Implementation of Spectral Learning of Latent-Variable Probabilistic Context-Free Grammars

University of St Andrews

Author:
Tatiana Matejovičová

Supervised by:
Dr Mark-Jan Nederhof
Dr Louis Theran

9 April 2018

Abstract

In this thesis, spectral learning of latent-variable probabilistic context-free grammars is described, implemented and tested. Spectral learning, based on singular value decomposition, is used to extract probabilistic distribution of context-free rules accompanied by latent states as introduced by Cohen et al. (2012). The updated version of the original algorithm based on clustering (Narayan and Cohen, 2015) is implemented in Python. A pipeline to test the grammar accuracy is assembled and used in multiple experiments. The effect of training corpus size and number of latent states on accuracy is evaluated finding the minimum training corpus size of 20 thousand sentences for 8 latent states. Furthermore, for our implementation, 8 latent states are shown to be the optimum. Finally, the implementation is compared to that of Cohen et al. (2013) and the reasons for lower accuracy of the implementation are analysed.

Contents

1 Introduction

In natural language processing (NLP), Context-free grammars (CFGs) are designed to model the syntax of a particular language. Being used for syntactic parsing, they are an integral part of multiple NLP applications. Syntactic parsing is an important intermediate step in semantic analysis and hence it is essential for tasks such as information extraction and question answering. To deal with ambiguity in natural language when parsing, probabilistic distribution is assigned to context-free rules resulting in probabilistic CFGs (PCFGs) (Charniak, 1997).

Probabilistic models with hidden or latent states are of a great value in natural language processing. These include for example hidden markov models (HMMs) for part-of-speech (POS) tagging and CFGs. Typically, the hidden or latent states are not observed in the labeled input data and are assigned during the training phase so that they maximise the likelihood of the data given the model. In HMMs for example, part-of-speech tags are hidden states that are assigned so that the likelihood of the corpus sentences given the tags is maximised. In PCFGs, nodes of parse trees are augmented with latent states resulting in latent-variable PCFGs (L-PCFGs). The model is defined over the latent random variables that detect patterns and explain correlations between the observed patterns.

The Baum-Welch/Expectation-Maximization (EM) algorithm is one of the traditional algorithms that has proven very successful to estimate hidden or latent variable models: HMMs, CFGs (Petrov et al., 2006) or generally, models with missing data (Dempster et al., 1977). However, it gives no guarantee to find the global maximum of the likelihood function and therefore to give consistent parameter estimates. In practice, it might become more difficult to reach the global maximum. Consequently, an alternative learning algorithm based on singular value decomposition (SVD) has been proposed by Hsu et al. (2012) that provides consistent estimates of HMMs in polynomial time.

Cohen et al. (2012) extend the work of Hsu et al. (2012) on spectral learning of HMMs and the work of Petrov et al. (2006) on L-PCFGs by proposing a spectral learning learnong algorithm of L-PCFGs that provides consistent parameter estimates, resistant to local maxima of the likelihood function, in contrast to the previously described EM algorithm. It is based on SVD of the training examples followed by the projection of the training examples to a lower-dimensional space, calculation of empirical averages and matrix operations. Later, Narayan and Cohen (2015) proposed a modified version of the algorithm based on k-means clustering that is more intuitive and easier to implement.

The aim of this work is to implement the latter version of the spectral learning algorithm of L-PCFGs in a single high-level programming language. The published implementation by Narayan and Cohen (2017) uses a combination of Java and Matlab for the algorithm implementation. In this work it is demonstrated that it is possible to implement the algorithm in a single programming language, namely Python. Furthermore, a number of experiments is performed to evaluate the effect of the training corpus size and number of latent states on the parsing accuracy of the resulting grammar. Finally, the results are compared to those of Cohen et al. (2013).

2 Background

In this section the necessary background is defined and illustrated.

2.1 Definition of L-PCFGs

In this section the formal definition and examples of CFGs, PCFGs and L-PCFGs are given. The motivation for latent states and probability is explained and their utility is illustrated.

2.1.1 CFGs

Context-free grammars are the most commonly used system to model the constituent structure of natural language i.e. the formal rules according to which words can be grouped into phrases and how these can form a sentence. A CFG consists of rules that define how symbols of the grammar can be grouped and ordered and a lexicon which translates a subset of the symbols to words. A simplified example of a CFG is shown in Figure 1.

S	→ NP VP
NP	→ Pronoun | Det Nominal
Nominal	→ Nominal Noun | Noun
VP	→ Verb NP

(a) Set of rules

Pro	→ *I*
Verb	→ *have* | *see*
Det	→ *a* | *the*
Noun	→ *house* | *dog*

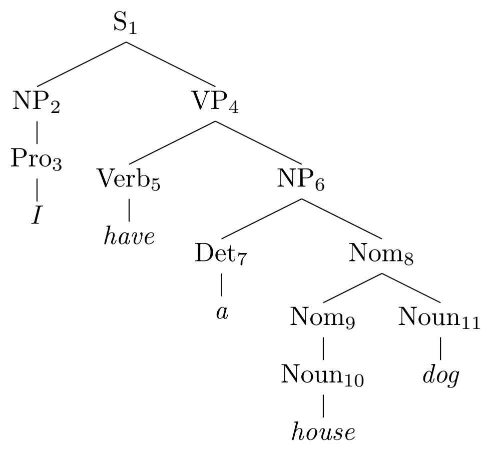
(b) Lexicon

Figure 1: Simple CFG

A CFG can be thought of in two ways. On one hand, as a system to generate sentences by application of rules in a sequence. For example symbol NP can be rewritten to *a dog* by a series of rule expansions called derivation:

$$\text{NP} \rightarrow \text{Det Nom} \rightarrow a\ \text{Nom} \rightarrow a\ \text{Noun} \rightarrow a\ dog$$

On the other hand, a CFG can be used to give structure to already existing sentences. The action of finding a structure given a sentence i.e. finding a parse tree, is called parsing and this latter view will be used in this work. Figure 2a shows an example parse of the sentence "I have a house dog". Figure 2b lists the parsing rules by always expanding the leftmost expandable unit.



(a) Parse tree

$r_1 = S \rightarrow NP\ VP$
$r_2 = NP \rightarrow Pro$
$r_3 = Pro \rightarrow I$
$r_4 = VP \rightarrow Verb\ NP$
$r_5 = Verb \rightarrow have$
$r_6 = NP \rightarrow Det\ Nom$
$r_7 = Det \rightarrow a$
$r_8 = Nom \rightarrow Nom\ Noun$
$r_9 = Nom \rightarrow Noun$
$r_{10} = Noun \rightarrow house$
$r_{11} = Noun \rightarrow dog$

(b) Sequence of rules

Figure 2: Parsing the sentence "I have a house dog" using grammar from Figure 1

Finally, formal definition of a CFG is given. Here, symbols that represent abstraction in the language such as S, NP and Det, that can be expanded, are called non-terminal symbols or non-terminals. Symbols that correspond to words in natural language such as "dog" and "I" are called terminal symbols or terminals. Consequently, parse tree node labeled with a non-terminal or terminal are called non-terminal node or terminal node respectively.

Definition 2.1 *Context-Free Grammar*
Context-free grammar is generated by a 4-tuple (N, n, R, S) where:

- N *- a set of non-terminal symbols*
- $[n]$ *- a set of terminal symbols such that* $N \cap [n] = \emptyset$
- R *- a set of rules of the form* $A \rightarrow \beta$ *where* $A \in N$ *and* $\beta \in (N \cup [n])*$
- $S \in N$ *- the start symbol*

2.1.2 PCFGs

In syntactic parsing, as in many NLP phenomena, ambiguity poses a challenge. A CFG generates structural ambiguity if it allows multiple parse trees to be assigned to a single sentence.

[[*red apples*] *and bananas*]

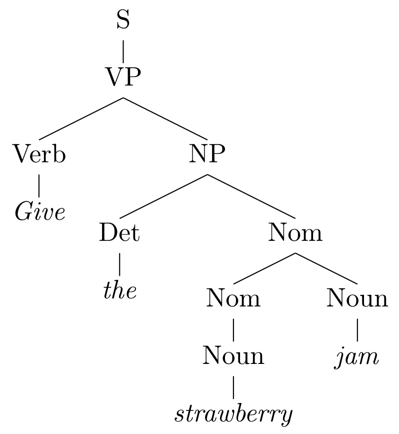
(a) Meaning 1

[*red* [*apples and bananas*]]

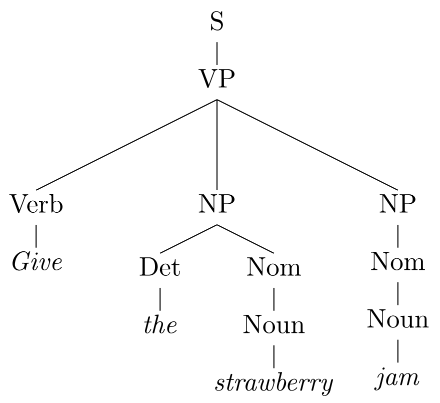
(b) Meaning 2

Figure 3: Two possible meanings for the phrase "red apples and bananas"

Figure 3 shows an example of coordination ambiguity. The meaning outlined on the left specifies the color of apples and the color of bananas remains unknown, whereas the meaning outlined on the right defines both types of fruit to be red. A human reader can immediately identify the first meaning to be more likely since it is not common for bananas to be red.



