6 UNIT 6 INTRODUCTION TO NATURAL LANGUAGE PROCESSING

Textbook: Chapter 22

This unit will cover (A) Overview of Linguistics, Grammars and Languages; (B) Basic Parsing techniques (C) Semantic Analysis & structures and (D) Natural Language generation and systems

An intelligent agent who wants to acquire knowledge needs to be able to understand, at least partially, the natural language that humans speak.

The following are knows as information-seeking tasks: text classification, information retrieval and information extraction. They all make use of **language models:** models that predict the probability distribution of language expressions.

The famous Sapir Whorf hypothesis claims that our understanding of the world is strongly influenced by the language we speak. Some languages have several words to express the same concept capable of capturing the concept from slightly different angles. There are also cases in which a certain concept that is well defined in one language, does not 'exist' or cannot be names with one word in other languages. Some linguists also believe that language influences our thoughts or ways of thinking through their grammatical structure or features.

A language can be defined as an infinite set of strings whose formation is governed by a set of rules called grammar. Unlike natural languages, formal languages also have rules that define the meaning or the semantics of the language.

In understanding meaning, probability plays an important role. For example, natural languages are ambiguous. So a sentence may have several meanings, so one considers the probability distribution of a sentence over several meanings.

6.1 An Overview of Linguistics, Grammar and Languages

Linguistics is the academic study of languages whether spoken or written. The academicians that study this are called **linguists**. Linguists study how we learn language (language acquisition), what feature characterize a language (phonology, morphology and syntax), how we attach meaning to sounds (semantics), how dictionaries are made (lexicography).

Language can take many forms: spoken, written, but also through signs and non-verbal cues. Speech and writing have several things in common but they are different systems. The most obvious common characteristic is that they both include words. Words in turn have their own structure. Words are made of symbols and these symbols have linguistic values in the sense that they correspond to sounds in spoken language. In English, for example, alphabet letters and speech sounds do not always fit. A symbol 'a' represents at least six sounds: f*a*ther, *a*sk, all, sens*a*tion, *a*round and m*a*re.

Most linguists define language as “a learned system of sounds that have an arbitrary value and meet a social need to communicate”. For linguists language is sound not writing. The sounds are called *phones*, groups of sounds are called *morphs*, *a*nd they have meaning.

Grammar, in the traditional sense, represents the accumulation of terms, rules, and methods for analysing how language is used. A language may have more that one standard, e.g., the English versus the American standard. Generally the role of a grammar is to devise rules governing the correct usage of a language by legislating constructions that must be adhered to or avoided. **Grammar focuses on the written language.**

The main method for syntactically analysing a given written sentence is called **parsing. P**arsing is described in Section 23.2 of Norvig textbook**.** Parsing is a rote process consisting of several steps:

1) Identifying the largest structural components of a sentence: subject and predicate, dependent and independent clauses.

2) Classifying each word as of of the eight **parts of speech (POS):** nouns, pronouns, verbs, adverbs, adjectives, prepositions, conjunctions, interjections.

3) Describing individual words in terms of their inflectional or derivational prefixes or suffixes.

4) Explaining the relationship of each word to other words in the sentence through a “sentence diagram”.

Context-Free Grammars

Context-free grammars (CFG) are a finite collection of rules which tells us that a certain sentence is grammatically correct (syntactically correct) and what their grammatical structure is.

Here is a CFG written in Prolog which contains 11 context free rules. The ‘- >’ defines a rule. The symbols s, np, n, det, vp, v are caled non-terminal symbols. Each of these symbols has a traditional meaning in linguistics: s is short for sentence, np is short for noun phrase, vp is short for verb phrase, and det is short for determiner. That is, each of these symbols is shorthand for a grammatical category. The symbols in italics are the terminal symbols or the lexical terms / or words.

s -> np vp
np -> n det
vp -> v np
vp -> v
det -> athu
det -> langa
det - > []
n -> *abambo*
n -> *bwino*
n - > *pano*
v -> *ali*

The language generated by a grammar consists of all the strings that the grammar classifies as grammatical. For example the following sentences are grammatically correct with respect to this CFG:
‘Abambo athu ali bwino’ meaning “Our father is well.”

