Unsupervised Morphological Segmentation for Low-Resource Polysynthetic Languages

Report submitted in fulfillment of the requirements for the Exploratory Project of

Second Year B.Tech.

by

Yash Malik

18075065

Under the guidance of

Dr. A.K. Singh

Department of Computer Science and Engineering

INDIAN INSTITUTE OF TECHNOLOGY (BHU) VARANASI
Varanasi 221005, India
June 2020

Dedicated to

My parents and my teachers.

Declaration

I certify that

1. The work contained in this report is original and has been done by myself and the general supervision of my supervisor.

2. The work has not been submitted for any project.

3. Whenever I have used materials (data, theoretical analysis, results) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references.

4. Whenever I have quoted written materials from other sources, I have put them under quotation marks and given due credit to the sources by citing them and giving required details in the references.

Place: IIT (BHU) Varanasi
Date: 16th June, 2020

Yash Malik
B.Tech. (Part II) Student
Department of Computer Science and Engineering,
Indian Institute of Technology (BHU) Varanasi,
Varanasi, INDIA 221005.

<u>Certificate</u>

*This is to certify that the work contained in this report entitled “**Unsupervised Morphological Segmentation for Low-Resource Polysynthetic Languages**” being submitted by **Yash Malik (Roll No. 18075065)**, carried out in the Department of Computer Science and Engineering, Indian Institute of Technology (BHU) Varanasi, is a bona fide work of our supervision.*

Place: IIT (BHU) Varanasi
Date: 20th June, 2020

Dr. A.K. Singh
Department of Computer Science and Engineering,
Indian Institute of Technology (BHU) Varanasi,
Varanasi, INDIA 221005.

Acknowledgments

I would like to express my sincere gratitude to my supervising Professor Dr. A.K. Singh and guiding mentor Mr. Rajesh Kumar Mundotiya for their constant guidance and support during the whole project work.

Place: IIT (BHU) Varanasi

Date: 16th June, 2020 **Yash Malik**

Abstract

Polysynthetic languages pose a unique challenge for morphological analysis due to their very high morpheme-to-word ratio and very high inflection in word-formation. These languages are morphologically rich and are highly synthetic, i.e., single words can be composed of many morphemes. In extreme cases, the entire sentence consists of only one single token. Morphological segmentation in such cases becomes a challenging task when we consider the fact that the training data can be extremely scarce for such languages.

This report presents some unsupervised approaches for morphological segmentation of low-resource polysynthetic languages based on Adaptor Grammars(AG) framework [1], as proposed by Eskander et al. (2019) in their research paper [2].

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

In Natural Language Processing (NLP), morphological segmentation is the task of segmenting words into meaning-bearing morphemes [1]. Morphological segmentation is a critical task in NLP. It has many direct or indirect applications in various fields, such as part-of-speech tagging, text classification, machine translation, speech recognition, etc.

This report presents an unsupervised approach for the morphological segmentation of polysynthetic languages based on the Adaptor Grammars framework [1].

We show that the Adaptor Grammars based approaches outperform other unsupervised methods — Morfessor [4] and MorphoChain [5] — which are considered as the state-of-the-art for this task.

A generalized and easy-to-use GUI app is also developed which partially automates and simplifies the workflow. The app can be used in similar research work.

[1] A morpheme is the smallest unit of language that has meaning or function, which includes words, prefixes, affixes, and other word structures that impart meaning.

1.2 Motivation of the Research Work

Computational morphological segmentation is an active research topic as it forms an essential basis for many natural language processing tasks.

Due to the high cost associated with manually labeling data for morphology and the increasing interest in low-resource and endangered languages, where they lack adequate morphologically annotated resources, need arises to find an unsupervised approach for the task of morphological segmentation, especially for low-resource languages. The main goal of this report is to examine the success of 'Adaptor Grammars' for the task of unsupervised morphological segmentation when applied to polysynthetic languages, where the morphology is synthetically complex (not simply agglutinative), and where resources are minimal.

1.3 Organisation of the Report

The first chapter contains some vital background information of the terms and tools used throughout the report, quoted from relevant sources.

The second chapter describes the theoretical aspects of the project work, including the results. We begin by defining the problem and highlighting the challenges associated with it. We then describe the proposed solution setup in detail. The last section of this chapter deals with the evaluation and results of our model versus the other state-of-the-art baselines.

The third chapter deals with the practical aspects of the project work. We outline the processes going on behind the scene. We also present our GUI app in this chapter and highlight its features. In the end, some screenshots of a sample run are provided.

We wrap up by discussing and concluding our results in the last chapter. We also propose the possibility of an extension of this model for Indian languages in the future.

Chapter 2

Background Knowledge

Definitions of some of the basic terminologies and tools that we use throughout the report are provided below. All the definitions and explanations are **quoted from relevant sources**.

2.1 Basic terms and their definitions

- **Morpheme** : "A morpheme is the smallest meaningful unit in a language."[6]
- **Affix** : "In linguistics, an affix is a morpheme that is attached to a word stem to form a new word or word form."[7]
- **Inflection** : "In linguistic morphology, inflection (or inflexion) is a process of word formation, in which a word is modified to express different grammatical categories such as tense, case, voice, aspect, person, number, gender, mood, animacy, and definiteness."[8]
- **Agglutination** : "Agglutination is a linguistic process pertaining to derivational morphology in which complex words are formed by stringing together morphemes without changing them in spelling or phonetics."[9] "An agglutinating language (e.g., Turkish or Finnish) is one in which word forms can be easily

and clearly segmented into individual morphs, each of which represents a single grammatical category."[3]

- **Isolating Language** : "In isolating languages, every word must be a single morpheme (no affixation). These are isolating and fully analytic languages. This makes segmentation more clear in this language type."[3]

- **Synthetic Language** : "A synthetic language uses inflection or agglutination to express syntactic relationships within a sentence. Inflection is the addition of morphemes to a root word that assigns grammatical property to that word, while agglutination is the combination of two or more morphemes into one word."[10]

- **Polysyntetic Language** : "Polysynthetic languages are morphologically rich languages which are highly synthetic, i.e., single words can be composed of many individual morphemes."[11]

- **Context Free Grammar (CFG)** : "In formal language theory, a context-free grammar (CFG) is a formal grammar in which every production rule is of the form

$$A \to \alpha$$

where A is a single nonterminal symbol, and α is a string of terminals and/or nonterminals (α can be empty). A formal grammar is considered "context free" when its production rules can be applied regardless of the context of a nonterminal."[12]

2.2 Adaptor Grammars

“Adaptor Grammars are a framework for specifying nonparametric Bayesian models[1] that can be used to learn latent tree structures from a corpus of strings. There are two components to an AG model: the *base distribution*, which is just a PCFG, and the *adaptor*, which “adapts” the probabilities assigned to individual subtrees under the PCFG model, such that the probability of a subtree under the complete model may be considerably higher than the product of the probabilities of the PCFG rules required to construct it. Although in principle the adaptor can be any function that maps one distribution onto another, Johnson et al. (2007) use a Pitman-Yor Process (PYP) (Pitman and Yor, 1997) as the adaptor because it acts as a caching model. Under a PYP AG model, the posterior probability of a particular subtree will be roughly proportional to the number of times that subtree occurs in the current analysis of the data (with the probability of unseen subtrees being computed under the base PCFG distribution).”[13]

[1]Bayesian inference is a statistical model that uses Bayes’ theorem to update the probability for a hypothesis.

Chapter 3

Project Work

Theoretical concepts of the Project Work are discussed in this chapter. We also describe and state the results from our experiments in the last section.

3.1 Problem Definition

Design a statistical machine learning algorithm to split a given word into the surface forms of its smallest meaning-bearing units, morphemes. The task needs to be performed for polysynthetic languages.

Examples (English Language):

treatable = *treat* + *able*

carelessness = *care* + *less* + *ness*

irresponsible = *ir* + *respons* + *ible*

unacceptability = *un* + *accept* + *ability*

For this particular task, we need to segment words from the four Uto-Aztecan languages: *Mexicanero* **(MX)**, *Nahuatl* **(NH)**, *Wixarika* **(WX)** and *Yorem Nokki* **(YN)** [11].

Note: To carry out our experiments and for future research purposes, we also develop a fully-functional GUI app to simplify the processes described in this chapter. The working and features of the app are described in next chapter.

3.2 Languages and Datasets

This section describes the relationship between language typology and morphological analysis, followed by a description of the datasets.

3.2.1 Language Typology and Morphological Analysis

In linguistic typology, the broader gradient of morpheme complexity in a language is: *isolating/analytic* to *synthetic* to *polysynthetic*. The more specific gradation is: ***agglutinating*** to ***mildly fusional*** to ***fusional***. [2]

Thus a polysynthetic language can be characterized from agglutinating to fusional. A *polysynthetic and agglutinating* language has a high number of morphemes per word, with clear boundaries between morphemes. While, a *polysynthetic and fusional* language also has many morphemes per word, but due to many phonological and other processes, the segmentation boundaries are not clear.

"Typically, polysynthetic languages demonstrate **holophrasis**, i.e. the ability of *an entire sentence* to be expressed as what is considered by native speakers to be *just one word*" [2]

Consider as an example a word from Westen Greenlandic, an exclusively suffixing polysynthetic language (taken from ("Polysynthetic language.", wikipedia.org)) [14]:

Aliikkusersuillammassuaanerartassagaluarpaalli

aliikku - sersu - i - llammas - sua - a - nerar - ta - ssa - galuar - paal - li

entertainment - provide - one.good.at - say.that - sure.but - but

'However, they will say that he is a great entertainer, but ...'

The following example is from WX, one of the languages in the dataset for our experiments (taken from (Mager et al., 2018c)) [15]:

(Note: '+' is a vowel in the Wixarika language)

yuhutame nep+weiwa

yu - huta - me ne - p+ - we - iwa

an - two - ns 1sg:s - asi - 2pl:o - brother

I have two brothers.

Description of the four Yuto-Aztecan languages (which are considered for analysis):

- "**Mexicanero** is a Western Peripheral Nahuatl variant, spoken in the Mexican state of Durango by *approximately one thousand people.*"[11]

- "**Nahuatl** is a large subgroup of the Yuto-Aztecan language family, and, including all of its variants, the most spoken native language in Mexico. The data collected for this work belongs to the Oriental branch spoken by *70 thousand people* in Northern Puebla."[11]

- "**Wixarika** is a language spoken in the states of Jalisco, Nayarit, Durango and Zacatecas in Central West Mexico by approximately *fifty thousand people.*"[11]

- "**Yorem Nokki** is part of Taracachita subgroup of the Yuto-Aztecan language family. Its Southern dialect is spoken by close to *forty thousand people* in the Mexican states of Sinaloa and Sonora, while its Northern dialect has about twenty thousand speakers. In this work, we consider the Southern dialect."[11]

All these four languages are classified as *polysynthetic* languages[2] [3]. "However, as noted above, there is a gradation of polysynthesis, so the delineation of language types is not clear-cut. For these four languages, the more agglutinative is WX; Leza(2004) has observed 20 morphemes per word for this language." [2]

The morphological analysis of polysynthetic languages thus is challenging due to the root-morpheme complexity. This property of polysynthetic languages makes the task of surface segmentation complex but also relevant for further analysis.

