Sprawozdanie

Zbieranie i Analiza Danych

Labolatorium nr 6 – Implementacja drzewa decyzyjnego ID3

Oskar Kwidziński

S156013

Fizyka Techniczna

Semestr zimowy 2022/2023

Wstęp

Algorytm drzewa decyzyjnego ID3 jest algorytmem wyznaczającym . Cechą charakterystyczną algorytmu jest wybór kolejnych atrybutów, na które dzielone jest drzewo, pod względem kryterium „information gain”. Kolejno przeprowadzane są testy, aby końcowe drzewo było jak najprostsze i jak najefektywniejsze. Wybór atrybutów opiera się na liczeniu entropii, co pozwala obliczyć, wybór którego z atrybutów da największy przyrost informacji. Takim atrybutem jest ten, który podzieli zbiór przykładów na jak najbardziej równe podzbiory.

Rozwiązanie

W zadaniu zaprezentowano rozwiązanie dla prostego zbioru danych, pozwalającego podjąć decyzję na temat wyboru pomiędzy restauracją i food truckiem na podstawie atrybutów: Mood, Hunger oraz Money:

Mood	Hunger	Money	Choice
Good	Extreme	High	Restaurant
Average	Peckish	Low	Foodtruck
Average	Extreme	High	Restaurant
Good	Peckish	High	Restaurant
Bad	Extreme	High	Restaurant
Bad	Peckish	High	Foodtruck
Average	Extreme	High	Restaurant
Average	Extreme	Low	Foodtruck
Average	Average	Low	Foodtruck
Good	Average	Low	Foodtruck
Bad	Peckish	Low	Foodtruck
Average	Peckish	High	Restaurant

Aby podjąć decyzję, który z atrybutów podzieli zbiór danych najlepiej, obliczany jest przyrost informacji osiągnięty na podstawie wyboru tego atrybutu. Jest to obliczane, dla każdego z dostępnych atrybutów, jako różnica pomiędzy istniejącą w zbiorze entropią i przefiltorowanymi danymi zawierającymi określoną wartość atrybutu:

```python
def find_next_node(dataset, label, classes):
    features_info_dict = {}
    feature_names = list(dataset.columns.drop(label))
    for feature in feature_names:
        feature_info_gain = calculate_information_gain(dataset, label, classes, feature)
        features_info_dict[feature] = feature_info_gain
    max_info_feature = max(features_info_dict, key=features_info_dict.get)
    return max_info_feature
```

Do obliczenia najbardziej korzystnego wyboru atrybutu wykorzystywane są pomocnicze funkcje calculate_entropy oraz calculate_information_gain, które kolejno wyznaczają entorpię zestawu danych oraz wyznaczają ilość informacji uzyskiwanej w wyniku podziału zbioru danych ze względu na każdy atrybut:

```python
def calculate_entropy(dataset, label, classes):
    total_entropy = 0
    dataset_size = len(dataset.index)
    for label_class in classes:
        class_count = dataset[label].value_counts()[label_class]
        class_probability = class_count / dataset_size
        class_entropy = - class_probability * np.log2(class_probability)
        total_entropy += class_entropy
    return total_entropy
```

```python
def calculate_information_gain(dataset, label, classes, feature):
    feature_information = 0
    feature_values = dataset[feature].unique()
    dataset_size = len(dataset.index)
    for feature_value in feature_values:
        feature_entropy = 0
        feature_size = dataset[feature].value_counts()[feature_value]
        for feature_label_class in classes:
            feature_label_class_count = dataset[(dataset[label] == feature_label_class) & (dataset[feature] == feature_value)].shape[0]
            if feature_label_class_count != 0:
                feature_class_probability = feature_label_class_count / feature_size
                feature_value_entropy = - feature_class_probability * np.log2(feature_class_probability)
                feature_entropy += feature_value_entropy
        feature_value_probability = feature_size / dataset_size
        feature_value_info_gain = feature_value_probability * feature_entropy
        feature_information += feature_value_info_gain
    information_gain = calculate_entropy(dataset, label, classes) - feature_information
    return information_gain
```

