
ID3

Use case ID3

For the ID3 algorithm evaluation we will use the METABRIC dataset from cBioportal. This breast cancer dataset contains clinical features. The aim is to predict survival class (vital status). For the development of the algorithm we prepared a small dataset derived from the METABRIC dataset. The dataset is called `sample.txt` and can be downloaded here

Once the algorithm is validated you can use the full METABRIC dataset from bioportal.

Part D. Implementation of ID3 algorithm

The ID3 algorithm (Iterative Dichotomiser 3) is an algorithm used for constructing decision trees, which are commonly used in machine learning for classification tasks. A decision tree can be visualized as a flowchart-like structure where each internal node represents a test on a feature, each branch represents the outcome of the test, and each leaf node represents a class label. The ID3 algorithm is basis to more advanced decision tree algorithms such as C45. The algorithm selects the features that best separates the data based on a measure called **information gain**. Information gain is calculated using the concept of **entropy** from information theory. Entropy measures the impurity or uncertainty in a dataset, Information Gain measures the reduction in entropy after splitting a dataset based on an feature (attribute). The higher the Information Gain, the more informative the feature is for classification.

Entropy

$$\text{Entropy} = -\sum_{i=1}^{n} p_i \cdot \log\left(p_i\right)$$

where:

- p_i represents the individual probabilities in the distribution.
- The summation is taken over all probabilities p_i where $p_i > 0$ ($p_i > 0$ avoids undefined operations for $\log\left(0\right)$)

The choice of base for log varies for different applications. Base 2 gives the unit of bits (or "shannons"), while base e gives "natural units" nat, and base 10 gives units of "dits", "bans", or "hartleys". https://en.wikipedia.org/wiki/Entropy_(information_theory)

Information Gain in Decision Trees

In the context of decision trees, Information Gain is used to determine the best feature to split the data at each node. The feature with the highest Information Gain is selected for the split.

1. **Calculate the entropy of the entire dataset**:

$$\text{Entropy}(D) = -\sum_{i=1}^{n} p_i \cdot \log(p_i)$$

where p_i is the probability of class i in the entire dataset D.

2. **For each attribute** A:
   - Calculate the entropy for each subset D_v (where v is a possible value of A):

$$\text{Entropy}(D_v) = -\sum_{i=1}^{n} p_{iv} \cdot \log(p_{iv})$$

   where p_{iv} is the probability of class i in subset D_v.
   - Calculate the weighted sum of the entropies of all subsets:

$$\sum_{v \in \text{Values}(A)} \frac{|D_v|}{|D|} \cdot \text{Entropy}(D_v)$$

   where $|D_v|$: The number of elements in subset D_v and $|D|$: The total number of elements in dataset D

3. **Calculate the Information Gain for attribute (A)**:

$$\text{IG}(A) = \text{Entropy}(D) - \sum_{v \in \text{Values}(A)} \frac{|D_v|}{|D|} \cdot \text{Entropy}(D_v)$$

where $\text{Values}(A)$: The set of all possible values that attribute A can take

ID3 algorithm

1. select best feature based on information gain
2. create a node for the selected feature
3. split the data into subsets based on the possible values of the selected feature
4. recursive apply algorithm on each subset, select next best feature until all samples in subset belong to a class or there are no more features
5. use manority voting for class if several class labels remain in a subset

NB:

- to prevent overfitting you can set a max_depth, like pruning a tree.

Quinlan, J. R. 1986. Induction of Decision Trees. Mach. Learn. 1, 1 (Mar. 1986), 81–106

Implementation Task

- Download the `sample.txt` dataset here

- Complete the `entropy()`, `information_gain()`, and `ID3()` functions. Feel free to add additional functions if necessary to enhance the implementation.
- Consider to adjust the code with a `max_depth` property.
- Use the code snippets to create your own ID3 classifier. This we will use in part E and F.

