Trying Tries Algorithm

Denizhan Pak
Indiana University - Computational Linguistics Department
denpak@iu.edu

Abstract

There are many reasons to apply morphological analysis at the sentence level. For the application of computational tools to corpus data it is important that all tokens are specified to allow the tools to fine grain as much of the data as possible. To determine the list of morphemes in a language however is a time consuming task and unsupervised algorithm could relieve quite a few researchers and grad students. In this paper we potentially useful unsupervised machine learning to accomplish just this task.

1 Introduction

Morpheme parsing is an important task from a computational linguistics standpoint as it provides a way to denote meaningful units within a corpus in turn this allows us to apply the many tools which use these units using the lowest components which still provide information. We propose that the semantic importance of a morpheme can be determined by its relative frequency. More explicitly if we are able to identify substrings which occur as a cohesive unit with a high relative frequency then those units correspond to morphemes or some other sort of semantic unit. The algorithm below provides an unsupervised method through which such semantic units can be distinguished.The algorithm uses a data structure similar to a "trie" however edges between nodes are assigned a transition probability, we shall call this a "p-trie." The algorithm requires two user determined hyper parameters: $0 < \lambda < 1 < \rho$. Where ρ is a reinforcement rate and λ is a learning rate. At the end of the algorithm the function will have returned a $p - trie$ with the estimated probability values. Once we have a $p-trie$ it is possible to extract po-

Algorithm 1 Build Trie

Require: $0 < \lambda < 1 < \rho$
  $pointer \leftarrow root$
  for $char$ in $corpus$ **do**
    if $pointer == root$ **then**
      $p_1 = 1, p_2 = 1$
    else
      $p_1 = \lambda, p_2 = \rho$
    end if
    if $char$ in $pointer \rightarrow children$ **then**
      $edge_{pointer \rightarrow char} \leftarrow edge_{pointer \rightarrow char} * \rho$
    else
      $edge_{pointer \rightarrow char} \leftarrow \lambda$
      $pointer \rightarrow children$ **append** $char$
    end if
    $r \leftarrow$ **random number(0,1)**
    if $r > edge_{pointer \rightarrow char}$ **then**
      $pointer \leftarrow root$
    else
      $pointer \leftarrow char_{pointer}$
    end if
  end for
  return $root$

tential morphemes as sequences of nodes that start at the root. We can even assign a certainty per morpheme which is the product of the weights along its edges. If the corpus is not large enough an easy approach to improving access to data would simply be to randomize the words in the corpus and run the algorithm through it again starting with the existing $p - trie$.

2 Proposed Goals

2.1 Minimum Viable Product

The minimum viable product of this project is a complete implementation of the algorithm defined above along with a comprehensive performance analysis. The analysis will include accuracy testing across multiple corpora in multiple languages along with the effects of different parameterizations.

2.2 Expected Product

The expected product will be the minimum viable product as well as specific optimizations such as simulated annealing for the parameter values, a more complex function for reinforcement, and an automated randomization of the corpus designed to improve performance.

2.3 High-Achievement Product

The high-achievement product will be the expected product as well as a generalization of the algorithm which could be applied to finding meaningful sub-sequences in any sequential data with a particular eye toward word embeddings. As well as a potential generative application of the generated $p - trie$.

3 Requirements

- A working python implementation of the algorithm.
- A well documented description of the algorithm and its performance.
- A report characterizing the algorithm including its strengths and weaknesses.

4 Timelines

- 10.15-10.22 A first implementation of the algorithm and basic testing using a single corpus.
- 10.22-10.29 Testing algorithm using varying parameter sets across multiple languages and corpora.
- 10.29-11.06 Implementation of simulated annealing and varied reinforcement and corpora randomization.
- 11.06-11.16 Evaluation of new implementation across corpora and languages.
- 11.16-11.26 Generalization of the algorithm to any subsequence and evaluation of performance on different data such as word embeddings.
- 11.26-12.03 Writing final report and preparing presentation.

This does not include literature review or the writing of documentation both of which I intend to do as they coincide with different parts of the timeline above.

