Exploratory Project of Second Year B.Tech.

Unsupervised Morphological Segmentation for Low-Resource Polysynthetic Languages

under the guidance of

Dr. A.K. Singh

Yash Malik
(18075065)
B.Tech. - 4th semester

24.06.2020

Overview

Problem

Morphological Segmentation for Polysynthetic Languages

Solution Presented

- An Adaptor Grammars based unsupervised setup
- A GUI app to carry out the experiments

Results

Our setup **outperforms** the state-of-the-art baselines on all the languages in the experiments

Motivation

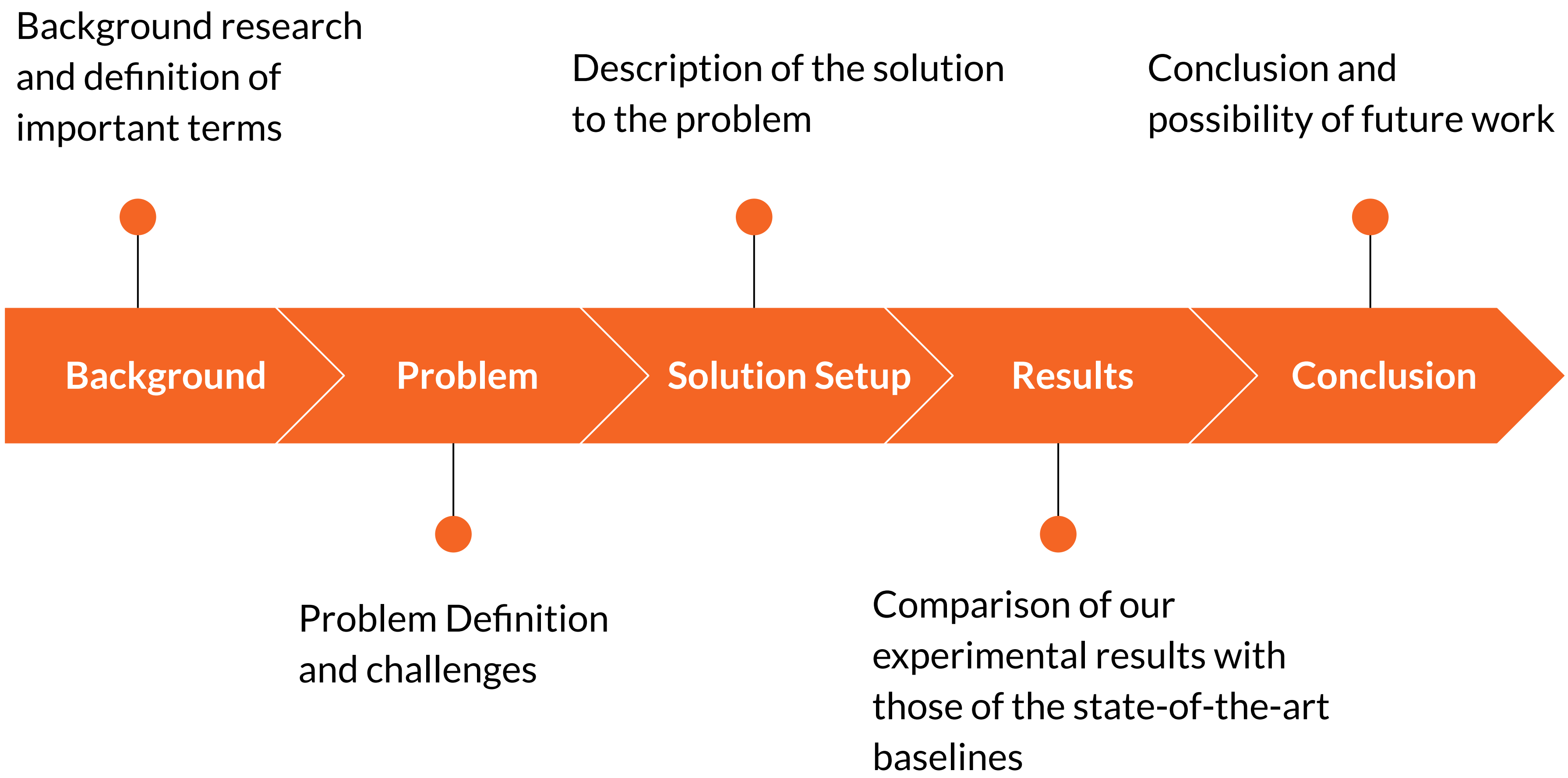
Morphological Segmentation

- Computational morphological segmentation is an active research topic as it forms an essential basis for many Natural Language processing tasks.
- Has applications in *POS tagging*, *text classification*, *machine translation*, *speech recognition* etc.

Polysynthetic Languages

- Pose a unique challenge to the task of morphological segmentation due to the root-morpheme complexity.
- High cost of manually labelling data for morphology.
- Training data is extremely scarce for these languages.

Structure of Presentation


Background research and definition of important terms
Description of the solution to the problem
Conclusion and possibility of future work
Background
Problem
Solution Setup
Results
Conclusion
Problem Definition and challenges
Comparison of our experimental results with those of the state-of-the-art baselines

Background Knowledge

Background Knowledge

- **Morpheme** : the smallest meaningful unit in a language. E.g., *cat*, *dog*, suffixes (*-es*, *-ing*, *-tion*), prefixes (*pre-*, *bi-*) etc.
- **Synthetic Language** : A synthetic language uses inflection or agglutination to express syntactic relationships within a sentence, i.e., words tend to be more morphologically complex
- **Inflection** : a word is **modified** to express different grammatical categories. E.g., plural forms (*mouse -> mice*, *foot -> feet*), tenses (*write -> wrote -> written*).
- **Agglutination** : complex words are formed by **stringing together** morphemes **without changing them in spelling or phonetics**. E.g., *care + less + ness*.
- **Context Free Grammar (CFG)** : a set of recursive rules that generate the patterns of strings.
- **Probabilistic CFG (PCFG)** : CFG with probability associated with its rules.

Background Research

Adaptor Grammars

- Adaptor Grammars (AG) is a framework introduced by Johnson et. al (2007) for specifying nonparametric Bayesian models that can be used to **learn latent tree structures from a corpus of strings**.
- The authors use a Pitman-Yor Process as the adaptor (PYP). 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.

Problem Description

Problem Definition

Statement

Design a statistical machine learning algorithm to split a given word (from a polysynthetic language) into the surface forms of its smallest meaning-bearing units, morphemes.

Examples for English Language:

- **treatable** = *treat + able*
- **carelessness** = *care + less + ness*
- **irresponsible** = *ir + respons + ible*
- **unacceptability** = *un + accept + ability*

Problem Definition

Polysynthetic Language

- Highly synthetic languages, i.e., **words are composed of many morphemes**
- Typically have long "sentence-words"

Consider for example the Yupik word

tuntussuqatarniksaitengqiggtuq

"He had not yet said again that he was going to hunt reindeer."

tuntu-ssur-qatar-ni-ksaite-ngqiggte-uq

reindeer-hunt-(future)-say-(negation)-again-(third person-singular-indicative)

and except for the morpheme **tuntu** "reindeer", none of other morphemes can appear in isolation.

Challenges associated with the task

- 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.
- The morphology is synthetically complex (not simply agglutinative).
- The resources for such languages are minimal, therefore we have very low training data available.

Approach to the Solution

Notable Points

- As highlighted earlier, that the **resources are scarce and manual annotation has high cost** (especially for unknown languages). We, therefore, find a completely **unsupervised** way for this task.
- **Language-Independent,** we don’t use any language dependent data, and our setup for all the languages will be the same.
- As we **only consider unsupervised way**, we never include the segmented form of any word in our training.

Using Adaptor Grammars for Morphological Segmentation

Grammar

We need to define a CFG for **the word grammar**. An example of such grammar is shown on the right.

We define and experiment with 9 such grammars.