3.2.2 Datasets for Experiment

Morphological data for polysynthetic languages are scarce. Kann et al. (2018) have made a step forward by releasing a small set of morphologically segmented datasets [11]. These datasets were constructed so they include both segmentable as well as non-segmentable words to ensure that methods can correctly decide against splitting up single morphemes. [2]

As we are using the data in an unsupervised way, we only use the unsegmented data to prepare our training set. There are two different possible approaches for the learning, transductive, and inductive.

- In **Transductive mode**, any word that needs to be segmented should be in the vocabulary of the training input. So the segmentation output is just a lookup to the mapping between training input and the segmentation output.
- In **Inductive mode**, the word need not be present in the training list. The learner does not process the query words; instead, the segmentation is performed by parsing the words using a PCFG parsing algorithm such as CYK [1].

We use the transductive mode of learning in our experiments ; therefore, we include the unsegmented words from the test sets in our training sets. Inductive learning did not improve any performance.

We do not use the segmented words for training (our approach is unsupervised). The datasets provided by Kann et al. (2018) consist of the train set, development set, and test set [11]. Each source file contains one word per line, and the corresponding target file contains their segmentations. We use these datasets to prepare

[1]CYK algorithm is a parsing algorithm for CFGs

	Mexicanero	Nahuatl	Wixarika	Yorem N.
train	427	540	665	511
dev	106	134	176	127
test	355	449	553	425

Table 3.1: Number of words in train, dev, test splits from Kann et al. (2018) datasets

	Mexicanero	Nahuatl	Wixarika	Yorem N.
Train set	888	1123	1394	1063
Test set	461	583	729	552

Table 3.2: Number of words in train and test splits in the prepared datasets (for transductive learning)

the datasets for our experiment. We use the unsegmented words (from all the three sets) to prepare our Training set, and we use the words as well as their respective segmentations to prepare our Test set (from dev and train sets). Table 3.1 and 3.2 describe this information and the repective word counts for each language.

Note: The datasets from Kann et al. have been slightly processed for clarity purposes and to satisfy the criteria for the input-output format of our application and of the evaluation software. Notably, we use a space character ‘ ’ instead of a ‘!’ to separate the segments, and redundant space between the letters of a word is removed. e.g.,

$$\text{p ! u ! k a ! w e} \implies \text{p u ka we}$$

Both the datasets (original as well as processed) are provided along with this report.

3.3 Describing the Solution Setup

Formal grammars, and particularly Context-Free Grammars (CFGs), are a keystone of linguistic description and provide a model for the structural description of linguistic objects. Probabilistic CFGs (PCFGs) extend this model by associating a probability to each context-free rewrite rule.

"Adaptor grammars (**AGs**) (Johnson et al., 2007) weaken the independence assumptions of PCFGs by inserting additional stochastic processes called adaptors into the procedure for generating structures. AGs provide a simple framework to implement Bayesian nonparametric learning of grammars and are usually trained in an unsupervised manner using sampling techniques."[3]

We try several AG learning setups for the experiments:

1. use the standard grammar in a completely unsupervised way
2. in case we have some linguistic knowledge, we seed our knowledge into the grammar (semi supervised way)
3. we try to approximate the effect of semisupervised learning, by automating the process of seeding

We describe these setups in detail in section 3.5.2.

3.3.1 Using Adaptor Grammars for Polysynthetic Languages

"An Adaptor Grammar is typically composed of a PCFG and an adaptor that adapts the probabilities of individual subtrees. For morphological segmentation, a PCFG is a morphological grammar that specifies word structure, where AGs learn latent tree structures given a list of words."[2]

The AGs take as input the vocabulary of the language we want to learn and the grammar rules the sampler has to follow.

- The vocabulary input is a list of unsegmented words.
- The grammar input consists of rules along with the adaptation information (see section 3.4 for details).

We briefly outline the experiment setup below (and describe them in detail in their respective sections):

1. **Grammars** We use the nine grammars from Eskander et al. (2016) [16].
2. **Learning Settings** We consider the three learning settings in Eskander et al., 2016 [16]: *Standard*, *Scholar-seeded Knowledge* and *Cascaded*.

3.4 Defining Grammars

The first step in learning morphological segmentation using Adaptor Grammars is to define the grammar. We use the nine grammars described by Eskander et al. (2016)[16].

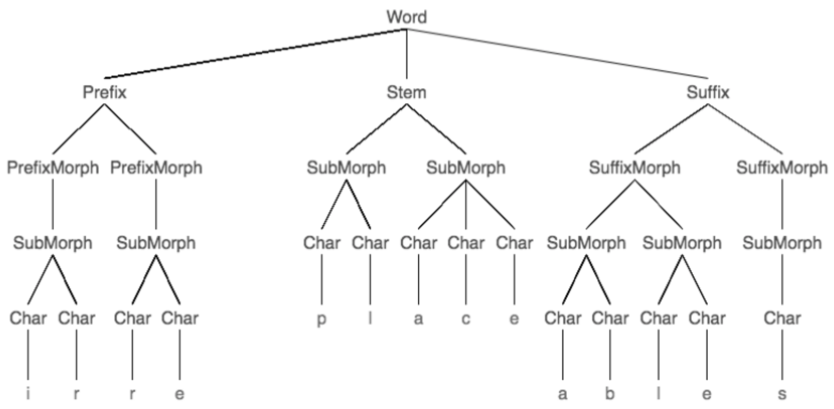


Figure 3.1: The representation of English word *irreplaceables* segmented using *PrStSu+SM*, from Eskander et al. (2020) [3]

Figures 3.1, 3.2 and 3.3 show the grammar trees of three of the nine grammars used: *PrStSu+SM*, *PrStSu2a+SM* and *Morph+SM*, representing the English word *irreplaceables.* (*Pr*, *St* and *Su* refer to *Prefix*, *Stem* and *Suffix,* respectively. *SM* represents *sub-morph*)

For more details about the specifications of these grammars and other grammars, see Eskander et al. (2016) [16].

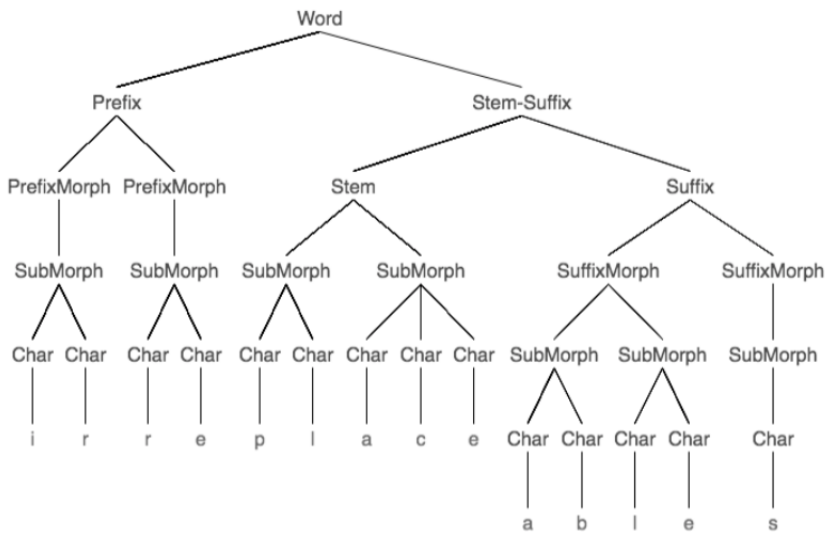


Figure 3.2: The representation of English word *irreplaceables* segmented using *PrStSu2a+SM*, from Eskander et al. (2020) [3]

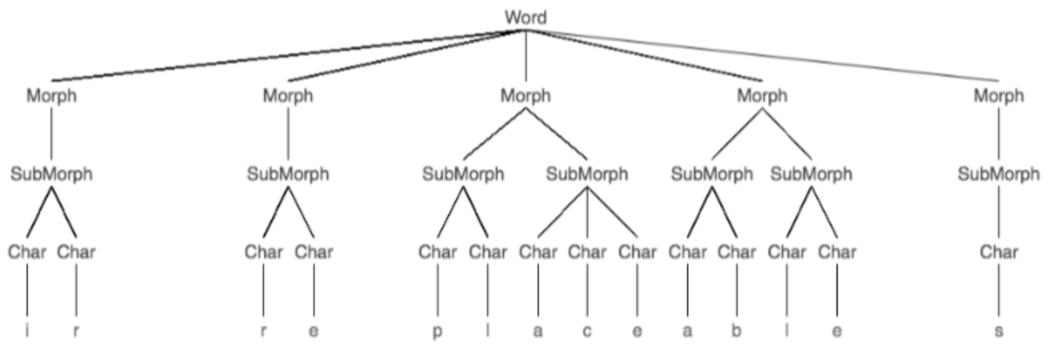


Figure 3.3: The representation of English word *irreplaceables* segmented using *Morph+SM*, from Eskander et al. (2020) [3]

Also, each production rule in our grammar has to be associated with three parameters; θ, a, and b, where θ is the probability of the rule in the generator, while a and b are the parameters of the Pitman-Yor process [17]. If the parameters are not specified, they are sampled by the trainer, or we can also set them to default values before running the learner.

"Setting a to one means the underlying non-terminal is not adapted and is sampled by the general Pitman-Yor process, while setting a to *zero* means the adaptor of the non-terminal is a Dirichlet process [18] with the concentration parameter b." [3]

When a non-terminal is adapted, each sub-tree that can be generated using the initial rule of that non-terminal is considered as a potential rule in the grammar. Otherwise, the non-terminal expands as in a regular PCFG."
For more details, see Johnson et al. (2007) [1].

3.5 Training The Model

3.5.1 Inputs

The main inputs to the learner are the grammar and the vocabulary of the language we want to learn the segmentation for.