5 Data Policy

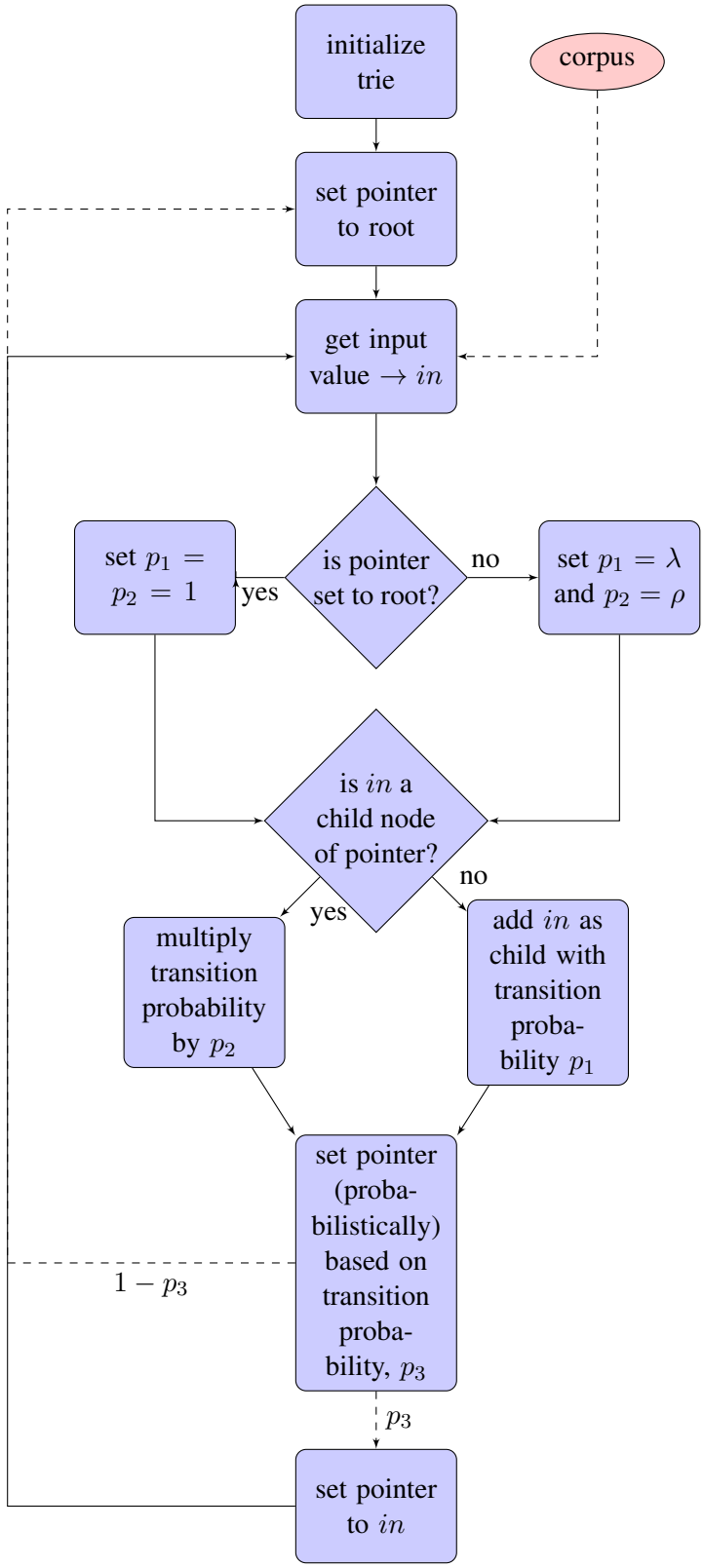
All data and code will be made available on a public git page. The code will be written in Python and the data for testing and evaluation will be taken from the open UD morpheme dataset.

6 References

- Byte Encoding Paper
- Resource on Bayesian Framework without tears
- UD documentation
- Morphessor documentation
- Still looking for more.

7 Appendix

Below is the flow diagram to explain the functioning of the algorithm.


initialize trie
corpus
set pointer to root
get input value $\rightarrow in$
set $p_1 = p_2 = 1$
yes
is pointer set to root?
no
set $p_1 = \lambda$ and $p_2 = \rho$
is in a child node of pointer?
yes
no
multiply transition probability by p_2
add in as child with transition proba-bility p_1
set pointer (proba-bilistically) based on transition proba-bility, p_3
$1 - p_3$
p_3
set pointer to in