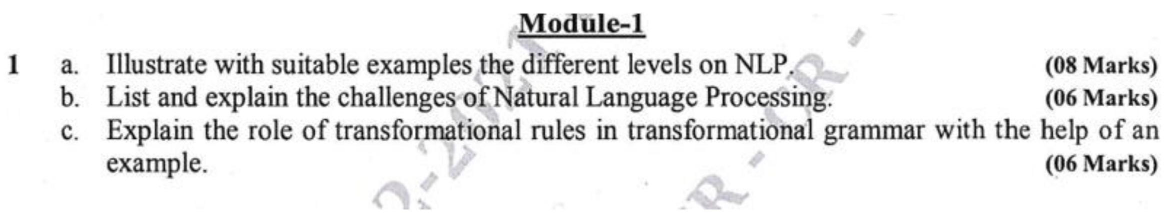
VTU Question Paper with Solution

17CS741 Natural Language processing Jan/Feb 2021



Module-1

1 a. Illustrate with suitable examples the different levels on NLP. **(08 Marks)**

b. List and explain the challenges of Natural Language Processing. **(06 Marks)**

c. Explain the role of transformational rules in transformational grammar with the help of an example. **(06 Marks)**

1.a. There are four types of process analysis:

1.Lexical Analysis

Lexical Analysis is analysis of words which is the fundamental for natural language.It includes word level processing i.e, morphological knowledge about the structure and formation of words.

2.Syntactic Analysis

Syntactic Analysis considers sequence of words that forms a unit or sentence. Decomposition of sentence into words and identify their relationship.

3.Semantic Analysis

Semantic Analysis is associated with the meaning of language and meaningful representation of linguistic input.

4.Discourse Level

Discourse Level interprets the structure and meaning of language units referred as Discourse Knowledge. Discourse Knowledge deals with how the meaning of a sentence is determined by preceeding sentence.

Eg: how pronoun refers to the preceeding noun.

It determines the function of sentence in the text. Prgamatic(Sensible) knowledge is needed for resolving anaphoric(ideas other than text) references.

Eg:I went with Joy___

1b. Challenges of NLP:

There are number of factors that make NLP difficult. It relates to the problems of representation and interpretation.The language computing requires precise representation ofcontext. It is highly ambiguous and vague(i.e, achieving such representation can be difficult).

- Inability to capture all the required knowledge : It is impossible to embody all sources of knowledge that humans use and not possible by humans to write procedures that initiate language processing.
- Greater difficulty: Identifying its semantics is a difficult task.The principle of compositional semantics considers the meaning of a sentence to be a composition of the meaning of words.

Also words alone donot make a sentence.

Example1: I do not like icecream

Do not I ice cream like $\rightarrow$ is wrong

Example2: Kabir and Ayan are married.

Kabir and Suha are married.

Both the sentence give different meaning, even though it has same structure.

- Only words along with the syntactic and semantic relation can give meaning to a sentence.
- So, the only way for a machine to learn the meaning of a specific words in a message is by considering the context defined by the concurring words.

  a. Idioms ,Metaphors and Ellipses: These leads to more complexity as it requires identification of meaning of written text.
     Ex: Oldman finally kicked the bucket
  b. Quantifier Scoping: Scoping of quantifier is not clear.
  c. Ambiguity of natura
  d. Incorporating contextual and word level knowledge poses a greater difficulty.

Various Sources of Ambiguity in Natural Languages:

1. Word Level Ambiguity: It is the first level of ambiguity.It involves identification of words that have multiple meaning associated with them.
   Words may be ambiguous. Example1 : can,but -> part of speech-can

   ->meaning –bat

   Example2: bank has two meanings ->1.Financial Institution Sense ->2.River Bank Sense

   These are solved by word sense disambiguation.

2. Structural Ambiguity: The ambiguity exist in the sentence not the words.
   Example: Stolen rifle found by tree.
   It can be solved by Verb SubCategorization or Probabilistic Parsing

1c. Transformational Grammar :

- Mapping from Deep Structure to Surface Structure by Chomsky in 1957.
- The deep structure can be transformed in number of ways to yield many different surface level representation.
- Sentence with deep surface level have same meaning and share a common deep level representation.
  Ex: Pooja plays Veena.
  Veena is played by Pooja.

Both the above sentences have the same meaning, but different surface structure.The sentences are generated from some deep structures in which:

Deep subject-Pooja

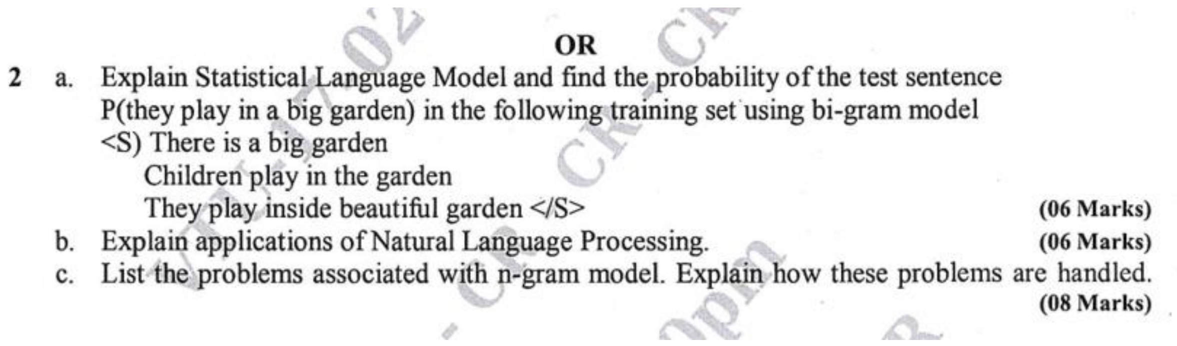
Deep object-Veena

- Transformational Grammar has three components:
  1. Phrase Structure
  2. Transformational Rules
  3. Morphophonemic Rules-> Match each sentence represented to a string of phonemes.
     Each of these components consists of a set of rules.
     Phrase Structure Grammar -> consists of rules that generate natural language sentence and assign a structural description to them
     S-> NP+VP
     VP->V+NP
     NP->Det+Noun
     V->Aux+VerB

Det->the, a, an
Verb-> catch, write, eat
Noun -> police , snatcher
Aux-> will,is,can

S=>Sentence
NP=>Noun Phrase
VP=>Verb Phrase
Det=>Determinants

- Sentence that can be generated by these rules are termed as grammatical.
  Second component of transformational grammar is set of transformational rules,it lets transformation of one underlying phrase __ into a derived phrase __, this is applied to the terminal of the phrases.
- Heterogeneous have more than one symbol on the left hand side, these rules are used to transform one surface representation into another.
  Ex: An active sentence into a passive sentence.
  Rules relating active and passive sentences in:
  NP1-Aux-V-NP2=> NP2-Aux+be+en-V-by-NP1
  =>underlying input having the structure:
  N-Aux-V-NP can be transformed to NP-Aux+be –en-V-by+NP
  It involves addition of strings 'be' and 'en' and certain rearrangement of the constituents of a sentence.
- Transformational Rules can either be obligatory(that which ensures agreement in number of subjects and verbs) and optional(modifies the structure of a sentence while preserving its meaning).


OR

2 a. Explain Statistical Language Model and find the probability of the test sentence P(they play in a big garden) in the following training set using bi-gram model
<S) There is a big garden
Children play in the garden
They play inside beautiful garden </S> **(06 Marks)**

b. Explain applications of Natural Language Processing. **(06 Marks)**

c. List the problems associated with n-gram model. Explain how these problems are handled. **(08 Marks)**

2a. Statistical Language model:

A statistical language model is a probability distribution P(s) over all possible words sequences.

1. N-gram model

The goal is to estimate the probability of sentence. By decomposing sentence probability into a product of conditional probabilities using the chain rule as follows:

P(S)=P(w1,w2…………wn)

=P(w1)p(w2/w1)……………….P(wn/w1w2w3……wn-1)

=π P(wi/hi) where i=1 to n

Hi is history of word wi defined as w1,w2…….wi-1

To calculate sentence probability, need to calculate the probability of word, given the sentence of words preceding it.

An n-gram model simplifies the task by approximating the probability of a word given all the previous words by the conditional probability given previous n-1 words only.

$P(w_i/h_i)=P(w_i/w_{i-n+1}\ldots\ldots\ldots w_{i-1})$

Thus , an n-gram model calculate P(wi/hi) by modeling language as marker model of order n-1 i,e. by looking n-1 words only.

A model that limits the history to the previous one word only, is termed a bi-gram (n=1) model.

Likewise, a model that conditions the probability of a word to the previous two words, is called a tri-gram (n=2) model.

Using bi-gram and tri-gram estimate, the probability of sentence can be calculates as,

$P(S)=\pi\ P(w_i/w_{i-1})$ where $i=1$ to n

$P(S)=\pi\ P(w_i/w_{i-2}.W_{i-1})$ where $i=1$ to n

Ex: The bi-gram approximation of P(east/The Arabian Knights are fairy of the) is

P(east/the)

Whereas a trigram approximation is

P(east/of the)

A special word <s> is introduced to mark the beginning of the sentence in bi-gram estimation. The probability of the first word in a sentence is conditioned as <S>. Similarly in tri-gram estimation, we introduce two pseudo words.

<s1> and <s2>

To estimate there probabilities, the training of n-gram model on the training corpus. To estimate n-gram parameters using the maximum likelihood estimation technique using relative frequencies.

Count a particular n-gram in the training corpus to divide it by sum of all n-grams that share the same prefix.

P(wi/wi-n+1………..wi-1)=C(wi-n+1…….wi-1,wi)/∑ C(wi-n+1………….wi-1,wi)

The sum of all n-grams that share first n-1 words is equal to the count of the common prefix

$W_{i-n+1}\ldots\ldots\ldots\ldots w_{i-1}$

P(wi/wi-n+1………..wi-1)=C(wi-n+1…….wi-1,wi)/C(wi-n+1………….wi-1)

The model parameter we get using these estimate, maximizes the probability of the training set T given the model M i,e. P(T/M).

Ex: Training set:

There is a big garden

Children play in the garden

They play inside beautiful garden.

Bigram model:

P(the/<S>)=0.67= 2 /3

P(Arabian/the)=0.4 = 2 /5

P(Knights/Arabian)=1.0=== 2 /2

P(are/these)=1.0

P(the/are)=0.5

P(fairy/the)=0.2

P(tales/fairy)=1.0

P(of/tales)=1.0

P(the/of)=1.0

P(east/the)=0.2

P(stories/the)=0.2

P(of/stories)=1.0

P(are/Knights)=1.0

P(translated/are)=0.5

P(in/translated)=1.0

P(many/in)=1.0

P(language/many)=1.0

Test sentence(S) : They play in big garden

P(the/<S>)*P(Arabian/the)*P(Knights/Arabian)*P(are/these)*P(the/are)*P(fairy/the)*P(tales/fairy)*P(of/tales)*P(the/of)*P(east/the)*P(stories/the)*P(of/stories)*P(are/Knights)*P(translated/are)*P(in/transalted)*P(many/in)*P(language/many)

=0.67*0.4*1.0*1.0*0.5*0.2*1.0*1.0*1.0*0.2

=0.0067

2b. The applications where NLP is used are:

1. Machine Translation->Automatic translation of text from one human language to another.It is necessary to have an understanding of words and phrases, grammars of two languages, semantics of world knowledge.
2. Speech Recognition->mapping Acoustic Speech Signals to a set of words.The difficulties involved are due to the wide variations in the pronunciation of words, homonyms(Ex: dear and deer),acoustic ambiguities in the sense of hearing(Ex: interest and in the rest).
3. Speech Synthesis->Automatic production of speech.Systems can read out mails on telephone or even read out a story book.
4. Natural Language interfaces to databases->Allows quering a structured database using natural language sentence.
5. Information Retrieval(IR)->It is done by identifying documents relevant to a users query. Application in Information Retrieval includes:
    1. Indexing(Stop word elimination,stemming and phrase extraction,etc).
    2. Word Sense Diasmbiguation
    3. Query Modification
    4. Knowledge Based
        - WordNet , LDOCE(Longman Dictionary Of Contemporary Engkish),Roget's Thesaurus.
6. Information Extraction(IE)->Captures and Outputs factual information contained within a document.Similar to Information Retrieval, Information Extraction also responds to a user's information need. The users information is not specified as keyword query but specified as pre defined database schema or templates.
    - An IR system identifies a subset of documents in a large repository of text databases. Ex: In a library scenario, a subset of resources in a library.
    - An IE system identifies a subset of information within a document that fits the predefined template.
7. Question Answering->On a question and a set of documents, a question answering system attempts to find the preciseanswer , or atlesat the presence of portion of text in which the answer appears.
    - It's not like the IR system, it returns the whole document that seems relevant to the user's query.
    - It's also different from IE system, --- that the content that is to be extracted is unkown.
    - So it has benefits from IE to identify entities the text and it requires more NLP ,------ it need not only precise analysis of question and portions of text, but also semantic as wwll as background knowledge to answer certain type of questions.
8. Text Summarization->It deals with creation of summaries of documents and involves syntactic, semantic discourse level(i.e, interpret the structure and meaning of large units) processing.

2c. The n-gram model suffers from data sparseness problem. An n-gram that does not occur in the training data is assigned zero probability, so that even a large corpus has several zero entries in the bigram matrix.

A number of smoothing techniques have been developed to handle the data sparseness problem.