‘Abambo langa ali pano.’ My father is here.

But also the sentences: “Bwino langa ali pano” which does not make sense in Chichewa.

A context free recogniser is a program which correctly tells us whether or not a string belongs to the language generated by a context free grammar. To put it another way, a recogniser is a program that correctly classifies strings as grammatical or ungrammatical (relative to some grammar).

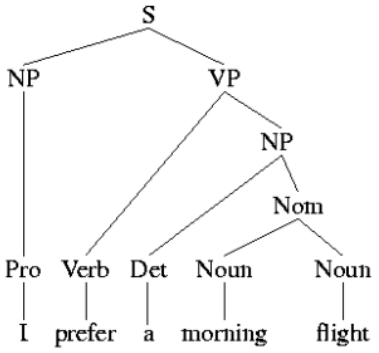
A context free language is a language that can be generated by a context free grammar. Some languages are context free, and some are not. For example, it seems plausible that English is a context free language. It is probably possible to write a context free grammar that generates all (and only) the sentences that native speakers find acceptable.

Example 6.1[1]

Noun → *flight* | *breeze* | *trip* | *morning* | ...
Verb → *is* | *prefer* | *like* | *need* | *want* | *fly* ...
Adjective → *cheapest* | *non-stop* | *first* | *latest* | *other* | *direct* | ...
Pronoun → *me* | *I* | *you* | *it* | ...
Proper-Noun → *Alaska* | *Baltimore* | *Los Angeles* | *Chicago* | *United* | *American* | ...
Determiner → *the* | *a* | *an* | *this* | *these* | *that* | ...
Preposition → *from* | *to* | *on* | *near* | ...
Conjunction → *and* | *or* | *but* | ...

The lexicon for L_0

S → *NP VP*	I + want a morning flight
NP → *Pronoun*	I
| *Proper-Noun*	Los Angeles
| *Det Nominal*	a + flight
Nominal → *Noun Nominal*	morning + flight
| *Noun*	flights
VP → *Verb*	do
| *Verb NP*	want + a flight
| *Verb NP PP*	leave + Boston + in the morning
| *Verb PP*	leaving + on Thursday
PP → *Preposition NP*	from + Los Angeles



The grammar for L_0

In English, the VP consists of the verb and a number of other constituents. For example:

1 Chapter 12 of An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, by Daniel Jurafsky and James H. Martin

VP → Verb	disappear
VP → Verb NP	prefer a morning flight
VP → Verb NP PP	leave Boston in the morning
VP → Verb PP	leaving on Thursday

TASK 6.1 Give four different VP in Chichewa with examples for each.

Do you think Chichewa is a context free language? Explain your answer using as many examples as you want.

Context-Free Grammars in Prolog

Read http://www.learnprolognow.org/lpnpage.php?pagetype=html&pageid=lpn-htmlse28

The CFG in the previous section can be written in Prolog using difference lists.

```
s(X,Z):- np(X,Y), vp(Y,Z).
np(X,Z):- n(X,Y),det(Y,Z).
vp(X,Z):- v(X,Y), np(Y,Z).
vp(X,Z):- v(X,Z).
det([athu|W],W).
det([langa|W],W).
det([],_).
n([abambo|W],W).
n([dengu|W],W).
n([bwino|W],W).
n([pano|W],W).
v([ali|W],W).
```

Here's how to recognise sentences:
?- s([abambo,athu,ali,bwino],[]).

True .

You can also generate all sentences that are accepted by this CFG.

?- s(X,[]).
X = [abambo, athu, ali, abambo, athu] ;
X = [abambo, athu, ali, abambo, langa] ;
X = [abambo, athu, ali, abambo] ;
X = [abambo, athu, ali, dengu, athu] ;
X = [abambo, athu, ali, dengu, langa] ;
X = [abambo, athu, ali, dengu]
etc….

