Dhynasah Çakir

Machine Learning: Project 3

Decision tree

Overview

This program implements the ID3 decision tree algorithm for classification. This approach uses entropy and information gain to perform the classification.

Approach

Python was used as the language for this program, although many different languages can be used for machine learning algorithms.

Overview of steps in program:

1. Combine testing and training data into one file. (reason is explained in discussion and results). Place data in a pandas data frame
2. Splits data into training and testing data frames
3. Calls decision tree algorithm function
    a. The parameter is the training data frame. Counter, minimum and maximum number of samples and maximum depth for tree are set
    b. The base case is set as:
        i. If the data is pure or the number of samples is less than the minimum samples or counter is equal to the maximum depth of tree, then we classify the data and stop there.
            1. Classify function takes the data as a parameter and determines the largest amount from that class column and returns that category.
    c. This next step is when recursion occurs
        i. Increment counter
        ii. Determine possible splits and the best split

1. Parameter for possible splits the data, this function creates a list of unique values for each attribute and returns that list
2. Best split then take that possible split list and determines the best column and split value based on overall entropy of that column returning those 2 variables.
    a. Overall entropy uses entropy function

iii. Then split at that value
    1. This function takes the best split and split value separates the given data abased on the split value

iv. Then we instantiate a dictionary for our tree

v. Call our decision tree algorithm again for each set of data from the split and attach the output for that branch to our tree.

vi. Return tree when complete

4. Print tree
5. Calculates accuracy of model

Discussion of results

The final tree has an accuracy of 72.7%. With a depth of 5. The root of the tree is persons = 2.

The result was originally was around 2% accuracy. I determined the reason for this extremely low accuracy is that many of the examples in the test set are more diverse then the training set.

As a way of troubleshooting this problem I decided to use only instances from the training set to also do the testing. I selected at random a small portion of instances for testing. This gave me an accuracy of 85%. Based on this outcome I knew my model was working correctly, But something else was the issue.

Next, I combined the training set and test set and used that for the training data, and kept the testing set the same as before. This gave me an accuracy of 54%. A lot better than 20% but not as good as 85%. At this point I realized the training set was extremely homogenous in the attribute characteristics and their classification. Once I combined the test set and the training set and selected the test sets randomly then I had a better predictive model. at around 72.7%. Although

the accuracy changes with every model, the most being 75%, and the least being 70% unless a seed is set.

I used the algorithm on the fishing examples from homework 2 and got an accuracy of 66% with a very short 1 branch tree. Therefore, this code can be applied to many different decision tree classification data sets.

Output of the dataset:

{'safety = low': ['unacc', {'persons = 2': ['unacc', {'maintenance = vhigh': [{'cost = vhigh': ['unacc', {'cost = high': ['unacc', 'acc']}]}, {'cost = low': [{'safety = med': ['acc', 'vgood']}, 'acc']}]}]}]}}

Accuracy is 2.0366598778004072 %

{'persons = 2': ['unacc', {'safety = low': ['unacc', {'maintenance = vhigh': [{'cost = vhigh': ['unacc', {'cost = high': ['unacc', 'acc']}]}, {'cost = low': [{'safety = med': ['acc', 'vgood']}, 'acc']}]}]}]}}

Accuracy is 82.0 %

{'persons = 2': [{'safety = medium': ['poor', {'trunk = medium': ['poor', {'cost = medium': ['poor', {'doors = 5': ['poor', 'unacc']}]}]}]}, {'safety = low': [{'persons = 6': ['poor', {'persons = more': ['unacc', {'maintenance = medium': ['poor', 'unacc']}]}]}, {'safety = medium': [{'cost = low': [{'maintenance = high': ['acceptable', 'good']}, {'trunk = small': ['poor', 'acceptable']}]}, {'safety = med': [{'trunk = small': ['unacc', 'acc']}, {'maintenance = vhigh': ['unacc', 'acc']}]}]}]}]}}

