data.
- Decision trees work best only with normalized data.
- Decision trees can be used for classification tasks.
- An increased depth of decision tree implies better performance.
- Decision trees require more percentage of total data to train.
- The misclassifications observed were as a result of abnormal data distribution.

7 References

Bhandari, A. (2020). *Confusion Matrix for Machine Learning*. https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/

Bohanec, M. (1997). *UCI Machine Learning Repository: Data Set*. http://archive.ics.uci.edu/ml/datasets/machine-learning-databases/car/

Bohanec, M., & Rajkovič, V. (1988). KNOWLEDGE ACQUISITION AND EXPLANATION FOR MULTI-ATTRIBUTE DECISION MAKING *. *Proceedings of the 8th International Workshop 'Expert Systems and Their Applications AVIGNON 88,' 1*(Avignon 1988), 59–78.

Bohanec, M., & Rajkovič, V. (1990). DEX: An Expert System Shell for Decision Support *. *Sistemica*, *1*(1), 145–147.

Darlington, A. (2018). *Car Evaluation Data Set | Kaggle*. https://www.kaggle.com/elikplim/car-evaluation-data-set/version/1

Gopalan, B. (2020). What is Entropy and Information Gain? How are they used to construct decision trees? *Numpy Ninja*. https://www.numpyninja.com/post/what-is-entropy-and-information-gain-how-are-they-used-to-construct-decision-trees

Krekel, H. (2004). *pytest: helps you write better programs — pytest documentation*. https://docs.pytest.org/en/6.2.x/

Mitchel, T., & Balcan, M.-F. (2015). *Decision tree learning*. Machine Learning, Course 10-601. https://www.cs.cmu.edu/~bhiksha/courses/10-601/decisiontrees/

Mitchell, T. (2006). Machine Learning. In *Copyright*.

Strobl, C., Malley, J., & Tutz, G. (2009). An Introduction to Recursive Partitioning: Rationale,

Application and Characteristics of Classification and Regression Trees, Bagging and Random Forests. *Psychological Methods*, *14*(4), 323–348. https://doi.org/10.1037/a0016973