(a) Parse 1



(b) Parse 2

Figure 4: Two possible parse trees for the sentence "Give the strawberry jam" using grammar from Figure 5

Figure 4 shows an example of attachment ambiguity. The meaning of the left parse is to give the strawberry jam (to someone). The meaning of the right parse on the other hand, is nonsensical and means to give the jam to the strawberry. To a human reader, it might be obvious that the first parse tree is probably the correct one using the semantics (meaning) of the sentences and its context.

To represent the likelihood of a certain parse tree and thus enable the parser to select the most likely parse, conditional probability is assigned to each rule resulting in probabilistic CFGs. Definition of PCFG is as outlined in Figure 5 with each rule $r = A \rightarrow \beta$ being assigned the probability $p(r) = p(A \rightarrow \beta | \ A)$. Probability of a parse tree $\tau = (r_1, r_2, r_3 ... r_N)$ is then calculated as:

$$p(\tau) = p(r_1, r_2 ... r_N) = p(r_1)p(r_2)...p(r_N) \tag{1}$$

Note that here we assume that there is only one possible root node S.

S	$\rightarrow$ VP [0.05]
VP	$\rightarrow$ Verb NP [0.20]
VP	$\rightarrow$ Verb NP NP 0.10
NP	$\rightarrow$ Det Nom [0.20]
NP	$\rightarrow$ Nom [0.15]
Nom	$\rightarrow$ Nom Noun [0.20]
Nom	$\rightarrow$ Noun [0.75]
Verb	$\rightarrow$ *Give* [0.30]
Det	$\rightarrow$ *the* [0.60]
Noun	$\rightarrow$ *strawberry* [0.10]
Noun	$\rightarrow$ *jam* [0.40]

Figure 5: Context-free grammar (incomplete)

Using grammar from Figure 5, probabilities of the left and right parse from Figure 4a are:

$$p(left) = 0.05 \times 0.20 \times 0.30 \times 0.20 \times 0.60 \times 0.20 \times 0.75 \times 0.10 \times 0.40 = 2.16 \times 10^{-6}$$
$$p(right) = 0.05 \times 0.10 \times 0.30 \times 0.20 \times 0.60 \times 0.75 \times 0.10 \times 0.15 \times 0.75 \times 0.40 = 6.075 \times 10^{-7}$$

And thus we formally established that the probability of the left parse is more likely than the probability of the right parse, as expected. In general, the task of a probabilistic syntactic parser given a sentence S is to find the parse tree with the highest probability.

2.1.3 L-PCFGs

In this section the need for latent states is explained and Latent-Variable PCFG is formally defined.

Even though PCFG is quite a powerful language formalism there is some information it is unable to capture. CFG imposes probabilistic independence assumption on rules which results in insufficient representation of structural dependencies in a parse tree. Moreover, CFG models don't represent syntactic facts about terminal symbols (words) which could help resolve certain ambiguities.

To improve the representation of structural and lexical dependencies, each non-terminal is "assigned" an additional variable to represent further properties such as those mentioned above. Capturing such properties then leads to improved parsing accuracy. Finally, the formal definition of L-PCFG formalism is given as defined by Cohen et al. (2012).

Definition 2.2 *Latent-Variable Probabilistic Context-Free Grammar*
An L-PCFG is an 8-tuple (N, I, P, m, n, t, q, π) where:

- *N - a set of non-terminal symbols (non-terminals)*
- *$I \subset N$ - a set of in-terminals*

- *$P \subset N$ - a set of pre-terminals such that $N = I \cup P$ and $I \cap P = \emptyset$*
- *$[n]$ - a set of terminal symbol*
- *$[m]$ - a set of possible hidden states*
- *For all $a \in I$; $b,\ c \in N$ and $h_1,\ h_2,\ h_3,\ \in [m]$, there is a (binary) rule $a(h_1) \rightarrow b(h_2)\ c(h_3)$ with an associated parameter $t = p(a(h_1) \rightarrow b(h_2)\ c(h_3)|a(h_1))$*
- *For all $a \in P$, $h \in [m]$, $x \in [n]$, there is a (terminal) rule $a(h) \rightarrow x$ with an associated parameter $q = p(a(h) \rightarrow x|a(h))$*
- *For all $a \in I$ and $h \in [m]$ there is parameter $\pi =$ probability of $a(h)$ being the root*

Note that in definition 2.2 pre-terminals are simply part-of-speech (POS) tags, terminals are possible words and the remaining symbols are in-terminals. Note that all in-terminals can act as roots with different probabilities in contrast to definition 2.1 of CFGs in which there was a designated start symbol S.

Also, note that this definition in contrast to 2.1 defines a CFG in Chomsky Normal Form (definition 2.3) which is required for the spectral learning algorithm. For example parse tree in Figure 4a is in Chomsky normal form but parse tree in Figure 4b is not since it contains the production rule VP $\rightarrow$ Verb NP NP.

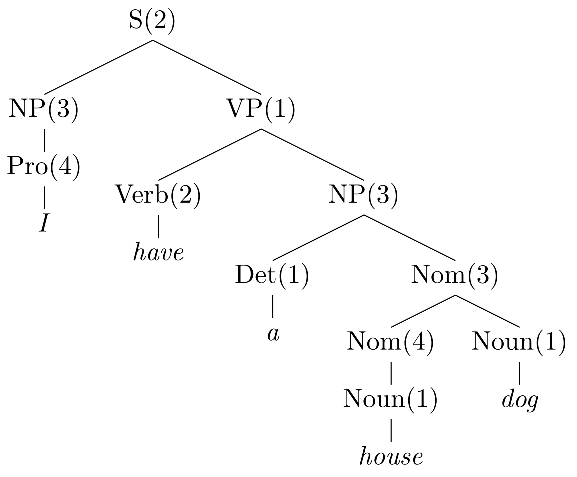
Definition 2.3 *Chomsky Normal Form (CNF)*
A parse tree is in Chomsky normal form if the right-hand side of each production rule is either a single terminal symbol or two non-terminal symbols.

Finally, the distinction between a skeletal and full parse tree is defined (definition 2.4) as outlined by Cohen et al. (2012). In simple terms, full trees are skeletal trees with latent-states associated to their nonterminal nodes. An example of an s-tree is given in Figure 2 and an example of a full parse tree is given in Figure 6.

Definition 2.4 *Skeletal tree (s-tree) and full tree*
Skeletal and full tree are defined as the following:

- *A skeletal tree is a sequence of rules $r_1, r_2, ...r_N$ where each r_i is either of the form $a \rightarrow b\ c$ or $a \rightarrow x$. The rule sequence forms a top-down, lef-most derivation under a CFG.*
- *A full tree consists of an s-tree $r_1...r_N$ with associated values $h_1...h_N$. Each h_i is the value for the latent state for the left hand side of the rule r_i. Each h_i can take any value in $[m]$.*

Note that each nonterminal has a separate set of latent states associated to it. Therefore, for example latent state 3 associated to both NP and VP is not related.



(a) Full tree - Parse tree annotated with latent states

$$
\begin{aligned}
r_1 &= \text{S} \rightarrow \text{NP VP} \\
r_2 &= \text{NP} \rightarrow \text{Pro} \\
r_3 &= \text{Pro} \rightarrow \textit{I} \\
r_4 &= \text{VP} \rightarrow \text{Verb NP} \\
r_5 &= \text{Verb} \rightarrow \textit{have} \\
r_6 &= \text{NP} \rightarrow \text{Det Nom} \\
r_7 &= \text{Det} \rightarrow \textit{a} \\
r_8 &= \text{Nom} \rightarrow \text{Nom Noun} \\
r_9 &= \text{Nom} \rightarrow \text{Noun} \\
r_{10} &= \text{Noun} \rightarrow \textit{house} \\
r_{11} &= \text{Noun} \rightarrow \textit{dog}
\end{aligned}
$$

(b) Sequence of (skeletal) rules

$$
\begin{aligned}
h_1 &= 2 \\
h_2 &= 3 \\
h_3 &= 4 \\
h_4 &= 1 \\
h_5 &= 2 \\
h_6 &= 3 \\
h_7 &= 1 \\
h_8 &= 3 \\
h_9 &= 4 \\
h_{10} &= 1 \\
h_{11} &= 1
\end{aligned}
$$

(c) Sequence of latent states

Figure 6: Full parse tree of the sentence "I have a house dog"

2.2 Spectral Learning Algorithm

Now that L-PCFGs are defined, the description of the spectral learning algorithm is given as proposed by Narayan and Cohen (2015) to extract L-PCFG from a corpus of parsed sentences. It is important to note that the latent states are assigned in this algorithm and are not present in the original corpus. Firstly, an important concept of inside and outside tree is defined.