Accuracy is 54.378818737270876 %

{'persons = 2': [{'safety = medium': ['poor', {'trunk = medium': ['poor', {'cost = medium': ['poor', {'doors = 5': ['poor', 'unacc']}]}]}]}, {'safety = low': [{'persons = 6': ['poor', {'maintenance = medium': ['poor', {'cost = medium': ['poor', 'unacc']}]}]}, {'safety = med': [{'trunk = small': [{'cost = low': ['acc', 'unacc']}, {'maintenance = vhigh': ['unacc', 'acc']}]}, {'safety = medium': [{'trunk = small': ['poor', 'acceptable']}, {'maintenance = vhigh': ['unacc', 'acc']}]}]}]}}

Accuracy is 75.09090909090908 %

{'persons = 2': [{'safety = medium': ['poor',

{'doors = 5': ['poor',

{'maintenance = medium': ['poor',

{'trunk = medium': ['poor',

'unacc']}]}}]},

{'safety = low': [{'persons = 6': ['poor',

{'maintenance = medium': ['poor',

{'cost = medium': ['poor', 'unacc']}]}]},

{'safety = medium': [{'cost = low': [{'trunk = small': ['poor', 'good']},

{'cost = medium': ['acceptable', 'poor']}]},

{'safety = med': [{'trunk = big': ['acc', 'unacc']},

{'maintenance = vhigh': ['unacc', 'acc']}]}]}]}

Accuracy is 72.72727272727273 %

```
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 14 19:14:18 2019

@author: Mauri
"""

import numpy as np
import pandas as pd

import random

def train_test_split(df, test_size):

    if isinstance(test_size, float):
        test_size = round(test_size * len(df))

    indices = df.index.tolist()
    test_indices = random.sample(population=indices, k=test_size)

    test_df = df.loc[test_indices]
    train_df = df.drop(test_indices)

    return train_df, test_df
# this function determines if there is a split in the data based on
classification
def split(df, test_len):

    if isinstance(test_len, float):
        test_len = round(test_len * len(df))

    ind = df.index.tolist()
    test_ind = random.sample(population=ind, k=test_len)

    test_df = df.loc[test_ind]
    train_df = df.drop(test_ind)

    return train_df, test_df

def purity(data):

    class_col = data[:, -1]
    unique_classes = np.unique(class_col)

    if len(unique_classes) == 1:
        return True
    else:
        return False

def classify(data):
    class_col = data[:, -1]
    unique_classes, unique_clss_num = np.unique(class_col,
return_counts=True)
```

```
    ind = unique_clss_num.argmax()
    classif = unique_classes[ind]
    return classif

def features(df):

    feature_types = []
    n_unique_values_treshold = 15
    for feature in df.columns:
        if feature != "class":
            unique_values = df[feature].unique()
            example_value = unique_values[0]

            if (isinstance(example_value, str)) or (len(unique_values) <=
n_unique_values_treshold):
                feature_types.append("categorical")
            else:
                feature_types.append("continuous")

    return feature_types

def possible_splits(data):

    possible_splits = {}
    _, n_columns = data.shape
    for column_ind in range(n_columns - 1):          # excluding the last
column which is the label
        values = data[:, column_ind]
        unique_values = np.unique(values)

        feature = FEATURE_TYPES[column_ind]
        if feature == "continuous":
            possible_splits[column_ind] = []
            for ind in range(len(unique_values)):
                if ind != 0:
                    value1 = unique_values[ind] #current value
                    value2 = unique_values[ind - 1] #previous value
                    possible_split = (value1 + value2) / 2

                    possible_splits[column_ind].append(possible_split)

        # feature is categorical
        # (there need to be at least 2 unique values, otherwise in the
        # split_data function data_below would contain all data points
        # and data_above would be empty)
        elif len(unique_values) > 1:
            possible_splits[column_ind] = unique_values

    return possible_splits

def split_data(data, split_col, split_val):

    split_column_values = data[:, split_col]
```

```
    type_of_feature = FEATURE_TYPES[split_col]
    if type_of_feature == "continuous":
        below = data[split_column_values <= split_val]
        above = data[split_column_values >  split_val]

    # feature is categorical
    else:
        below = data[split_column_values == split_val]
        above = data[split_column_values != split_val]

    return below, above

def calc_overall_entropy(below, above):

    n = len(below) + len(above)
    p_below = len(below) / n