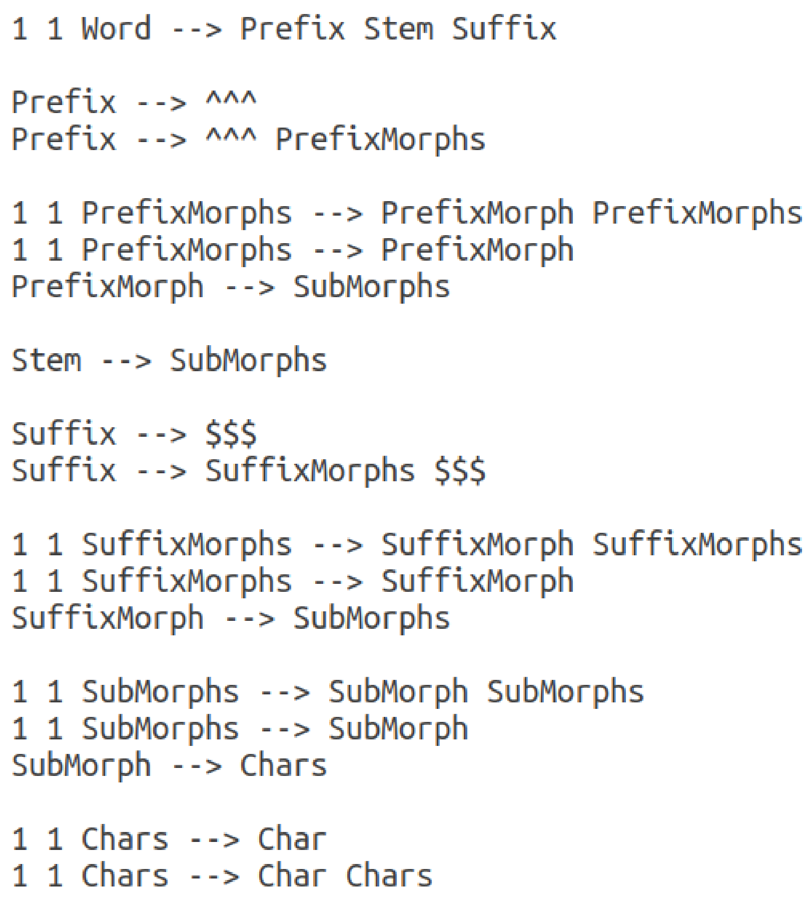
It is to be noted that the standard grammar is a comprehensive set of rules and is **entirely independent of the language**. Also, note that as **we are using the transductive learning method, the test set's unsegmented words should be included in the train set**.

We use $\wedge\wedge\wedge$ *and* $\$\$\$$ *to indicate the beginning and end of words respectively.*
The input grammar to the AGs should follow the format similar to one in Fig. 3.4.

3.5.2 Learning Settings

Escander et al., (2016) [16] define three learning settings: *Standard*, *Scholar-seeded* and *Cascaded*. We briefly define them here.
Note : *We* ***directly quote*** *the setup for cascaded setting* ***from the latest source****, Eskander et. al (2020)[3] , as this is the* ***single most important setup for our solution****.*

- **Standard**: The standard setting is language independent. The grammar does not have any language-specific rules, and *the learning is fully unsupervised.* Fig. 3.4 shows the input of *PrStSu+SM* grammar in the standard mode.



```
1 1 Word --> Prefix Stem Suffix

Prefix --> ^^^
Prefix --> ^^^ PrefixMorphs

1 1 PrefixMorphs --> PrefixMorph PrefixMorphs
1 1 PrefixMorphs --> PrefixMorph
PrefixMorph --> SubMorphs

Stem --> SubMorphs

Suffix --> $$$
Suffix --> SuffixMorphs $$$

1 1 SuffixMorphs --> SuffixMorph SuffixMorphs
1 1 SuffixMorphs --> SuffixMorph
SuffixMorph --> SubMorphs

1 1 SubMorphs --> SubMorph SubMorphs
1 1 SubMorphs --> SubMorph
SubMorph --> Chars

1 1 Chars --> Char
1 1 Chars --> Char Chars
```

Figure 3.4: The standard *PrStSu+SM* grammar (note the simple recursive rules of the grammar and the relation with fig 3.1)

- **Scholar-seeded**: When some linguistic knowledge is available (such as a list of morphemes) this knowledge can be seeded into the grammar trees as additional production rules, allowing for a *semi-supervised learning* setup [3]. Fig. 3.5 shows the input of *PrStSu+SM* grammar in the scholar-seeded mode, where a sample of English prefixes and suffixes are added.

- **Cascaded**: “The cascaded setting approximates the effect of scholar-seeded setting in a language-independent setup. This is done by first obtaining a list of morphemes from a segmentation model that is trained on a high-precision grammar, and then seeding those morphemes into another grammar. In this setup, both the standard grammar and the segmentation output of another grammar are provided. The system then extracts the top morphemes, typically affixes, from the segmentation output and seeds them into the standard grammar prior to running the learner."[3]

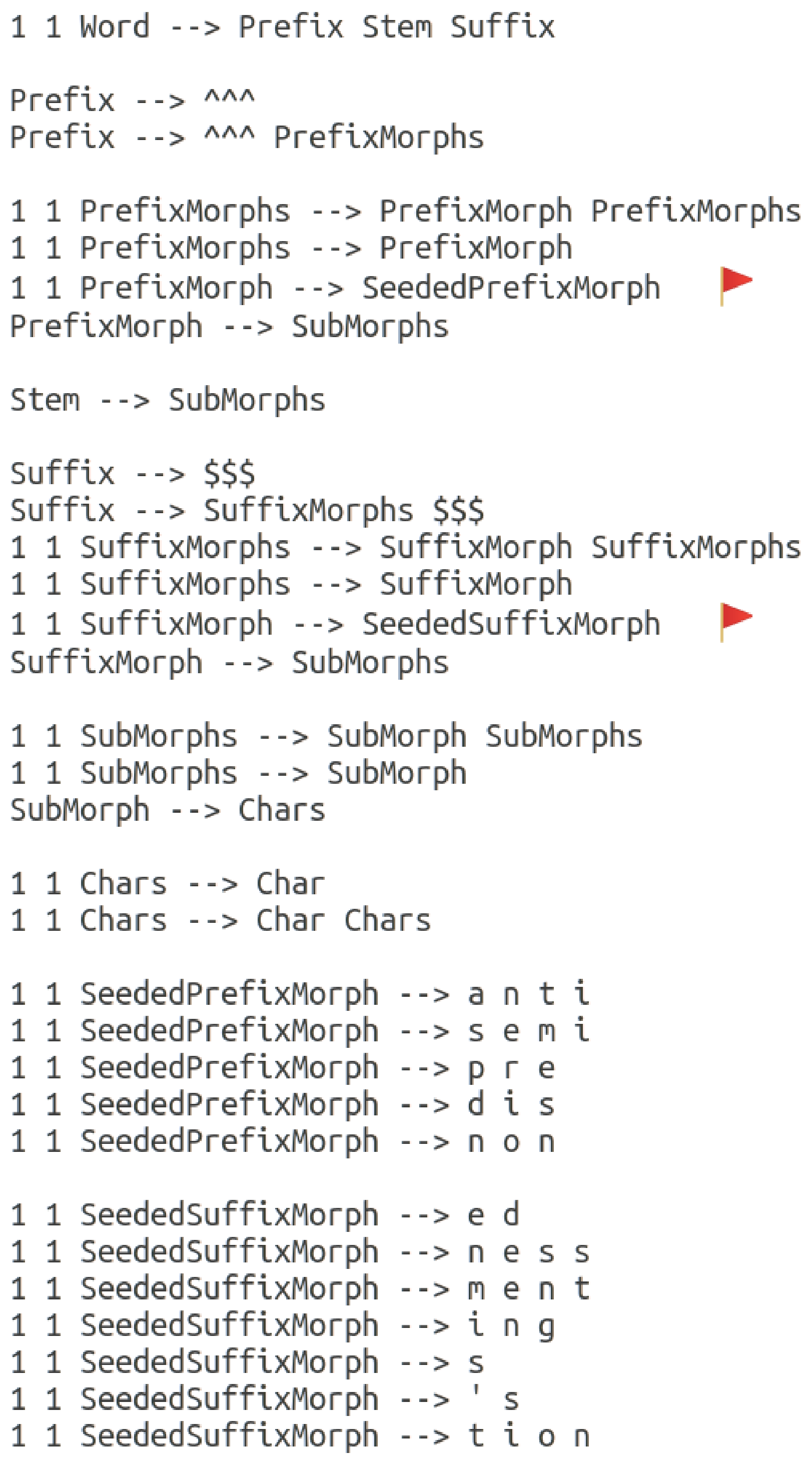
Note: "A production rule that is not preceded by parameters in Fig. 3.4 and Fig. 3.5 has a default zero value for the a parameter in the Pitman-Yor process, which means the rule is adapted. On the other hand, those rules preceded by "1 1" are not adapted, where the first number represents the value of the probability of the rule in the generator, θ, and the second number is the value of the a parameter in the Pitman-Yor process."[3]

Also, note that the grammars in Fig. 3.4, and Fig. 3.5 are not ready to be used as input for the Adaptor Grammars framework. The $Char$ (in grammar rules) needs to be assigned to every letter terminal in the alphabet of the language for which we want to learn the segmentation. However, this process can be automated by picking up the characters from the training set and defining the grammar then. **The implementation details are described in chapter 4.**

The nine grammars used in the experiments are provided for both *Standard* and *Scholar-seeded* settings along with the report. The *Cascaded* learning setting also works using these two grammars.

3.6 Evaluation and Results

In this section we evaluate our morphological segmentation setup, qualitatively and analytically. We process the datasets outlined in section 3.2 . We first describe the evaluation setup, then we describe the evaluation metrics and the baselines. We conclude by comparing our results to the **state-of-the-art** baselines.



```
1 1 Word --> Prefix Stem Suffix

Prefix --> ^^^
Prefix --> ^^^ PrefixMorphs

1 1 PrefixMorphs --> PrefixMorph PrefixMorphs
1 1 PrefixMorphs --> PrefixMorph
1 1 PrefixMorph --> SeededPrefixMorph    🚩
PrefixMorph --> SubMorphs

Stem --> SubMorphs

Suffix --> $$$
Suffix --> SuffixMorphs $$$
1 1 SuffixMorphs --> SuffixMorph SuffixMorphs
1 1 SuffixMorphs --> SuffixMorph
1 1 SuffixMorph --> SeededSuffixMorph    🚩
SuffixMorph --> SubMorphs

1 1 SubMorphs --> SubMorph SubMorphs
1 1 SubMorphs --> SubMorph
SubMorph --> Chars

1 1 Chars --> Char
1 1 Chars --> Char Chars

1 1 SeededPrefixMorph --> a n t i
1 1 SeededPrefixMorph --> s e m i
1 1 SeededPrefixMorph --> p r e
1 1 SeededPrefixMorph --> d i s
1 1 SeededPrefixMorph --> n o n

1 1 SeededSuffixMorph --> e d
1 1 SeededSuffixMorph --> n e s s
1 1 SeededSuffixMorph --> m e n t
1 1 SeededSuffixMorph --> i n g
1 1 SeededSuffixMorph --> s
1 1 SeededSuffixMorph --> ' s
1 1 SeededSuffixMorph --> t i o n
```

Figure 3.5: The scholar-seeded *PrStSu+SM* grammar for English (note the addition of flagged lines and knowledge seeds in the end) (the flags are not a part of the grammar)

3.6.1 Evaluation Setups

LIMS (Language-Independent Morphological Segmenter) is a system that provides the best on average results for morphological segmentation using Adaptor Grammars. It uses the cascade, starting with PrStSu2b+Co+SM grammar, and inserting the obtained affixes into PrStSu+SM and running this modified PrStSu+SM.[16]

- For our model we conduct the evaluation using the **LIMS setup**, as proposed by Eskander et al., (2016) [16]. We found that the LIMS setup gave the best performance on average for all these languages. LIMS is a cascaded setup with a high precision grammar (such as *PrStSu2b+Co+SM*) as the input grammar for the first run and a *PrStSu+SM* grammar as the second grammar input to the AG. We only report the results for this setup, as this provides the best results for our model.