Expected outcome

The expected outcome using `sample.txt` (without splitting) for the initial gain calculations is as follows:

```
Entropy of the target (Vital Status): 0.97
feature Histological Type gain 0.00
feature Grade gain 0.02
feature Tumor Stage gain 0.07
feature ER Status gain 0.02
best feature: Tumor Stage
```

The result of the ID3 algorithm is a nested dictionary with the best features and the targets like this

```
{'Tumor Stage': {'stage iii': {'Histological Type': {'ilc': 'alive',
'other': 'dead', 'idc': 'alive'}}, 'stage i': {'Histological Type':
{'ilc': 'dead', 'mixed type': 'dead', 'other': 'alive', 'idc':
'alive'}}, 'stage iv': {'Histological Type': {'ilc': 'alive', 'idc':
'dead'}}, 'stage ii': {'Grade': {'grade 2': 'alive', 'grade 3':
'dead'}}}}
```

Which can be represented in a tree using the `plotter.print_tree(decision_tree)` function

```
Decision Tree:
Tumor Stage
      stage iii ->
            Histological Type
                  ilc ->                    alive
                  other ->              dead
                  idc ->                    alive
      stage i ->
            Histological Type
                  ilc ->                    dead
                  mixed type ->             dead
                  other ->              alive
                  idc ->                    alive
      stage iv ->
            Histological Type
                  ilc ->                    alive
                  idc ->                    dead
      stage ii ->
            Grade
```

```
                    grade 2 ->                  alive
                    grade 3 ->                  dead
```

A more fancy plot can be created using the `plotter.create_plot(decision_tree)` function tree.png

Bonus

Use networkX to present the tree in a graph

```
# YOUR CODE HERE TO IMPORT THE SAMPLE.TXT DATASET
import pandas as pd
df = pd.read_csv('sample.txt', delimiter='\t')
# print(df)
# data = df.to_numpy()
data = df
data
```

