CPT Documentation

Bluesheeptoken

Feb 28, 2023

CONTENTS:

1 A sequence prediction algorithm **1**

1.1 Introduction . . . 1

1.2 Examples . . . 1

1.3 The `Cpt` class . . . 2

1.4 Tuning . . . 5

Index **7**

CHAPTER

ONE

A SEQUENCE PREDICTION ALGORITHM

1.1 Introduction

This project is a cython open-source implementation of the Compact Prediction Tree algorithm using multithreading.

CPT is a sequence prediction algorithm. It is a highly explainable model and good at predicting, in a finite alphabet, next value of a sequence. However, given a sequence, CPT cannot predict an element already present in this sequence (Cf "Tuning" part).

This implementation is based on the following research papers

http://www.philippe-fournier-viger.com/ADMA2013_Compact_Prediction_trees.pdf

http://www.philippe-fournier-viger.com/spmf/PAKDD2015_Compact_Prediction_tree+.pdf

1.2 Examples

1.2.1 Hello World example

You can test the model with the following code

```python
from cpt.cpt import Cpt
model = Cpt()

model.fit([['hello', 'world'],
           ['hello', 'this', 'is', 'me'],
           ['hello', 'me']
          ])

model.predict([['hello'], ['hello', 'this']])
# Output: ['me', 'is']
```

1.2.2 Sklearn Example

This code is also compatible with sklearn tools such as `Gridsearch`

```python
from sklearn.base import BaseEstimator
from cpt.cpt import Cpt
from sklearn.model_selection import GridSearchCV


class SKCpt(Cpt, BaseEstimator):
    def __init__(self, split_length=0, noise_ratio=0, MBR=0):
        super().__init__(split_length, noise_ratio, MBR)

    def score(self, X):
        # Choose your own scoring function here
        predictions = self.predict(list(map(lambda x: x[self.split_length:-1], X)))
        score = sum([predictions[i] == X[i][-1] for i in range(len(X))]) / len(X) * 100
        return score

data = [['hello', 'world'], ['hello', 'cpt'], ['hello', 'cpt']]


tuned_params = {'MBR': [0, 5], 'split_length': [0, 1, 5]}

gs = GridSearchCV(SKCpt(), tuned_params)

gs.fit(data)

gs.cv_results_
```

1.3 The `Cpt` class

class `cpt.cpt.`**`Cpt`**

Compact Prediction Tree class.

Attributes

split_length
[int, default 0 (all elements are considered)] The split length is used to delimit the length of training sequences.

noise_ratio
[float, default 0 (no noise)] The threshold of frequency to consider elements as noise.

MBR
[int, default 0 (at least one update)] Minimum number of similar sequences needed to compute predictions.

alphabet
[Alphabet] The alphabet is used to encode values for Cpt. `alphabet` should not be used directly.

Methods

compute_noisy_items	Compute noisy elements.
find_similar_sequences	Find similar sequences.
fit	Train the model with a list of sequence.
predict	Predict the next element of each sequence of the parameter `sequences`.
predict_k	Predict the next elements of each sequence of the parameter `sequences`, sorted by descending confidence.
retrieve_sequence	Retrieve sequence from the training data.

`fit`(*sequences*)

Train the model with a list of sequence.

The model can be retrained to add new sequences. `model.fit(seq1);model.fit(seq2)` is equivalent to `model.fit(seq1 + seq2)` with seq1, seq2 list of sequences.

Parameters

sequences
[list] A list of sequences of any hashable type.

Returns

None

Examples

```
>>> model.fit([['hello', 'world'], ['hello', 'cpt']])
```

`predict`(*sequences*, *multithreading=True*)

Predict the next element of each sequence of the parameter `sequences`.

Parameters

sequences
[list] A list of sequences of any hashable type.

multithreading
[bool, default True] True if the multithreading should be used for predictions.

Returns

predictions
[list of length `len(sequences)`] The predicted elements.

Raises

ValueError
noise_ratio should be between 0 and 1. MBR should be non-negative.

Examples

```
>>> model = Cpt()
```

```
>>> model.fit([['hello', 'world'],
     ['hello', 'this', 'is', 'me'],
     ['hello', 'me']
    ])
```

```
>>> model.predict([['hello'], ['hello', 'this']])
['me', 'is']
```

`predict_k`(*sequences*, *k*, *multithreading=True*)

Predict the next elements of each sequence of the parameter `sequences`, sorted by descending confidence.

Parameters

sequences
[list] A list of sequences of any hashable type.

k: int
Number of predictions to make per sequence, ordered by descending confidence.

multithreading
[bool, default True] True if the multithreading should be used for predictions.

Returns

predictions
[List[List[Any]] of dimension `len(sequences)` * k] The predicted elements.

Raises

ValueError
noise_ratio should be between 0 and 1. MBR should be non-negative.

Examples

```
>>> model = Cpt()
```

```
>>> model.fit([['hello', 'world'],
     ['hello', 'this', 'is', 'me'],
     ['hello', 'me']
    ])
```

```
>>> model.predict_k([['hello']], 2)
[['me', 'this']]
```

`compute_noisy_items`(*noise_ratio*)

Compute noisy elements.

An element is considered as noise if the frequency of sequences in which it appears at least once is below `noise_ratio`.

Parameters

noise_ratio
[float] The threshold of frequency to consider elements as noise.

Returns

noisy_items
[list] The noisy items.

Raises

ValueError
noise_ratio should be between 0 and 1

`find_similar_sequences`(*sequence*)

Find similar sequences.

A sequence similar `X` of a sequence `S` is a sequence in which every element of `S` is in `X`

Parameters

sequence
[list]

Returns

similar_sequences
[list] The list of similar_sequences.

`retrieve_sequence`(*index*)

Retrieve sequence from the training data.

Parameters

index
[int] Index of the sequence to retrieve.

Returns

sequence
[list]

Examples

```
>>> model = Cpt()
>>> model.fit([['sample', 'data'], ['should', 'not', 'be', 'retrieved']])
>>> model.retrieve_sequence(0)
['sample', 'data']
```

1.4 Tuning

CPT has 3 meta parameters that need to be tuned

1.4.1 MBR

MBR indicates the number of similar sequences that need to be found before predicting a value.

The higher this parameter, the longer the prediction. Having more similar sequences can result in a higher accuracy.

1.4.2 split_length

split_length is the number of elements per sequence to be stored in the model. (Choosing 0 results in taking all elements)

split_length needs to be finely tuned. As the model cannot predict an element present in the sequence, giving a too long sequence might result in lower accuracy.

1.4.3 noise_ratio

The noise_ratio determines which elements are defined as noise and should not be taken into account.

INDEX

C

compute_noisy_items() (*cpt.cpt.Cpt method*), 4
Cpt (*class in cpt.cpt*), 2

F

find_similar_sequences() (*cpt.cpt.Cpt method*), 5
fit() (*cpt.cpt.Cpt method*), 3

P

predict() (*cpt.cpt.Cpt method*), 3
predict_k() (*cpt.cpt.Cpt method*), 4

R

retrieve_sequence() (*cpt.cpt.Cpt method*), 5