The word smoothing is used to denote these techniques. Because they tend to make distributions more uniform by moving the extreme probabilities towards the average.

1. **Add one smoothing:**

It adds a value of one to each n-gram frequency before normalizing them into probabilities. Thus the conditional probability becomes:

P(wi/wi-n+1…………wi-1)=C(wi-n+1…….wi-1,wi)/C(wi-n+1………….wi-1)+v

V is the vocabulary size i.e. size of the set of all the words being considered.

2. Good Turing smoothing:

Adjusts the frequency of an n-gram using the count of n-grams having a frequency of occurrence f+1.

If converts the frequency of an n-gram from f to f*

$f^*=(f+1)n_{f+1}/n_f$

n_f - number of n-gram that occur exactly f times in the training corpus.

Ex: Occur 4 times is 25,108 and the number of n-gram that occur 5 times is 20,542 ,smoothing count for 5 will be

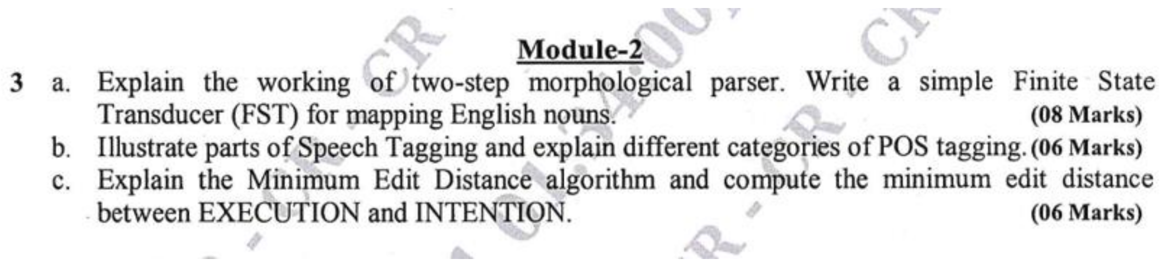
20542/25108*5=4.09

2. **Caching:**

The frequency of n-gram is not uniform across the text segments or corpus. Certain words occur more frequently in certain segments and rarely in others.

ex: The frequency of the word 'n-gram' is high, whereas it occurs rarely. The basic n-gram model ignores this sort of variation of n-gram frequency.

The cache model combines the most recent n-gram frequency with the standard n-gram model to improve its performance locally.



Module-2

3 a. Explain the working of two-step morphological parser. Write a simple Finite State Transducer (FST) for mapping English nouns. **(08 Marks)**

b. Illustrate parts of Speech Tagging and explain different categories of POS tagging. **(06 Marks)**

c. Explain the Minimum Edit Distance algorithm and compute the minimum edit distance between EXECUTION and INTENTION. **(06 Marks)**

3a. Finite state transducer is a 6 tuple

simple transducer that accepts two strings,hot and cat maps them into cot and bat.
 FSA encode regural languages and regular relation.
-Regular relation is the relation between regular languages.
-The regular language encoded pn the upper side of a FST is called upper language,lower side is lower language.
 If T->transducer
    S->string
then T(S)->represent the set of strings encoded by T such that the pair (s,t) in is relation.

FST's are closed under:
            *union
            *cocatenation
            *composition
            *kleene closure
     Not closed under:
            *Intersection
            *complementation
Two step morphological parser
->Split the words up into its possible components
   eg:bird+s out of birds->morpheme boundries
:Spelling rules are cnsidered
->2 possible ways of splitting up:
   boxes->boxe+s,box+s
1)boxes->boxe+s
   Assume,
     boxe-stem
     s-suffix
2)boxes->box+s
     box-stem
     s- suffix
     e- has been introduced due to the spelling rule
=>output is the concatenation of morphemes i.e stem+affixes

FST

=>FST represents the information the the comparative form of adjective less is lesser,e(belons to) here is empty string.
=>The automation is inherently bi-directional the same transducer can be used for analysis->surface i/p upword
for generation->lexical
i/p,downword application
In boxes->box+s
   used a lexicon to look up categories of the stems and meaning of the affixes.
  bird+s will be mapped to bird+N+PL
  box+s will be mapped to box+N+PL
so,boxe+s is incorrect way of splitting boxes so discarded.

But,spouses=>spouse+s
     parses=>parse+s
->It's correct ,in this orthographic rules are used to handle these spelling variations,
=>SO,one of the spelling rules says:
  add e after -s,-z,-x,-ch,-sh
  before the s-dish->dishes,box->boxes
Each of these steps can be implemented with the help of transducer.
  ->Two transducers are needed:
1)One that mas th surface form to the intermediate form
2)another that maps the intermediate form to lexical form.
  Develop an FST based morphological parser for singular and plural nouns in English
->The plural form of regular nouns usually end with -s or -es
However a word ending in 's' need not be plural like miss,ass

->One of the required translatins is the deletion of the 'e' when introducing a morpheme boundary.This deletion is usually required for words ending in 'xes,ses,zes' eg:(suffixes,boxes)

Simplified FST,mapping english nouns to the intermediate form
->Next step is to develop a transducer,that does the mapping from the intermediate level to the lexical level.
->Thee input to transducer has one of the following forms:
   *Regular noun stem,eg:bird,cat
   *Regular noun stem +s,eg:bird+s
   *Singular irregular noun stem,eg:goose
   *Plural irregular nun stem,eg:geese
->First case,the transducer has to map all symbols of the stem to themselves and then output N and sg
->Second case ,it has to map all symbols of the stem of the stem,to themselves,but then output N replaces PL with S
->Third case ,it has to do the same as in the first case
->Fourth case,the transducer has rto map the irregular plural noun stem to the corresponding singular stem.(eg:geese to goose) and then it should add N and PL

The mapping from state 1 to state 2,3,4 is carried out with the help of transducer encoding a lexicon
-The transducer implementing the lexicon maps the individual regular and irregular noun stems to their correct noun stem,replacing labels like regular noun form,etc.
-This lexicon maps the surface form geese,which is an irregular noun to its correct stem goose in the following way:
    g:g e:o e:o s:s e:e
-Mapping for the regular surface form of bird
  b:b i:i r:r d:d
-Representing pairs like a:a with a single letter,these two representations are reduced to
  g e:o e:o s e and b i r d
-Composing this reducer with the previous one,we get a single two-level transducer with one input tape and one output tape.
-This maps plural nouns into the stem plus the morphological marker+pl and singular nouns into the stem plus the morpheme+sg
-Thus a surface word form birds will be mapped to bird+ N+PL
  b:b i:i r:r d:d e:n s:pl
-Each letter maps to itself,while e maps to morphological feature+N,and s maps to morhological feature pl.
A transducer mapping nouns to thier stem and morphological features
Spelling Error Detection and Correction:
-In computer based information system errors of typing and spelling causes variation between strings
-These errors are investigated and that are:
    *Single character omission
    *Insertion
    *Substitute
    *Reversal
=>Most common typing mistake
-Domearu (1964) reported that over 80% of the typing errors were single_error misspelling
   1)Substitution of a single letter
   2)omission of of a single letter
   3)Insertion of of a single letter
   4)Transposition of two adjacent letters

3b.Parts of speech tagging methods has 3 categories.

1. Rule based(linguistic)
2. Stochastic (data driven)
3. Hybrid.

Rule Based Taggers

- Use hand coded rules to assign tags to words. These rules use a lexicon to obtain a list of candidate tags and then use rules to discard incorrect tags.
- Have a 2 stage architecture:

- First stage is simply a dictionary look up procedure, which returns a set of potential tags and appropriate syntactic features for each word.
- Second stage uses a hand coded rules to discard contextually illegitimate tags to get a single part of speech for each word.

  Eg: The noun-verb ambiguity

  The show must go on.

  The potential tags for the word show in this sentence is {VB,NN}.
- If preceding word is determine THEN eliminate VB tags. This rule simply disallows verb after a determine using this rule the word show is noun only.

  In addition to contextual information, many taggers use morphological information to help in the disambiguation process.
- If word ends in _ing and preceding word is a verb THEN label it a verb(VB).
- Capitalization information can be utilized in the tagging of unknown nouns.
- Rule based tagger require supervised training
- Instead rules can be induced automatically
- To induce rules, untagged text is run through a tagger
- The output is then manually corrected
- The corrected text is then submitted to the tagger, which learns correction rules by comparing the 2 sets of data. This process may be repeated several times.
- TAGGIT (Greene and Rubin 1971) :- Initial tagging of Brown Corpus(Francis and Kucera 1982). Rule based s/m, it uses 3,300 disambiguation rules and able to tag 77% of the words in the Brown Corpus with their correct part of speech.
- Another rule based tagger is ENGTWOL (voutilainen 1995)

Advantages:

- Speed, deterministic than stochastic
- Arguments against them is the skill and effort required in writing disambiguation rules
- Stochastic taggers require manual work if good performance is to be achieved
- Rule based, time is spent in writing a rule set
- Stochastic, time is spent developing restriction on transitions and emissions to improve tagger performance

Disadvantages:

- It's usable for only one language. Using it for another one requires a rewrite of most of the programs.

2) Stochastic tagger

- Standard stochastic tagger is HMM tagger algorithm
- Markov model applies the simplifying assumption that the probability of a chain of symbols can be approximated in terms of its parts of n-grams
- Simplest n-gram model is unigram model, which assigns the most likely tag to each token
- Unigram model needs to be trained using a tagged training corpus before it can be used to tag data
- The most likely statistics are gathered over the corpus and used for tagging
- The context used by the unigram tagger is the text of the word itself

  Ex: It will assign the tag JJ for each occurrence of fast is used as an adjective than used as noun, verb or adverb.

  She had a fast[noun]

  Muslims fast[verb] during Ramadan

  Those who were injured in the accident need to be helped fast[adverb]
- Bigram tagger uses the current word and the tag of previous word in tagging process
- As the tag sequence "DT NN" is more likely than the tag sequence "DD JJ"

- In general, n gram model considers the current word and the tag of the previous n-1 words in assigning a tag to a word
- The context considered by a tri-gram model. The shaded area represents context.

Tokens	**W_{n-2}**	**W_{n-1}**	**W_n**	**W_{n+1}**
Tags	**t_{n-2}**	**t_{n-1}**	**t_n**	**t_{n+1}**

Context used by trigram tagger.

- So, far we have considered how a tag is assigned to a word given the previous tag
- The objective of a tagger is to assign a tag sequence to a given sentence

  HMM uses 2 layers of states:

  1) Visual layer corresponds to the input words
  2) Hidden layer learnt by the s/m corresponding to the tags
- While ragging the input data, observe only the words the tags are hidden
- States of the model are visible in training, not during the tagging task
- HMM makes use of lexical and bigram probabilities estimated over a tagged corpus in order to compute the most likely tag sequence for each sentence
- One way to store the statistical information is to build a probability matrix. This conations both the probability that an individual word belongs to a word clan as well as the n-gram analysis eg: For a bigram model, the probability that a word of class X follows a word of class Y. This matrix is then used to drive the HMM tagger while tagging a unknown text

Given a sequence of words, the objective is to find the most probable tag sequence for the sentence

Let w be the sequence of words.

$W=w_1,w_2,w_3,,,,,w_n$

The task is to find the tag sequence

$T=t_1,t_2,t_3,,,,,t_n$

Which maximizes P(T/W) i.e,,

$T'= \text{argmax}_T\ P(T/W)$

Applying Bayes Rule, P(T/W) can be the estimated using the expression:

$P(T/W)= P(T/W) *P(T)/P(W)$

- The probability of applying for the word sequence, P(W) remains the same for each tag sequence, we can drop it. So, the expression:

  $T'= \text{argmax}_T\ P(W/T) * P(T)$

- Using the Markov assumptions, the probability of a tag sequence can be estimated as the product of the probability of its constituent n-grams, i.e,,

  $P(T)=P(t_1)*P(t_2/t_1)*P(t_3/t_2/t_1)\ldots..*P(t_n/t_1,,,,,t_{n-1})$

P(W/T) is the probability of seeing a word sequence, given a tag sequence.

Ex: It's assigning the probability of seeing " The egg is rotten" given 'DT NNP VB JJ' we make the following two assumptions:

- The words are independent of each other
- The probability of a word is dependent only on it's tag.

Using the assumptions, we obtain

$P(W/T)=P(w_1/t_1)*P(w_2/t_2)\ldots\ldots.P(w_i/t_i) *\ldots\ldots P(w_n/t_n)$

i.e,, $P(W/T) = \pi^n_{i=1} P(w_i/t_i)$

so, $P(W/T)*P(T) = \pi^n_{i=1} \; P(w_i/t_i) * P(t_1)*P(t_2/t_1)*P(t_3/t_2/t_1)\ldots..*P(t_n/t_1,,,,,t_{n-1})$

Approximately the tag history using only the two previous tags, the transition probability, P(t) becomes

$$P(T)= (t_1)*P(t_2/t_1)*P(t_3/t_2/t_1)*\ldots..*P(t_n/t_{n-2}\; t_{n-1})$$

Hence, P(t/w) can be estimated as

$$P(w/t)*P(T) = \pi^n_{i=1} \; P(w_i/t_i) * P(t_1)*P(t_2/t_1)*P(t_3/t_2/t_1)*\ldots..* P(t_n/t_{n-2}\; t_{n-1})$$

$$= \pi^n_{i=1} P(w_i/t_i) * P(t_1)*P(t_2/t_1)*\pi^n_{i=3} P(t_i/t_{i-2}\; t_{i-1})$$

We estimate these probabilities from relative frequencies via maximum likelihood estimation.

$$P(t_i/t_{i-2}\; t_{i-1}) = \frac{C(t_{i-2}, t_{i-1}, t_i)}{C(t_{i-2}, t_{i-1})}$$

$$P(w_i/t_i) = \frac{C(w_i, ti)}{C(t_i)}$$

Where $C(t_{i-2}, t_{i-1}, t_i)$ is the number of occurrences of t i , followed by t i-2 , t i-1.