To find out if *abambo athu* is a noun phrase we ask:

?- np([abambo, athu],[]).

And we generate all the noun phrases in the grammar as follows:

?- np(X,[]).

TASK 6.2 Write the CFG that accepts the following sentences in Chichewa. Then test your CFG. “Zinthu zanga zili bwino.” “Chipinda chake chili uko.”

The way CFG are written using lists is not intuitive. Prolog has a nicer notation to write grammars called the **Definite Clause Grammars.**

```
s --> np, vp.
np --> n,det.
vp --> v, np.
vp --> v.
det --> [athu].
det --> [langa].
```

```
det --> [].
n --> [abambo].
n --> [dengu].
n --> [bwino].
n --> [pano].
v --> [ali].
```

You can test this grammar with the same queries as before: ?- s([abambo,athu,ali,bwino],[]).
The two grammars are equivalent as Prolog translates the DFG into CFG. To see this run the following query.

```
?- listing(s).
s(A, B) :-
    np(A, C),
    vp(C, B).
```

TASK 6.3 Rewrite your CFG from the previous task using now DFG instead of CFG.

TASK 6.4 (Exercise 7.3 from online tutorial learnprolognow.org) Let $a^n b^{2n}$ be the formal language which contains all strings of the following form: an unbroken block of a s of length n followed by an unbroken block of b s of length 2n , and nothing else. For example, abb , aabbbb , and aaabbbbbb belong to $a^n b^{2n}$, and so does the empty string. Write a DCG that generates this language.

Chomsky Normal Forms (CNF)

A CFG is said to be in Chomsky Normal form if the production rules are all of the following forms:

A → a (a terminal symbol)
A → BC (at most two symbols on the right hand side RHS)
S → ε (null string)

where A,B,S,S are non-terminal symbols and a is a terminal symbol and ε is the empty symbol.

The conversion to Chomsky Normal Form has four main steps:

1. Get rid of all ε productions, where ε is the empty symbol. For example, for step 1: if P → AxB with both A and B nullable, add productions P → xB|Ax|x. After this, delete all productions with empty RHS.

2. Get rid of all productions where RHS is one variable. A unit production is where RHS has only one symbol. Consider the production A → B. Then for every production B → α, add the production A → α.

3. Replace every production that is too long by shorter productions. For example, if have production A → BCD, then replace it with A → BE and E → CD.

4. Move all terminals to productions where RHS is one terminal. For every terminal on the right of a non-unit production, add a substitute variable. For example, replace production A → bC with productions A → BC and B → b.

Example 6.1 (Source https://www.cs.bgu.ac.il/~auto191/wiki.files/9a.pdf):

Consider the CFG:

S → aXbX

X → aY | bY | ε

Y → X | c

After the elimination of ε we obtain

S → aXbX | abX | aXb | ab

X → aY | bY | a | b (Note that the variable X is nullable; and so therefore is Y).

Y → X | c

After elimination of the unit production Y → X, we obtain:
S → aXbX | abX | aXb | ab
X → aY | bY | a | b
Y → aY | bY | a | b | c

Now, break up the RHSs of S; and replace a by A, b by B and c by C wherever not units:

S → EF | AF | EB | AB

X → AY | BY | a | b

Y → AY | BY | a | b | c

E → AX

F → BX

A → a

B → b

C → c

TASK 6.5 (Convert the following CFG into Chomsky Normal Form:

S → AbA

A → Aa|ε

TASK 6.6 (Is it true that any context-free grammar can be automatically transformed into Chomsky Normal Form? Explain your answer.

6.2 Parsing Techniques: The CYK Algorithm

The word problem of a language L asks for deciding whether a given input word w is a member of L. The Cocke–Younger–Kasami-Algorithm (CYK or CKY) is a highly efficient **parsing algorithm** for context-free grammars written in CNF.

The algorithm uses a “dynamic programming” or “table filling algorithm”.