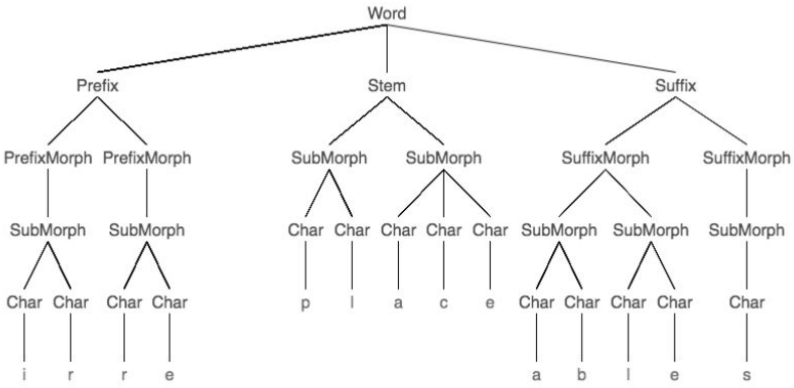


Fig 1. The representation[1] of English word irreplaceables segmented using PrStSu+SM

[1]Taken from Eskander et al. (2020)

Probabilistic Context Free Grammar

- PCFGs are used as the input to Adaptor Grammars
- The grammar on the right is a valid PCFG input for the Adaptor Grammars
- Note the simple recursive rules
- ‘^^^’ represents the beginning of word
- ‘$$$’ represents the end

```
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
```

Fig 2. The standard PrStSu+SM grammar

Standard Settings (Unsupervised Way)

We use the grammars described in Fig 2 as is, we don't add any linguistic knowledge.

We only use unsegmented words as the training input.

Fig 3. Sample results of a Std. Run (on EMMA-2 metrics)

```
RESULT:
=======
gold standard: /home/yash/Desktop/Demo/Code+Data/Data/language_data/processed/mayo.test_set.dic
prediction   : /home/yash/Desktop/Demo/Code+Data/Outputs/TestRun/mayo.test_set.prediction
evaluation time: 0.06s

precision: 0.7849756690997571
recall   : 0.6953163017031626
fmeasure : 0.7374307095232839
```

```
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
```

Grammars with Linguistic Knowledge

We can seed linguistic knowledge to our grammars as additional production rules.

We seed a list of affixes (Prefix/Suffix) in our grammar.

Fig 4. An example of a scholar-seeded grammar
(notice the prefix and suffix seeds in the end)

Scholar Seeded Settings
(Semi Supervised Way)

As expected, experiments suggest that we achieve better results on addition of linguistic knowledge to our grammars.

But this is not what we want as our aim is to go for completely unsupervised way.

Cascaded Setup

As noted, that scholar-seeded knowledge generally improves the performance of our model.

We try to automate the task of seeding linguistic knowledge to our grammars.

- Run a **high precision**[1] grammar with standard settings
- Pick the top affixes
- Seed them into the final grammar
- Run the modified grammar

[1]We don't want the affixes picked up as the seeds to be incorrect.

Best Cascaded Setup

The cascade of grammars gives a significant improvement.

We find that the high precision grammar **PrStSu2b+Co+SM**, followed by **PrStSu+SM** grammar, gives us the best results.

We describe our results in the next section, where we call this setup of cascades of grammar as **AG-LIMS** (as **LIMS** is the best on average cascaded setup proposed by Eskander et al. (2016), and our setup is basically the same setup as LIMS).