- We conduct the evaluation in a transductive learning scenario, where the unsegmented test words are included in our training set.

- We run the sampler for 500 iterations for all languages (the performance is almost similar to that at 1000 iterations).

- No annealing is used as it does not improve the results. All the parameters to the Adaptor Grammars are automatically inferred (we do not specify any default value for any of the hyperparameters).

- We compute the results as the average of five runs.

3.6.2 Evaluation Metrics

We evaluate the performance of our setup using two metrics: Boundary Precision and Recall (**BPR**) and Evaluation Metric for Morphological Analysis - 2 (**EMMA-2**)[19].

"BPR is the classical evaluation method for morphological segmentation, where the boundaries in the proposed segmentation are compared to the boundaries in the reference. In contrast, EMMA-2 is based on matching the morphemes, and is a variation of EMMA (Spieglerand Monson, 2010). In EMMA, each proposed morpheme is matched to each morpheme in the gold segmentation through one-to-one mappings. However, EMMA-2 allows for shorter computation times as it replaces the one-to-one assignment problem in EMMA by two many-to-one assignment problems, where two or more proposed morphemes can be mapped to one reference morpheme. EMMA-2 also results in higher precision and recall as it tolerates failing to join two allomorphs or to distinguish between identical syncretic morphemes."[3]

3.6.3 Baselines

We evaluate our system versus two **state-of-the-art** baselines: **Morfessor** [4] and **MorphoChain** [5].

- Morfessor is a commonly used framework for unsupervised and semi-supervised morphological segmentation and is publicly available for free [2].
- MorphoChain is another publicly available system for unsupervised morphological segmentation [3].

Note: All the data for the results of Baselines has been taken from Eskander et al., (2020) [3].

3.6.4 System Performance

Table 3.3 reports the performance of our system and Table 3.4 reports our performance compared to Morfessor and MorphoChain based on the respective $F1$ - scores for the four polysynthetic languages when tested on Test sets. The results by our

[2] `https://morfessor.readthedocs.io/en/latest/`
[3] `https://github.com/karthikncode/MorphoChain`

system are under the **AG-LIMS** column.

Language	**BPR**			**EMMA-2**		
	Precision	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**
Mexicanero	67.8	77.3	71.8	85.6	81.8	82.4
Nahuatl	65.1	72.3	67.4	81.1	78.0	78.6
Wixarika	84.3	66.8	73.3	87.2	65.5	72.1
Yorem Nokki	84.0	75.9	78.5	93.1	80.8	84.8

Table 3.3: Result on Test sets using the AG based LIMS model (AG-LIMS).

Fig. 3.6 and 3.7 show the graphical representations of Table 3.3.

The high precision and low recall for WX suggests that our model tends to under-segment the words of this language. This can be attributed to the high morpheme-to-word ratio for WX and existence of many single letter morphemes in the language. Our model fails to segment these single letter morphemes.

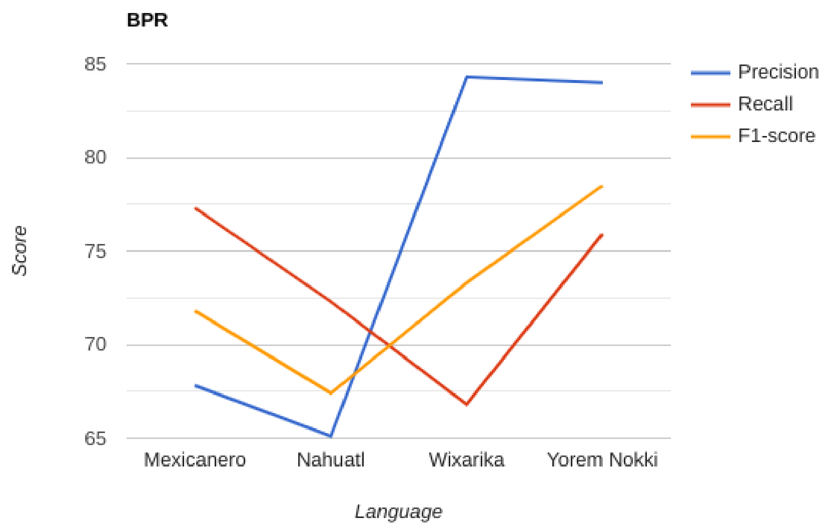


Figure 3.6: BPR scores of each language (from table 3.4)

As shown in Table 3.4, we obtain better scores on both metrics, compared to our baselines, on all four languages.

“It is worth noting that models that tend to under-segment achieve significantly

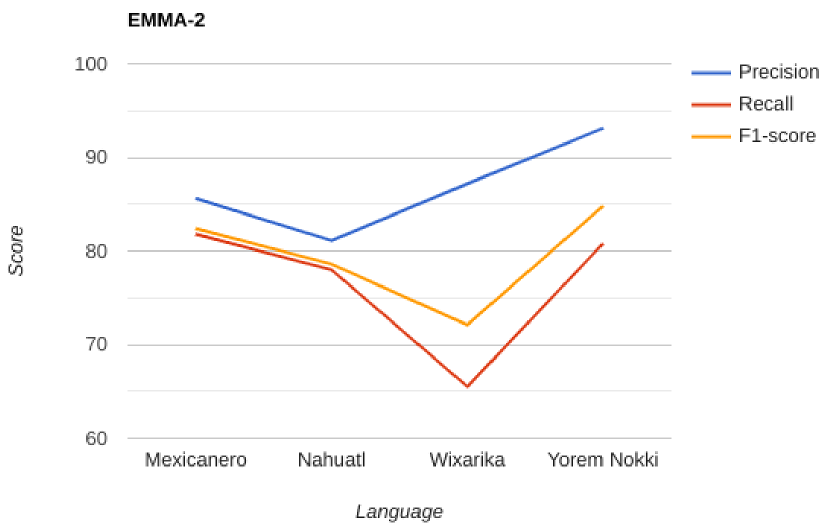


Figure 3.7: EMMA-2 scores of each language (from table 3.4)

better EMMA-2 scores as opposed to the BPR ones, which is due to the one-to-many mappings in EMMA-2. This is one of the main reasons why system rankings may differ depending on the evaluation metric." [3]

Language	**BPR**			**EMMA-2**		
	Morfessor	**MorphoChain**	**AG-LIMS**	**Morfessor**	**MorphoChain**	**AG-LIMS**
Mexicanero	70.5	64.3	71.8	79.6	77.4	82.4
Nahuatl	61.2	55.9	67.4	73.4	74.8	78.6
Wixarika	72.9	50.3	73.3	71.7	62.3	72.1
Yorem Nokki	71.7	58.0	78.5	79.0	77.6	84.8
Average	69.1	57.1	72.8	75.9	73.0	79.5

Table 3.4: Result on Test sets using the AG based LIMS model (AG-LIMS) for each language compared to two baselines; Morfessor and MorphoChain. The results are reported on both the BPR and EMMA-2 F1-scores.

Fig. 5.1 and 5.2 show the graphical representations of Table 3.4 and Fig. 5.3. compares the average results.

We analyse and conclude the results in chapter 5.

Chapter 4

Technical Details of the Project Work

This chapter outlines the implementation details of the project. We also present our Graphical User Interface (**GUI**) based app, which is used to perform all the experiments described in chapter 3. The app can be used in future to carry out similar research work. In the end, we provide screenshots from a sample run of the app.

Note: All the code for the work described in this chapter is provided along with the report.

4.1 Framework Used

We use the open source software **Pitman-Yor Adaptor Grammar Sampler** (AG) by Mark Johnson [1]. The complete information about how the sampler works is described in their NIPS 2006 paper [1].

The sampler requires two inputs: *the grammar* and *the training data*. For the purpose of Morphological Segmentation the grammar should specify the word structure

[1]available at `http://web.science.mq.edu.au/~mjohnson/Software.htm`

rules and the training data should be a list of raw words.

4.2 Programming Language Used

We use the **Python** language for all our major work. Python has support of many excellent libraries and modules that simplify the development work.

Python code is easy to read and has the advantage of being user-friendly.

4.3 Implementation details

We describe the implementation work of the various phases:

1. Preprocessing Phase
2. Training Phase
3. Segmentation Phase

4.3.1 Preprocessing

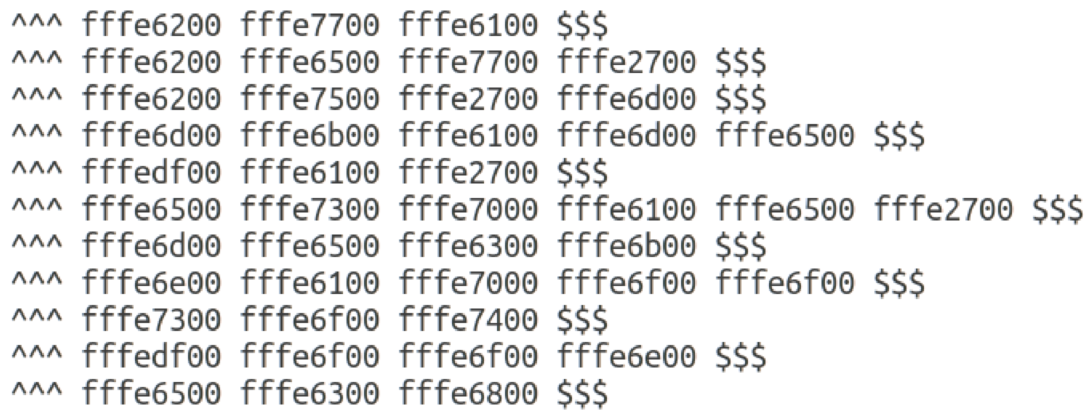
In this phase we prepare the data to be fed into the AG.