How the CYK-allgorithm works

Fix a context-free grammar $G(N, T, P, S)$ in Chomsky normal form.

The algorithm decides for an input w whether $w \in \mathrm{L}(G)$.

For a given input $w = a_1 \cdots a_n \in T^n$, let for all $1 \leq i \leq j \leq n$

$$w_{ij} = a_i a_{i+1} \cdots a_j.$$

The algorithm computes inductively for $j - i = 0, \ldots, n-1$ the sets

$$N_{ij} = \{X \in N : X \overset{G,*}{\Longrightarrow} w_{ij}\}.$$

Obviously, the word w is in $\mathrm{L}(G)$ if and only if S is in N_{1n}.

(Source: https://www.math.uni-heidelberg.de/logic/ss17/formsprach/fs20-cky-algorithm-41.pdf)

The CYK algorithm starts by constructing a triangular table so that each row corresponds to one length of sub-strings. The following are then calculated:

- Xi, i is the set of variables A such that A → wi is a production of G
- Compare at most n pairs of previously computed sets:

$$(X_{i,i}, X_{i+1,j}), (X_{i,i+1}, X_{i+2,j}) \ldots (X_{i,j-1}, X_{j,j})$$

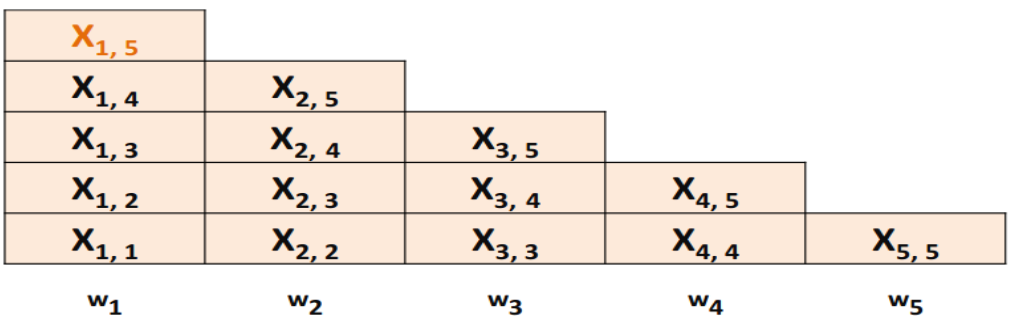


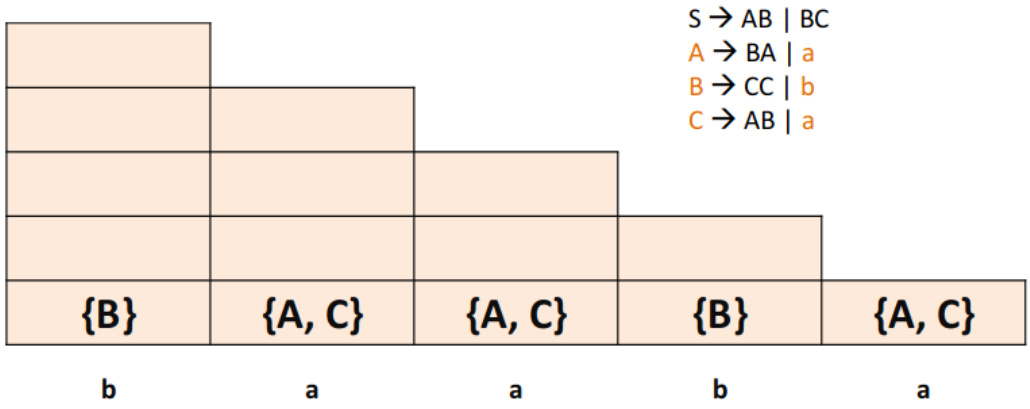
Table for string ‘w’ that has length 5

Example 6.2[2]:

Show the CYK Algorithm with the following example:

- CNF grammar **G**
  - S → AB | BC
  - A → BA | a
  - B → CC | b
  - C → AB | a
- **w** is baaba
- Question Is **baaba** in L(G)?