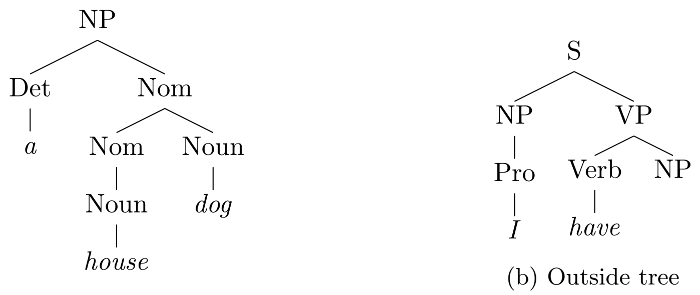
Definition 2.5 *Inside and Outside Tree*
Given a parse tree τ and a node $\nu \in \tau$, τ can be split in the following way:

- *Inside tree denoted t contains the entire subtree below ν including ν*

- *Outside tree contains everything in τ excluding t but including ν*

Given a CFG, we denote the space of inside trees T and the space of outside trees O.

The example sentence parse "I have a house dog" from Figure 2a is split into an inside and outside tree at node NP_6 in Figure 7.



(a) Inside tree

(b) Outside tree

Figure 7: Tree from Figure 2a split at node NP_6

Algorithm Inputs

1. A corpus of parsed sentences P with all non-terminal nodes in each parse tree transformed into training examples in the following way:

   Let $\tau \in P$ be a parse tree and ν be one of the non-terminal nodes of τ. Then the corresponding training example of ν is of the form (t, o):

   - t - inside tree of τ
   - o - outside tree of τ

   Moreover, for all $a \in N$ define S^a as the set of all training examples such that a is the non-terminal symbol of the corresponding parse tree node.

2. Feature functions of inside and outside trees:

   - $\phi : T \to \mathbb{R}^d$ - inside tree feature function maps an inside tree t to a vector $\phi(t) \in \mathbb{R}^d$
   - $\psi : O \to \mathbb{R}^{d'}$ - outside tree feature function maps an outside tree o to a vector $\psi(o) \in \mathbb{R}^{d'}$

3. An integer k denoting the SVD rank.

4. An integer m denoting the number of latent states.

Algorithm Steps

1. Singular Value Decomposition
   - For all $a \in N$ calculate:

$$\Omega^a = \frac{1}{|S^a|} \sum_{(t,o)\in S^a} \phi(t)\psi(o)^T \in \mathbb{R}^{d\times d'} \tag{2}$$

   - For all $a \in N$ calculate SVD of rank k on Ω^a.
   - Obtain:
     - $U^a \in \mathbb{R}^{k\times d}$ as a matrix of the left singular vectors of Ω^a corresponding to the k largest singular values
     - $V^a \in \mathbb{R}^{k\times d'}$ as a matrix of the right singular vectors of Ω^a corresponding to the k largest singular values
2. Projection
   For all $a \in N$:
   - For all training examples $(t, o) \in S^a$ compute:
     - $y = U^a\phi(t)$
     - $z = V^a\psi(o)$
     - Set x to be the concatenation of y and z
   - Define Z^a as the set of vectors x from training examples in S^a
3. Cluster Projections and Tree Annotation
   - For all $a \in N$ cluster the set Z^a to m clusters using k-means clustering
   - Obtain a clustering function $\gamma : \mathbb{R}^{2k} \to [m]$ that maps a projected vector x to a cluster in $[m]$
   - Annotate each node in corpus P with $\gamma(x)$ i.e. the latent state of the corresponding non-terminal node

Algorithm Outputs Compute the following to obtain the final parameters:

- Probability of binary rules $p(a(h_1) \to b(h_2)\ c(h_3)|a(h_1))$ as the relative frequency of its appearance in the annotated corpus:

$$p(a(h_1) \to b(h_2)\ c(h_3)|a(h_1)) = \frac{|a(h_1) \to b(h_2)\ c(h_3)|}{|a(h_1)|} \tag{3}$$

- Probability of terminal rules $p(a(h) \to x|a(h))$ as the relative frequency of its appearance in the annotated corpus:

$$p(a(h_1) \to b(h_2)\ c(h_3)|a(h_1)) = \frac{|p(a(h) \to x|a(h))|}{|a(h)|} \tag{4}$$

- Probability of an annotated in-terminal symbol $a(h)$ being a root as the relative frequency of it being a root in the whole corpus.

The details of how the algorithm is implemented in this project are given in section 3.

2.3 Singular Value Decomposition

In this section the intuition behind using singular value decomposition in step 1 of spectral learning algorithm is explained. SVD provides a way to approximate a matrix with one of lower rank.

Definition 2.6 *Singular value decomposition*
Suppose $\Omega \in \mathbb{R}^{d \times d'}$ *has rank*$(\Omega) = r$. *Then we can write*

$$\Omega = U \Sigma V^T$$

where the columns of $U \in \mathbb{R}^{d \times r}$ *and* $V \in \mathbb{R}^{d' \times r}$ *are orthonormal, and* $\Sigma \in \mathbb{R}^{r \times r}$ *is zero everywhere except for entries on the main diagonal, where the* (j, j) *entry is* σ_j, *for* $j = 1...r$ *and* $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r \geq 0$.
The diagonal entries $\sigma_1...\sigma_r$ *are called the singular values of* Ω. *The columns* $u_1, ..., u_r$ *of* U *are the left singular vectors and the columns* $v_1, ..., v_r$ *of* V *are the right singular vectors.*

Now we have

$$\Omega = U\Sigma V^T = \begin{bmatrix} | & | & & | \\ u_1 & u_2 & \dots & u_r \\ | & | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_r \end{bmatrix} \begin{bmatrix} — & v_1^T & — \\ — & v_2^T & — \\ & \vdots & \\ — & v_r^T & — \end{bmatrix} \tag{5}$$

Multiply $U\Sigma$ to obtain

$$\Omega = \begin{bmatrix} | & | & & | \\ \sigma_1 u_1 & \sigma_2 u_2 & \dots & \sigma_r u_r \\ | & | & & | \end{bmatrix} \begin{bmatrix} — & v_1^T & — \\ — & v_2^T & — \\ & \vdots & \\ — & v_r^T & — \end{bmatrix} \tag{6}$$

Which can be written as

$$\Omega = \sum_{j=1}^{r} \sigma_j u_j v_j^T \tag{7}$$

The expression in equation 7 is called the dyadic form of SVD in which each term of the sum is a rank-1 matrix. Because singular values $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r \geq 0$ are ordered in descending order, the leading terms of this sum dominate the partial sum. Therefore, $\sum_{j=1}^{k} \sigma_j u_j v_j^T$ with $k < r$ will give a good lower-rank approximation of Ω. This corresponds to step 1 of spectral learning algorithm where we assign U^a and V^a to be the k largest left and right singular vectors of Ω^a respectively. Consequently, U^a and V^a are used to project feature functions of inside and outside trees down to a lower-dimensional vector. These vectors then form a suitable input for k-means clustering.

Intuition behind SVD As Narayan and Cohen (2015) described, in the spectral learning algorithm (section 2.2) Ω^a is an empirical estimate for the cross-covariance matrix between the inside and outside trees of a given nonterminal a. Under the L-PCFG model, the inside and outside tree are independent given the latent state at their connecting point which means that latent state can be defined by finding patterns in inside and outside trees that occur together. Therefore by reducing the dimensions of this matrix by SVD, we get the representation that corresponds to the latent states.

2.4 Parsing with Latent States

In this section it is outlined how latent states are treated when syntactic parsing is performed. For a given sentence $x = x_1...x_N$ a syntactic parser is used to find the most likely skeletal parse tree

$$\widehat{\tau}(x) = \underset{x=yield(\tau)}{\arg\max}\; p(\tau)$$

To do this, given an L-PCFG, latent states are marginalized over in these two central calculations:

- For a given s-tree $r_1...r_N$, calculate its probability by $\sum_{h_1...h_N} p(r_1...r_N, h_1...h_N)$
- For a given input sentence $x = x_1...x_N$, calculate the marginal probabilities of nonterminal a spanning $x_i...x_j$ by $\mu(a, i, j) = \sum_{x=yield(\tau):(a,i,j)\in\tau} p(\tau)$

These are calculated by variants of inside and outside dynamic algorithms as described by Cohen et al. (2012). These are further used in a syntactic parsing algorithm by Goodman (1996).

3 Implementation

In this section we give details on how the algorithm was implemented and how important decisions were made.

