Natural Language Processing Principles and Applications (Elective -II) [AML 5233]

Marks: 50 Duration: 90 mins.

I Sessional Exam – Answer Scheme

Distinguish the following with relevant examples: (10 Marks)

1. Homonyms 2. Homographs 3. Homophones 4. Heteronyms 5. Heterographs (10)

Homonyms are defined as words that share the same spelling or pronunciation but have different meanings. An alternative definition restricts the constraint on same spelling. The relationship between these words is termed as *homonymy*.

Homonyms are often said to be the superset of homographs and homophones. An example of homonyms for the word *bat* can be demonstrated in the following sentences: *The bat hangs upside down from the tree* and *That baseball bat is really sturdy.*

Homographs are words that have the same written form or spelling but have different meanings. Several alternate definitions say that the pronunciation can also be different.

Some examples of homographs include the word *lead* as in *I am using a lead pencil* and *Please lead the soldiers to the camp* , and also the word *bass* in *Turn up the bass for the song* and *I just caught a bass today while I was out fishing* . Note that in both cases, the spelling stays the same but the pronunciation changes based on the context in the sentences.

Homophones are words that have the same pronunciation but different meanings, and they can have the same or different spellings. Examples would be the words *pair* (meaning couple) and *pear* (the fruit). They sound the same but have different meanings and written forms. Often these words cause problems in NLP because it is very difficult to find out the actual context and meaning using machine intelligence.

Heteronyms are words that have the same written form or spelling but different pronunciations and meanings. By nature, they are a subset of homographs. They are also often called *heterophones*, which means “different sound.” Examples of heteronyms are the words *lead* (metal, command) and *tear* (rip off something, moisture from eyes).

Heterographs are words that have the same pronunciation but different meanings and spellings. By nature, they are a subset of homonyms. Their written representation might be different, but they sound very similar or often exactly the same when spoken.

Some examples include the words *to*, *too* , and *two* , which sound similar but have different spellings and meanings.

Describe the hierarchical syntax of a structured sentence by annotating it with POS tags for a suitable sentence. (5 Marks)

Syntax and structure usually go hand in hand, where a set of specific rules, conventions, and principles usually govern the way words are combined into phrases, phrases get combined into clauses, and clauses get combined into sentences. Let us consider specifically about the English language syntax and structure as we are dealing with textual data that belongs to the English language. But a lot of these concepts can be extended to other languages too. Knowledge about the structure and syntax of language is helpful in many areas like text processing, annotation, and parsing for further operations such as text classification or summarization.

In English, words usually combine together to form other constituent units. These constituents include **words**, **phrases**, **clauses**, and **sentences**. All these constituents exist together in any message and are related to each other in a hierarchical structure.

Moreover, a sentence is a structured format of representing a collection of words provided they follow certain syntactic rules like grammar.

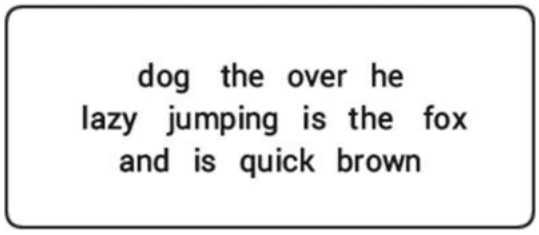


Fig: A collection of words without any relation or structure

From the collection of words in Fig , it is very difficult to ascertain what it might be trying to convey or mean. Indeed, languages are not just comprised of groups of unstructured words. Sentences with proper syntax not only help us give proper structure and relate words together but also help them convey meaning based on the order or position of the words. Considering our previous hierarchy of sentence → clause → phrase → word, we can construct the hierarchical sentence tree in Fig given below using shallow parsing , a technique using for finding out the constituents in a sentence.