Step 1: Construct the triangular table
Step 2: Calculate the bottom row



Step 3: Calculate second bottom row
$X_{1,2} = (X_{i,i},X_{i+1,j}) = (X_{1,1} , X_{2,2}) = \{B\}\{A,C\}=\{BA,BC\}$ Look for production rules to generate BA or BC. $X_{1,2} = \{S,A\}$

$X_{2,3} = (X_{i,i},X_{i+1,j}) = (X_{2,2} , X_{3,3}) = \{A,C\}\{A,C\} = \{AA,AC,CA,CC\} = \{B\}$
$X_{3,4} = (X_{i,i},X_{i+1,j}) = (X_{3,3} , X_{4,4}) = \{A,C\}\{B\} = \{AB,CB\} = \{S,C\}$
$X_{4,5} = (X_{i,i},X_{i+1,j}) = (X_{4,4} , X_{5,5}) = \{B\}\{A,C\} = \{BA,BC\} = \{S,A\}$

Step 4: Calculate the third bottom row

$X_{1,3} = (X_{i,i},X_{i+1,j})(X_{i,i+1},X_{i+2,j}) = (X_{1,1} , X_{2,3})(X_{1,2} , X_{3,3}) = \{B\}\{B\}U\{S,A\}\{A,C\} = \{BB,SA,SC,AA,AC\} = \{\emptyset\}$ (The empty set)

2 Source https://web.cs.ucdavis.edu/~rogaway/classes/120/winter12/CYK.pdf

$X_{2,4} = (X_{i,i},X_{i+1,j})(X_{i,i+1},X_{i+2,j}) = (X_{2,2}, X_{3,4})(X_{2,3}, X_{4,4}) = \{A,C\}\{S,C\} \cup \{B\}\{B\} = \{AS,AC,CS,CC,BB\} = \{B\}$

$X_{3,5} = (X_{i,i},X_{i+1,j})(X_{i,i+1},X_{i+2,j}) = (X_{3,3}, X_{4,5})(X_{3,4}, X_{5,5}) = \{A,C\}\{S,A\} \cup \{S,C\}\{A,C\} = \{AS,AA,CS,CA,SA,SC,CA,CC\} = \{B\}$

Step 5/6: Calculate the last two bottom rows. The final table is:

{S, A, C}	← $X_{1,5}$			
Ø	{S, A, C}			
Ø	{B}	{B}		
{S, A}	{B}	{S, C}	{S, A}	
{B}	{A, C}	{A, C}	{B}	{A, C}
b	a	a	b	a

A visual simulation of the CYK algorithm can be found here: https://www.xarg.org/tools/cyk-algorithm/

TASK 6.7 Follow the example below and check that the solution shown is the correct one.

Example Application of the CYK-algorithm

Consider the language $L = \{ww^R : w \in \{0,1\}^+\}$ of all palindromes over the binary alphabet of even nonzero length.

$j =$	1	2	3	4	5	6	$i =$
$a_1 = 0$	$\{X_0\}$	$\{S\}$	$\{R_1\}$	$\emptyset$	$\emptyset$	$\{S\}$	1
$a_2 = 0$	$\{X_0\}$	$\emptyset$	$\emptyset$	$\{S\}$	$\{R_0\}$		2
$a_3 = 1$	$\{X_1\}$	$\{S\}$	$\{R_0\}$	$\emptyset$			3
$a_4 = 1$	$\{X_1\}$	$\emptyset$	$\emptyset$				4
$a_5 = 0$	$\{X_0\}$	$\{S\}$					5
$a_6 = 0$	$\{X_0\}$						6

Let G be a grammar in Chomsky normal form that generates L and has the rules

$$S \to X_0R_0 \mid X_1R_1 \mid X_0X_0 \mid X_1X_1,$$

$$R_0 \to SX_0, \quad R_1 \to SX_1,$$

$$X_0 \to 0, \quad X_1 \to 1,$$

Applying the CKY-algorithm to G and the input $w = 001100$ yields the sets N_{ij} as shown in the table on the left.