3.1 Choice of Projection Functions

The second part of the input to the spectral algorithm are projection functions $\phi(t)$, $\psi(o)$ of inside and outside trees (see Figure 7) respectively. The goal is to define a set of features that provide sufficient information about latent-state values by providing a good estimate for cross-covariance between the inside and outside trees for each nonterminal. Cohen et al. (2013) uses a set of 7 features for inside tree function and a set of 7 features for the outside tree function. However, results of their experiments show that a more compact subset of the features also achieves a relatively high accuracy. Therefore, for simplicity, we select the following subset of 4 features for each projection function.

Inside tree features For inside tree feature function $\phi(t)$, assume that binary rule $a \rightarrow b\ c$ or alternatively terminal rule $a \rightarrow x$ is the root rule of the inside tree t. Examples are given for inside tree in Figure 7a.

- The production rule $a \rightarrow b\ c$ or $a \rightarrow x$ e.g. NP $\rightarrow$ Det Nom
- Nonterminal b (empty if terminal rule) e.g. Det
- Nonterminal c (empty if terminal rule) e.g. Nom
- Number of terminal nodes spanned by the root nonterminal a e.g. 3

Outside tree features For outside tree feature function $\psi(o)$, the root that was used to split the original parse tree is called the foot note. Examples are given for outside tree in Figure 7b.

- The parent of the foot note (if any) e.g. VP
- The grandparent of the foot note (if any) e.g. S
- The rule one level above foot note (if any) e.g. VP $\rightarrow$ Verb NP
- The rule two levels above foot note (if any) e.g. S $\rightarrow$ NP VP

For each feature, a one-dimensional vector is created with size of the number of possibilities for the given feature. It is set to zero everywhere except at the position that corresponds to the value to represent where it is set to one. Inside and outside tree feature functions are defined as concatenations of all corresponding feature vectors.

3.2 Programming Language, Libraries and Corpus

Programming Language The software is implemented in Python 3.6.4. using object-oriented design. This programming language was chosen because of its scientific suitability and many useful libraries available.

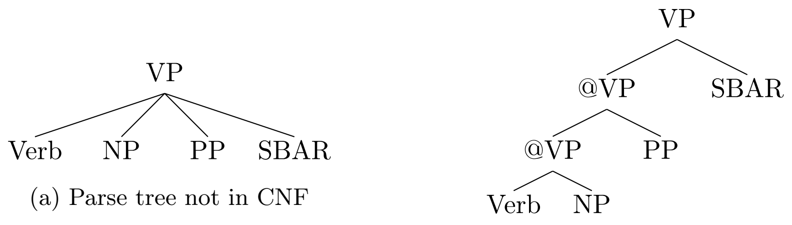
Libraries The following Python libraries are utilised:

- Natural Language Toolkit (NLTK) provides an essential backbone for the software. Classes such as Tree and Nonterminal are heavily utilised.
- Scientific Python (SciPy) and its methods for sparse matrices are crucial for time feasibility when dealing with large and sparse matrices that were abundant in this project.
- Numerical Python (NumPy) is used for manipulation of dense vectors and matrices.
- Scikit-learn (sklearn) is a python machine learning library and its implementation of k-means clustering is utilised.

Corpus For experiments, Wall Street Journal (WSJ) sections of the Penn Treebank (Marcus et al., 1993) is used. It has 25 sections and for the first and second experiment, sentences from sections 0 - 22 are used for development and testing in random order. In the third experiment, sections 1 - 21 are used for training and section 22 is used for testing. The treebank has 37015 parsed sentences altogether.

3.3 Conversion to Chomsky Normal Form

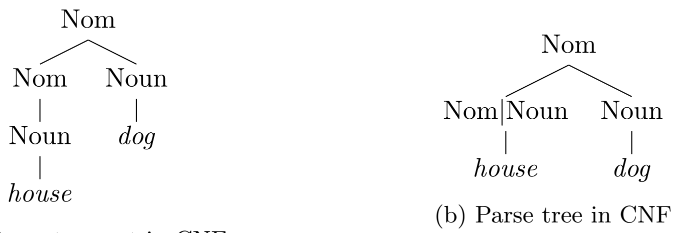
As mentioned in section 2.1.3, L-PCFGs are defined over CFG rules in Chomsky normal form (definition 2.3). However naturally, rules often occur in a non-CNF form. Therefore before training, rules are normalised in the same way as described in Petrov et al. (2006). Figures 8 and 9 show examples of how this is done. Figure 10 shows an example of normalisation of a tree from Penn WSJ treebank.



(a) Parse tree not in CNF

(b) Parse tree in CNF

Figure 8: Conversion to CNF by addition of nonterminal @VP



(a) Parse tree not in CNF

(b) Parse tree in CNF

Figure 9: Conversion to CNF by merging of two nonterminals to Nom|Noun

```
                   TOP
                    |
                    S
       _____________|_____________________________
      |             VP                            |
      |       ______|____________                 |
   NP-SBJ  |   ADVP-TMP       NP-PRD              |
      |    |      |         _____|________        |
     PRP  VBD     RB      NN             NN       .
      |    |      |        |              |       |
     He   was previously vice         president   .
```

(a) Parse tree not in CNF

```
                              S^g
                  ____________|___________________
              @S^g                                |
   _______________|______                         |
  |                     VP^g                      |
  |             _____________|_________           |
  |        @VP^g                    NP^g          |
  |      ____|_______           ____|_______      |
NP^g|PRP VBD      ADVP^g|RB     NN          NN    .
  |       |           |          |           |    |
  He     was      previously   vice    president  .
```

(b) Parse tree in CNF

Figure 10: Conversion to CNF example tree from Penn WSJ treebank

All parse trees are normalised using scripts provided by Narayan and Cohen (2017). As a result of the definition of L-PCFG sentences of length one can not be parsed.

3.4 Parsing

Parsing algorithm using grammars with latent states as described in section 2.4 was implemented as a part of Rainbow Parser (Narayan and Cohen, 2017) which is reused for this project.

Rare Words Cutoff In any language there is large number of words and it is not possible to encounter all of them during the training phase. Therefore it is essential to account for this. Similarly as in Narayan and Cohen (2017), rare words are replaced by their POS tag before training and the following vocabulary threshold is selected for all experiments:

$$Vocabulary\ threshold = 5$$

Before spectral learning a vocabulary of all words is created with their counts and those that have a count lower than the threshold are replaced by corresponding POS tags. The same vocabulary is used during parsing and before parsing is performed, words that are below threshold are replaced by the corresponding POS tags.

Pruning Threshold Pruning threshold is the minimum probability of a rule with latent states to be used by the parser. Any rule with lower probability will be replaced by its pruned version with no latent states in the parsing algorithm. As in Cohen et al. (2013), the following pruning threshold is selected for all experiments:

$$\mathit{Pruning\ threshold} = 0.00005$$

3.5 Evaluation Metrics

In this section the metric used for evaluation of the grammar is described. Given a candidate parse and the correctly annotated solution (gold standard), the metric evaluates the conceptual distance between the two. The standard metric is parseval (Black et al., 1991) which is also used by Cohen et al. (2013). It calculates constituency differences between two parse trees where constituent is any complete subtree of a given tree.

The following measures are then defined for each candidate parse tree and the corresponding gold standard. Note that by convention, unary constituents are excluded from the measures.

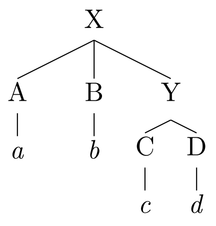
- Precision: Number of correct constituents out of all constituents in the candidate parse tree.
- Recall: Number of correct constituents out of all constituents in the gold standard.
- F_1 measure: Harmonic mean of precision and recall.

$$F_1 = 2 \times \frac{\mathit{Precision} \times \mathit{Recall}}{\mathit{Precision} + \mathit{Recall}} \times 100$$

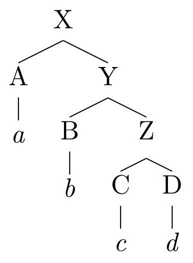
F_1 measure for the whole corpus of candidate parses is the average of all F_1 measures. Moreover, two variants of the measure are used:

- Unlabeled: The yield of each constituent is considered.
- Labeled: The yield of each constituent together with its root label is considered.

As default, the unlabeled scores are used as also done in Cohen et al. (2013). For some experiments, labeled scores are used for comparison. To illustrate the calculation the following artificial example is given.