Hybrid Taggers:

Hybrid approaches to tagging combine the features of both the rule based and stochastic approaches .They use rules to assign tags to words .Like the stochastic taggers ,this is a machine learning techniques and rules are automatically induced from the data. Transformation-based learning (TBL) of tags ,also known as Brill tagging ,is a example of hybrid approach . Tranformation-based error-driven learning has been applied to a number of natural learning problems, including parts-of-speech tagging, speech generation ,and syntactic parsing .

The input to Brill's TBL tagging algorithm is a tagged corpus and a lexicon .The initial state annotator uses the lexicon to assign the most likely tag to each word as the start state.An ordered set of transformational rules are are applied sequentially.The rule that that result in the most improved tagging is selected.

A manually tagged corpus is used as reference for truth. The process is iterated until some stopping criteria is reached ,such as when no significant information is acheived over the previous iteration. At each iteration ,the tranformation that results in the highest score is selected.The output of the algorithm is a ranked list of learned transformation that transform the initial tagging close close to correct tagging .

New text can then be annotated by first assigning the most frequent tag and then applying the ranked list of learned tranformation in order.

?????fig:TBL Learner

TBL tagging algorithm:

Input:	Tagged corpus and lexicon(with most frequent information)
step1:	Label every word with most likely tag(from dictionary)
step2:	Check every possible transformation and select one which most improves tagging
step3:	Re-tag corpus applying the rules
Repeat 2-3	Until some stopping criterion is reached
RESULT:	Ranked sequence of tranformation rules

Each tranformation is a pair of re-write rule of the form t1->t2 and a contextual condition .In order to limit the set of transformations ,a small set of templates is constructed .Any allowable

tranformation is an instantiation of these templates.

@ Examples of transformation templates and rules learned by the tagger

change tag a to tag b when :

1. The preceding (following) word is tagged z.
2. The preceding (following) word is w.
3. The word two before (after) is w.
4. One of the preceding two words is w.
5. One of the two preceding (following) words is tagged z.
6. The current word is w and the preceding (following) word is x.
7. One of the previous three words is tagged z.

#	change tags		contextual condition	Example
	from	to		
1.	NN	VB	The previous tag is TO	To/TO fish/NN
2.	JJ	RB	The previous tag is VBZ	runs/VBZ fast/JJ

The rules are applied in TBL tagger,

Example::Assume that in a corpus,fish is the most likely to be a noun.

P(NN/fish) = 0.91

P(VB/fish) = 0.09

now,consider the following two sentences and their initial tags :

I/PRP like/VB to/TO eat/VB fish/NNP

I/PRP like/VB to/TO fish/NNP

The most likely tag for fish in NNP,the tagger assigns this tag to the word in both sentence .In the second case,it is a mistake .

After initial tagging when the tranformation rules are applied ,the tagger learns a rule that applies exactly to this mistagging of fish.Change NNP to VB of teh previous tag in TO.

As the contextual condition is satisfied ,this rule with change fish/NN to fish/VB:

like/VB to/TO fish/NN->like/VB to/TO fish/VB

The algorithm can be made efficient by indexing the words in a trainig corpus using potential tranformation .Most of the work in part-of-speech is done for English and some European languages.In other languages,part-of-speech tagging and

NLP research in general ,is constrained by the lack of annotated corpus.

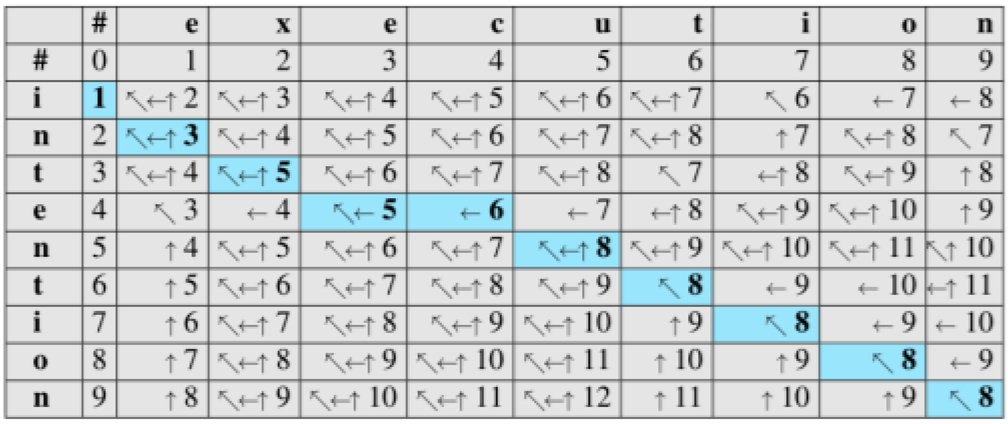
A few parts of speech tagging systems reported in recent years use morphological analysers along with a tagged corpus.

eg: Bengali tagger based on HHH developed by Sandipan et.al(2004)

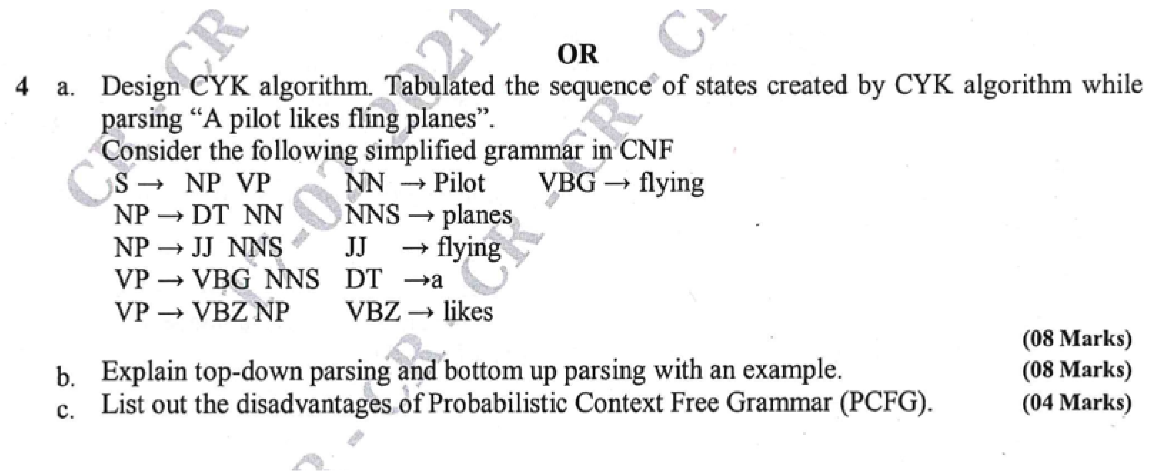
Hindi taggers developed by Smriti et.al(2006)

Smriti et.al used a decision tree based learning algorithm.

3c. Minimum Edit Distance of EXECUTION and INTENTION



	#	e	x	e	c	u	t	i	o	n
#	0	1	2	3	4	5	6	7	8	9
i	**1**	↖←↑ 2	↖←↑ 3	↖←↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖ 6	← 7	← 8
n	2	↖←↑ **3**	↖←↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ 8	↑ 7	↖←↑ 8	↖ 7
t	3	↖←↑ 4	↖←↑ **5**	↖←↑ 6	↖←↑ 7	↖←↑ 8	↖ 7	←↑ 8	↖←↑ 9	↑ 8
e	4	↖ 3	← 4	↖← **5**	← **6**	← 7	←↑ 8	↖←↑ 9	↖←↑ 10	↑ 9
n	5	↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ **8**	↖←↑ 9	↖←↑ 10	↖←↑ 11	↖↑ 10
t	6	↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ 8	↖←↑ 9	↖ **8**	← 9	← 10	←↑ 11
i	7	↑ 6	↖←↑ 7	↖←↑ 8	↖←↑ 9	↖←↑ 10	↑ 9	↖ **8**	← 9	← 10
o	8	↑ 7	↖←↑ 8	↖←↑ 9	↖←↑ 10	↖←↑ 11	↑ 10	↑ 9	↖ **8**	← 9
n	9	↑ 8	↖←↑ 9	↖←↑ 10	↖←↑ 11	↖←↑ 12	↑ 11	↑ 10	↑ 9	↖ **8**



OR

4 a. Design CYK algorithm. Tabulated the sequence of states created by CYK algorithm while parsing "A pilot likes fling planes".
Consider the following simplified grammar in CNF

S → NP VP	NN → Pilot	VBG → flying
NP → DT NN	NNS → planes	
NP → JJ NNS	JJ → flying	
VP → VBG NNS	DT → a	
VP → VBZ NP	VBZ → likes	

(08 Marks)

b. Explain top-down parsing and bottom up parsing with an example. **(08 Marks)**

c. List out the disadvantages of Probabilistic Context Free Grammar (PCFG). **(04 Marks)**

4a. Cocke Younger Kosami

Dynamic programming parsing algorithm

Bottom up approach

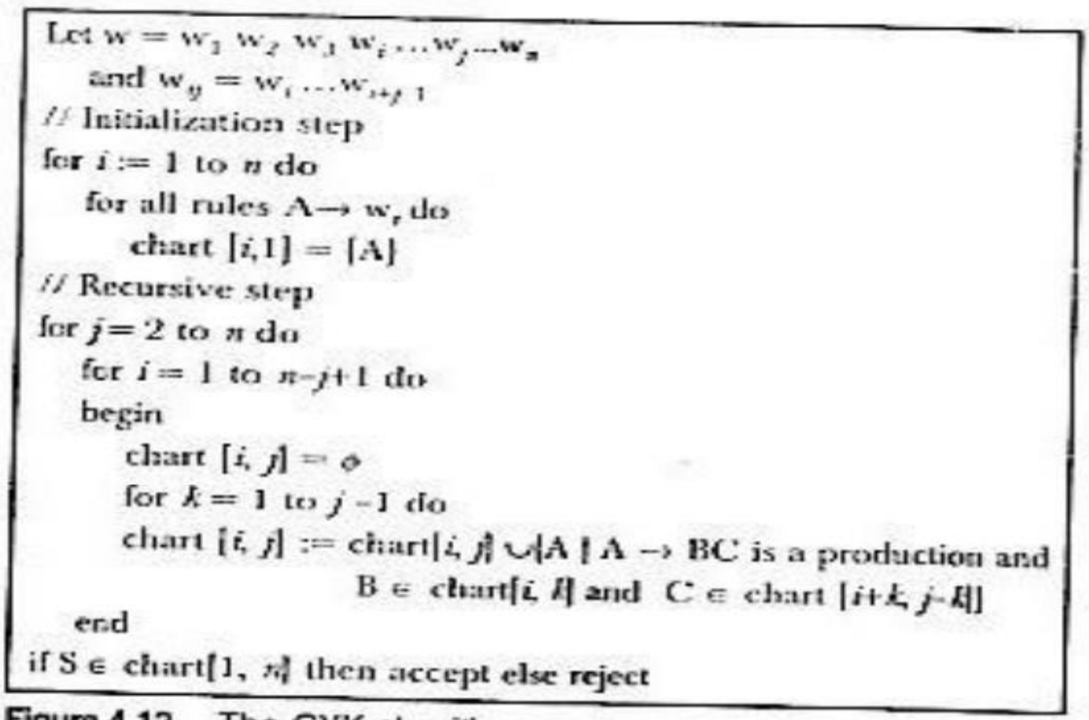
Build parse tree incrementally

Each entry based on previous entry.

CNF

A->BC

A->w



```
Let $w = w_1 w_2 w_3 w_i \dots w_j \dots w_n$
    and $w_{ij} = w_i \dots w_{i+j-1}$
// Initialization step
for $i := 1$ to $n$ do
    for all rules $A \rightarrow w_i$ do
        chart $[i,1] = \{A\}$
// Recursive step
for $j = 2$ to $n$ do
    for $i = 1$ to $n-j+1$ do
    begin
        chart $[i, j] = \emptyset$
        for $k = 1$ to $j-1$ do
        chart $[i, j]$ := chart$[i, j] \cup \{A \mid A \rightarrow BC$ is a production and
                                 $B \in$ chart$[i, k]$ and $C \in$ chart $[i+k, j-k]\}$
    end
if $S \in$ chart$[1, n]$ then accept else reject
```

Figure 4.12 The CYK algorithm

4b. Top down and Bottom up parser

1. Top down parsing

- Starts its search from the root node S and works downward towards the leaves
- The input can be derived from the designated start symbol S, of the grammar
- The next step is to find all sub trees which can start with S
- To generate the sub trees of all second level search expand the root node using all the grammar rules with S on their left hand side
- Each non-terminal symbol in the resulting sub trees is expanded next using the grammar rules having a matching non – terminal symbol on their left hand side
- The right hand side of the grammar rules provide the nodes to be generated, which are then expanded recursively

- As the expansion continues, the tree grows downward and eventually reaches a state where the bottom of the tree consists of part of speech categories
- All trees whose leaves do not match words in the input sentence are rejected, learning only trees that represents successful parses, matches exactly with the words in the input sequence.