THEOREM (CYK): The CYK Algorithm correctly computes X_{ij} for all i and j; thus w is in L(G) if and only if S is in X_{1n}. The running time of the algorithm is O(n3).

TASK 6.8 Consider the following CNF grammar G with 4 production rules. Is w=ababa in L(G)?
S → AB | BC

A → BA | a
B → CC | b
C → AB | a

The CYK Parser is the most efficient one but requires the grammar to be in CNF. There are other parsers as summarised in the table below[3].

ALGORITHM	MAIN FEATURES	USE CASES
CYK	• great worst-case performance ($O(n^3)$) • grammars require special CNF form	Specific problems (e.g., membership problem)
Earley	• great worst-case performance ($O(n^3)$), usually linear performance • can handle all kinds of grammars and languages	Parser generators that must be able to handle all grammars and languages
LL	• easy to implement • less powerful than most other algorithms (e.g., cannot handle left-recursive rules) • historically was the most popular choice	Hand built parsers; parser generators that are easier to build
LR	• hard to implement (some variants are easier than others) • powerful (can handle most grammars, some variants can handle all of them) • great performance (usually linear, more powerful variants can have worst-case performance of $O(n^3)$)	Parser generators with the best performance
Packrat (PEG)	• linear time of performance • uses a special format • designed for parsing computer languages	Simple, but powerful parsers or parser generators for computer languages
Parser Combinator	• modular and easy to build • comparatively bad perfomance ($O(n^4)$) in the worst case)	Parser generators that are easy to use and understand
Pratt	• peculiar and lesser known algorithm • no need for a grammar	Parsing expressions or other cases in which precedence is not dictated by the order of appearance

TASK 6.9 Do some research and explain how the LL parser works. Give 2 examples that show how this algorithm runs. Compare it with the CYK algorithm. What are the similarities and differences?

3 Source https://tomassetti.me/guide-parsing-algorithms-terminology/

6.3 N-gram Character Models

One of the simplest language models is a probability distribution over sequences of characters.

We write P($c_{1:N}$) is the probability of a sequence of characters c_1 to c_N . A sequence of written symbols of length n is called an **n-gram.** Special such n-grams are 1-gram, 2-gram (bigram) or 3-gram (trigram). A model of the probability distribution of n-letter sequences is called an **n-gram model.**

In an n-gram model, the probability of a character c_1 depends only on the immediately preceding characters not on all other characters (we say that an n-gram is a Markov chain of order n-1). So in a trigram:

$$P(c_{1:i} \mid c_{1:i-1}) = P(c_{1:i} \mid c_{i-2:i-1})$$

Example 6.10 In Chichewa nouns are split in several classes. Each class has associated to it a determining prefix which is usually a bi-gram that is then added to the qualifying verb or adjective for that noun in a sentence construction. For example the noun class **Mu/A** in which words such as **mu**nthu/ **a**nthu and **m**kazi/**a**kazi, **mw**amuna/**a**muna.

Example 6.11 The 1-grams generated from the word ‘munthu’ are [m,u,n,t,h,u]. The 2-grams generated are mu,un,nt,th,hu.

Task 6.12 Can you write all the 3-grams and 4-grams generated from the word ‘munthu’.

Automatic Language Recognition

What can we do with n-gram character models? One task for which they are well suited is **language identification**: given a text, determine what natural language it is written in.

The first task is to collect a corpus of text for each language. About 100,000 characters of each language are needed. One approach to language identification is to first build a trigram character model of each candidate language. For each of the model is built by counting trigrams in a corpus of that language. That gives us a model of each language, but we want to select the most probable language given the text, so we apply Bayes' rule.

So for each tri-gram we calculate the probability that that trigram appears in each language. To compute the total sentence probability, we multiply each of the n-gram probabilities. Notice that when we multiply a lot of small probabilities our final result gets very small; for this reason we usually use log probabilities instead and add them

Example 6.13[4]

The following table shows the calculated probabilities for 1-grams in four European languages. Note that each character is converted to its binary hexadecimal encoding for easy processing.