(a) Candidate parse tree



(b) Gold standard parse tree

$$Y : cd$$
$$X : abcd$$

(c) Candidate constituents

$$Z : cd$$
$$Y : bcd$$
$$X : abcd$$

(d) Gold standard constituents

Figure 11: Parse trees of the artificial sentence "a b c d"

Figure 11 shows two different parses for artificial sentence "a b c d" and the corresponding constituents. For unlabeled constituents we get the following parseval metrics:

$$\begin{aligned}
&Precision = \frac{2}{2} = 1 \\
&Recall = \frac{2}{3} \\
&F_1 = 2 \times \frac{1 \times \frac{2}{3}}{1 + \frac{2}{3}} \times 100 = 80
\end{aligned}$$

For labeled constituents we get the following, lower, parseval metrics:

$$\begin{aligned}
&Precision = \frac{1}{2} \\
&Recall = \frac{1}{3} \\
&F_1 = 2 \times \frac{\frac{1}{2} \times \frac{2}{3}}{\frac{1}{2} + \frac{2}{3}} \times 100 = 40
\end{aligned}$$

Note that for parse trees in CNF we always have $Precision = Recall = F_1$. In our experiments, the implementation of parseval by Sekine and Collins (2013) is used.

3.6 Pipeline

Finally, in this section the overall structure and dependencies of the project pipeline reiterated, summarised and visualised.

Pipeline steps To extract and evaluate a grammar using the spectral learning algorithm the following steps are performed:

1. **Convert parse trees:** Given a corpus of parsed sentences such as WSJ Penn treebank, all sentences are transformed to CNF which is required by the definition of L-PCFGs. For this we use a script included in Rainbow Parser (Narayan and Cohen, 2017). The corpus, now in CNF, is then used to select sentences for training and testing (gold standard). Furthermore, POS tagged test sentences are extracted to be used later as an input to parser.

2. **Spectral learning:** This is the fundamental part of the pipeline during which the training sentences are annotated with latent states. The input to spectral learning algorithm are training sentences, feature functions, number of latent states and vocabulary threshold. Note that by convention, we use the same value for number of latent states (m) and the SVD rank (k). Moreover, vocabulary with word counts is generated.

3. **Calculate relative frequencies:** In this step, grammar is generated using the annotated corpus from the previous step. Relative frequencies of rules and roots of the grammar are calculated, producing the grammar file for parsing. Pruned version of the grammar with no latent states is created as well.

4. **Parse test sentences:** In this step the generated grammar from the previous step is used to parse test sentences to measure how well it performs. The input to syntactic parser is the grammar, pruned grammar, POS tagged test sentences and vocabulary with word counts. Furthermore, the parser requires vocabulary and pruning threshold. The output is a set of parsed test sentences to be evaluated. The Rainbow Parser by Narayan and Cohen (2017) is used for this step.

5. **Calculate F_1 measure:** Finally, the parsing accuracy is evaluated by comparing the gold standard and candidate parsed sentences to produce the mean F_1 measure. This measure is used to evaluate the quality of the generated grammar. The parseval implementation of Sekine and Collins (2013) is utilised for this step.

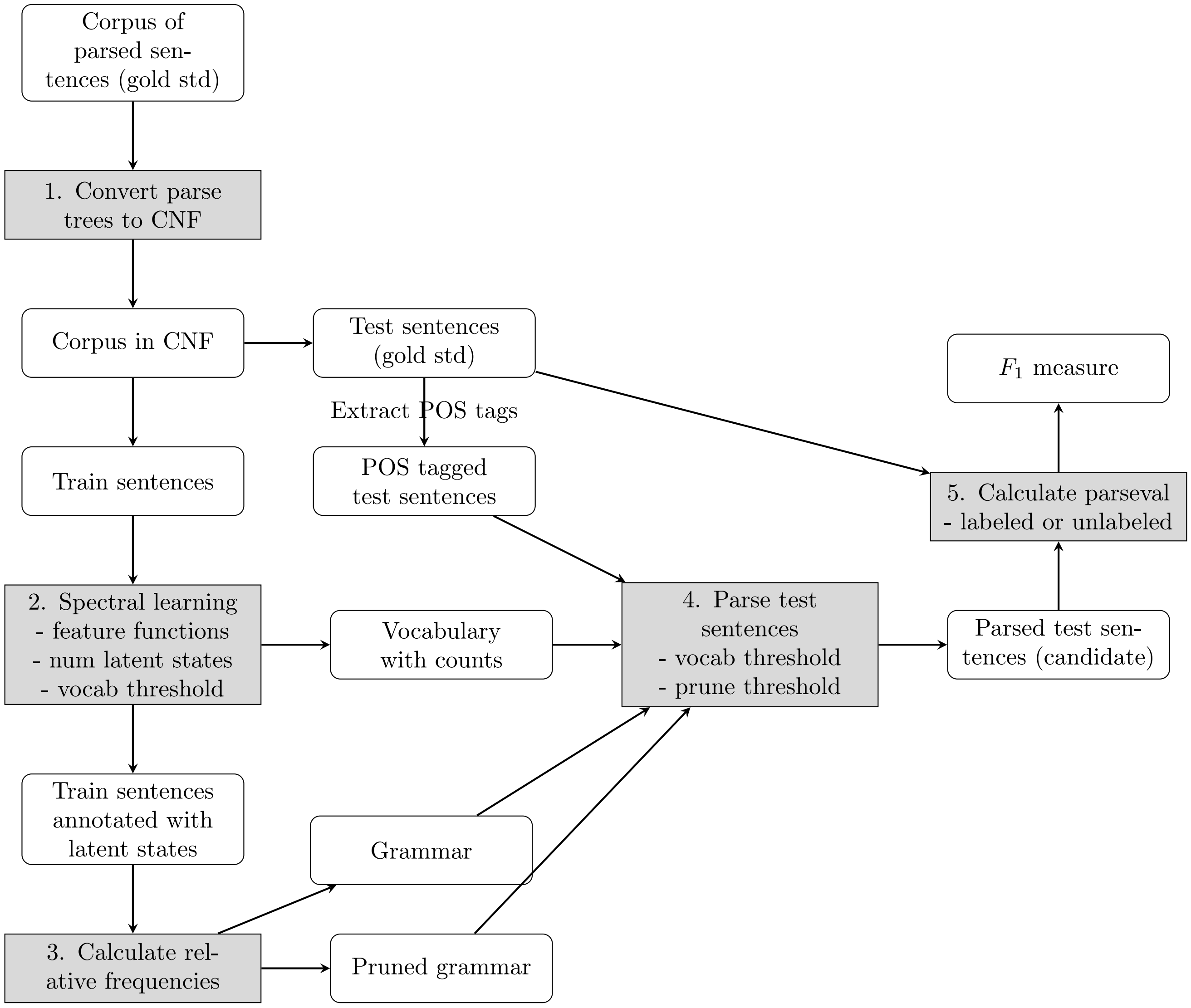

Corpus of parsed sentences (gold std)
1. Convert parse trees to CNF
Corpus in CNF
Test sentences (gold std)
Extract POS tags
POS tagged test sentences
Train sentences
2. Spectral learning - feature functions - num latent states - vocab threshold
Vocabulary with counts
4. Parse test sentences - vocab threshold - prune threshold
Parsed test sentences (candidate)
5. Calculate parseval - labeled or unlabeled
F_1 measure
Train sentences annotated with latent states
Grammar
3. Calculate relative frequencies
Pruned grammar

Figure 12: Flowchart of the project pipeline

Note here, that POS tags are extracted directly from the parsed sentences (gold standard) for simplicity. In reality the POS tags would need to be assigned by a POS tagger which would in return decrease the parsing accuracy. However, the current state of art for POS tagging is a little over 97% (Manning, 2011) and so the difference in parsing accuracy should not be significant. This is further discussed in section 5.3.

4 Experiments

In this section, the experiment setup and results are given to test our implementation of spectral learning algorithm and to demonstrate various properties of L-PCFGs and their usefulness.

For the first two experiments, sentences for training and testing are selected randomly from sections 0 - 22 of WSJ Penn treebank. For the final experiment sections 1 - 21 of WSJ Penn treebank are used for training and section 22 is used for testing. For the first two experiments, the accuracy of the resulting grammar is tested on 1 thousand sentences, training and testing sentences are selected randomly and experiment is repeated 3 times for each experiment setup. Full results are attached in section 7.

4.1 Size of Training Corpus and Accuracy

In this experiment we explore what the effect of the size of the training corpus on the parsing accuracy is. The hypothesis is that the larger training set should result in a grammar with greater parsing accuracy (F_1 measure). SVD rank (which is equal to number of latent states) is selected to be 8, since in Cohen et al. (2013) this has shown to be the lowest SVD rank that leads to good accuracy.

4.1.1 Experimental Setup

In this experiment the following corpus sizes are used:

- Size of training corpus: 5, 10, 15, 20, 25 and 30 thousand sentences

The following parameters are fixed throughout the experiment:

- Feature functions: All features used
- Number of latent states (m), SVD rank (k): 8
- Vocabulary threshold: 5
- Number of test sentences: 1000
- Pruning threshold: 0.00005
- Parseval metric: Unlabeled F_1 measure.

The experiment is run 3 times for each setup and the resulting F_1 measures and training times are averaged. All results can be found in section 7.1.