    p_above = len(above) / n

    overall_entropy =  (p_below * calculate_entropy(below)
                      + p_above * calculate_entropy(above))

    return overall_entropy

def calculate_entropy(data):

    class_col = data[:, -1]
    _, counts = np.unique(class_col, return_counts=True)

    prob = counts / counts.sum()
    entropy = sum(prob * -np.log2(prob))

    return entropy

def best_split(data, possible_splits):

    overall_entropy = 9999
    for column_ind in possible_splits:
        for value in possible_splits[column_ind]:
            below, above = split_data(data, split_col=column_ind,
split_val=value)
            current_overall_entropy = calc_overall_entropy(below, above)

            if current_overall_entropy <= overall_entropy:
                overall_entropy = current_overall_entropy
                best_split_column = column_ind
                best_split_value = value

    return best_split_column, best_split_value

def decision_tree_algorithm(df,counter=0, min_samples=2, max_samples=2,
max_depth=5):

    if counter ==0:
        global COLUMNS, FEATURE_TYPES
```

```
        COLUMNS = df.columns
        FEATURE_TYPES= features(df)
        data= df.values
    else:
        data= df


    #base case
    if(purity(data)) or (len(data) <min_samples) or (counter ==
max_depth):
        classification = classify(data)
        return classification
    #recursion
    else:
        counter+=1
        possible_split = possible_splits(data)
        col_split, split_val = best_split(data, possible_split)
        below, above = split_data(data, col_split, split_val)
        feature_name = COLUMNS[col_split]
        type_of_feature = FEATURE_TYPES[col_split]
        if type_of_feature == "continuous":
            question = "{} <= {}".format(feature_name, split_val)
        else:
            question = "{} = {}".format(feature_name, split_val)

        # instantiate sub-tree
        tree = {question: []}
        yes_answer = decision_tree_algorithm(below, counter, min_samples,
max_depth)
        no_answer = decision_tree_algorithm(above, counter, min_samples,
max_depth)

        if yes_answer == no_answer:
            tree = yes_answer
        else:
            tree[question].append(yes_answer)
            tree[question].append(no_answer)

        return tree

def classify_example(example, tree):
    question = list(tree.keys())[0]
    feature_name, comparison_operator, value = question.split(" ")

    # ask question
    if comparison_operator == "<=":
        if example[feature_name] <= float(value):
            answer = tree[question][0]
        else:
            answer = tree[question][1]

    # feature is categorical
    else:
        if str(example[feature_name]) == value:
```

```
            answer = tree[question][0]
        else:
            answer = tree[question][1]

    # base case
    if not isinstance(answer, dict):
        return answer

    # recursive part
    else:
        residual_tree = answer
        return classify_example(example, residual_tree)


def main():
    global FEATURE_TYPES
    # place dataset in a pandas dataframe
    train_data = pd.read_csv('is it a good day to fish.txt')
    train_data.columns =['wind','temp','water','time','sky','day','class']
    #train_data.columns =
['cost','maintenance','doors','persons','trunk','safety','class']
    test_data= pd.read_csv('car_test.txt')
    test_data.columns =
['cost','maintenance','doors','persons','trunk','safety','class']
    random.seed(0)

    train_df, test_df = train_test_split(train_data, test_size=3)

    tree = decision_tree_algorithm(train_df, max_depth=5)
    print(tree)
    test_df['classification']= test_df.apply(classify_example,
args=(tree,), axis=1)
    test_df['classification_correct'] = test_df['classification'] ==
test_df['class']
    accuracy =(test_df['classification_correct'].mean())*100

#    fish_data = pd.read_csv('is it a good day to fish.txt')
#    fish_data.columns =
['wind','temp','water','time','sky','day','class']
#    train_fish, test_fish = train_test_split(fish_data, test_size=3)
#    tree = decision_tree_algorithm(train_fish, max_depth=5)
#    test_fish['classification']= test_fish.apply(classify_example,
args=(tree,), axis=1)
#    test_fish['classification_correct'] = test_fish['classification'] ==
test_fish['class']

    accuracy = (test_df['classification_correct'].mean())*100
    print('Accuracy is ', accuracy, '%')

if __name__ == "__main__":
    main()
```