	German	English	French	Danish	Swedish
`0x63 'c'`	0.021	0.022	0.024	0.025	0.009
`0x6C 'l'`	0.028	0.032	0.040	0.041	0.041

To compute the total sentence probability, we multiply each of the n-gram probabilities. Notice that when we multiply a lot of small probabilities our final result gets very small; for this reason we usually use log probabilities instead and add them (using the following identity: `log(a*b) = log(a) + log(b)`). Previously we calculated that 0x63 has a probability of 0.021 in German, this corresponds to a log probability of `log(0.021) = -3.85`. From here on we will use the log probabilities of the n-grams.

An Example: we wish to determine the probability that the sentence '**The cat**' comes from each of our 5 languages using 2-grams. Below are the log probabilities for each of the 2-grams in the sentence from our 5 languages:

	German	English	French	Danish	Swedish
`0x5468 'Th'`	-8.67	-6.37	-9.60	-9.40	-10.00
`0x6865 'he'`	-4.96	-4.08	-6.51	-6.03	-6.23
`0x6520 'e '`	-3.94	-3.56	-3.24	-3.71	-4.74
`0x2063 ' c'`	-10.10	-5.00	-4.69	-7.66	-7.78
`0x6361 'ca'`	-9.46	-5.89	-6.28	-8.27	-8.47
`0x6174 'at'`	-5.63	-4.80	-5.37	-5.04	-4.81
total:	-42.79	-29.73	-35.72	-40.13	-42.05

For English the sentence probability is (-6.37) + (-4.08) + (-3.56) + (-5.00) + (-5.89) + (-4.80) = -29.73, which is a much higher probability than any of the other languages.

4 Source: http://practicalcryptography.com/miscellaneous/machine-learning/tutorial-automatic-language-identification-ngram-b/

Some n-grams are just rare and they might not occur in the corpus we considered. How do we calculate the probabilities of n-grams that don't appear in our training corpus? A convention is to assign a count of 1 to unseen n-grams. This prevents our total sentence probability from becoming zero as soon as an unusual n-gram is seen.

The Bayes Classifier

The **Conditional Probability** for two events A and B

$$P(A \mid B) = P(B \mid A) / P(A)$$

For **independent A and B** we have

$$P(A|B) = P(A)\ P(B).$$

The Bayes Rule is

$$P(B \mid A) = \frac{P(A|B)P(B)}{P(A)}$$

Joint Probabilities:

$$P(X,Y) = P(X \mid Y)\ P(Y)$$

The Markov chain rule:

$$P(X1,X2,...Xn)=P(X1)P(X2|X1)P(X3|X2,X1)...P(Xn|X1,...Xn-1)$$

N-gram language model assumes each word depends only on the last n-1 words (Markov assumption).

To decide what language a sentence was in, we calculate the probability each sentence comes from each language, then choose the language with the highest probability as our guessed label. **This is known as a naive Bayes classifier.**

If S is a sentence then the probability of the sentence P(S) is the joint probability of each of its individual n-grams s1, …, s4:

$$P(S) = P(s1, s2, ..., sn)$$

The probability of the sentence is reduced to the probabilities of the sentence's individual bigrams.

```
P('munthu') ~ P('mu')P('un'|'mu')P('nt'|'un')P('th'|'nt')P('hu'|'th')
```

A naive Bayes classier works best it we have more data. So longer sentences will usually be more reliably classified than short sentences.

The size of our n-grams has an effect on our classification ability. It has been shown that the correctness jumps considerably going from 1- to 2-grams, then steadily climbs. Most models use tri-grams and 4-grams.

6.4 N-gram Word Models

In a similar way as we defined n-grams over characters we can define n-grams over words. a contiguous sequence of n tokens from a given piece of text.