App

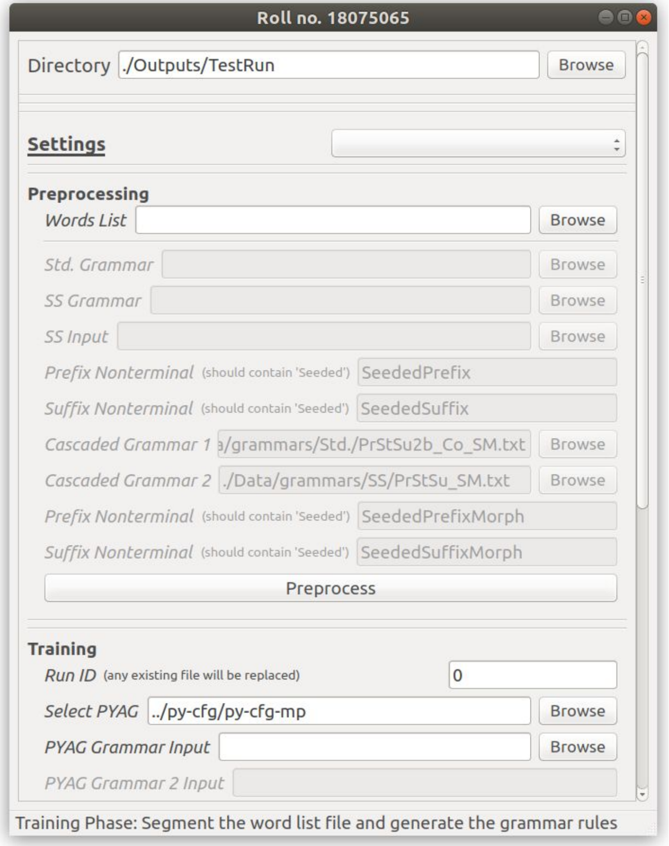
Slight changes in our parameters, required changing of code at several places

Automates trivial processes, e.g., filling out the grammar and word list input to the trainer after preprocessing