4.1.2 Results

Note that for no latent states the pruned version of the grammar was used for parsing. Therefore we indicate that the training time is the same for both 8 and no latent states.

Size [sents]	F_1 measure 8 latent states	F_1 measure No latent states	Time [Hours]
5000	69.29	66.36	0.5
10000	77.36	67.73	1.55
15000	77.41	68.43	2.26
20000	81.24	69.69	3.28
25000	79.71	68.46	3.75
30000	81.57	69.23	5.54

Figure 13: Size of training corpus with mean accuracy and time

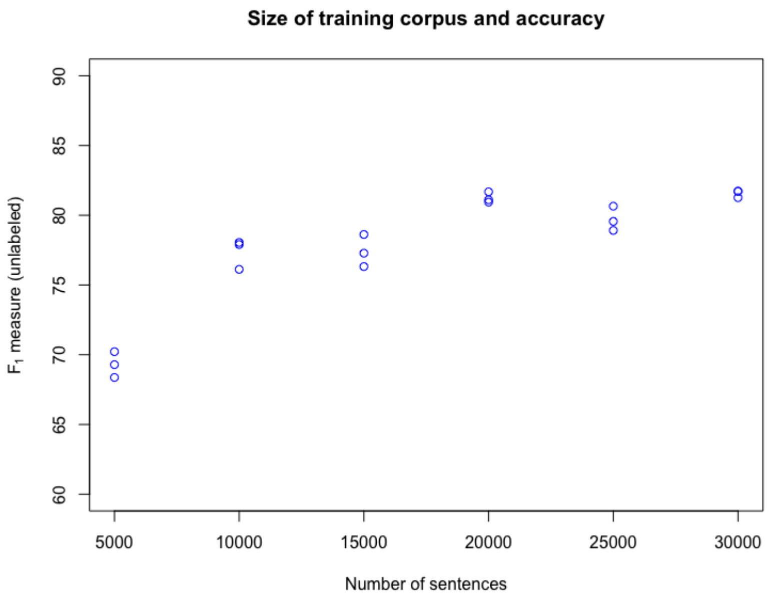


Figure 14

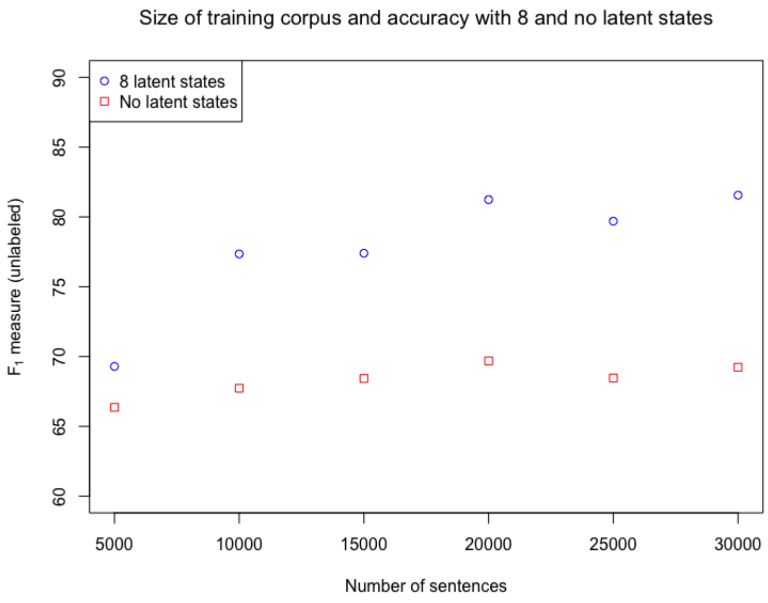


Figure 15

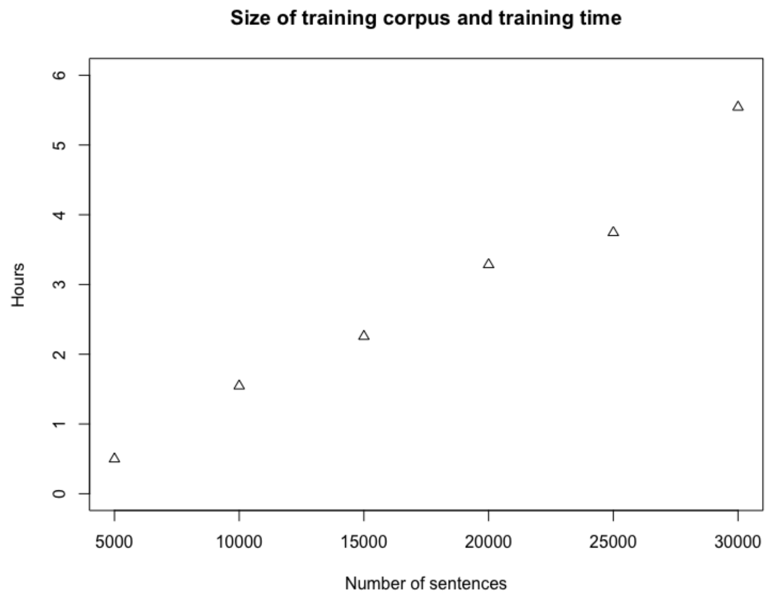


Figure 16

4.2 Number of Latent States and Accuracy

In this experiment we explore the effect of the number of latent states on the parsing accuracy. The hypothesis is that greater number of latent states should record more correlations between hidden variables. However it might be that high number of latent states will result in a grammar that is too sparse to parse accurately.

4.2.1 Experimental Setup

In this experiment the following numbers of latent states are used:

- Number of latent states (m), SVD rank (k): 1, 2, 4, 8, 16, 24 and 32

The following parameters are fixed throughout the experiment.

- Feature functions: All features used
- Vocabulary threshold: 5
- Size of training corpus: 10000 sentences
- Number of test sentences: 1000
- Pruning threshold: 0.00005
- Parseval metric: Unlabeled and labeled F_1 measure.

The experiment is run 3 times for each setup and the resulting F_1 measures and training times are averaged. All results can be found in section 7.2.

4.2.2 Results

	F_1 measure	F_1 measure	F_1 measure	
Latent states	Unlabeled F_1	Labeled F_1	Unlabeled - Labeled	Time [Hours]
1	68.38	60.88	7.5	1.51
2	71.19	64.03	7.16	1.21
4	76.36	69.94	6.42	1.4
8	78.52	72.3	6.22	1.81
16	75.82	69.07	6.75	1.43
24	65.47	55.61	9.86	1.78
32	65.17	55.82	9.35	1.94