  S→ NP VP
  S→VP
  NP→ Det Nominal NP→ noun
  NP→ Det noun PP
  Nominal→Noun
  Nominal→ Noun nominal
  VP→ Verb NP
  VP→ Verb
  PP→ preposition NP
  Det→this/that/a/the
  Verb→ sleep/sing/saw/open/paint
  Preposition→from/with/on/to
  Pronoun→she/he/they

  Sample Grammar

  Consider the grammar shown in the table and the sentence. “paint the door”

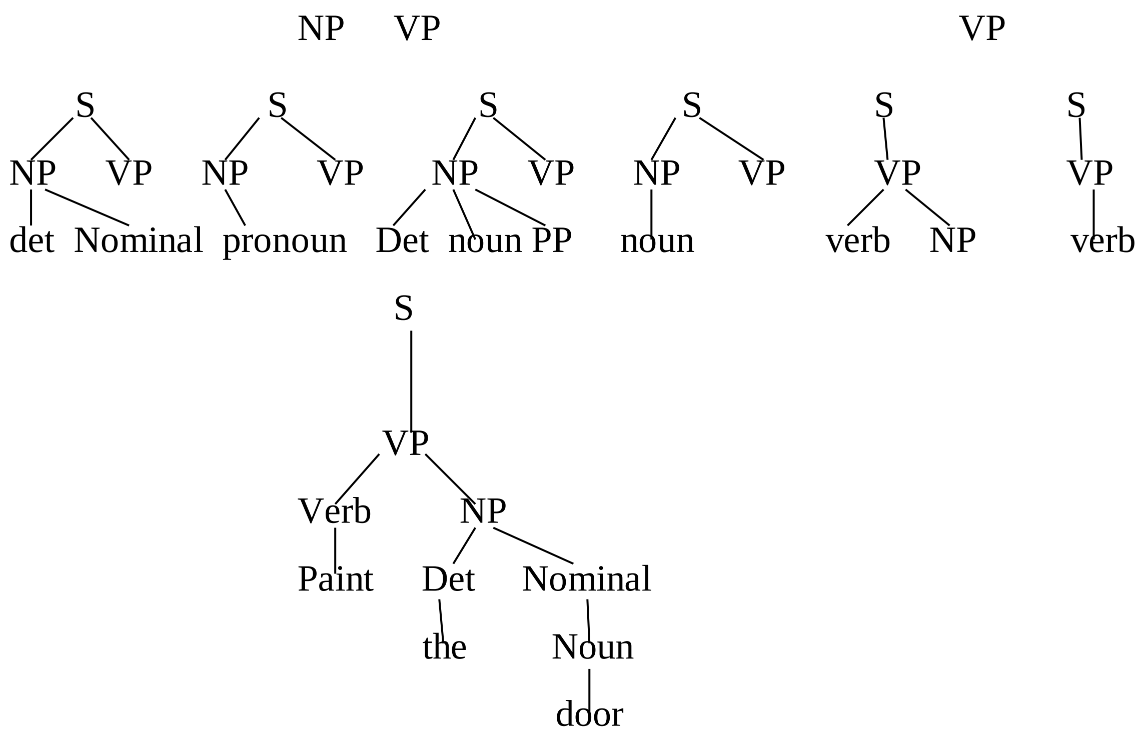
  A top down search begins with the start symbol of the grammar. Thus the first level(ply) search tree consists of a single node labeled S.

  The grammar in table has two rules with S on their left hand side. These rules are used to expand the tree, which gives us two partial trees at the second level search.

  The third level is generated by expanding the non – terminal at the bottom of the search tree is the previous ply. Due to space constraints, only the expansion corresponding to the left most non- terminals has been shown .

  The subsequent steps in the parse are left. The correct parse tree is obtained by expanding the fifth parse tree by the third level.

Level 1: S
Level 2:

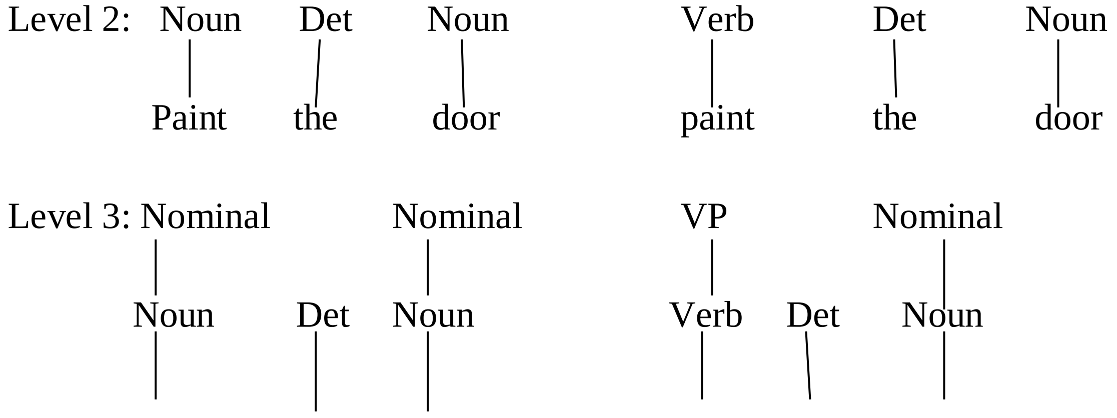


A top down search space

2. Bottom – up parsing

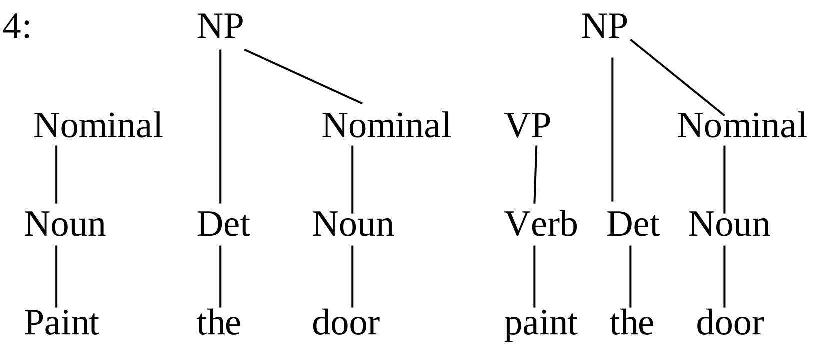
- Starts with the words is the Input sentence and attempts to construct a parse tree is an upward direction towards the root
- At each step, the parser looks for rules in grammar where the right hand side matches some of the portions in the parse tree constructed and reduces it using the left hand side of the production
- The parse is considered successful of the parser reduces the tree to the start symbol of the grammar.

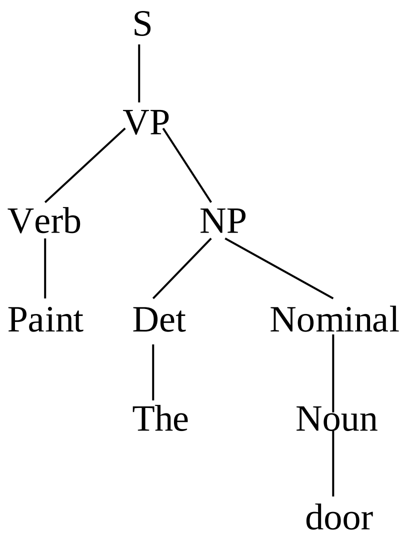
Level 1: Pain the door



Paint the door paint the door

Level 4:



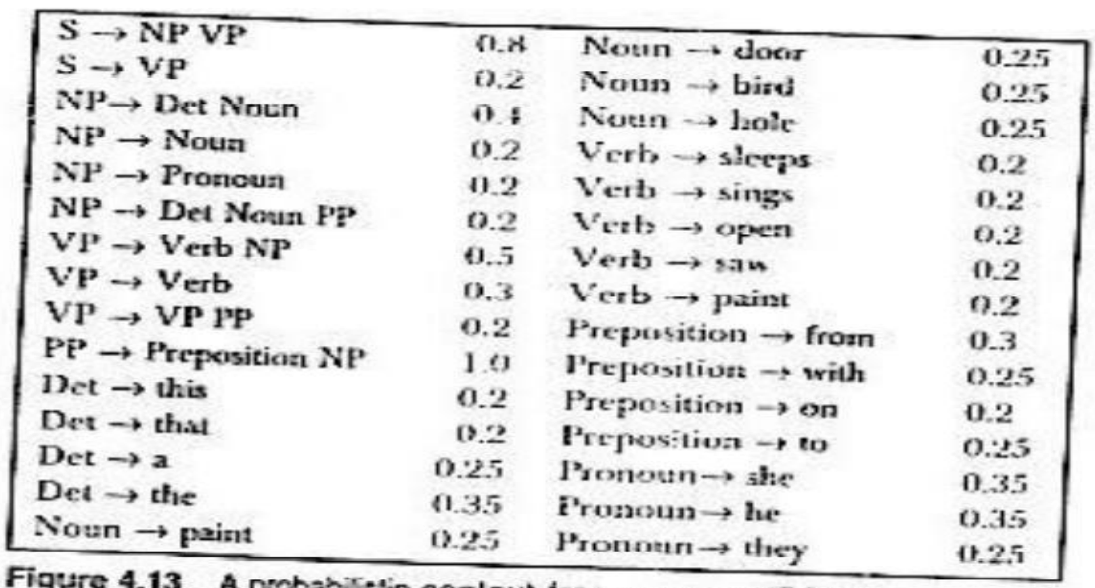


- Top down search starts generating trees with the start symbol of the grammar, it never wastes time exploiting a tree leading to a different root
- It wastes considerable time exploiting S trees that eventually result in words that are inconsistent with the input, because a top down parser generates trees before seeing the input
- Bottom up parser→Never explores a tree that does not match the input . However it wastes time generating trees that have no chance of leading to an S-rooted tree.

4c. Probabilistic Parsing

- Statistical parser- stochastic tagger-hand parsed text
- Assigning probabilities to possible parses of a sentence→ Most likely
- Find, Assign, Return

- PCFG
- Probabilistic parser→ assign probabilities to parses.
- A->α[p]
- PCFG is defined by pair(G,f) where G is a CFG and f is a positive function defined over the set of rules such that, the probabilities associated with rules expanding particular non terminal is 1
- ∑f(A->α)=1



S → NP VP	0.8	Noun → door	0.25
S → VP	0.2	Noun → bird	0.25
NP→ Det Noun	0.4	Noun → hole	0.25
NP → Noun	0.2	Verb → sleeps	0.2
NP → Pronoun	0.2	Verb → sings	0.2
NP → Det Noun PP	0.2	Verb → open	0.2
VP → Verb NP	0.5	Verb → saw	0.2
VP → Verb	0.3	Verb → paint	0.2
VP → VP PP	0.2	Preposition → from	0.3
PP → Preposition NP	1.0	Preposition → with	0.25
Det → this	0.2	Preposition → on	0.2
Det → that	0.2	Preposition → to	0.25
Det → a	0.25	Pronoun→ she	0.35
Det → the	0.35	Pronoun→ he	0.35
Noun → paint	0.25	Pronoun→ they	0.25

Figure 4.13 A probabilistic context-free grammar (PCFG)

[illegible]
the probabilities for each of the rules used in generating the parse tree:

$$P(\varphi, s) = \prod_{n \in \varphi} p\{r(n)\}$$

where n is a node in the parse tree φ and r is the rule used to expand n.

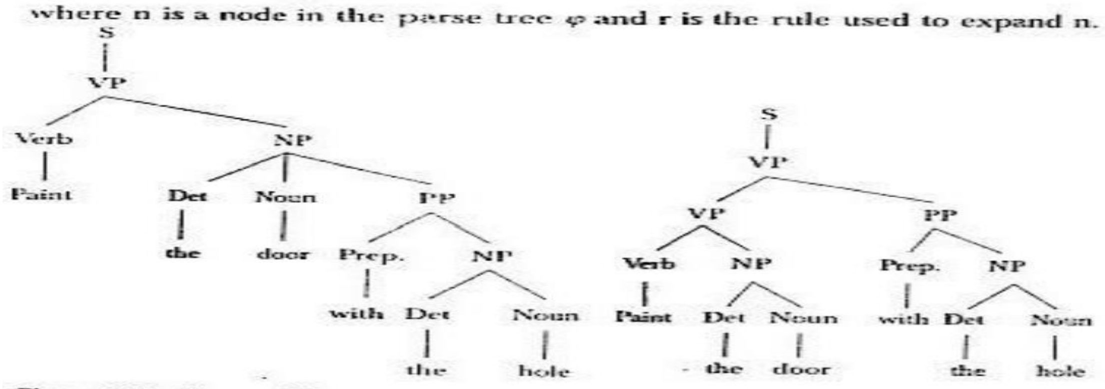


Figure 4.14 Two possible parse trees

Module-3

5 a. Explain the four patterns used to extract relationship between two entries with an example for each. **(08 Marks)**

b. Explain a dependency path Kernel for Relation Extraction. **(08 Marks)**

c. Discuss the knowledge roles for below sentences with the same domain concepts.

i) The calculated <u>insulating</u> <u>resistance</u> values lay in the safe operating area

ii) Compared to the last examination, lower values for the <u>insulating</u> <u>resistance</u> were ascertained due to dirtiness at the surface. **(04 Marks)**

5a.