	Histological Type	Grade	Tumor Stage	ER Status	Vital Status
0	ILC	Grade 3	Stage III	Positive	Alive
1	ILC	Grade 2	Stage I	Negative	Dead
2	Mixed Type	Grade 3	Stage I	Negative	Dead
3	Other	Grade 1	Stage I	Negative	Alive
4	ILC	Grade 2	Stage IV	Negative	Dead
5	Mixed Type	Grade 1	Stage I	Negative	Dead
6	Other	Grade 1	Stage III	Positive	Dead
7	IDC	Grade 3	Stage I	Negative	Alive
8	Mixed Type	Grade 2	Stage II	Positive	Alive
9	ILC	Grade 3	Stage II	Positive	Dead
10	IDC	Grade 3	Stage IV	Negative	Dead
11	IDC	Grade 2	Stage II	Positive	Alive
12	IDC	Grade 1	Stage III	Positive	Alive
13	Mixed Type	Grade 1	Stage I	Positive	Dead
14	ILC	Grade 2	Stage IV	Negative	Alive
15	IDC	Grade 1	Stage I	Negative	Dead
16	Other	Grade 2	Stage I	Positive	Dead
17	IDC	Grade 2	Stage III	Negative	Dead
18	IDC	Grade 3	Stage II	Negative	Dead
19	Mixed Type	Grade 2	Stage II	Negative	Alive

```
# YOUR CODE HERE
import numpy as np
def entropy(y):
    _, counts = np.unique(y, return_counts=True)
    p = counts / counts.sum()
    D = -np.sum(p * np.log2(p))
    return D
```

```
def information_gain(data, feature, target):
    IG = entropy(target)
    for val in np.unique(data[feature]):
        mask = data[feature] == val
        subdata = data[mask]
        IG -= len(subdata)/len(data) * entropy(target[mask])
    return IG


def id3(data, features, target, depth=0, max_depth=None,
gain_threshold=1e-3):
    y = data[target]
    gain, best_feature = max(((information_gain(data, feature, y),
feature) for feature in features))

    result = {best_feature: {}}
    for value in data[best_feature].unique():
        subdata = data[data[best_feature] == value]
        sub_y = y[data[best_feature] == value]
        if (max_depth and depth == max_depth) or gain <
gain_threshold:
            # BASE CASE
            subresult = sub_y.mode()[0]
        else:
            # RECURSIVE CASE
            subresult = id3(subdata, features, target, depth + 1,
max_depth, gain_threshold)
            # COMBINATION
        result[best_feature][value] = subresult
    return result

    # Expected result to return:
    # {'Tumor Stage': {'stage iii': {'Histological Type': {'ilc':
'alive', 'other': 'dead', 'idc': 'alive'}},
    #                  'stage i': {'Histological Type': {'ilc':
'dead', 'mixed type': 'dead', 'other': 'alive', 'idc': 'alive'}},
    #                  'stage iv': {'Histological Type': {'ilc':
'alive', 'idc': 'dead'}},
    #                  'stage ii': {'Grade': {'grade 2': 'alive',
'grade 3': 'dead'}}
    #                 }
    # }

    pass


# Define target and features explicitly
target = data.columns[-1]
features = data.columns[:-1]
```

```python
# Calculate the entropy of the target variable (Class)
y = data[target]
print(f"Entropy of the target ({target}): {entropy(y)}")

# Build the decision tree using the ID3 algorithm
decision_tree = id3(data, features, target, max_depth=2)

# Print the resulting decision tree
print("Decision Tree:")
from pprint import pprint
pprint(decision_tree)
#plot results
# plotter.print_tree(decision_tree)
#plotter.create_plot(decision_tree)

sample = data[features].iloc[0]
print(sample)
tree = decision_tree
while True:
    feature = list(tree.keys())[0]
    value = sample[feature]
    try:
        tree = tree[feature][value]
    except KeyError:
        decision = np.nan
    else:
        if isinstance(tree, str):
            decision = tree
            break

print(f'You are {decision}!')
```

```
Entropy of the target (Vital Status): 0.9709505944546686
Decision Tree:
{'Tumor Stage': {'Stage I': {'Histological Type': {'IDC': {'Grade':
{'Grade 1': 'Dead',

'Grade 3': 'Alive'}},
                                                   'ILC': {'Tumor
Stage': {'Stage I': 'Dead'}},
                                                   'Mixed Type':
{'Tumor Stage': {'Stage I': 'Dead'}},
                                                   'Other': {'Grade':
{'Grade 1': 'Alive',

'Grade 2': 'Dead'}}}},
                 'Stage II': {'Grade': {'Grade 2': {'Tumor Stage':
{'Stage II': 'Alive'}},
```

```
                                          'Grade 3': {'Tumor Stage':
{'Stage II': 'Dead'}}}},
                 'Stage III': {'Histological Type': {'IDC': {'Grade':
{'Grade 1': 'Alive',

'Grade 2': 'Dead'}},
                                                    'ILC': {'Tumor
Stage': {'Stage III': 'Alive'}},
                                                    'Other': {'Tumor
Stage': {'Stage III': 'Dead'}}}},
                 'Stage IV': {'Histological Type': {'IDC': {'Tumor
Stage': {'Stage IV': 'Dead'}},
                                                   'ILC': {'Tumor
Stage': {'Stage IV': 'Alive'}}}}}}
Histological Type          ILC
Grade                   Grade 3
Tumor Stage           Stage III
ER Status              Positive
Name: 0, dtype: object
