Decision Tree

March 19, 2021

0.1 A simple ID3 (Iterative Dichotomiser 3) Decision Tree Python Implementation

0.1.1 By Vishakan Subramanian

[1]:
```python
import math
```

[2]:
```python
def calc_entropy(p):
    """Calculate the entropy of a given numerical value."""

    if p != 0:
        return -p * math.log2(p)
    else:
        return 0
```

0.2 Finding Total Entropy

[3]:
```python
def calc_total_entropy(data):
    """Calculates the total entropy of the given dataset."""

    output_labels = list(set(label[3] for label in data))
    output_probs = []
    #print(output_labels)

    for label in output_labels:
        count = 0

        for data_row in data:
            if data_row[-1] == label:     #Assumes that the classifying label is␣
 ↪in the last column
                count += 1

        output_probs.append((count/len(data)))

    entropy = round(sum(calc_entropy(prob) for prob in output_probs), 3)

    return entropy
```

0.3 Calculating the Information Gain by filtering each feature

[4]:
```python
def calc_info_gain(total_entropy, feature_data, full_dataset):
    """Calculates the information gained by branching out towards a specific␣
 ↪feature. """

    feature_labels = list(set(label for label in feature_data))
    feature_probs = []
    subfeature_entropy_values = []
    info_gain = total_entropy
    #print(feature_labels)

    for label in feature_labels:
        count = 0
        indices = []
        filtered_data = []


        for i in range(len(feature_data)):
            if feature_data[i] == label:
                count += 1
                indices.append(i)


        for i in range(len(full_dataset)):
            if i in indices:
                indices.remove(i)
                filtered_data.append(full_dataset[i])

        subfeature_entropy_values.append(calc_total_entropy(filtered_data))

        feature_probs.append((count/len(feature_data)))

    #print(subfeature_entropy_values, feature_probs)

    entropy_lost = sum(subfeature_entropy_values[i] * feature_probs[i] for i in␣
 ↪range(len(feature_probs)))


    info_gain = total_entropy - entropy_lost
    #print(info_gain)

    return round(info_gain, 3)
```

0.4 Calculating the Information Gain for each feature of the dataset

[5]:
```python
def print_info_gains(features, data):
    """Print the information gain by branching out towards all existing features␣
↪of the dataset. """

    total_entropy = calc_total_entropy(data)
    feature_info_gains = []

    for i in range(len(features)):
        feature_values = [feature_set[i] for feature_set in data]
        feature_info_gains.append((features[i], calc_info_gain(total_entropy,␣
↪feature_values, data)))

    feature_info_gains.sort(key = lambda x: x[1], reverse = True)

    print("Feature ----- Gain\n")

    for gains in feature_info_gains:

        print("{0} ----- {1}".format(gains[0], gains[1]))

    if feature_info_gains[0][0] == features[-1]:     # if the maximum gain␣
↪feature is the o/p feature in list
        return feature_info_gains[1]
    else:
        return feature_info_gains[0]
```

0.5 Making the Decision Tree

[6]:
```python
def make_tree(features, data):
    """Make a decision tree based on branching out towards maximum information␣
↪gain feature."""

    iter = 1

    while True:
        print("Iteration:", iter, "\n")
        iter += 1
        max_gain_feature, max_info_gain = print_info_gains(features, data)

        if max_info_gain == 0:
            print("\n\t\tClassification complete!\n")
            break

        feature_index = features.index(max_gain_feature)
```

```
        temp_set = []
        flag = True

        for i in range(len(data)):
            temp_set.append([i, data[i][feature_index], data[i][-1]])

        #print(temp_set)
        feature_values = list(set(value[1] for value in temp_set))

        temp_set.sort(key = lambda x: x[1])

        for value in feature_values:
            feat_flag = True
            prev = temp_set[0][2]

            for i in range(len(temp_set)):
                if value != temp_set[i][1] and (i + 1) < len(temp_set):
                    prev = temp_set[i + 1][1]
                    break

                if temp_set[i][1] != prev:
                    feat_flag = False

            #print(feat_flag, value)
            if feat_flag == True:
                decrement = 1
                del_indices = []
                print("\nRemoved Data:")

                for i in range(len(temp_set)):
                    if value == temp_set[i][1] and (temp_set[i][0] - decrement)␣
↪< len(data):
                        print(temp_set[i])
                        del_indices.append(temp_set[i][0])
                        #del data[temp_set[i][0]]
                        decrement += 1

                for index in sorted(del_indices, reverse=True):  ␣
↪#discontinguous array, so sorting
                    del data[index]                               #necessary to␣
↪remove properly


                print("\nRemaining Data:")
                for d in data:
                    print(d)
```

```
                temp_set = []
                for i in range(len(data)):   #bug fix, recalculate the temp set␣
↪to avoid index mismatch
                    temp_set.append([i, data[i][feature_index], data[i][-1]])

                print("\n")
                #break
```

[7]:

```
#Dataset-1 : Activity is the output label. Deadline?, Party? and Lazy? are␣
↪feature labels.

features = ["Deadline?", "Party?", "Lazy?", "Activity"]

data = [["Urgent", "Yes", "Yes", "Party"],
        ["Urgent", "No", "Yes", "Study"],
        ["Near", "Yes", "Yes", "Party"],
        ["None", "Yes", "No", "Party"],
        ["None", "No", "Yes", "Pub"],
        ["None", "Yes", "No", "Party"],
        ["Near", "No", "No", "Study"],
        ["Near", "No", "Yes", "TV"],
        ["Near", "Yes", "Yes", "Party"],
        ["Urgent", "No", "No", "Study"]]
```