Divided into phases

Help for every label/button/field

Can be easily extended to be used in any similar research

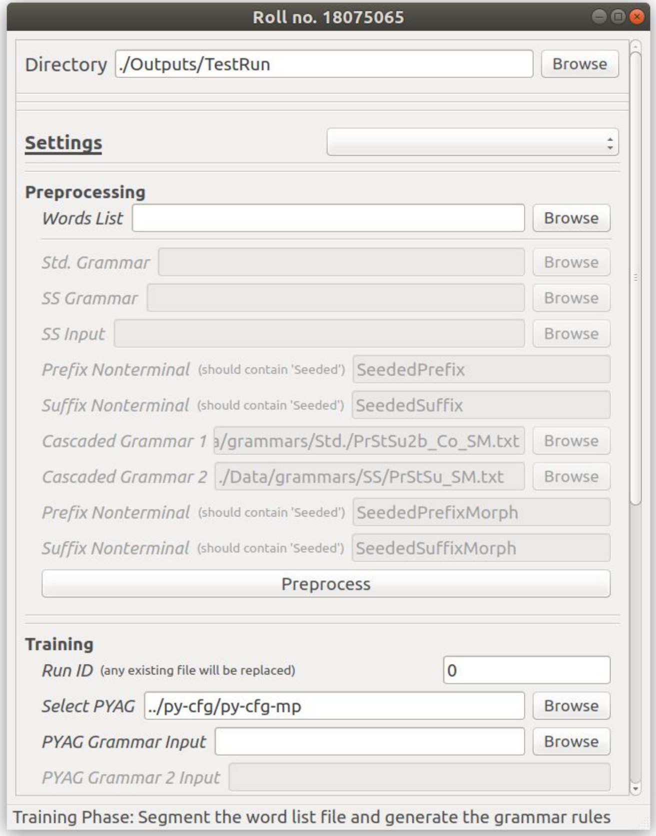


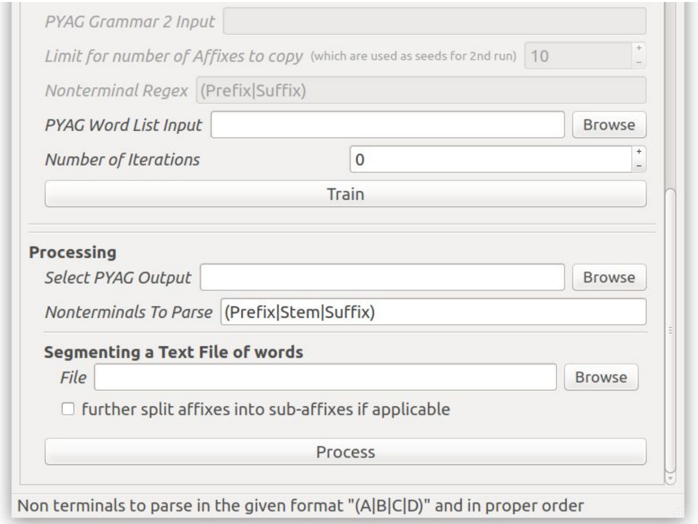
App layout

Directory where all the outputs are stored

Choose the settings for the run

Various grammar options that are highlighted for input, according to the chosen settings





Status Bar displays help

Overview of working of the app

```
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
```

Grammar Rules

\+

```
uplifiting
happier
treatable
carelessness
irresponsible
unacceptability
```

Word list

→

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

Segmented Word List