Computing the dot-product (i.e., the kernel) between the features vectors associated with two relation examples amounts to calculating the number of common anchored subsequences between the two sentences. This is done efficiently by modifying the dynamic programming algorithm used in the string kernel from [2] to account only for common sparse subsequences constrained to contain the two protein-name tokens. The feature space is further prunned down by utilizing the following property of natural language statements: when a sentence asserts a relationship between two entity mentions, it generally does this using one of the following four patterns:

- **[FB]** **F**ore–**B**etween: words before and between the two entity mentions are simultaneously used to express the relationship. Examples: 'interaction of $\langle P_1 \rangle$ with $\langle P_2 \rangle$,' 'activation of $\langle P_1 \rangle$ by $\langle P_2 \rangle$.'
- **[B]** **B**etween: only words between the two entities are essential for asserting the relationship. Examples: '$\langle P_1 \rangle$ interacts with $\langle P_2 \rangle$,' '$\langle P_1 \rangle$ is activated by $\langle P_2 \rangle$.'
- **[BA]** **B**etween–**A**fter: words between and after the two entity mentions are simultaneously used to express the relationship. Examples: '$\langle P_1 \rangle$ – $\langle P_2 \rangle$ complex,' '$\langle P_1 \rangle$ and $\langle P_2 \rangle$ interact.'
- **[M]** **M**odifier: the two entity mentions have no words between them. Examples: *U.S. troops* (a ROLE:STAFF relation), *Serbian general* (ROLE:CITIZEN).

5b.

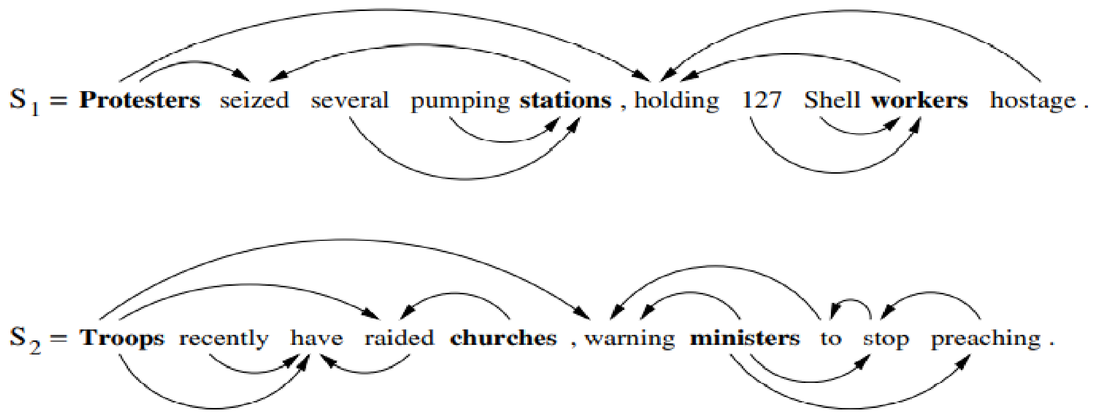

S_1 = Protesters seized several pumping stations , holding 127 Shell workers hostage .
S_2 = Troops recently have raided churches , warning ministers to stop preaching .

Fig. 3.4. Sentences as dependency graphs.

these examples reveals that all relevant words form a shortest path between the two entities in a graph structure where edges correspond to relations between a word (head) and its dependents. For example, Figure 3.4 shows the full dependency graphs for two sentences from the ACE (Automated Content Extraction) newspaper corpus [12], in which words are represented as nodes and word-word dependencies are represented as directed edges. A subset of these word-word dependencies capture the predicate-argument relations present in the sentence. Arguments are connected to their target predicates either directly through an arc pointing to the predicate ('troops $\rightarrow$ raided'), or indirectly through a preposition or infinitive particle ('warning $\leftarrow$ to $\leftarrow$ stop'). Other types of word-word dependencies account for modifier-head relationships present in adjective-noun compounds ('several $\rightarrow$ stations'), noun-noun compounds ('pumping $\rightarrow$ stations'), or adverb-verb constructions ('recently $\rightarrow$ raided').

Word-word dependencies are typically categorized in two classes as follows:

- [**Local Dependencies**] These correspond to local predicate-argument (or head-modifier) constructions such as 'troops $\rightarrow$ raided', or 'pumping $\rightarrow$ stations' in Figure 3.4.
- [**Non-local Dependencies**] Long-distance dependencies arise due to various linguistic constructions such as coordination, extraction, raising and control. In Figure 3.4, among non-local dependencies are 'troops $\rightarrow$ warning', or 'ministers $\rightarrow$ preaching'.

5c

(1) He had no regrets for **his** actions in **Brcko**.

his → actions ← in ← **Brcko**

(2) U.S. **troops** today acted for the first time to capture an alleged Bosnian war criminal, rushing from unmarked vans parked in the northern Serb-dominated **city** of Bijeljina.

troops → rushing ← from ← vans → parked ← in ← **city**

(3) Jelisic created an atmosphere of terror at the **camp** by killing, abusing and threatening the **detainees**.

detainees → killing ← Jelisic → created ← at ← **camp**
detainees → abusing ← Jelisic → created ← at ← **camp**
detainees → threatning ← Jelisic → created ← at ← **camp**
detainees → killing → by → created ← at ← **camp**
detainees → abusing → by → created ← at ← **camp**
detainees → threatening → by → created ← at ← **camp**

OR

6 a. With a neat diagram, explain the architecture used in the task of learning to annotate cases with knowledge Roles. **(10 Marks)**

b. Explain Functional overview of Infact system with a neat diagram. **(10 Marks)**

6a.

4.4 Learning to Annotate Cases with Knowledge Roles

To perform the task of learning to annotate cases with knowledge roles, we implemented a software framework, as shown in Figure 4.5. Only the preparation of documents (described in Section 4.4.1) is performed outside of this framework. In the remainder of the section, every component of the framework is presented in detail.

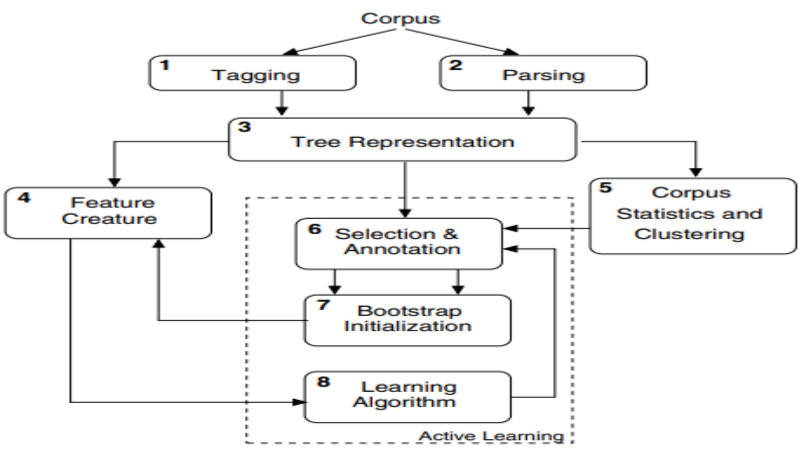


Fig. 4.5. The Learning Framework Architecture.

4.4.1 Document Preparation

In Section 4.2.1 it was mentioned that our documents are official diagnostic reports hierarchically structured in several sections and subsections, written by using MS® Word. Actually, extracting text from such documents, while preserving the content structure, is a difficult task. In completing it we were fortunate twice. First, with MS® Office 2003 the XML based format WordML was introduced that permits storing MS® Word documents directly in XML. Second, the documents were originally created using a MS® Word document template, so that the majority of them had the same structure. Still, many problems needed to be handled. MS® Word mixes formatting instructions with content very heavily and this is reflected also in its XML format. In addition, information about spelling, versioning, hidden template elements, and so on are also stored. Thus, one needs to explore the XML output of the documents to find out how to distinguish text and content structure from unimportant information. Such a process will always be a heuristic one, depending on the nature of the documents. We wrote a program that reads the XML document tree,

and for each section with a specified label (from the document template) it extracts the pure text and stores it in a new XML document, as the excerpt in Figure 4.6 shows.

4.4.2 Tagging

The part-of-speech (POS) tagger (TreeTagger[4]) that we used [26] is a probabilistic tagger with parameter files for tagging several languages: German, English, French, or Italian. For some small problems we encountered, the author of the tool was very cooperative in providing fixes. Nevertheless, our primary interest in using the tagger was not the POS tagging itself (the parser, as is it shown in Section 4.4.3, performs tagging and parsing), but getting stem information (since the German language has a very rich morphology) and dividing the paragraphs in sentences (since the sentence is the unit of operation for the next processing steps).

The tag set used for tagging German is slightly different from that of English.[5] Figure 4.7 shows the output of the tagger for a short sentence.[6]

As indicated in Figure 4.7, to create sentences it suffices to find the lines containing: `".      \$.      ."` (one sentence contains all the words between two such lines). In general, this is a very good heuristic, but its accuracy depends on the nature of the text. For example, while the tagger correctly tagged abbreviations found in its list of abbreviations (and the list of abbreviations can be customized by adding abbreviations common to the domain of the text), it got confused when the same abbreviations were found inside parentheses, as the examples in Figure 4.8 for the word ‘ca.’ (circa) show.

4.4.3 Parsing

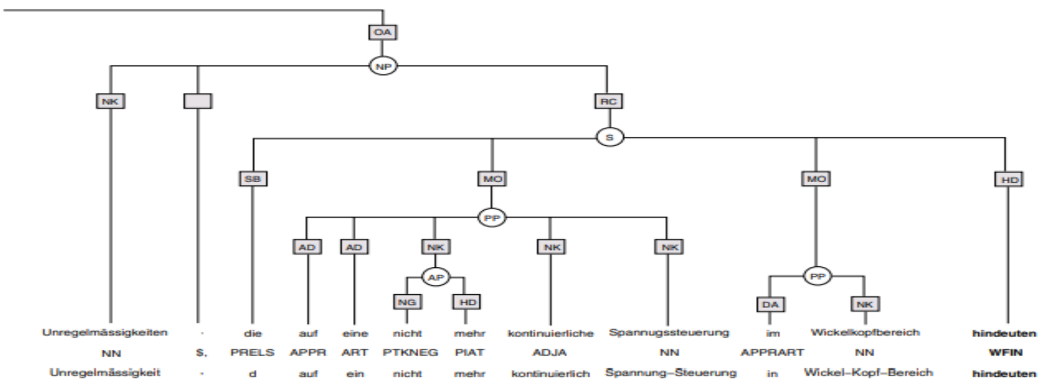
Syntactical parsing is one of the most important steps in the learning framework, since the produced parse trees serve as input for the creation of features used for learning. Since we are interested in getting qualitative parsing results, we experimented with three different parsers: the Stanford parser (Klein 2005), the BitPar parser [27, 25], and the Sleepy parser [7]. What these parsers have in common is that they all are based on unlexicalized probabilistic context free grammars (PCFG) [18], trained on the same corpus of German, Negra[7] (or its superset Tiger[8]), and their source code is publicly available. Still, they do differ in the degree they model some structural aspects of the German language, their annotation schemas, and the infor-

[7] http://www.coli.uni-saarland.de/projects/sfb378/negra-corpus/
[8] http://www.ims.uni-stuttgart.de/projekte/TIGER/TIGERCorpus/

mation included in the output. Figure 4.9 shows the output of the same sentence[9] parsed by each parser, and in the following, we discuss each of them.

4.4.4 Tree Representation

The bracketed parse tree and the stem information of tagging serve as input for the step of creating a tree data structure. The tree is composed of terminals (leaf nodes) and non-terminals (internal nodes), all of them known as constituents of the tree. For export purposes as well as for performing exploration or annotation of the corpus, the tree data structures are stored in XML format, according to a schema defined in the TigerSearch[10] tool. The created tree, when visualized in TigerSearch, looks like the one shown in Figure. 4.10.[11] The terminals are labeled with their POS tags and also contain the corresponding words and stems; the inside nodes are labeled with their phrase types (NP, PP, etc.); and the branches have labels, too, corresponding to the grammatical functions of the nodes. The XML representation of a portion of the tree is shown in Figure 4.11.


OA
NP
NK
RC
S
SB
MO
MO
HD
PP
AD
AD
NK
NK
NK
AP
NG
HD
PP
DA
NK
Unregelmässigkeiten NN Unregelmässigkeit
. $, .
die PRELS d
auf APPR auf
eine ART ein
nicht PTKNEG nicht
mehr PIAT mehr
kontinuierliche ADJA kontinuierlich
Spannugssteuerung NN Spannung–Steuerung
im APPRART in
Wickelkopfbereich NN Wickel–Kopf–Bereich
hindeuten WFIN hindeuten

4.4.5 Feature Creation

Features are created from the parse tree of a sentence. A feature vector is created for every constituent of the tree, containing some features unique to the constituent, some features common to all constituents of the sentence, and some others calculated with respect to the target constituent (the predicate verb).

4.4.6 Annotation

To perform the manual annotation, we used the Salsa annotation tool (publicly available) [11]. The Salsa annotation tool reads the XML representation of a parse tree and displays it as shown in Figure 4.12. The user has the opportunity to add frames and roles as well as to attach them to a desired target verb. In the example of Figure 4.12 (the same sentence of Figure 4.10), the target verb *hindeuten* (point to) evokes the frame Evidence, and three of its roles have been assigned to constituents of the tree. Such an assignment can be easily performed using point-and-click. After this

process, an element <frames> is added to the XML representation of the sentence, containing information about the frame. Excerpts of the XML code are shown in Figure 4.13.