Następnie budowany jest podział drzewa w oparciu o wybrany atrybut z funkcji „find_next_node” nazwany „max_info_feature”:

```python
def generate_sub_tree(dataset, label, feature):
    tree = {}
    feature_values = dataset[feature].unique()
    for feature_value in feature_values:
        feature_label_values = dataset[dataset[feature] == feature_value]
        has_unique_label = feature_label_values[label].eq(feature_label_values[label].iloc[0]).all()
        if has_unique_label:
            feature_label_class = feature_label_values.iloc[0, feature_label_values.columns.get_loc(label)]
            tree[feature_value] = feature_label_class
        else:
            tree[feature_value] = 'Unknown'
    return tree
```

W przypadku gdy dla danej wartości wybranego atrybutu otrzymujemy liść, do słownika „tree” dodawane jest hasło w postaci: atrybut – wartość atrybutu. W przeciwnym przypadku (gdy nie otrzymano czystego węzła), do słownika „tree” dodawane jest hasło w postaci: atrybut – Unknown. Taki zabieg ma posłużyć dalszemu rozwijania gałęzi w tę stronę w późniejszej implementacji algorytmu.

Zabieg dla każdej z gałęzi jest kontynuowany w rekursywnie wywoływanej funkcji „make_tree”, aż do momentu gdy wyeliminowane zostaną wszystkie wartości „Unknown” w tej gałęzi drzewa. Proces kończy swoje działanie w momencie gdy nieprzypisane rekordy do żadnego z liści się wyczerpią:

```python
def make_tree(dataset, label, classes, old_tree, feature=None):
    if not dataset.empty:
        max_info_feature = find_next_node(dataset, label, classes)
        tree = generate_sub_tree(dataset, label, max_info_feature)
        if not bool(old_tree):
            old_tree[max_info_feature] = tree
            next_tree = old_tree[max_info_feature]
        else:
            old_tree[feature] = {}
            old_tree[feature][max_info_feature] = tree
            next_tree = old_tree[feature][max_info_feature]
        for node, branch in next_tree.items():
            if branch == 'Unknown':
                feature_value_dataset = dataset[dataset[max_info_feature] == node]
                make_tree(feature_value_dataset, label, classes, next_tree, node)
```

Warto zwrócić uwagę, że w podanej implementacji, po każdym utworzeniu gałęzi zbiór danych jest „przycinany”, aby wyeliminować powtórne klasyfikowanie wartości atrybutu do wybranego liścia drzewa. W kolejnych krokach implementacji wybierane są z zestawu danych tylko te rekordy, dla których wartość obecnie rozpatrywanego atrybutu nie dała odpowiedzi jak go sklasyfikować - ‘Uknown’. W ten sposób, dzięki ponownemu wywołaniu funkcji „make_tree” drzewo jest rozbudowywane o kolejne gałęzie i liście.

Podsumowanie

W wyniku działania programu udało się poprawnie sklasyfikować każdą z decyzji na podstawie utworzonego zbioru danych, co zostało zaprezentowane w postaci drzewa jako output działania programu:


```
{
    "Money": {
        "High": {
            "Mood": {
                "Average": "Restaurant",
                "Bad": {
                    "Hunger": {
                        "Extreme": "Restaurant",
                        "Peckish": "Foodtruck"
                    }
                },
                "Good": "Restaurant"
            }
        },
        "Low": "Foodtruck"
    }
}
```


Warto jednak zwrócić uwagę na wady tego algorytmu:

- ID3 nie zakłada pruningu (przycinania) drzew,

- ID3 nie radzi sobie z danymi numerycznymi (istnieją co prawda rozwiązania pozwalające na grupowanie określonych danych numerycznych) oraz wartościami brakującymi,

- w przypadku gdy ID3 nie jest w stanie osiągnąć „czystego” liścia algorytm wpada w rekursję prowadzącą do wywoływania w kółko tej samej funkcji – w przypadku języka Python wyrzuca to wyjątek „max depth recursion”.