The AG framework requires an initial CFG (Context Free Grammar) and a list of words (one word per line). An example of CFG is given in fig. 3.4.

4.3.1.1 Preparing the Word List

1. The letters need to be space separated as is the requirement by the AG.
2. For the purpose of **language independence**, we convert all the characters in the dataset to their utf-8 encodings before using them.
3. We use two hypothetical characters '$\wedge\wedge\wedge$' and '\$\$\$' as *word-begin* and *word-end* respectively, as described in section 3.5.1.

Fig. 4.1 represents the sample ouput of the *processed word list file*. This processed word list is now ready to be used as the input for AG.



```
^^^ fffe6200 fffe7700 fffe6100 $$$
^^^ fffe6200 fffe6500 fffe7700 fffe2700 $$$
^^^ fffe6200 fffe7500 fffe2700 fffe6d00 $$$
^^^ fffe6d00 fffe6b00 fffe6100 fffe6d00 fffe6500 $$$
^^^ fffedf00 fffe6100 fffe2700 $$$
^^^ fffe6500 fffe7300 fffe7000 fffe6100 fffe6500 fffe2700 $$$
^^^ fffe6d00 fffe6500 fffe6300 fffe6b00 $$$
^^^ fffe6e00 fffe6100 fffe7000 fffe6f00 fffe6f00 $$$
^^^ fffe7300 fffe6f00 fffe7400 $$$
^^^ fffedf00 fffe6f00 fffe6f00 fffe6e00 $$$
^^^ fffe6500 fffe6300 fffe6800 $$$
```

Figure 4.1: Example of a processed word list. The codes here represent the encodings of characters.

4.3.1.2 Preparing the Grammar

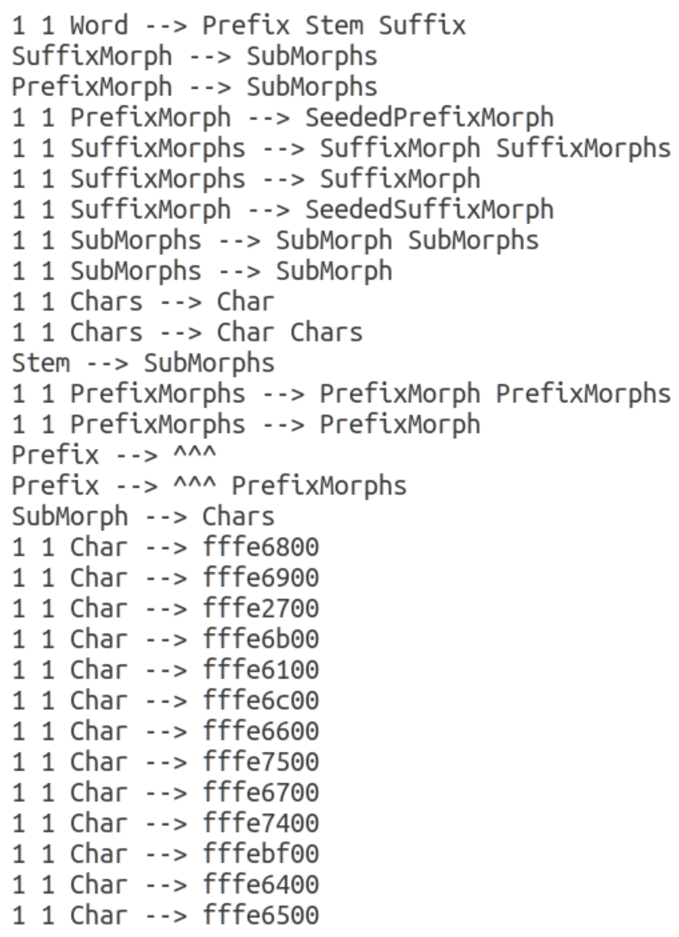
The grammar input to the AG varies significantly for the three settings:

- Standard
- Scholar Seeded
- Cascaded

Standard Settings The grammar for this setting is the same as the one in fig. 3.4, with the addition of character terminals.

This task is automated by picking up the unique characters from the raw dataset, converting them to their encodings, and adding them as terminals to the grammar rules in fig. 3.4.

This processed grammar (fig. 4.2) is now ready to be used as input grammar for AG.



```
1 1 Word --> Prefix Stem Suffix
SuffixMorph --> SubMorphs
PrefixMorph --> SubMorphs
1 1 PrefixMorph --> SeededPrefixMorph
1 1 SuffixMorphs --> SuffixMorph SuffixMorphs
1 1 SuffixMorphs --> SuffixMorph
1 1 SuffixMorph --> SeededSuffixMorph
1 1 SubMorphs --> SubMorph SubMorphs
1 1 SubMorphs --> SubMorph
1 1 Chars --> Char
1 1 Chars --> Char Chars
Stem --> SubMorphs
1 1 PrefixMorphs --> PrefixMorph PrefixMorphs
1 1 PrefixMorphs --> PrefixMorph
Prefix --> ^^^
Prefix --> ^^^ PrefixMorphs
SubMorph --> Chars
1 1 Char --> fffe6800
1 1 Char --> fffe6900
1 1 Char --> fffe2700
1 1 Char --> fffe6b00
1 1 Char --> fffe6100
1 1 Char --> fffe6c00
1 1 Char --> fffe6600
1 1 Char --> fffe7500
1 1 Char --> fffe6700
1 1 Char --> fffe7400
1 1 Char --> fffebf00
1 1 Char --> fffe6400
1 1 Char --> fffe6500
```

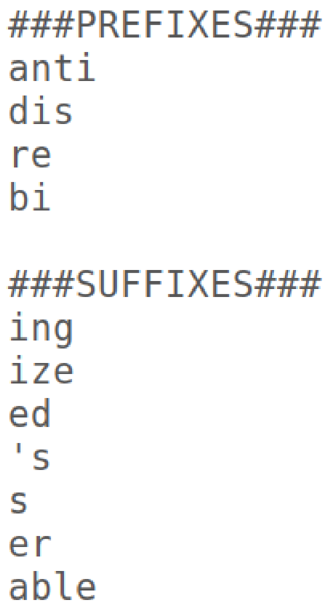
Figure 4.2: Example of a processed standard grammar. The codes here represent the encodings of characters.

Scholar-seeded Settings The preparation of grammar for this setting requires the addition of linguistic knowledge to the grammar (refer to fig. 3.5).

To automate this process we take an input of Affixes (Prefixes and Suffixes) as shown in fig 4.3.

After reading the *seeds* from the seed file, they are processed to their encodings, and are then automatically added to the grammar.

The processed grammar (Fig. 4.4) is now ready to be used as input grammar for AG.



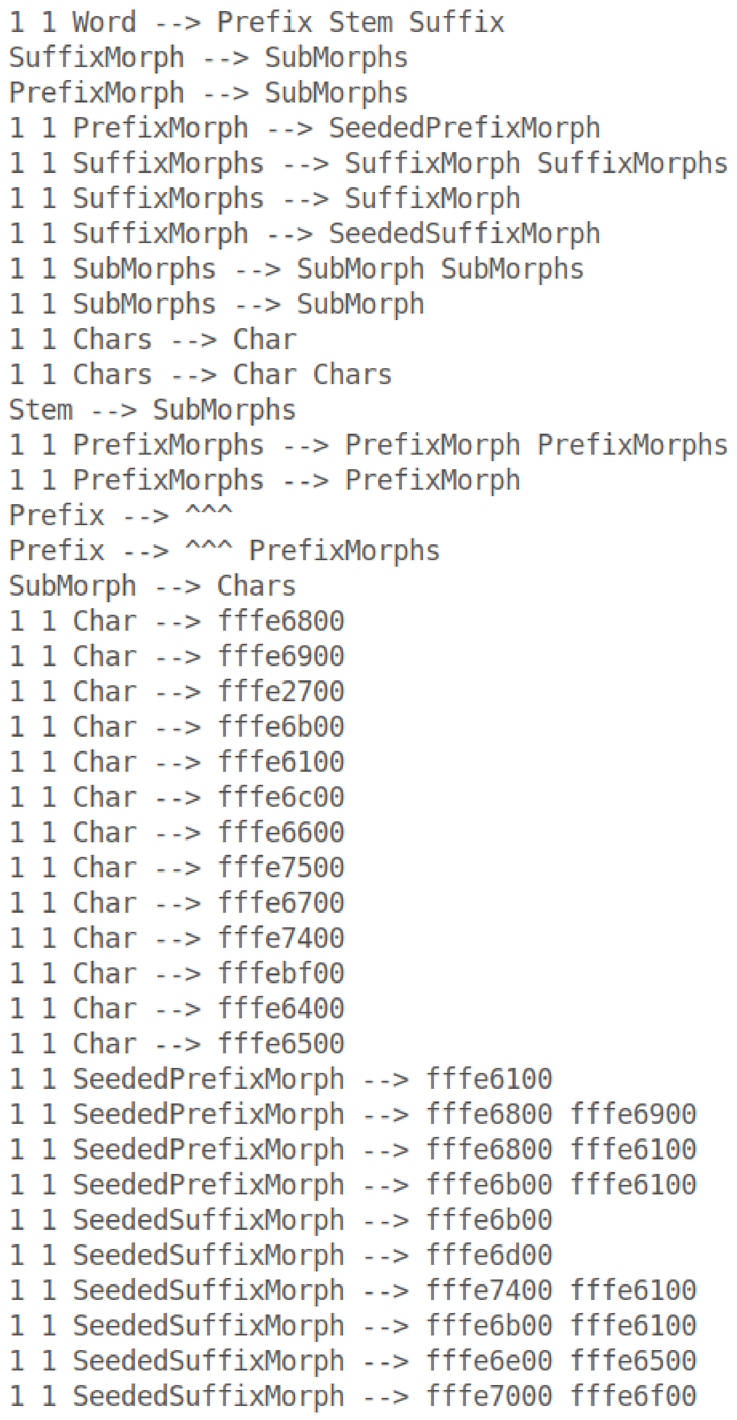
```
###PREFIXES###
anti
dis
re
bi

###SUFFIXES###
ing
ize
ed
's
s
er
able
```

Figure 4.3: An example of an input file for the affix-seeds for Scholar-seeded settings. The affixes are read from this file and are used to prepare the final grammar (fig. 4.4).

Cascaded Settings Cascaded settings make two runs of the trainer. So we prepare the grammar inputs one by one.