4.4.7 Active Learning

Research in IE has indicated that using an active learning approach for acquiring labels from a human annotator has advantages over other approaches of selecting instances for labeling [16]. In our learning framework, we have also implemented an active learning approach. The possibilities for designing an active learning strategy are manifold; the one we have implemented uses a committee-based classification scheme that is steered by corpus statistics. The strategy consists of the following steps:

a) Divide the corpus in clusters of sentences with the same target verb. If a cluster has fewer sentences than a given threshold, group sentences with verbs evoking the same frame into the same cluster.
b) Within each cluster, group the sentences (or clauses) with the same parse sub-tree together.
c) Select sentences from the largest groups of the largest clusters and present them to the user for annotation.
d) Bootstrap initialization: apply the labels assigned by the user to groups of sentences with the same parse sub-tree.
e) Train all the classifiers of the committee on the labeled instances; apply each trained classifier to the unlabeled sentences.
f) Get a pool of instances where the classifiers of the committee disagree and present to the user the instances belonging to sentences from the next largest clusters not yet manually labeled.
g) Repeat steps d)–f) a few times until a desired accuracy of classification is achieved.

6b.

5.2 InFact System Overview

InFact consists of an indexing and a search module. With reference to Figure 5.1, indexing pertains to the processing flow on the bottom of the diagram. InFact models text as a complex multivariate object using a unique combination of deep parsing, linguistic normalization and efficient storage. The storage schema addresses the fundamental difficulty of reducing information contained in parse trees into generalized data structures that can be queried dynamically. In addition, InFact handles the problem of linguistic variation by mapping complex linguistic structures into semantic and syntactic equivalents. This representation supports dynamic relationship and event search, information extraction and pattern matching from large document collections in real time.

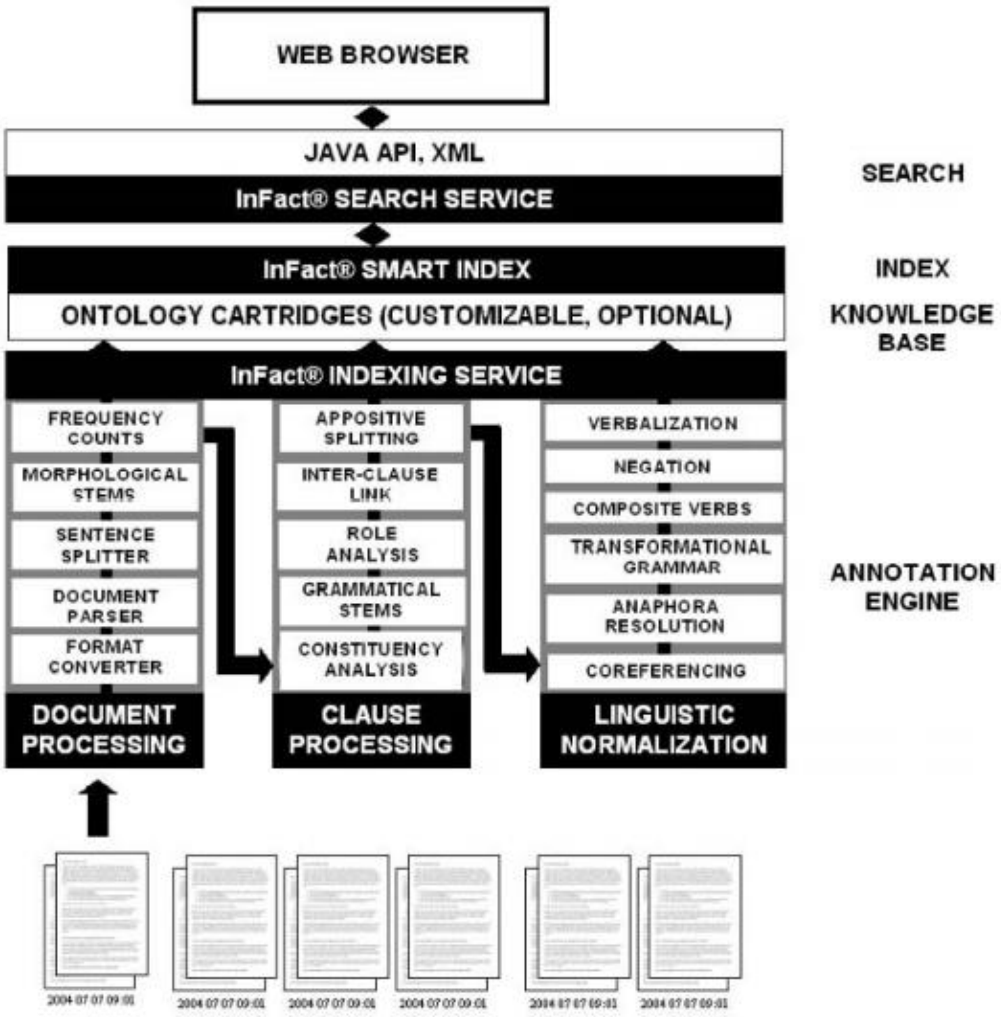


Fig. 5.1. Functional overview of InFact.

Document Processing

The first step in document processing is format conversion, which we handle through our native format converters, or optionally via search export conversion software from Stellant^{TM} (www.stellent.com), which can convert 370 different input file types. Our customized document parsers can process disparate styles and recognized zones within each document. Customized document parsers address the issue that a Web page may not be the basic unit of content, but it may consist of separate sections with an associated set of relationships and metadata. For instance a blog post may contain blocks of text with different dates and topics. The challenge is to automatically recognize variations from a common style template, and segment information in the index to match zones in the source documents, so the relevant section can be displayed in response to a query. Next we apply logic for sentence splitting in preparation for clause processing. Challenges here include the ability to unambiguously recognize sentence delimiters, and recognize regions such as lists or tables that

are unsuitable for deep parsing. Last, we extract morphological stems and compute frequency counts, which are then entered in the index.

Clause Processing

The indexing service takes the output of the sentence splitter and feeds it to a deep linguistic parser. A sentence may consist of multiple clauses. Unlike traditional models that store only term frequency distributions, InFact performs clause level indexing and captures syntactic category and roles for each term, and grammatical constructs, relationships, and inter-clause links that enable it to understand events. One strong differentiator of our approach to information extraction [4, 5, 7, 8, 14, 19] is that we create these indices automatically, without using predefined extraction rules, and we capture all information, not just predefined patterns. Our parser performs a full constituency and dependency analysis, extracting part-of-speech (POS) tags and grammatical roles for all tokens in every clause. In the process, tokens undergo grammatical stemming and an optional, additional level of tagging. For instance, when performing grammatical stemming on verb forms, we normalize to the infinitive, but we may retain temporal tags (e.g., past, present, future), aspect tags (e.g., progressive, perfect), mood/modality tags (e.g., possibility, subjunctive, irrealis, negated, conditional, causal) for later use in search.

Next we capture inter-clause links, through: 1) explicit tagging of conjunctions or pronouns that provide the link between the syntactic structures for two adjacent clauses in the same sentence; and 2) pointing to the list of annotated keywords in the antecedent and following sentence. Note that the second mechanism ensures good recall in those instances where the parser fails to produce a full parse tree for long and convoluted sentences, or information about an event is spread across adjacent sentences. In addition, appositive clauses are recognized, split into separate clauses and cross-referenced to the parent clause.

For instance, the sentence: "Appointed commander of the Continental Army in 1775, George Washington molded a fighting force that eventually won independence from Great Britain" consists of three clauses, each containing a governing verb (appoint, mold, and win). InFact decomposes it into a primary clause ("George Washington molded a fighting force") and two secondary clauses, which are related to the primary clause by an appositive construct ("Appointed commander of the Continental Army in 1775") and a pronoun ("that eventually won independence from Great Britain"), respectively. Each term in each clause is assigned a syntactic category or POS tag (e.g., noun, adjective, etc.) and a grammatical role tag (e.g., subject, object, etc.). InFact then utilizes these linguistic tags to extract relationships that are normalized and stored in an index, as outlined in the next two sections.

Table 5.2. The index abstraction of a sentence.

Subject	Subject-Modifier	Object	Object-Modifier	Verb	Verb-Modifier	Prep	Pcomp	Dist	Neg
		Washington	George	appoint				1	F
		commander		appoint				1	F
		Army	Continental	appoint				3	F
				appoint		in	1775	2	F
Washington	George	force	fighting	mold				2	F
force	fighting	independence		win				1	F
				win		from	Great Britain	3	F
				win	eventually			1	F

InFact stores the normalized triplets into dedicated index structures that

- are optimized for efficient keyword search
- are optimized for efficient cross-document retrieval of arbitrary classes of relationships or events (see examples in the next section)
- store document metadata and additional ancillary linguistic variables for filtering of search results by metadata constraints (e.g., author, date range), or by linguistic attributes (e.g., retrieve negated actions, search subject modifier field in addition to primary subject in a relationship search)
- (optionally) superimposes annotations and taxonomical dependencies from a custom ontology or knowledge base.

Module-4

7 a. Explain the functioning of Word Matching Feedback Systems. (08 Marks)

b. Discuss iSTART system and their modules. (08 Marks)

c. Illustrate Topic Models (TM) Feedback system. (04 Marks)

7a.

6.2.1 Word Matching Feedback Systems

Word matching is a very simple and intuitive way to estimate the nature of a self-explanation. In the first version of iSTART, several hand-coded components were built for each practice text. For example, for each sentence in the text, the "important words" were identified by a human expert and a length criterion for the explanation was manually estimated. Important words were generally content words that were deemed important to the meaning of the sentence and could include words not found in the sentence. For each important word, an association list of synonyms and related terms was created by examining dictionaries and existing protocols as well as by human judgments of what words were likely to occur in a self-explanation of the sentence. In the sentence "All thunderstorms have a similar life history," for example, important words are *thunderstorm, similar, life*, and *history*. An association list for *thunderstorm* would include *storms, moisture, lightning, thunder, cold, tstorm, t-storm, rain, temperature, rainstorms*, and *electric-storm*. In essence, the attempt was made to imitate LSA.

A trainee's explanation was analyzed by matching the words in the explanation against the words in the target sentence and words in the corresponding association lists. This was accomplished in two ways: (1) Literal word matching and (2) Soundex matching.

Literal word matching - Words are compared character by character and if there is a match of the first 75% of the characters in a word in the target sentence (or its association list) then we call this a literal match. This also includes removing suffix -s, -d, -ed, -ing, and -ion at the end of each words. For example, if the trainee's self-explanation contains 'thunderstom' (even with the misspelling), it still counts as a literal match with words in the target sentence since the first nine characters are exactly the same. On the other hand, if it contains 'thunder,' it will not get a match with the target sentence, but rather with a word on the association list.

Soundex matching - This algorithm compensates for misspellings by mapping similar characters to the same soundex symbol [1, 5]. Words are transformed to their soundex code by retaining the first character, dropping the vowels, and then converting other characters into soundex symbols. If the same symbol occurs more than once consecutively, only one occurrence is retained. For example, 'thunderstorm' will be transformed to 't8693698'; 'communication' to 'c8368.' Note that the later example was originally transformed to 'c888368' and two 8s were dropped ('m' and 'n' are both mapped to '8'). If the trainee's self-explanation contains 'thonderstorm' or 'tonderstorm,' both will be matched with 'thunderstorm' and this is called a soundex match. An exact soundex match is required for short words (i.e., those with fewer than six alpha-characters) due to the high number of false alarms when soundex is used. For longer words, a match on the first four soundex symbols suffices. We are considering replacing this rough and ready approach with a spell-checker.

7b.

6.3 iSTART: Evaluation of Feedback Systems

Two experiments were used to evaluate the performance of various systems of algorithms that vary as a function of approach (word-based, LSA, combination of word-based and LSA, and combination of word-based TM). In Experiment 1, we

compare all eight systems in terms of the overall quality score by applying each system to a database of self-explanation protocols produced by college students. The protocols had been evaluated by a human expert on overall quality. In Experiment 2, we investigated two systems using a database of explanations produced by middle-school students. These protocols were scored to identify particular reading strategies.

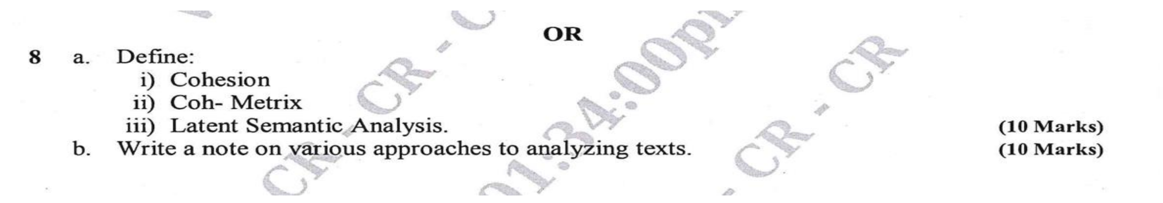
7c.

