Final Project: Context-Tree Weighting

Author: Gabriel Hofer

Instructor: Dr. Rebenitsch
Course: CSC-372 Analysis of Algorithms
Date: December 1, 2020

Algorithm, Application, Language Choice

- *Algorithm: Context-Tree Weighting (CTW)*
- *Application: File Compression*
- *Language: Python 3*

Where It Is Used

- *CTW solves the class of polynomial-time problems.*
- *It's primarily used in text compression.*
- *CTW can be used in applications where there is a data source and we want to predict the probability of the next symbol.*

Other applications

- *CTW can be used in route prediction applications. "Route prediction is a problem which deals with, given a sequence of road network graph edges already traveled by the user, predict the most probable edge of the network to be traveled (Tiwari 2008)."*
- *CTW can be used for content recognition purposes (*Z. Dawy 2004)

Alternative algorithms

- *Lempel-Ziv 78 (LZ78)*
- *Probabilistic Suffix Trees (PST)*
- *Prediction by Partial Match (PPM)*

Reason for choice

The choice of CTW for this project was motivated by the Hutter Prize. The CTW was recommended on the Hutter Prize FAQ page as a place to start with the challenge.

How Your Project Works

Constructor

I wrote a class called ctw. We create a new instance of the ctw class and pass a sequence s (with a binary alphabet {0,1}) and a depth d to the constructor. The constructor ‘constructs’ a d-bounded context-tree from the initial sequence.

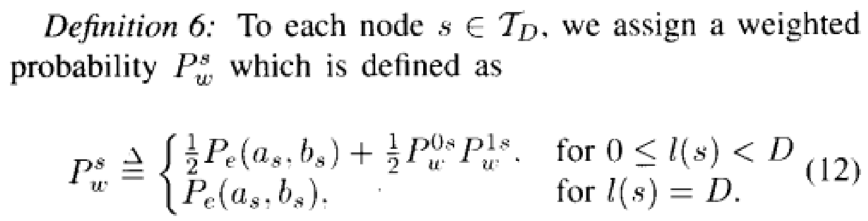
```python
# build the tree
def __init__(self,s,d):
    self.seq, self.depth, self.n, self.tree, self.Pw = s, d, len(s), {}, {}
    self.tree['root']=[0,0]
    for i in range(d,len(s)):
        for j in range(0,d+1):
            context= 'root' if j==0 else s[i-j:i]
            if context not in self.tree: self.tree[context]=[0,0]
            self.tree[context][int(s[i:i+1])]+=int(1)
```

addSymbol

addSymbol appends a new symbol b to the sequence. b must be in the alphabet {0,1}. The context-tree is updated by traversing the context path (from root to leaf) and incrementing zero-counts in nodes if symbol b is a zero or incrementing one-counts in nodes if symbol b is a one.

```python
# get next symbol and update context path
def addSymbol(self,b):
    self.tree['root'][int(b)]+=int(1)
    for i in range(1,self.depth+1):
        context=self.seq[self.n-i:]
        if context not in self.tree:
            self.tree[context]=[0,0]
        self.tree[context][int(b)]+=int(1)
    self.seq+=b
    self.n+=1
```

probs



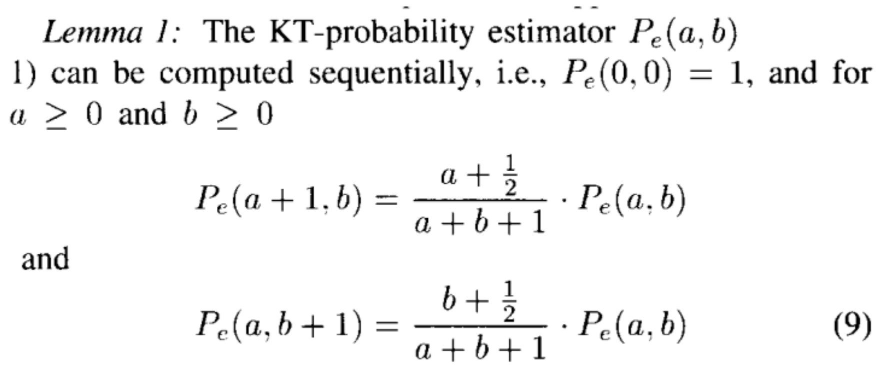
Definition 6: To each node $s \in \mathcal{T}_D$, we assign a weighted probability P_w^s which is defined as

$$P_w^s \triangleq \begin{cases} \frac{1}{2}P_e(a_s, b_s) + \frac{1}{2}P_w^{0s}P_w^{1s}, & \text{for } 0 \leq l(s) < D \\ P_e(a_s, b_s), & \text{for } l(s) = D. \end{cases} \quad (12)$$

```
# calculate probabilities for nodes in the tree
def probs(self,s):
    [a,b] = self.tree[s] if s in self.tree else [0,0]
    if s=='': [a,b]=self.tree['root']
    if(len(s)==self.depth):
        self.Pw[s]=self.Pe(a,b)
    else:
        self.probs('0'+s)
        self.probs('1'+s)
        self.Pw[s] = (self.Pe(a,b) + self.Pw['0'+s] * self.Pw['1'+s])/2
```

Pe (Krichevsky–Trofimov estimator)

Here is the definition of the KT-probability estimator that was used in the ‘probs’ function above. Note that the mathematical definition has been modified (a is substituted with (a-1)) for the purpose of writing a top-down recursive function (so that we can use P(0,0) as the base case).