1. The grammar input for the first run of AG is prepared in the same manner as in Standard Settings.

2. After picking up the top affixes (prefixes and suffixes) from Run-1, we prepare the grammar inputs for the second run in a manner similar to the Scholar-seeded grammar.



```
1 1 Word --> Prefix Stem Suffix
SuffixMorph --> SubMorphs
PrefixMorph --> SubMorphs
1 1 PrefixMorph --> SeededPrefixMorph
1 1 SuffixMorphs --> SuffixMorph SuffixMorphs
1 1 SuffixMorphs --> SuffixMorph
1 1 SuffixMorph --> SeededSuffixMorph
1 1 SubMorphs --> SubMorph SubMorphs
1 1 SubMorphs --> SubMorph
1 1 Chars --> Char
1 1 Chars --> Char Chars
Stem --> SubMorphs
1 1 PrefixMorphs --> PrefixMorph PrefixMorphs
1 1 PrefixMorphs --> PrefixMorph
Prefix --> ^^^
Prefix --> ^^^ PrefixMorphs
SubMorph --> Chars
1 1 Char --> fffe6800
1 1 Char --> fffe6900
1 1 Char --> fffe2700
1 1 Char --> fffe6b00
1 1 Char --> fffe6100
1 1 Char --> fffe6c00
1 1 Char --> fffe6600
1 1 Char --> fffe7500
1 1 Char --> fffe6700
1 1 Char --> fffe7400
1 1 Char --> fffebf00
1 1 Char --> fffe6400
1 1 Char --> fffe6500
1 1 SeededPrefixMorph --> fffe6100
1 1 SeededPrefixMorph --> fffe6800 fffe6900
1 1 SeededPrefixMorph --> fffe6800 fffe6100
1 1 SeededPrefixMorph --> fffe6b00 fffe6100
1 1 SeededSuffixMorph --> fffe6b00
1 1 SeededSuffixMorph --> fffe6d00
1 1 SeededSuffixMorph --> fffe7400 fffe6100
1 1 SeededSuffixMorph --> fffe6b00 fffe6100
1 1 SeededSuffixMorph --> fffe6e00 fffe6500
1 1 SeededSuffixMorph --> fffe7000 fffe6f00
```

Figure 4.4: Example of a processed Scholar-seeded grammar. The seeds are taken from the input as shown in fig.4.3. Compare this output with fig. 3.5, to note the similarity.

4.3.2 Training

In this phase we just pass the files prepared in section 4.3.1 as inputs to the AG.

```
(Word (Prefix#1 ^^^) (Stem#3 (SubMorphs (SubMorph#10 (Chars (Char fffe6200) (Chars (Char fffe7700) (
Chars (Char fffe6100))))) (SubMorphs (SubMorph#1 (Chars (Char fffe7300) (Chars (Char fffe7300) (
Chars (Char fffe6900) (Chars (Char fffe6100)))))))) (Suffix#2 (SuffixMorphs (SuffixMorph (
SeededSuffixMorph fffe7400 fffe6100))) $$$))

(Word (Prefix#3 ^^^) (Stem#2 (SubMorphs (SubMorph#3 (Chars (Char fffe6200) (Chars (Char fffe6500) (
Chars (Char fffe7700) (Chars (Char fffe2700) (Chars (Char fffe7500) (Chars (Char fffe7200) (Chars (
Char fffe7500)))))))))) (Suffix#2 (SuffixMorphs (SuffixMorph (SeededSuffixMorph fffe6b00
fffe6100))) $$$))

(Word (Prefix#1 ^^^ (PrefixMorphs (PrefixMorph (SubMorphs (SubMorph#1 (Chars (Char fffe6200) (Chars
(Char fffe7500)))))))) (Stem#3 (SubMorphs (SubMorph#2 (Chars (Char fffe2700))) (SubMorphs (
SubMorph#4 (Chars (Char fffe7200) (Chars (Char fffe7500)))))) (Suffix#3 (SuffixMorphs (SuffixMorph
(SubMorphs (SubMorph#9 (Chars (Char fffe6b00) (Chars (Char fffe6100)))))) (SuffixMorphs (
SuffixMorph (SeededSuffixMorph fffe6d00)))) $$$))

(Word (Prefix#6 ^^^ (PrefixMorphs (PrefixMorph (SubMorphs (SubMorph#1 (Chars (Char fffe6d00) (Chars
(Char fffe6500)))) (SubMorphs (SubMorph#4 (Chars (Char fffe2700)))))))) (Stem#1 (SubMorphs (
SubMorph#1 (Chars (Char fffe6100))))) (Suffix#2 (SuffixMorphs (SuffixMorph (SeededSuffixMorph
fffe6b00 fffe6100)) (SuffixMorphs (SuffixMorph (SubMorphs (SubMorph#3 (Chars (Char fffe6d00) (Chars
(Char fffe6500)))))))) $$$))
```

Figure 4.5: An example output of the parse file by the AG. Grammar tree of each word is printed.

The AG produces 3 output files.

- A parse file, that contains the grammar tree of each words in the training set.
- An output grammar file, where the AG writes the final grammar used by it for segmentation.
- A trace file, which traces the hyperparameters over several iterations, for that particular run.

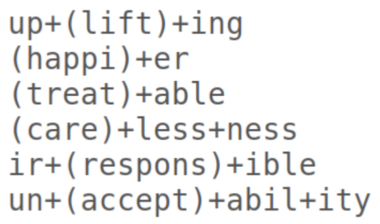
For the Cascaded Settings, the top affixes are picked up from the output of Run-1, and are used to prepare the grammar input for Run-2.

4.3.3 Segmentation

In this phase we segment the words from the given input list. The AG parse output is also converted to human readable format (for analysis purposes).

To do the word segmentation, we first process the parse ouput of AG and map the words with their respective segmentations.

As we use transductive mode of learning in our experiments, any query for the word segmentation just returns the corresponding value from the segmentation map.



```
up+(lift)+ing
(happi)+er
(treat)+able
(care)+less+ness
ir+(respons)+ible
un+(accept)+abil+ity
```

Figure 4.6: An example output of the segmentation of a word list.

4.4 GUI App

This section outlines the python based GUI app developed using PyQt4 toolkit. This app tries to simplify and automate the processes described in section 4.3

Salient features:

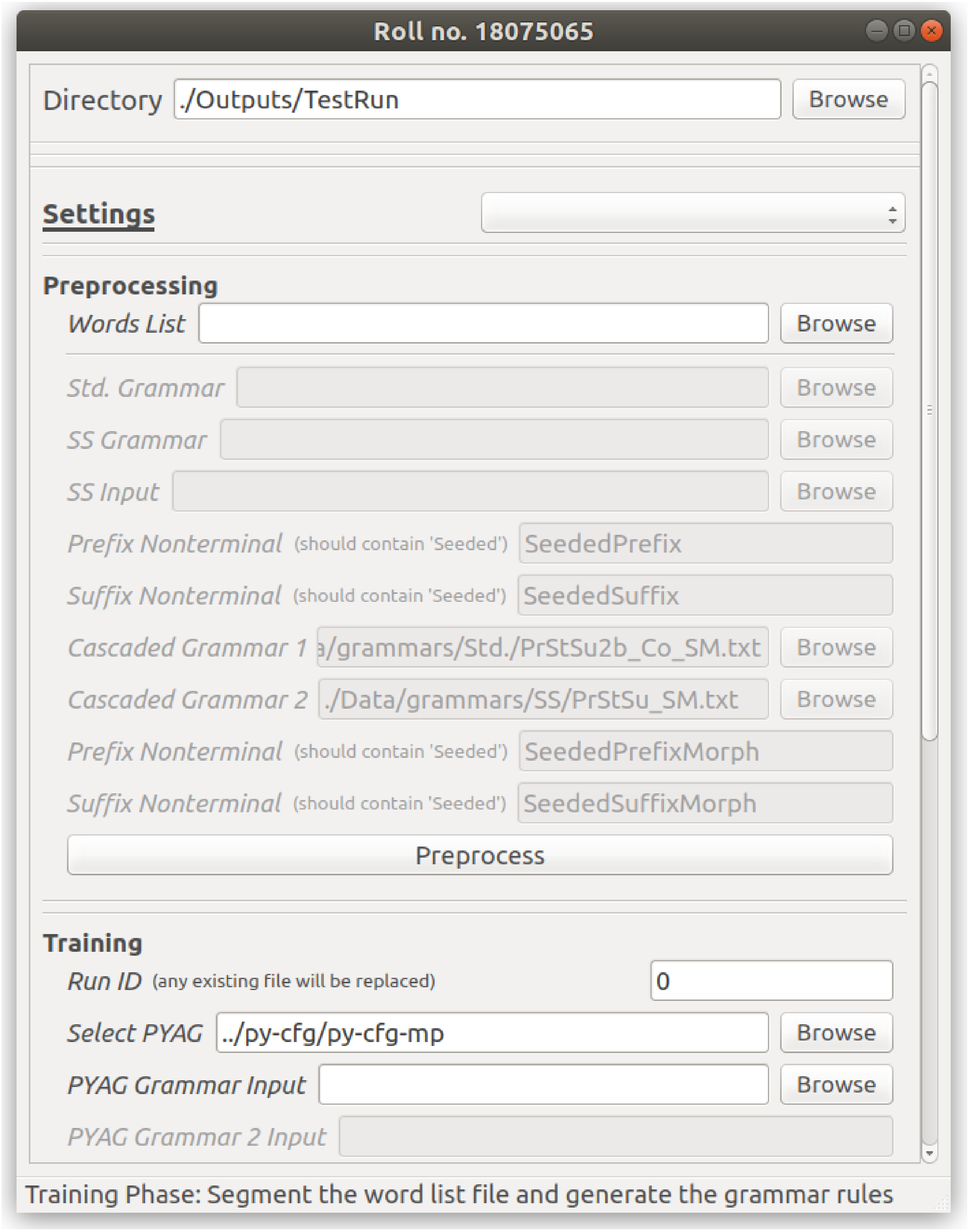
- The app automates trivial processes.
- All the processes are grouped under respective phases. The app, therefore, describes the workflow clearly.
- The app is complete in the sense that it allows the adjustment of all the parameters associated with the experiment.
- The graphical interface allows for faster manual work and makes it easy to work with, for *non-programmers.*
- The app is future ready and can be easily extended to be used in any similar research work.

Some features of the app that are visible in fig. 4.7 are described below.

- The most common values for various fields have already been set as the default values to speed up the process. These values can still be changed if one wants.
- The choice of the Settings highlights the appropriate fields for input from the user. For example, for Scholar Seeded setting, the fields, *'SS Grammar'*, *'SS Input'*, *'Prefix Nonterminal'* and *'Suffix Nonterminal'* are highlighted.
- **The status bar (at the bottom of the app) shows the help for the button/field/label when pointed at by the mouse cursor.** The app can, therefore, be used by anyone with a general idea of the model.