6.2.3 Topic Models (TM) Feedback System

The Topic Models approach (TM; [10, 27]) applies a probabilistic model in finding a relationship between terms and documents in terms of topics. A document is conceived of as having been generated probabilistically from a number of topics and each topic consists of number of terms, each given a probability of selection if that topic is used. By using a TM matrix, we can estimate the probability that a certain topic was used in the creation of a given document. If two documents are similar, the estimates of the topics they probably contain should be similar. TM is very similar to LSA, except that a term-document frequency matrix is factored into two matrices instead of three.

$$\{X_{normalized}\} = \{W\}\{D\} \tag{6.4}$$

The dimension of matrix $\{W\}$ is W x T, where W is the number of words in the corpus and T is number of topics. The number of topics varies, more or less, with the size of corpus; for example, a corpus of 8,000 documents may require only 50 topics while a corpus of 40,000 documents could require about 300 topics. We use the TM Toolbox [28] to generate the $\{W\}$ or TM matrix, using the same science corpus as we used for the LSA matrix. In this construction, the matrix $\{X\}$ is for all terms in the corpus, not just those appearing in two different documents. Although matrix $\{X\}$ is supposed to be normalized, the TM toolbox takes care of this normalization and outputs for each topic, the topic probability, and a list of terms in this topic along with their probabilities in descending order (shown in Table 6.1). This output is easily transformed into the term-topic-probability matrix.



OR

8 a. Define:
i) Cohesion
ii) Coh- Metrix
iii) Latent Semantic Analysis. **(10 Marks)**
b. Write a note on various approaches to analyzing texts. **(10 Marks)**

8a.

7.2 Cohesion

Recent research in text processing has emphasized the importance of the cohesion of a text in comprehension [5, 44, 43]. Cohesion is the degree to which ideas in the text are explicitly related to each other and facilitate a unified situation model for the reader. As McNamara and colleagues have shown, challenging text (such as science) is particularly difficult for low-knowledge students. These students are cognitively burdened when they are forced to make inferences across texts [22, 34, 35, 38, 44]. Adding cohesion to text alleviates this burden by filling conceptual and structural gaps. Recent developments in computational linguistics and discourse processing have now made it possible to measure this textual cohesion. These developments

have come together in a computational tool called *Coh-Metrix* [22] that approximates over 200 indices of textual cohesion and difficulty. Armed with this technology, text-book writers and teachers have the opportunity to better assess the appropriateness of a text for particular students [47], and researchers have the opportunity to assess cohesion patterns in text-types so as to better understand what constitutes a prototypical text from any given domain, genre, register, or even author [37, 42, 12, 14].

7.3 Coh-Metrix

Coh-Metrix assesses characteristics of texts by using parts of speech classifiers [4, 49, 7, 8, 9, 10, 11], and latent semantic analysis [32, 33]. The indices generated from Coh-Metrix offer an assessment of the cohesiveness and readability of any given text . These indices have been used to indicate textual cohesion and difficulty levels in a variety of studies. For example, Ozuru et al. [47] used Coh-Metrix to rate high and low cohesion versions of biology texts, the study showing that participants benefited most from the high cohesion versions. And Best, Floyd, and McNamara [1] used Coh-Metrix to compare 61 third-graders' reading comprehension for narrative and expository texts, the study suggesting that children with low levels of world knowledge were more inclined to have comprehension problems with expository texts. While research into assessing the benefits of cohesion continues apace, the utility of the Coh-Metrix tool has also allowed the pursuit of other avenues of textual investigation.

approach.

LSA is a technique that uses a large corpus of texts together with singular value decomposition to derive a representation of world knowledge [33]. LSA is based on the idea that any word (or group of words) appears in some contexts but not in others. Thus, words can be compared by the aggregate of their co-occurrences. This aggregate serves to determine the degree of similarity between such words [13]. LSA's practical advantage over shallow word overlap measures is that it goes beyond lexical similarities such as *chair/chairs* or *run/ran*, and manages to rate the relative semantic similarity between terms such as *chair/table*, *table/wood*, and *wood/forest*. As such, LSA does not only tell us whether two items are the same, it tells us how similar they are. Further, as Wolfe and Goldman [54] report, there is substantial evidence to support the notion that the reliability of LSA is not significantly different from human raters when asked to perform the same judgments.

8b.

7.4 Approaches to Analyzing Texts

Traditional approaches to categorizing discourse have tended to treat text as if it were a homogeneous whole. These wholes, or bodies of text, are analyzed for various textual features, which are used to classify the texts as belonging to one category or another [2, 3, 26, 27, 37, 42]. To be sure, such approaches have yielded impressive findings, generally managing to significantly discriminate texts into categories such as dialect, domain, genre, or author. Such discrimination is made possible because

different kinds of texts feature different quantities of features. For example, Biber [2] identified *if clauses* and singular person pronoun use as key predictors in distinguishing British- from American-English. Louwerse et al. [37] used cohesion scores generated from Coh-Metrix to distinguish both spoken from written texts and narratives from non-narratives. And Stamatatos, Fakotatos, and Kokkinakis [50] used a number of style markers including punctuation features and frequencies of verb- and noun-phrases to distinguish between the authors of a variety of newspaper columns. Clearly, discriminating texts by treating them as homogenous wholes has a good track record. However, texts tend to be *heterogeneous*, and treating them as such may substantially increase the power of corpus analyses.

Module-5

9 a. Explain design features of information retrieval systems, with a neat diagram. **(10 Marks)**

b. Define term weighting. Consider a document represented by the 3 terms {tornado, swirl wind} with the raw tf 4, 1 and 1 respectively. In a collection of 100 documents 15 documents contains the term tornado, 20 contains swirl and 10 contains wind. Find the idf and term weight of the 3 terms. **(06 Marks)**

c. Explain the benefits of eliminating stop words. Give examples in which stop word elimination may be harmful. **(04 Marks)**

9a.

9.2 DESIGN FEATURES OF INFORMATION RETRIEVAL SYSTEMS

Figure 9.1 illustrates the basic process of IR. It begins with the user's information need. Based on this need, he/she formulates a query. The IR system returns documents that seem relevant to the query. This is an engineering account of the IR system. The basic question involved is, 'what constitutes the information in the documents and the queries'. This, in turn is related to the problem of representation of documents and queries. The retrieval is performed by matching the query representation with document representation.

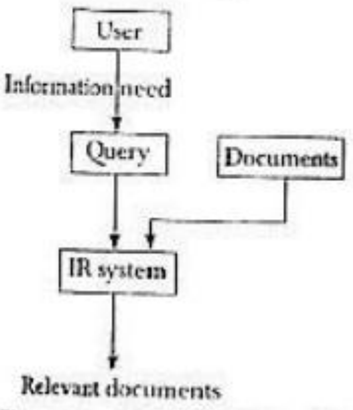
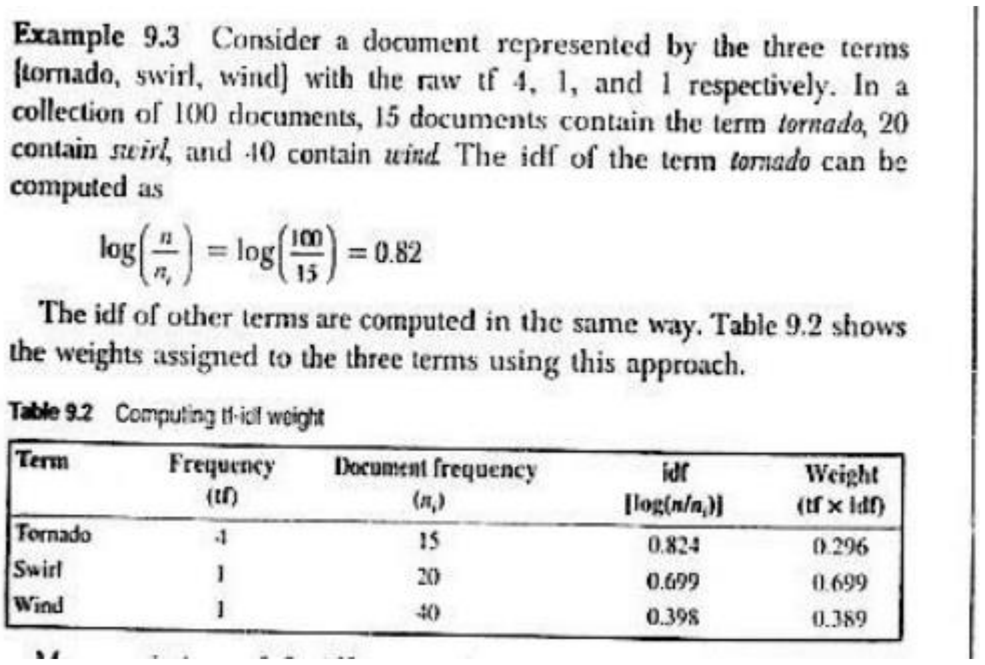


Figure 9.1 Basic information retrieval process

9.2.1 Indexing

In a small collection of documents, an IR system can access a document to decide its relevence to a query. However, in a large collection of documents, this technique poses practical problems. Hence, a collection of raw documents is usually transformed into an easily accessible representation. This process is known as indexing. Most indexing techniques involve identifying good document descriptors, such as keywords or terms, which describe the information content of documents. A good descriptor is one that helps describe the content of the document and discriminate it from other documents in the collection. Luhn (1957, 1958) is considered the first person to advance the notion of automatic indexing of documents based on their content. He assumed that the frequency of certain word-occurrences in an article gave meaningful

9b.



Example 9.3 Consider a document represented by the three terms {tornado, swirl, wind} with the raw tf 4, 1, and 1 respectively. In a collection of 100 documents, 15 documents contain the term *tornado*, 20 contain *swirl*, and 40 contain *wind*. The idf of the term *tornado* can be computed as

$$\log\left(\frac{n}{n_i}\right) = \log\left(\frac{100}{15}\right) = 0.82$$

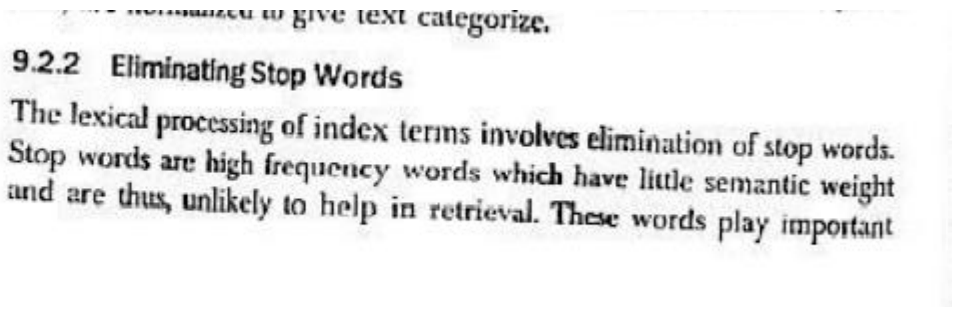
The idf of other terms are computed in the same way. Table 9.2 shows the weights assigned to the three terms using this approach.

Table 9.2 Computing tf-idf weight

Term	Frequency (tf)	Document frequency (n_i)	idf [$\log(n/n_i)$]	Weight (tf × idf)
Tornado	4	15	0.824	0.296
Swirl	1	20	0.699	0.699
Wind	1	40	0.398	0.389

.

9c.

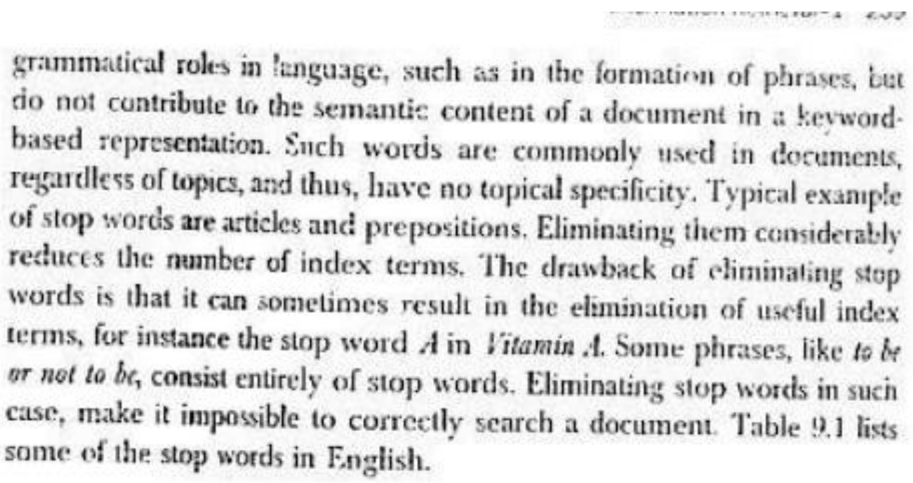


... normalized to give text categorize.

9.2.2 Eliminating Stop Words

The lexical processing of index terms involves elimination of stop words. Stop words are high frequency words which have little semantic weight and are thus, unlikely to help in retrieval. These words play important

.