Example 6. 14

- **Lilongwe** is an 1-gram.
- **Nkhata Bay** is a 2-gram containing [Nkhata, Bay]
- **Dzina Lanu ndani** is a 3-gram containing [Dzina, Lanu, ndani]
- **Amayi anu ali bwanji?** is a 4-gram containing the 4 words.

There are several n-grams models based on the Markov chain principle that the probability distribution of a word only depends on the immediate words.

Unigram model: $P(w1)P(w2)P(w3)...P(wn)$

Bigram model: $P(w1)P(w2|w1)P(w3|w2)...P(wn|wn-1)$

Trigram model: $P(w1)P(w2|w1)P(w3|w2,w1)...P(wn|wn-1wn-2)$

N-gram model: $P(w1)P(w2|w1)...P(wn|wn-1wn-2...wn-N)$

Task 6.15 Write the 3-gram model for the following sentence: '**Achimene anga akugona muno.**'

Task 6.16 Suppose we have the following corpus (From Julia Hockenmeier Introduction to NLP) Write down the probabilities as indicated.

```
Alice was beginning to get very tired of
sitting by her sister on the bank, and of
having nothing to do: once or twice she
had peeped into the book her sister was
reading, but it had no pictures or
conversations in it, 'and what is the use
of a book,' thought Alice 'without
pictures or conversation?'
```

$P(w_{i+1}=\text{of} \mid w_i=\text{tired}) =$
$P(w_{i+1}=\text{of} \mid w_i=\text{use}) =$
$P(w_{i+1}=\text{sister} \mid w_i=\text{her}) =$
$P(w_{i+1}=\text{beginning} \mid w_i=\text{was}) =$
$P(w_{i+1}=\text{reading} \mid w_i=\text{was}) =$

$P(w_{i+1}=\text{bank} \mid w_i=\text{the}) =$
$P(w_{i+1}=\text{book} \mid w_i=\text{the}) =$
$P(w_{i+1}=\text{use} \mid w_i=\text{the}) =$

Task 6.17 Have a look at https://books.google.com/ngrams/ and https://ai.googleblog.com/2006/08/all-our-n-gram-are-belong-to-you.html and writea short explanation of what these tools offer and then write down 2 or 3 examples you have run using these tools.

Text Classification

The task of text classification is that of assigning documents or text to one or more specific categories. For example, classifying a piece of text from a newspaper article as being a text containing art news or political news. Another example is to classify an email as being spam or non-spam or a post on a forum as being 'acceptable' (e.g., not containing abusive words) or not acceptable.

Task 6.18 Conduct a research to find out various applications of text classification and give en example for each.

Today the main techniques for text classification rely on machine learning that is learning classification rules from examples. In order to build such classifiers, we need labeled data, which consists of documents and their corresponding categories (or tags, or labels).

However, there are still classifiers based on Naive Bayes.

6.5 Information Retrieval Model

An IR model consists of the following:

1. A corpus of documents. Each system must decide what it wants to treat as a document: a paragraph, a page, or a multipage text.
2. Queries posed in a query language. A query specifies what the user wants to know. The query language can be just a list of words, such as [AI book]; or it can specify a phrase of words that must be adjacent, as in ("Al book"]; it can contain Boolean operators as in [AI AND book]; it can include non-Boolean operators such as [AI NEAR book] or [AI book site:www.aaai.org].
3. A result set. This is the subset of documents that the IR system judges to be relevant to the query. By relevant, we mean likely to be of use to the person who posed the query, for the particular information need expressed in the query.
4. A presentation of the result set. This can be as simple as a ranked list of document titles or as complex as a rotating color map of the result set projected onto a three-dimensional space, rendered as a two-dimensional display.

The earliest IR systems worked on a Boolean keyword model. Each word in the document collection is treated as a Boolean feature that is true of a document if the word occurs in the document and false if it does not.

Task 6.19 Conduct a research using your textbook and explain the following technique for IR

- BM25 scoring function
- IR Evaluation
- The Page-Rank Algorithm

Give an example for each.