However, behind the scenes many processes are going on (described in report)

—

A sample run of the app with

- Cascaded settings
- Yorem Nokki
- 500 iterations of the sampler

```
(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________________
```

Results

Experimental Results

- **Precision** : how many selected items are relevant
- **Recall** : how many relevant items are selected
- **F1-score** : harmonic mean of precision and recall

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

Our Scores on the metrics BPR and EMMA-2

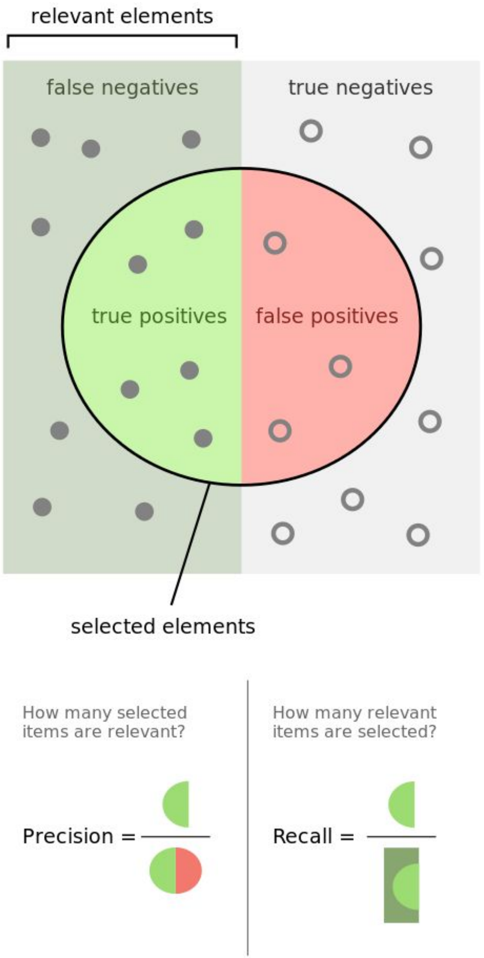
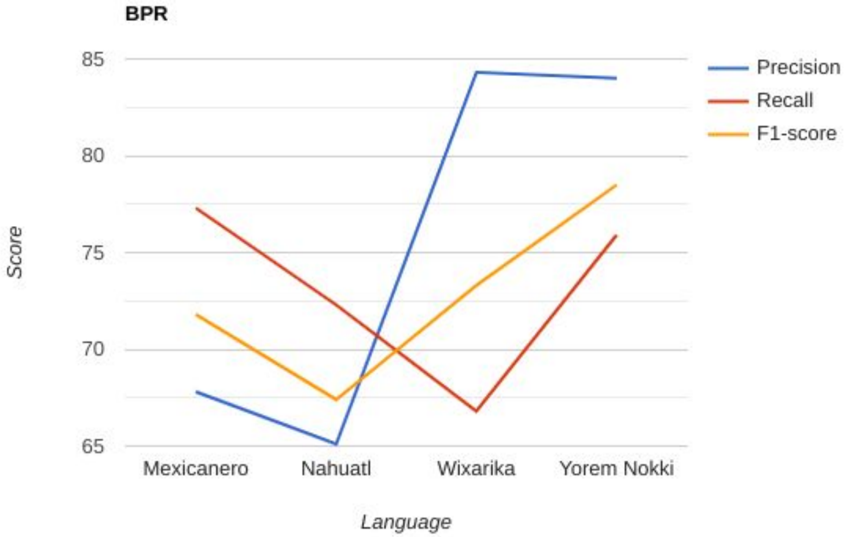
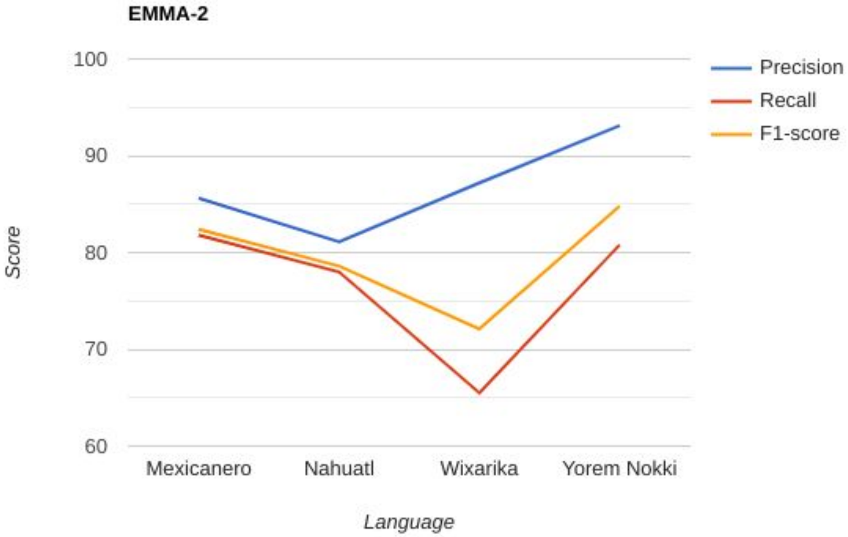