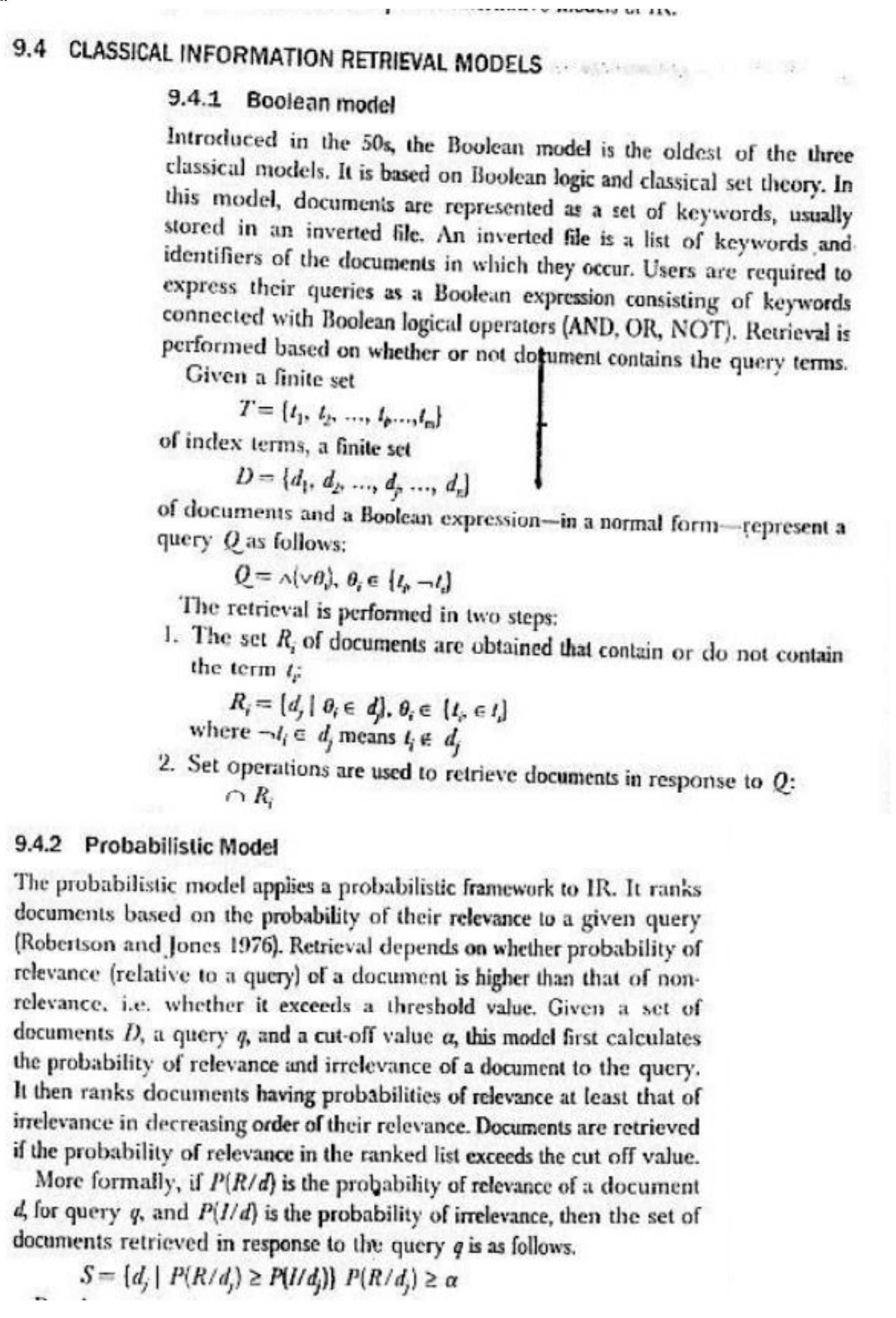
grammatical roles in language, such as in the formation of phrases, but do not contribute to the semantic content of a document in a keyword-based representation. Such words are commonly used in documents, regardless of topics, and thus, have no topical specificity. Typical example of stop words are articles and prepositions. Eliminating them considerably reduces the number of index terms. The drawback of eliminating stop words is that it can sometimes result in the elimination of useful index terms, for instance the stop word *A* in *Vitamin A*. Some phrases, like *to be or not to be*, consist entirely of stop words. Eliminating stop words in such case, make it impossible to correctly search a document. Table 9.1 lists some of the stop words in English.

OR

10 a. List different IR models. Explain classical Information Retrieval models. **(10 Marks)**

b. Explain Wordnet and list the applications of Wordnet. **(10 Marks)**

10.a.



9.4 CLASSICAL INFORMATION RETRIEVAL MODELS

9.4.1 Boolean model

Introduced in the 50s, the Boolean model is the oldest of the three classical models. It is based on Boolean logic and classical set theory. In this model, documents are represented as a set of keywords, usually stored in an inverted file. An inverted file is a list of keywords and identifiers of the documents in which they occur. Users are required to express their queries as a Boolean expression consisting of keywords connected with Boolean logical operators (AND, OR, NOT). Retrieval is performed based on whether or not document contains the query terms.

Given a finite set

$$T = \{t_1, t_2, \ldots, t_i, \ldots, t_m\}$$

of index terms, a finite set

$$D = \{d_1, d_2, \ldots, d_j, \ldots, d_n\}$$

of documents and a Boolean expression—in a normal form—represent a query Q as follows:

$$Q = \wedge(\vee\theta_i),\ \theta_i \in \{t_i, \neg t_i\}$$

The retrieval is performed in two steps:

1. The set R_i of documents are obtained that contain or do not contain the term t_i:

   $$R_i = \{d_j \mid \theta_i \in d_j\},\ \theta_i \in \{t_i, \neg t_i\}$$

   where $\neg t_i \in d_j$ means $t_i \notin d_j$

2. Set operations are used to retrieve documents in response to Q:

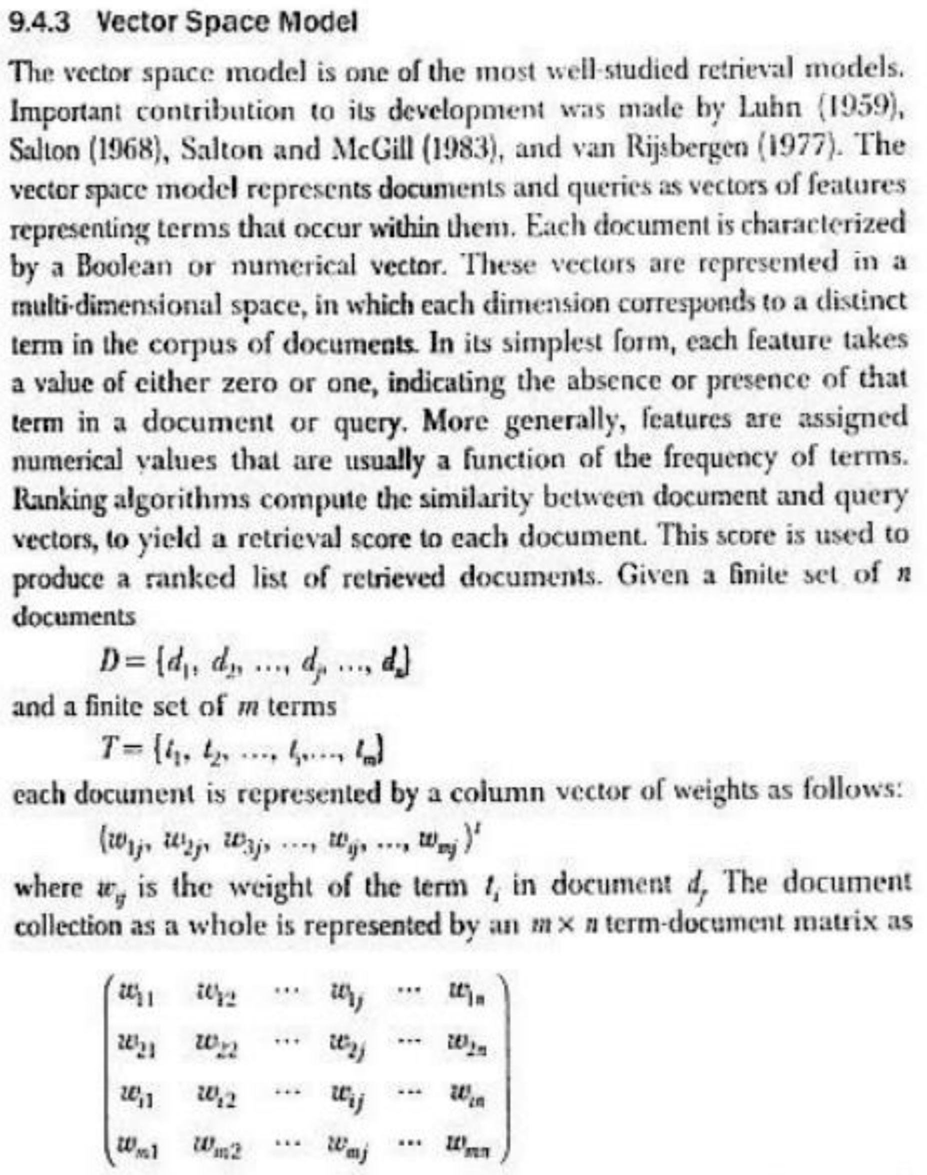
   $$\cap R_i$$

9.4.2 Probabilistic Model

The probabilistic model applies a probabilistic framework to IR. It ranks documents based on the probability of their relevance to a given query (Robertson and Jones 1976). Retrieval depends on whether probability of relevance (relative to a query) of a document is higher than that of non-relevance, i.e. whether it exceeds a threshold value. Given a set of documents D, a query q, and a cut-off value α, this model first calculates the probability of relevance and irrelevance of a document to the query. It then ranks documents having probabilities of relevance at least that of irrelevance in decreasing order of their relevance. Documents are retrieved if the probability of relevance in the ranked list exceeds the cut off value.

More formally, if $P(R/d)$ is the probability of relevance of a document d, for query q, and $P(I/d)$ is the probability of irrelevance, then the set of documents retrieved in response to the query q is as follows.

$$S = \{d_j \mid P(R/d_j) \geq P(I/d_j)\}\ P(R/d_j) \geq \alpha$$



9.4.3 Vector Space Model

The vector space model is one of the most well-studied retrieval models. Important contribution to its development was made by Luhn (1959), Salton (1968), Salton and McGill (1983), and van Rijsbergen (1977). The vector space model represents documents and queries as vectors of features representing terms that occur within them. Each document is characterized by a Boolean or numerical vector. These vectors are represented in a multi-dimensional space, in which each dimension corresponds to a distinct term in the corpus of documents. In its simplest form, each feature takes a value of either zero or one, indicating the absence or presence of that term in a document or query. More generally, features are assigned numerical values that are usually a function of the frequency of terms. Ranking algorithms compute the similarity between document and query vectors, to yield a retrieval score to each document. This score is used to produce a ranked list of retrieved documents. Given a finite set of n documents

$$D = \{d_1, d_2, \ldots, d_j, \ldots, d_n\}$$

and a finite set of m terms

$$T = \{t_1, t_2, \ldots, t_i, \ldots, t_m\}$$

each document is represented by a column vector of weights as follows:

$$(w_{1j}, w_{2j}, w_{3j}, \ldots, w_{ij}, \ldots, w_{mj})^t$$

where w_{ij} is the weight of the term t_i in document d_j. The document collection as a whole is represented by an $m \times n$ term-document matrix as

$$\begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1j} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2j} & \cdots & w_{2n} \\ w_{i1} & w_{i2} & \cdots & w_{ij} & \cdots & w_{in} \\ w_{m1} & w_{m2} & \cdots & w_{mj} & \cdots & w_{mn} \end{pmatrix}$$

10b.

12.2 WORDNET

WordNet[1] (Miller 1990, 1995) is a large lexical database for the English language. Inspired by psycholinguistic theories, it was developed and is being maintained at the Cognitive Science Laboratory, Princeton University, under the direction of George A. Miller. WordNet consists of three databases—one for nouns, one for verbs, and one for both adjectives and adverbs. Information is organized into sets of synonymous words called *synsets*, each representing one base concept. The synsets are linked to each other by means of lexical and semantic relations. Lexical relations occur between word forms (i.e., senses) and semantic relations between word meanings. These relations include synonymy, hypernymy/hyponymy, antonymy, meronymy/holonymy, troponymy, etc. A word may appear in more than one synset and in more than one part-of-speech. The meaning of a word is called sense. WordNet lists all senses of a word, each sense belonging to a different synset. WordNet's sense-entries consist of a set of synonyms and a gloss. A gloss consists of a dictionary-style definition and examples demonstrating the use of a synset in a sentence, as shown in Figure 12.1. The figure shows the entries for

Concept Identification in Natural Language

WordNet can be used to identify concepts pertaining to a term, to suit them to the full semantic richness and complexity of a given information need.

Word Sense Disambiguation

WordNet combines features of a number of the other resources commonly used in disambiguation work. It offers sense definitions of words, identifies synstes of synonyms, defines a number of semantic relations and is freely available. This makes it the (currently) best known and most utilized resource for word sense disambiguation. One of the earliest attempts to use WordNet for word sense disambiguation was in IR by Voorheese (1993). She used WordNet noun hierarchy (hypernym / hyponym) to achieve disambiguation. A number of other researchers have also used WordNet for the same purpose (Resnik 1995, 1997; Sussna 1993).

Automatic Query Expansion

WordNet semantic relations can be used to expand queries so that the search for a document is not confined to the pattern-matching of query terms, but also covers synonyms. The work performed by Voorhees (1994) is based on the use of WordNet relations, such as synonyms, hypernyms, and hyponyms, to expand queries.

Document Structuring and Categorization

The semantic information extracted from WordNet, and WordNet conceptual representation of knowledge, have been used for text categorization (Scott and Matwin 1998).

Document Summarization

WordNet has found useful application in text summarization. The approach presented by Barzilay and Elhadad (1997) utilizes information from WordNet to compute lexical chains.