4.5 Screenshots

In this section, some screenshots of a sample run are shown.



Roll no. 18075065

Directory ./Outputs/TestRun Browse

Settings

Preprocessing

Words List Browse

Std. Grammar Browse

SS Grammar Browse

SS Input Browse

Prefix Nonterminal (should contain 'Seeded') SeededPrefix

Suffix Nonterminal (should contain 'Seeded') SeededSuffix

Cascaded Grammar 1 a/grammars/Std./PrStSu2b_Co_SM.txt Browse

Cascaded Grammar 2 ./Data/grammars/SS/PrStSu_SM.txt Browse

Prefix Nonterminal (should contain 'Seeded') SeededPrefixMorph

Suffix Nonterminal (should contain 'Seeded') SeededSuffixMorph

Preprocess

Training

Run ID (any existing file will be replaced) 0

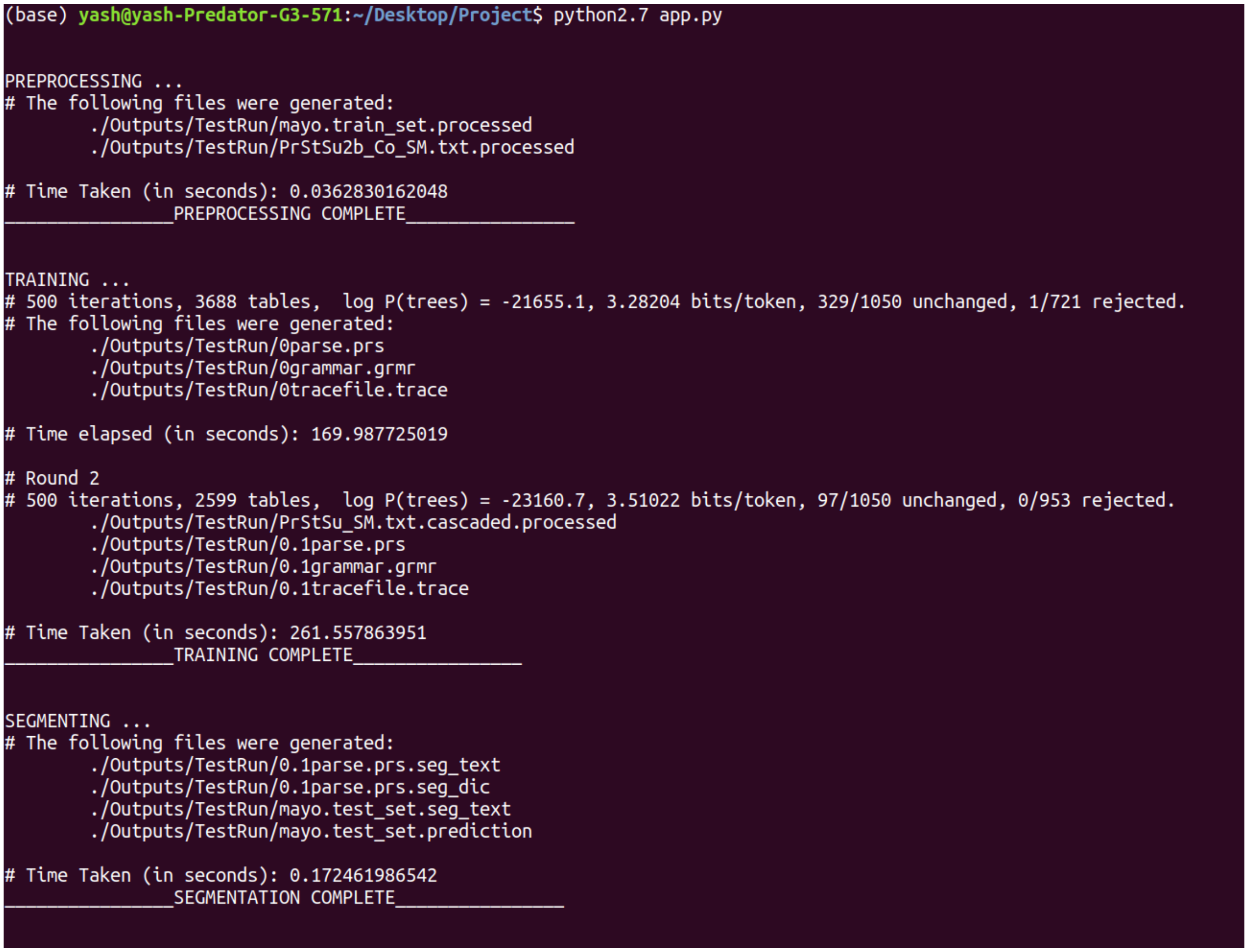
Select PYAG ../py-cfg/py-cfg-mp Browse

PYAG Grammar Input Browse

PYAG Grammar 2 Input

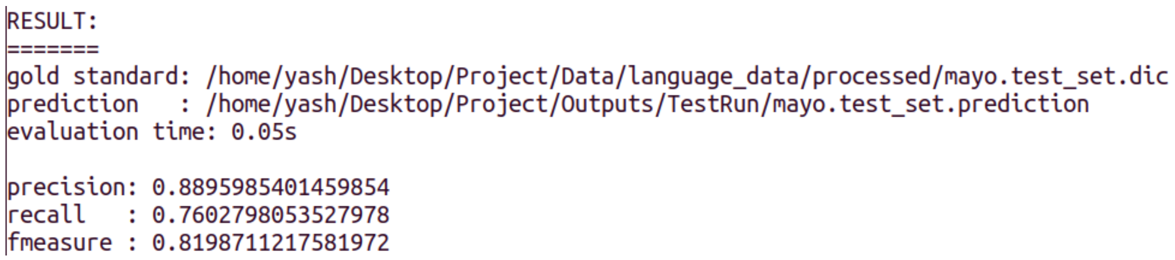
Training Phase: Segment the word list file and generate the grammar rules

Figure 4.7: Screenshot of the actual app in working.



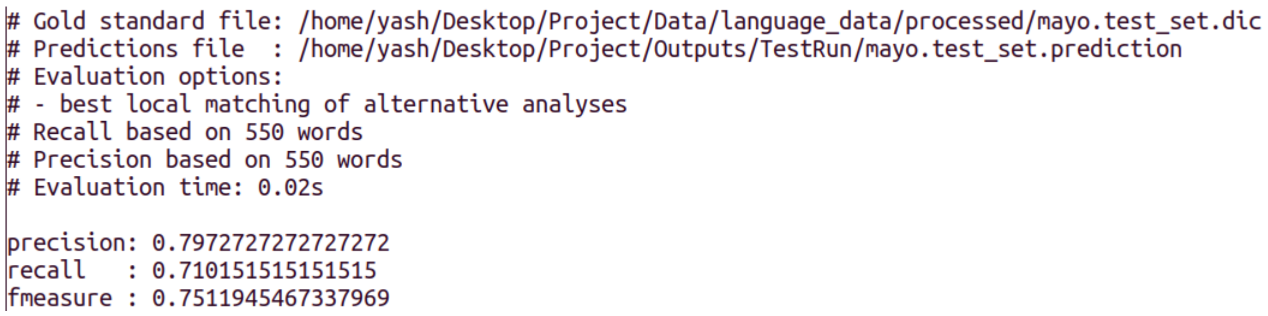
```
(base) yash@yash-Predator-G3-571:~/Desktop/Project$ python2.7 app.py


PREPROCESSING ...
# The following files were generated:
        ./Outputs/TestRun/mayo.train_set.processed
        ./Outputs/TestRun/PrStSu2b_Co_SM.txt.processed

# Time Taken (in seconds): 0.0362830162048
________________PREPROCESSING COMPLETE________________


TRAINING ...
# 500 iterations, 3688 tables,  log P(trees) = -21655.1, 3.28204 bits/token, 329/1050 unchanged, 1/721 rejected.
# The following files were generated:
        ./Outputs/TestRun/0parse.prs
        ./Outputs/TestRun/0grammar.grmr
        ./Outputs/TestRun/0tracefile.trace

# Time elapsed (in seconds): 169.987725019

# Round 2
# 500 iterations, 2599 tables,  log P(trees) = -23160.7, 3.51022 bits/token, 97/1050 unchanged, 0/953 rejected.
        ./Outputs/TestRun/PrStSu_SM.txt.cascaded.processed
        ./Outputs/TestRun/0.1parse.prs
        ./Outputs/TestRun/0.1grammar.grmr
        ./Outputs/TestRun/0.1tracefile.trace

# Time Taken (in seconds): 261.557863951
________________TRAINING COMPLETE________________


SEGMENTING ...
# The following files were generated:
        ./Outputs/TestRun/0.1parse.prs.seg_text
        ./Outputs/TestRun/0.1parse.prs.seg_dic
        ./Outputs/TestRun/mayo.test_set.seg_text
        ./Outputs/TestRun/mayo.test_set.prediction

# Time Taken (in seconds): 0.172461986542
________________SEGMENTATION COMPLETE________________
```

Figure 4.8: Terminal after completing all the processes.



```
RESULT:
=======
gold standard: /home/yash/Desktop/Project/Data/language_data/processed/mayo.test_set.dic
prediction   : /home/yash/Desktop/Project/Outputs/TestRun/mayo.test_set.prediction
evaluation time: 0.05s

precision: 0.8895985401459854
recall   : 0.7602798053527978
fmeasure : 0.8198711217581972
```

Figure 4.9: Results of EMMA-2 metric for a sample run.



```
# Gold standard file: /home/yash/Desktop/Project/Data/language_data/processed/mayo.test_set.dic
# Predictions file  : /home/yash/Desktop/Project/Outputs/TestRun/mayo.test_set.prediction
# Evaluation options:
# - best local matching of alternative analyses
# Recall based on 550 words
# Precision based on 550 words
# Evaluation time: 0.02s

precision: 0.7972727272727272
recall   : 0.710151515151515
fmeasure : 0.7511945467337969
```

Figure 4.10: Results of the BPR metric for a sample run.