Image taken from https://en.wikipedia.org/wiki/Precision_and_recall

Experimental Results





Comparison of our scores on all four languages

Example results of a sample run (MX)

```
# Gold standard file: /home/yash/Desktop/Demo/Code+Data/Data/language_data/processed/mexicanero.test_set.dic
# Predictions file  : /home/yash/Desktop/Demo/Code+Data/Outputs/TestRun/mexicanero.test_set.prediction
# Evaluation options:
# - best local matching of alternative analyses
# Recall based on 458 words
# Precision based on 458 words
# Evaluation time: 0.02s


precision: 0.6827272727272727
recall   : 0.7630303030303029
fmeasure : 0.7206486156894876
```

Results on BPR metric

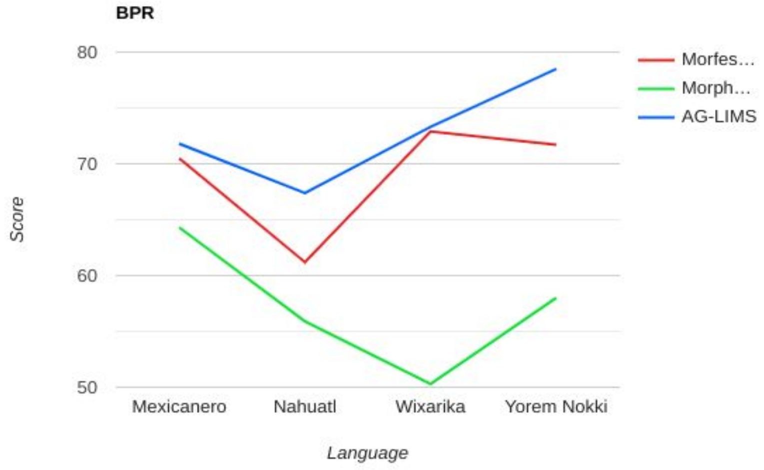
```
RESULT:
=======
gold standard: /home/yash/Desktop/Demo/Code+Data/Data/language_data/processed/mexicanero.test_set.dic
prediction   : /home/yash/Desktop/Demo/Code+Data/Outputs/TestRun/mexicanero.test_set.prediction
evaluation time: 0.05s

precision: 0.8704379562043799
recall   : 0.7551094890510944
fmeasure : 0.8086825915522501
```