Lemma 1: The KT-probability estimator $P_e(a, b)$

1) can be computed sequentially, i.e., $P_e(0, 0) = 1$, and for $a \geq 0$ and $b \geq 0$

$$P_e(a+1, b) = \frac{a + \frac{1}{2}}{a + b + 1} \cdot P_e(a, b)$$

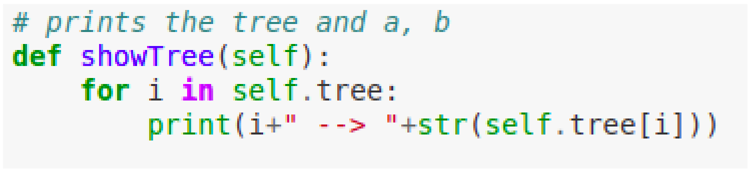
and

$$P_e(a, b+1) = \frac{b + \frac{1}{2}}{a + b + 1} \cdot P_e(a, b) \quad (9)$$

```
# Krichevsky–Trofimov estimator
def Pe(self,a,b):
    if a: return ((a-0.5)/(a+b))*self.Pe(a-1,b)
    elif b: return ((b-0.5)/(a+b))*self.Pe(a,b-1)
    else: return 1
```

showTree

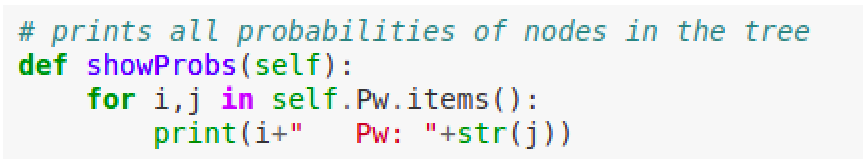
This function iterates through all nodes in the tree and prints each node's context string along with its corresponding pair: [a,b] which are the number of zeros and ones at that context



```
# prints the tree and a, b
def showTree(self):
    for i in self.tree:
        print(i+" --> "+str(self.tree[i]))
```

showProbs

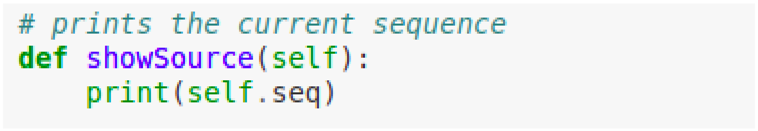
This function iterates through every node in the tree and prints the context string and the corresponding probability or weight for that context.



```
# prints all probabilities of nodes in the tree
def showProbs(self):
    for i,j in self.Pw.items():
        print(i+"   Pw: "+str(j))
```

showSource

Simply prints the sequence of symbols that have been seen so far.



```
# prints the current sequence
def showSource(self):
    print(self.seq)
```

Run time

The runtime of my context-tree constructor (__init__ function) is O(ND) where N is the length of the binary string sequence s and D is the depth of the context-tree.
The runtime of addSymbol is O(D) because the function traverses the context-path which has a length equal to the depth of the tree.
The runtime of probs is O(2^(D+1)) because there are 2^(D+1)-1 nodes in the tree.

Program usage or README

Run ctw.py on the command line with the following command:

```
$ python3 ctw.py
```

The ctw.py file contains a demo function which shows how to use the ctw class.

References

Tiwari, V.S., Arya, A. Distributed Context Tree Weighting (CTW) for route prediction. Open geospatial data, softw. stand. 3, 10 (2018). https://doi.org/10.1186/s40965-018-0052-9

Z. Dawy, J. Hagenauer and A. Hoffmann, "Implementing the context tree weighting method for content recognition," Data Compression Conference, 2004. Proceedings. DCC 2004, Snowbird, UT, USA, 2004, pp. 536-, doi: 10.1109/DCC.2004.1281512.

Willems, Frans, Yuri Shtarkov, and Tjalling Tjalkens. "Reflections on “the context tree weighting method: Basic properties”." Newsletter of the IEEE Information Theory Society 47.1 (1997).

Volf, Paulus Adrianus Jozef. Weighting techniques in data compression: Theory and algorithms. Technische Universiteit Eindhoven, 2002.

Willems, Frans MJ, Yuri M. Shtarkov, and Tjalling J. Tjalkens. "The context-tree weighting method: basic properties." IEEE transactions on information theory 41.3 (1995): 653-664.

Begleiter; El-Yaniv; Yona (2004), On Prediction Using Variable Order Markov Models, 22, Journal of Artificial Intelligence Research: Journal of Artificial Intelligence Research, pp. 385–421

Sadakane, Kunihiko. "Unifying Text Search and Compression-Suffix Sorting, Block Sorting and Suffix Arrays." (2000).

https://web.archive.org/web/20150302190939/http://www.ele.tue.nl/ctw/

http://prize.hutter1.net/