You are Alive!
```

```
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.utils.multiclass import unique_labels
import numpy as np

class ID3Tree(ClassifierMixin, BaseEstimator):
    def __init__(self, max_depth=None, gain_threshold=1e-3):
        self.max_depth = max_depth
        self.gain_threshold = gain_threshold

    def id3(self, data, y, depth=0):
        gain, best_feature = max(((self.information_gain(data,
feature, y), feature)
                                  for feature in range(X.shape[1])))
        result = {best_feature: {}}
        for value in np.unique(data[:, best_feature]):
            mask = data[:, best_feature] == value
            subdata = data[mask, :]
            sub_y = y[mask, :]
            if (self.max_depth and depth == self.max_depth) or gain <
self.gain_threshold:
                # BASE CASE
                subresult = sub_y.mean(axis=0)
            else:
                # RECURSIVE CASE
                subresult = self.id3(subdata, sub_y, depth + 1)
            # COMBINATION
            result[best_feature][value] = subresult
        return result
```

```
    def entropy(self, y):
        _, counts = np.unique(y, return_counts=True, axis=0)
        p = counts / counts.sum()
        D = -np.sum(p * np.log2(p))
        return D

    def information_gain(self, data, feature, target):
        IG = self.entropy(target)
        for val in np.unique(data[:, feature]):
            mask = data[:, feature] == val
            subdata = data[mask, :]
            IG -= len(subdata)/len(data) *
self.entropy(target[mask, :])
        return IG

    def fit(self, X, y):
        self.classes_ = unique_labels(y)
        onehot = y[:, np.newaxis] == self.classes_[np.newaxis]
        # Our tree is a dictionary, which maps a single feature, to a
dictionary of values.
        # These values map either to a tree, or a leaf node
(=probabilities)
        self.tree_ = self.id3(X, onehot, 0)
        return self

    def predict_proba(self, X):
        result = []
        for sample in X:
            tree = self.tree_
            while True:
                feature = list(tree.keys())[0]
                value = sample[feature]
                try:
                    tree = tree[feature][value]
                except KeyError:
                    # What happens if you have a stage V tumour?
                    result.append([np.nan] * len(self.classes_))
                    break
                else:
                    if not isinstance(tree, dict):
                        # We've reached a leaf node.
                        result.append(tree)
                        break
        return np.array(result)

    def predict(self, X):
        probas = self.predict_proba(X)
        idxs = np.argmax(probas, axis=1)
        return self.classes_[idxs]
```

```
data = df.to_numpy()
X = data[:, :-1]
y = data[:, -1]
tree = ID3Tree(max_depth=2)
tree.fit(X, y)
result = tree.predict(X)
pprint(tree.tree_)
print(result)
print(y)
```

```
{2: {'Stage I': {0: {'IDC': {1: {'Grade 1': array([0., 1.]),
                                 'Grade 3': array([1., 0.])}},
                     'ILC': {3: {'Negative': array([0., 1.])}},
                     'Mixed Type': {3: {'Negative': array([0., 1.]),
                                        'Positive': array([0., 1.])}},
                     'Other': {3: {'Negative': array([1., 0.]),
                                   'Positive': array([0., 1.])}}}},
     'Stage II': {1: {'Grade 2': {3: {'Negative': array([1., 0.]),
                                      'Positive': array([1., 0.])}},
                      'Grade 3': {3: {'Negative': array([0., 1.]),
                                      'Positive': array([0., 1.])}}}},
     'Stage III': {1: {'Grade 1': {0: {'IDC': array([1., 0.]),
                                       'Other': array([0., 1.])}},
                       'Grade 2': {3: {'Negative': array([0., 1.])}},
                       'Grade 3': {3: {'Positive': array([1.,
0.])}}}},
     'Stage IV': {1: {'Grade 2': {3: {'Negative': array([0.5, 0.5])}},
                      'Grade 3': {3: {'Negative': array([0.,
1.])}}}}}}}
['Alive' 'Dead' 'Dead' 'Alive' 'Alive' 'Dead' 'Dead' 'Alive' 'Alive'
 'Dead' 'Dead' 'Alive' 'Alive' 'Dead' 'Alive' 'Dead' 'Dead' 'Dead'
'Dead'
 'Alive']
['Alive' 'Dead' 'Dead' 'Alive' 'Dead' 'Dead' 'Dead' 'Alive' 'Alive'
'Dead'
 'Dead' 'Alive' 'Alive' 'Dead' 'Alive' 'Dead' 'Dead' 'Dead' 'Dead'
'Alive']
```

```
from sklearn.model_selection import GridSearchCV

tree = ID3Tree()
param_grid = {
    'max_depth': [1, 2, 3, 4, 5],
    'gain_threshold': [1e-2, 1e-3, 1e-4, 1e-5]
}
grid = GridSearchCV(tree, param_grid, cv=5)
grid.fit(X, y)
print(grid.best_params_)
print(grid.best_score_)
```

```
{'gain_threshold': 0.01, 'max_depth': 1}
0.55
```