[8]:

```
make_tree(features, data)
```

```
Iteration: 1

Feature ----- Gain

Activity ----- 1.685
Party? ----- 1.0
Deadline? ----- 0.534
Lazy? ----- 0.21

Removed Data:
[0, 'Yes', 'Party']
[2, 'Yes', 'Party']
[3, 'Yes', 'Party']
[5, 'Yes', 'Party']
[8, 'Yes', 'Party']

Remaining Data:
['Urgent', 'No', 'Yes', 'Study']
['None', 'No', 'Yes', 'Pub']
['Near', 'No', 'No', 'Study']
['Near', 'No', 'Yes', 'TV']
```

```
['Urgent', 'No', 'No', 'Study']


Iteration: 2

Feature ----- Gain

Activity ----- 1.371
Deadline? ----- 0.971
Lazy? ----- 0.42
Party? ----- 0.0

Removed Data:
[0, 'Urgent', 'Study']
[4, 'Urgent', 'Study']

Remaining Data:
['None', 'No', 'Yes', 'Pub']
['Near', 'No', 'No', 'Study']
['Near', 'No', 'Yes', 'TV']


Iteration: 3

Feature ----- Gain

Activity ----- 1.585
Deadline? ----- 0.918
Lazy? ----- 0.918
Party? ----- 0.0

Removed Data:
[0, 'None', 'Pub']

Remaining Data:
['Near', 'No', 'No', 'Study']
['Near', 'No', 'Yes', 'TV']


Iteration: 4

Feature ----- Gain

Lazy? ----- 1.0
Activity ----- 1.0
Deadline? ----- 0.0
Party? ----- 0.0
```

```
Removed Data:
[1, 'Yes', 'TV']

Remaining Data:
['Near', 'No', 'No', 'Study']


Iteration: 5

Feature ----- Gain

Deadline? ----- 0.0
Party? ----- 0.0
Lazy? ----- 0.0
Activity ----- 0.0

                Classification complete!
```

[9]:

```python
#Dataset-2 : Attractive? is the output label. Height, Hair and Eyes are feature␣
↪labels.

features = ["Height", "Hair", "Eyes", "Attractive?"]

data = [["Small", "Blonde", "Brown", "No"],
        ["Tall", "Dark", "Brown", "No"],
        ["Tall", "Blonde", "Blue", "Yes"],
        ["Tall", "Dark", "Blue", "No"],
        ["Small", "Dark", "Blue", "No"],
        ["Tall", "Red", "Blue", "Yes"],
        ["Tall", "Blonde", "Brown", "No"],
        ["Small", "Blonde", "Blue", "Yes"]]

make_tree(features, data)
```

```
Iteration: 1

Feature ----- Gain

Attractive? ----- 0.954
Hair ----- 0.454
Eyes ----- 0.347
Height ----- 0.003

Removed Data:
[5, 'Red', 'Yes']

Remaining Data:
```

```
['Small', 'Blonde', 'Brown', 'No']
['Tall', 'Dark', 'Brown', 'No']
['Tall', 'Blonde', 'Blue', 'Yes']
['Tall', 'Dark', 'Blue', 'No']
['Small', 'Dark', 'Blue', 'No']
['Tall', 'Blonde', 'Brown', 'No']
['Small', 'Blonde', 'Blue', 'Yes']



Removed Data:
[1, 'Dark', 'No']
[3, 'Dark', 'No']
[4, 'Dark', 'No']

Remaining Data:
['Small', 'Blonde', 'Brown', 'No']
['Tall', 'Blonde', 'Blue', 'Yes']
['Tall', 'Blonde', 'Brown', 'No']
['Small', 'Blonde', 'Blue', 'Yes']


Iteration: 2

Feature ----- Gain

Eyes ----- 1.0
Attractive? ----- 1.0
Height ----- 0.0
Hair ----- 0.0

Removed Data:
[0, 'Brown', 'No']
[2, 'Brown', 'No']

Remaining Data:
['Tall', 'Blonde', 'Blue', 'Yes']
['Small', 'Blonde', 'Blue', 'Yes']


Iteration: 3

Feature ----- Gain

Height ----- 0.0
Hair ----- 0.0
Eyes ----- 0.0
Attractive? ----- 0.0
```

```
Classification complete!
```

[10]:
```python
#Dataset-3 : Goes to Pub? is the output label. Drink, Gender, Student are␣
 ↪feature labels.

features = ["Drink", "Gender", "Student", "Pub"]

data = [["Beer", "T", "T", "T"]
        ,["Beer", "T", "F", "T"]
        ,["Vodka", "T", "F", "F"]
        ,["Vodka", "T", "F", "F"]
        ,["Vodka", "F", "T", "T"]
        ,["Vodka", "F", "F", "F"]
        ,["Vodka", "F", "T", "T"]
        ,["Vodka", "F", "T", "T"]]

make_tree(features, data)
```

```
Iteration: 1

Feature ----- Gain

Pub ----- 0.954
Student ----- 0.548
Drink ----- 0.204
Gender ----- 0.048

Removed Data:
[0, 'T', 'T']
[4, 'T', 'T']
[6, 'T', 'T']
[7, 'T', 'T']

Remaining Data:
['Beer', 'T', 'F', 'T']
['Vodka', 'T', 'F', 'F']
['Vodka', 'T', 'F', 'F']
['Vodka', 'F', 'F', 'F']


Iteration: 2

Feature ----- Gain

Drink ----- 0.811
Pub ----- 0.811
```

```
Gender ----- 0.123
Student ----- 0.0

Removed Data:
[1, 'Vodka', 'F']
[2, 'Vodka', 'F']
[3, 'Vodka', 'F']

Remaining Data:
['Beer', 'T', 'F', 'T']


Iteration: 3

Feature ----- Gain

Drink ----- 0.0
Gender ----- 0.0
Student ----- 0.0
Pub ----- 0.0

                Classification complete!
```