Figure 17: Number of latent states with mean F_1 measures and time

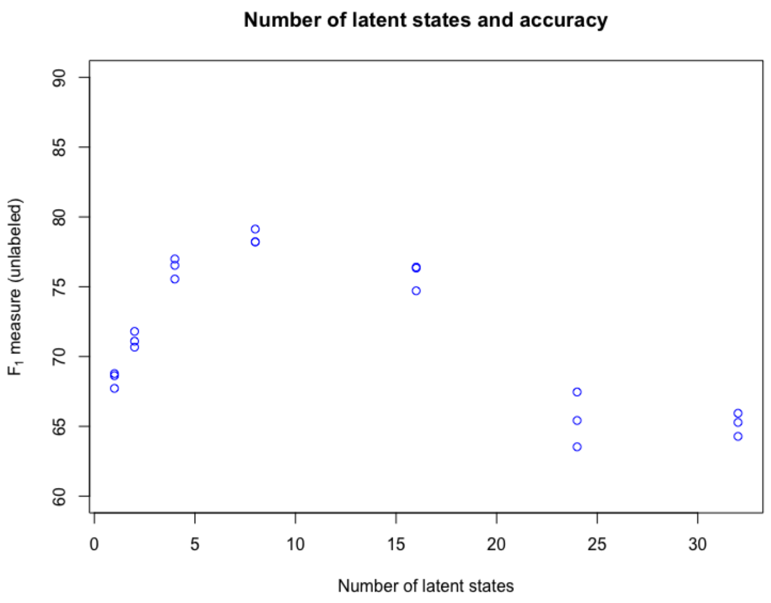


Figure 18

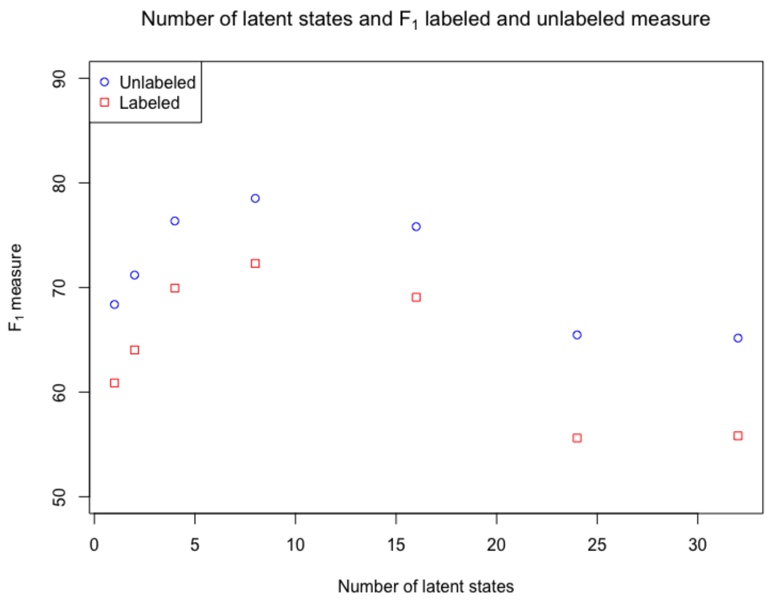


Figure 19

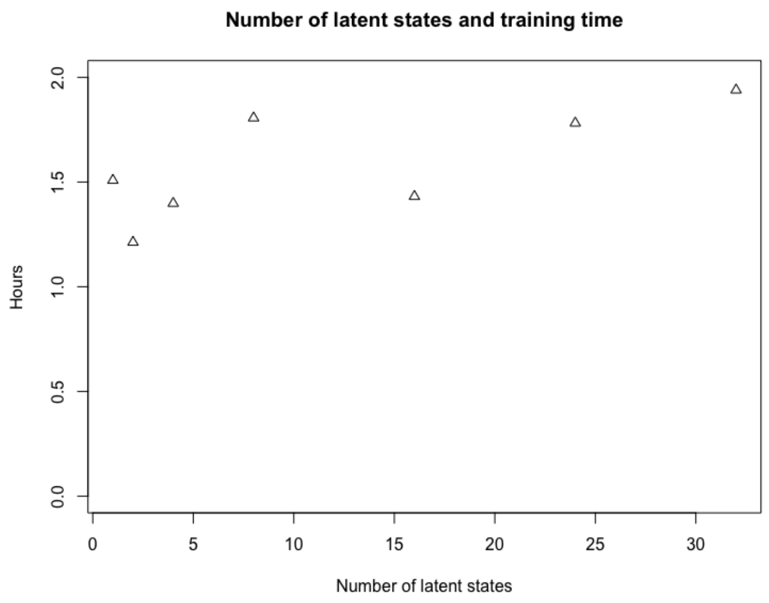


Figure 20

4.3 Comparison to Previous Implementation

In this section we compare our implementation of spectral algorithm (*Spectral*^{m}) to previous implementation by Cohen et al. (2013) (*Spectral*^{c}). The hypothesis is that *Spectral*^{m} will not achieve lower accuracy because of multiple differences between the two implementations. These are discussed further in section 5. Note that resulting values for *Spectral*^{c} implementation are taken directly from the paper (Cohen et al., 2013).

4.3.1 Experimental Setup

Here we give define experimental setup for *Spectral*^{m} experimental setup. In this experiment the following numbers of latent states are used:

- Number of latent states (m), SVD rank (k): 8, 16, 24 and 32

The following parameters are fixed throughout the experiment:

- Feature functions: All features used
- Vocabulary threshold: 5
- Size of training corpus: 32358 sentences (sections 1 - 21 of WSJ Penn treebank)
- Number of test sentences: 1334 (section 22 of WSJ Penn treebank)
- Pruning threshold: 0.00005
- Parseval metric: Unlabeled F_1 measure.

4.3.2 Results

Latent states	F_1 measure $Spectral^c$	F_1 measure $Spectral^m$
8	85.6	78.46
16	87.77	77.87
24	88.53	76.15
32	88.82	72.73

Figure 21: Comparison to previous implementation

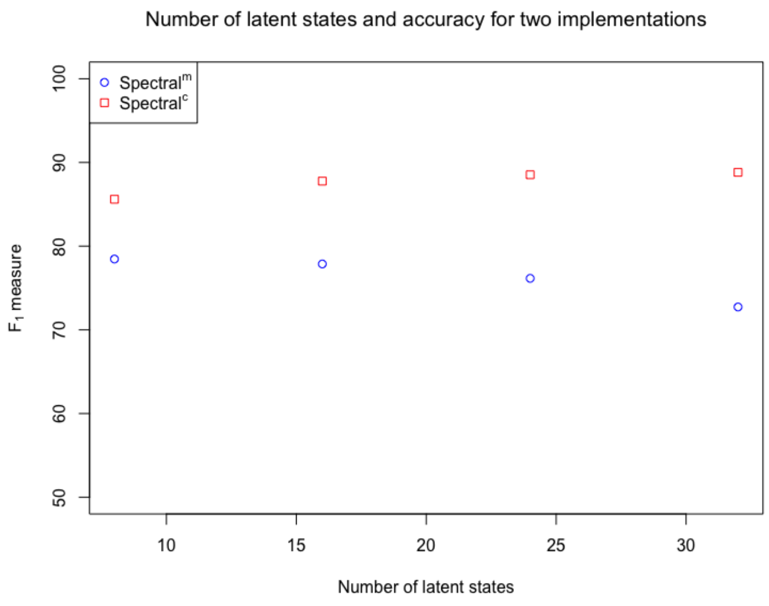


Figure 22

5 Discussion

In this section the analysis of experiments results is given.

5.1 Training Data and Accuracy

In this section the effect of the size of the training corpus on parsing accuracy is analysed.

Observing Figure 13 it appears that the size of training corpus has a significant effect on the resulting accuracy. The lowest accuracy 69.29% was observed for training corpus of 5 thousand sentences. Increasing the number of sentences to 10 thousand led to a significant increase to 77.36%. The highest accuracy was observed for training corpus of size 30 thousand sentences. However corpus of size 20 thousand sentences is enough to achieve a similar accuracy of 81.24%. Finally, observing Figure 14 we can conclude that to achieve a close to optimal accuracy for 8 latent states a training corpus of at least 20 thousand sentences is required.

For all training corpus sizes, the trained grammar with 8 latent states results in greater accuracy than without latent states (see Figure 15). This is as expected and it proves that latent states are of great necessity to achieve high accuracy. Even with increasing training corpus size, with no latent states, accuracy remains below 70% whereas for latent states accuracy goes up to 81.57%. However, it appears that with no latent states, less data, i.e. 10 thousand training sentences, are sufficient for a result that is close to the optimum whereas as mentioned in the previous paragraph, with latent states at least 20 thousand sentences are required.

Mean training time (see Figure 13) ranges from 0.5 hour to 5.5 hours for 5 thousand and 30 thousand sentences in corpus respectively. Observing Figure 16, the relation between number of sentences and training time appears to be linear.

5.2 Number of Latent States and Accuracy

In this section the effect of the number of latent states on parsing accuracy is analysed.

The highest unlabeled F_1 measure of 78.52% (see Figure 17) is observed for 8 latent states. 2, 4 and 16 latent states also result in accuracy that is greater than that for 1 latent state which is equivalent to a regular PCFG model i.e. having no latent states. However, for 24 and 32 latent states we observe worse accuracy than having no latent states at all. This is probably because there is not enough training data and overfitting occurs. Observing trends in Figure 18, we can conclude that for 10 thousand sentences accuracy increases with number of latent states increasing from 1 to about 8 after which it starts to decline. This is probably because the training corpus size of 10 thousand sentences is not sufficient and overfitting occurs. Results from section 4.3 suggest that increasing number of training corpus partly mitigates this but 8 latent states appears to remain the optimal number which is contrasting to results in Cohen et al. (2013) achieving highest accuracies with 32 latent states (see Figure 21).

When comparing unlabeled and labeled F_1 measures (see Figure 17), labeled F_1 measure is lower by 7.5% to 9.35% for all training corpus sizes. This is as expected since matching labeled constituents are harder to achieve than matching unlabeled constituents. Figure 19 suggests that the difference is larger for higher number of latent states.

Finally time doesn't change significantly for various number of latent states (see Figure 20). All training times were within the range of 1.2 to 2 hours. This together with the results from the first experiment suggests that time complexity of this implementation is heavily dependent on the number of training sentences in the corpus rather than the number of latent states.

5.3 Comparison to Previous Implementation

Finally, the implementation is compared to the original implementation.

Both implementations resulted in accuracies above 72%. However, the original $Spectral^c$ implementation achieves higher accuracy than our $Spectral^m$ implementation, for all numbers of latent states.

It is plausible that accuracies of implementations $Spectral^c$ and $Spectral^m$ differ for the following reasons:

1. **Algorithm:** $Spectral^c$ is an implementation of the original spectral algorithm as described in Cohen et al. (2012). $Spectral^m$ implements the updated version of the algorithm as described in Narayan and Cohen (2015) that results in more compact models. Experiments in Narayan and Cohen (2015) show that on its own the latter algorithm performs slightly worse than the original one. However they propose various noising schemes to produce multiple models which are then used for parsing. Using the latter algorithm with noising scheme then leads to results equivalent to results of the original algorithm. In $Spectral^m$ there is no noising scheme implemented, decreasing the accuracy of $Spectral^m$ implementation.