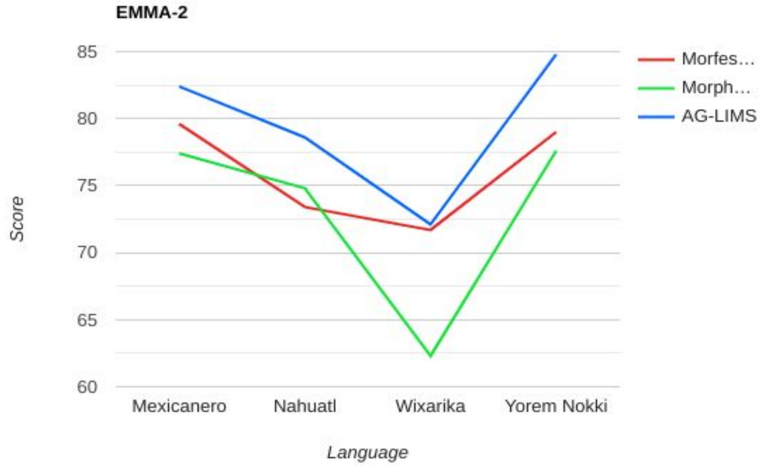
Results on EMMA-2 metric

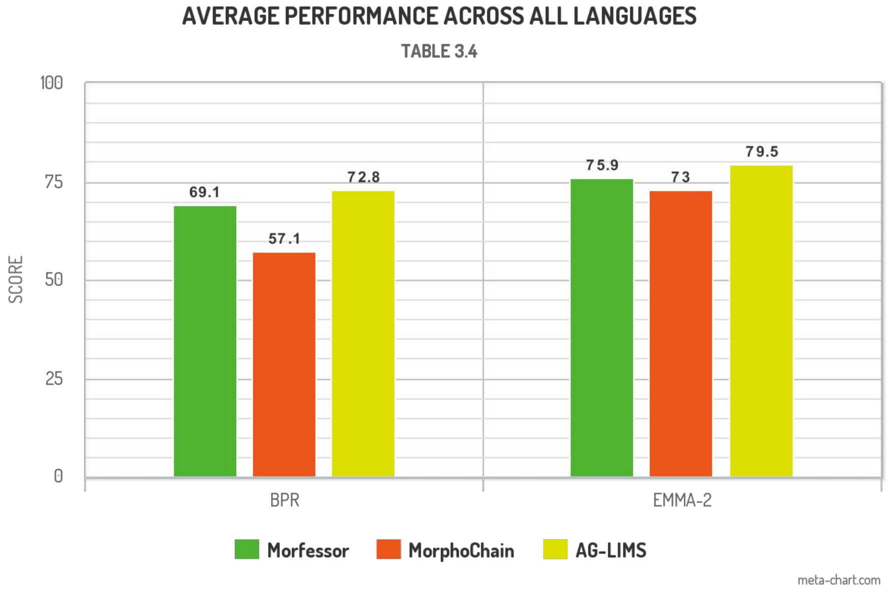
Experimental Results (Comparison)

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

Comparison of our model against Morfessor and MorphoChain







Comparison of our Model against Morfessor and MorphoChain across all languages and average performance

Conclusion

1. the setup achieves the **state-of-the-art** results on all languages on both metrics
2. even with very small amount of unsegmented data (training data), the setup produces promising results
3. Adaptor Grammars based system is able to generalize and learn well from a small amount of data

Scope for Future Work

We note that such a setup performs well even with low resources.

Can be used for **morphological segmentation of Indian Languages** which are agglutinative in nature (**Dravidian Languages**) and don't have large morphological dataset available.

The app released can be quite useful for this task.