Chapter 5

Conclusions and Discussion

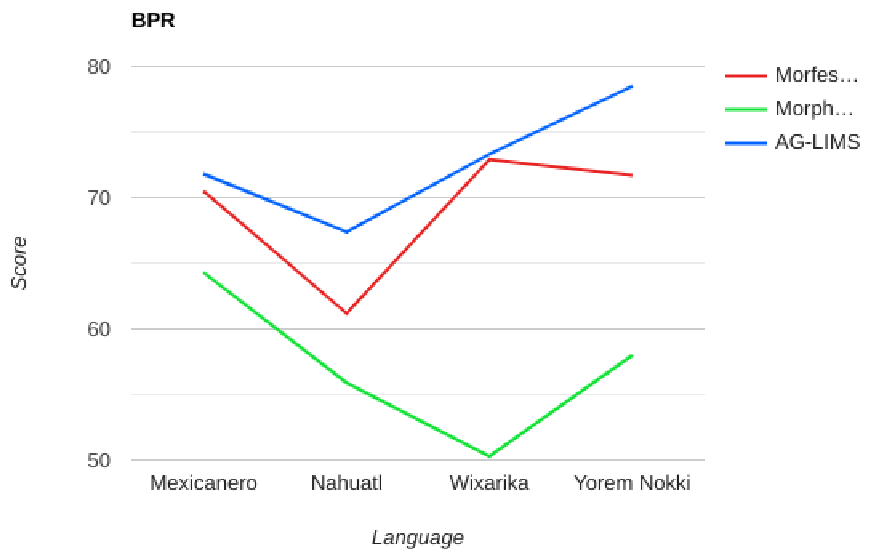


Figure 5.1: Comparison of BPR F1-scores of the setups, across all languages (from table 3.5)

As can be seen from Table 3.4, Fig. 5.1 and Fig. 5.2, we constantly obtain better results versus Morfessor and MorphoChain, for each language when tested using BPR and EMMA-2 metrics. The AG-LIMS setup achieves the best result on all languages in unsupervised manner. The AG-LIMS setup achieves absolute average F1-score increases of 3.7% and 15.7% over Morfessor and MorphoChain respectively, when

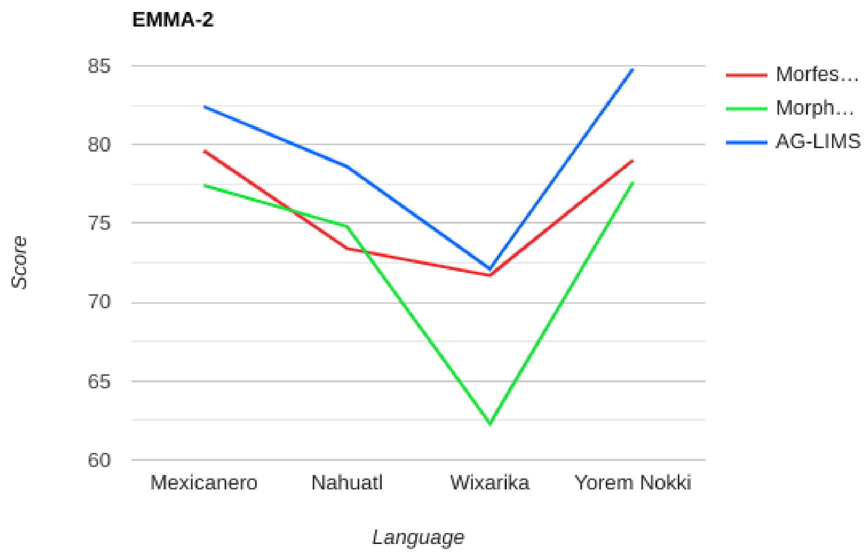


Figure 5.2: Comparison of EMMA-2 F1-scores of the setups, across all languages (from table 3.5)

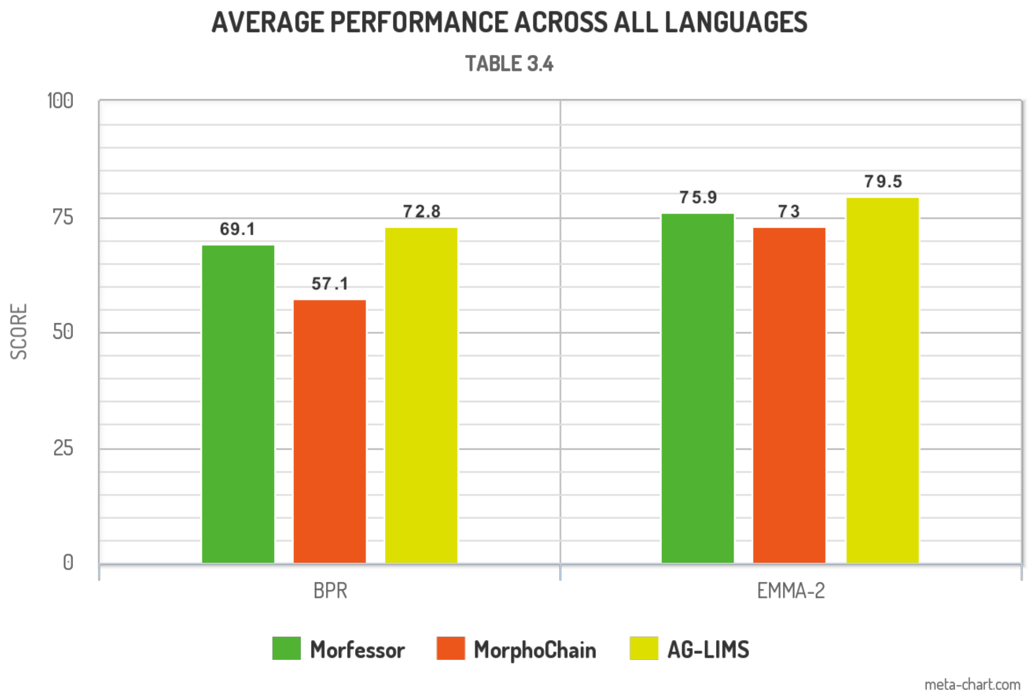


Figure 5.3: Comparison of the average F1 score of the setups (BPR scores on the left side and EMMA-2 scores on the right side of the graph)

using BPR metric.

We also establish that Adaptor Grammars based system is able to **generalize and learn well from a small amount of data**, which is the case when learning the segmentation for the polysynthetic languages.

Result Analysis and scope for Future Work

Unsupervised approaches based on Adaptor Grammars show promise in the area of morphological segmentation of low-resource polysynthetic languages.

- We showed that the setup achieves the state-of-the-art results on all languages on both metrics.
- We showed that even with very small amount of unsegmented data (training data), the setup is able to produce promising results.

In future this setup can be used for morphological segmentation of some Indian agglutinative languages (for e.g., Dravidian languages). Large morphological dataset is not available for most Indian languages, hence the described setup is promising for this task.

We also release our GUI app that can help to carry out such research work in future.

Bibliography

[1] M. Johnson, T. L. Griffiths, and S. Goldwater, "Adaptor grammars: A framework for specifying compositional nonparametric bayesian models," in *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. C. Platt, and T. Hoffman, Eds. MIT Press, 2007, pp. 641–648. [Online]. Available: http://papers.nips.cc/paper/3101-adaptor-grammars-a-framework-for-specifying-compositional-nonparametric-bayesian-models.pdf

[2] R. Eskander, J. Klavans, and S. Muresan, "Unsupervised morphological segmentation for low-resource polysynthetic languages," in *Proceedings of the 16th Workshop on Computational Research in Phonetics, Phonology, and Morphology*. Florence, Italy: Association for Computational Linguistics, Aug. 2019, pp. 189–195. [Online]. Available: https://www.aclweb.org/anthology/W19-4222

[3] R. Eskander, F. Callejas, E. Nichols, J. Klavans, and S. Muresan, "MorphAGram, evaluation and framework for unsupervised morphological segmentation," in *Proceedings of The 12th Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, May 2020, pp. 7112–7122. [Online]. Available: https://www.aclweb.org/anthology/2020.lrec-1.879

[4] M. Creutz and K. Lagus, "Unsupervised models for morpheme segmentation and

morphology learning," *ACM Trans. Speech Lang. Process.*, vol. 4, pp. 3:1–3:34, 02 2007.

[5] K. Narasimhan, R. Barzilay, and T. S. Jaakkola, "An unsupervised method for uncovering morphological chains," *CoRR*, vol. abs/1503.02335, 2015. [Online]. Available: http://arxiv.org/abs/1503.02335

[6] Wikipedia contributors, "Morpheme — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=Morpheme&oldid=960634482, 2020, [Online; accessed 19-June-2020].

[7] ——, "Affix — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=Affix&oldid=961671292, 2020, [Online; accessed 18-June-2020].

[8] ——, "Inflection — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=Inflection&oldid=955388236, 2020, [Online; accessed 20-June-2020].

[9] ——, "Agglutination — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=Agglutination&oldid=960909785, 2020, [Online; accessed 20-June-2020].

[10] ——, "Synthetic language — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=Synthetic_language&oldid=961702857, 2020, [Online; accessed 20-June-2020].

[11] K. Kann, J. M. Mager Hois, I. V. Meza-Ruiz, and H. Schütze, "Fortification of neural morphological segmentation models for polysynthetic minimal-resource languages," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers).* New Orleans, Louisiana:

Association for Computational Linguistics, Jun. 2018, pp. 47–57. [Online]. Available: https://www.aclweb.org/anthology/N18-1005

[12] Wikipedia contributors, "Context-free grammar — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=Context-free_grammar&oldid=959884744, 2020, [Online; accessed 20-June-2020].

[13] K. Sirts and S. Goldwater, "Minimally-supervised morphological segmentation using adaptor grammars," *Transactions of the Association for Computational Linguistics*, vol. 1, pp. 255–266, 2013. [Online]. Available: https://www.aclweb.org/anthology/Q13-1021

[14] Wikipedia contributors, "Polysynthetic language — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=Polysynthetic_language&oldid=955997075, 2020, [Online; accessed 18-June-2020].

[15] M. Mager, X. Gutierrez-Vasques, G. Sierra, and I. Meza-Ruiz, "Challenges of language technologies for the indigenous languages of the Americas," in *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, Aug. 2018, pp. 55–69. [Online]. Available: https://www.aclweb.org/anthology/C18-1006

[16] R. Eskander, O. Rambow, and T. Yang, "Extending the use of adaptor grammars for unsupervised morphological segmentation of unseen languages," in *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. Osaka, Japan: The COLING 2016 Organizing Committee, Dec. 2016, pp. 900–910. [Online]. Available: https://www.aclweb.org/anthology/C16-1086

[17] J. Pitman and M. Yor, “The two-parameter poisson-dirichlet distribution derived from a stable subordinator,” *Ann. Probab.*, vol. 25, no. 2, pp. 855–900, 04 1997. [Online]. Available: https://doi.org/10.1214/aop/1024404422

[18] H. Ishwaran and L. F. James, “Generalized weighted chinese restaurant processes for species sampling mixture models,” *STATISTICA SINICA*, vol. 13, p. 2003, 2003.

[19] S. Virpioja, V. T. Turunen, S. Spiegler, O. Kohonen, and M. Kurimo, “Empirical comparison of evaluation methods for unsupervised learning of morphology,” *Traitement Automatique des Langues*, vol. 52, no. 2, pp. 45–90, 2011.