2. **POS tagging:** As mentioned in section 3.6, instead of using a POS tagger, POS tags are extracted directly from the parsed sentences. This fact has two effects on the resulting parsing accuracy. Accuracy of state of art POS taggers is about 97% whereas using POS tags as given by the parsed sentences has by definition accuracy of 100% so this should lead to higher parsing accuracy. On the other hand, reusing POS tags as given by the gold standard, means that some of the POS tags will be unseen in the training data and the grammar will miss the production rules for these unseen POS tags. Any such sentence will then fail to be parsed resulting in lower accuracy. It is therefore questionable which of these two effects is greater. If we used POS tagger as well, our results would be more easily comparable.

3. **Projection features:** $Spectral^c$ implementation uses a set of seven feature functions for each of inside and outside trees whereas in our implementation, only four of these are used for each projection function (see section 3.1). This means that less patterns that occur together between inside and outside trees are identified leading to lower accuracy when estimating latent states.

4. **Feature scaling:** Cohen et al. (2013) introduces a way of feature scaling so that more frequent features are assigned greater value than the less frequent one. Feature scaling is utilised in $Spectral^c$ but it is not implemented in this project. Furthermore, experiments in Cohen et al. (2013) show that feature scaling is an important factor to achieve greater accuracy. They observe that without feature scaling accuracy decreases from 88.82% down to 84.40% which is significant.

$Spectral^c$ implementation achieves highest accuracy of 88.82% with 32 latent states whereas $Spectral^m$ implementation achieves highest accuracy of 78.46% with 8 latent states. Moreover, increasing number of latent states for $Spectral^c$ implementation increases the accuracy whereas for $Spectral^m$ it decreases accuracy (see Figure 22). One possible explanation of this behaviour is third and fourth argument above. Because $Spectral^s$ uses more features, more patterns can be recorded by more latent states. Experiments in Cohen et al. (2013) have not shown this behaviour when feature functions are reduced but perhaps this can occur in combination with abandoning the feature scaling.

6 Conclusion

The main goal of this project was to show that it is possible to implement spectral learning algorithm for L-PCFG as described by Narayan and Cohen (2015) in a single high-level programming language. To do this, we utilised Python and multiple libraries such as nltk, sklearn, numpy and scipy. Because the algorithm is abundant with sparse matrices, the use of scipy.sparse library for manipulation of sparse matrices was crucial and made the algorithm time feasible.

The algorithm was tested with corpora ranging from 5 up to 30 thousand sentences with number of latent states ranging from 2 to 32. The running time was shown to be more dependent on the size of the training corpus rather than number of latent states. For 30 thousand sentences, the algorithm completes in about 6 hours, which shows that the implementation is time feasible even with large training corpora.

Results of the experiments showed that with 8 latent states at least 20 thousand sentences are required to achieve close to optimal accuracy of about 80%. Having no latent states at all resulted in significantly lower accuracies under 70% but only half of the training corpus to achieve close to optimal accuracy. When trained on 10 thousand sentences 8 latent states were shown to result in optimal grammar. Grammars with 24 and 32 latent states performed worse than a PCFG model with no latent states which was likely the result of overfitting.

Our implementation performed worse than that by Cohen et al. (2013) for all numbers of latent states which was linked to multiple differences in the implementation. Experiments show that optimal number of latent states in Cohen et al. (2013) is 32 which is higher than that of 8 in our implementation. Our hypothesis is that this is due to having fewer feature functions and abandoning feature scaling.

To achieve greater validity of our results in comparison to other implementations it would be essential to eliminate the mentioned differences, mainly in the feature functions and their scaling. Moreover, rather than extracting POS tags from parsed sentences, they should be obtained using a POS tagger. Apart from these, our experiments have shown the expected properties in relation to training corpus size and number of latent states, proving the implementation correctness. Furthermore, the pipeline for testing the grammar parsing accuracy was correctly designed and utilised.

References

Black, E., Abney, S., Flickenger, D., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., et al. (1991). A procedure for quantitatively comparing the syntactic coverage of english grammars. In *Speech and Natural Language: Proceedings of a Workshop Held at Pacific Grove, California, February 19-22, 1991.*

Charniak, E. (1997). Statistical parsing with a context-free grammar and word statistics. *AAAI/IAAI*, 2005(598-603):18.

Cohen, S. B., Stratos, K., Collins, M., Foster, D. P., and Ungar, L. (2012). Spectral learning of latent-variable pcfgs. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 223–231. Association for Computational Linguistics.

Cohen, S. B., Stratos, K., Collins, M., Foster, D. P., and Ungar, L. H. (2013). Experiments with spectral learning of latent-variable pcfgs.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38.

Goodman, J. (1996). Parsing algorithms and metrics. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 177–183. Association for Computational Linguistics.

Hsu, D., Kakade, S. M., and Zhang, T. (2012). A spectral algorithm for learning hidden markov models. *Journal of Computer and System Sciences*, 78(5):1460–1480.

Manning, C. D. (2011). Part-of-speech tagging from 97% to 100%: is it time for some linguistics? In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 171–189. Springer.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330.

Narayan, S. and Cohen, S. B. (2015). Diversity in spectral learning for natural language parsing. *arXiv preprint arXiv:1506.00275.*

Narayan, S. and Cohen, S. B. (2017). shashiongithub/rainbow-parser: The rainbow parser. `https://github.com/shashiongithub/Rainbow-Parser`. (Accessed on 03/25/2018).

Petrov, S., Barrett, L., Thibaux, R., and Klein, D. (2006). Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 433–440. Association for Computational Linguistics.

Sekine, S. and Collins, M. J. (2013). Evalb - bracket scoring program. `http://nlp.cs.nyu.edu/evalb/`. (Accessed on 04/03/2018).

7 Appendix

7.1 Size of Training Corpus and Accuracy

Size [sents]	F_1 measure Run 1	F_1 measure Run 2	F_1 measure Run 3	F_1 measure Mean
5000	68.37	68.37	70.22	69.29
10000	76.12	77.9	77.9	77.36
15000	77.36	77.28	77.28	77.41
20000	81.68	81.68	81.68	81.24
25000	78.91	78.91	78.91	78.91
30000	78.91	78.91	78.91	78.91

Figure 23: Size of training corpus with F_1 metric using grammar with 8 latent states

Size [sents]	F_1 measure Run 1	F_1 measure Run 2	F_1 measure Run 3	F_1 measure Mean
5000	65.7	66.96	66.96	66.96
10000	67.26	67.26	67.26	67.26
15000	67.26	68.14	69.16	69.16
20000	69.16	69.39	69.39	69.69
25000	67.68	68.82	68.82	68.82
30000	68.82	70.2	70.2	70.2

Figure 24: Size of training corpus with accuracy using grammar with no latent states

Size [sents]	Time [h] Run 1	Time [h] Run 2	Time [h] Run 3	Mean [h] Mean
5000	0.45	0.47	0.58	0.5
10000	1.56	1.54	1.54	1.55
15000	2.44	2.44	1.88	2.26
20000	2.63	3.12	4.11	3.28
25000	4.23	3.38	3.62	3.75
30000	5.88	4.57	6.18	5.54

Figure 25: Size of training corpus and training time

7.2 Number of Latent States and Accuracy

Latent states	F_1 unlabeled Run 1	F_1 unlabeled Run 2	F_1 unlabeled Run 3	F_1 unlabeled Mean
1	68.78	67.72	68.63	68.38
2	71.8	71.1	70.67	71.19
4	76.53	76.99	75.55	76.36
8	78.22	78.21	79.13	78.52
24	74.71	76.33	76.41	75.82
16	63.53	67.46	65.42	65.47
32	65.28	64.28	65.94	65.17

Figure 26: Number of latent states and unlabeled F_1 measure

Latent states	F_1 labeled Run 1	F_1 labeled Run 2	F_1 labeled Run 3	F_1 labeled Mean
1	61.27	60.05	61.31	60.88
2	64.78	63.98	63.33	64.03
4	70.17	70.57	69.08	69.94
8	72.07	71.79	73.04	72.3
24	67.62	69.48	70.1	69.07
16	53.54	57.02	56.27	55.61
32	56.22	54.61	56.64	55.82

Figure 27: Number of latent states and labeled F_1 measure

Latent states	Time [h] Run 1	Time [h] Run 2	Time [h] Run 3	Mean [h] Mean
1	1.5	1.5	1.52	1.51
2	1.26	1.19	1.19	1.21
4	1.28	1.34	1.57	1.4
8	1.76	2.04	1.61	1.81
24	1.13	0.88	2.28	1.43
16	2.24	1.77	1.33	1.78
32	3.27	1.29	1.26	1.94

Figure 28: Number of latent states and training time