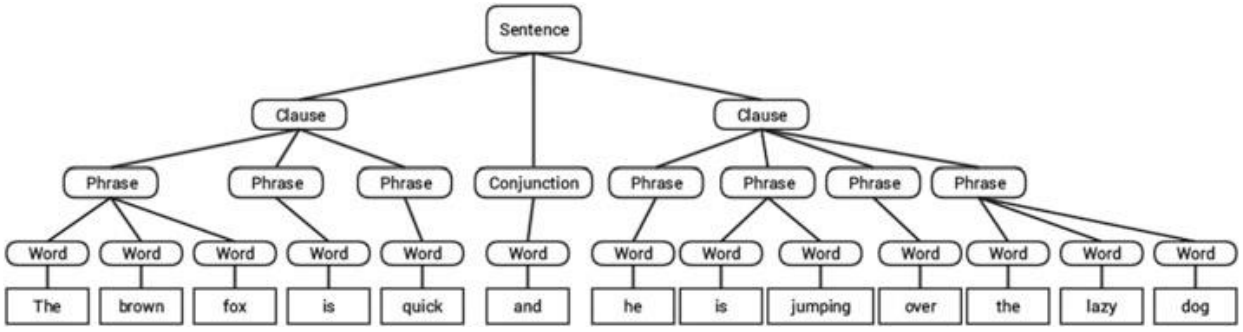


Fig: Structured sentence following the hierarchical syntax

From the hierarchical tree in Fig, we get the sentence "*The brown fox is quick and he is jumping over the lazy dog*" . We can see that the leaf nodes of the tree consist of words, which are the smallest unit

here, and combinations of words form phrases, which in turn form clauses. Clauses are connected together through various filler terms or words such as conjunctions and form the final sentence.

Write the categories and significance of different POS tags.

Words are the smallest units in a language that are independent and have a meaning of their own. Although morphemes are the smallest distinctive units, morphemes are not independent like words, and a word can be comprised of several morphemes. It is useful to annotate and tag words and analyze them into their parts of speech (POS) to see the major syntactic categories. Here, we have the main categories and significance of the various POS tags.

Usually, words can fall into one of the following major categories.

• N(oun) : This usually denotes words that depict some object or entity which may be living or nonliving. Some examples would be fox , dog , book , and so on. The POS tag symbol for nouns is N .

• V(erb) : Verbs are words that are used to describe certain actions, states, or occurrences. There are a wide variety of further subcategories, such as auxiliary, reflexive, and transitive verbs (and many more). Some typical examples of verbs would be running , jumping , read , and write . The POS tag symbol for verbs is V .

• Adj(ective) : Adjectives are words used to describe or qualify other words, typically nouns and noun phrases. The phrase beautiful flower has the noun (N) flower which is described or qualified using the adjective (ADJ) beautiful . The POS tag symbol for adjectives is ADJ .

• Adv(erb) : Adverbs usually act as modifiers for other words including nouns, adjectives, verbs, or other adverbs. The phrase very beautiful flower has the adverb (ADV) very , which modifies the adjective (ADJ) beautiful , indicating the degree to which the flower is beautiful. The POS tag symbol for adverbs is ADV.

Besides these four major categories of parts of speech , there are other categories that occur frequently in the English language. These include pronouns, prepositions, interjections, conjunctions, determiners, and many others. Also, each POS tag like the noun (N) can be further subdivided into categories like singular nouns (NN), singular proper nouns (NNP), and plural nouns (NNS).

Considering our previous example sentence (The brown fox is quick and he is jumping over the lazy dog) where we built the hierarchical syntax tree, if we were to annotate it using basic POS tags, it would look like Fig below.

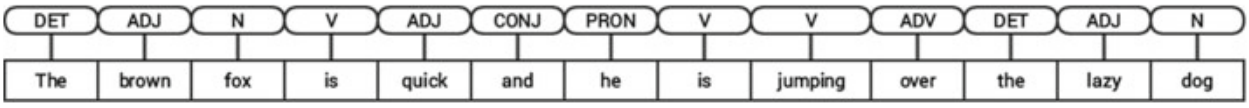


Fig: Annotated words with their POS tags

Compare the need for propositional Logic and First Order Logic.

Propositional logic is an analytical statement which is either true or false. It is basically a technique that represents the knowledge in logical & mathematical form.

There are two types of propositional logic; Atomic and Compound Propositions.

Proposition logic can be either true or false it can never be both. In this type of logic, symbolic variables are used in order to represent the logic and any logic can be used for representing the variable. It is comprised of objects, relations, functions, and logical connectives. Proposition formula which is always false is called 'Contradiction' whereas a proposition formula which is always true is called 'Tautology'.

First-Order Logic is another knowledge representation in AI which is an extended part of PL. FOL articulates the natural language statements briefly. Another name of First-Order Logic is 'Predicate Logic'. FOL is known as the powerful language which is used to develop information related to objects in a very easy way. Unlike PL, FOL assumes some of the facts that are related to objects, relations, and functions. FOL has two main key features or you can say parts that are; 'Syntax' & 'Semantics'.

Propositional Logic converts a complete sentence into a symbol and makes it logical whereas in First-Order Logic relation of a particular sentence will be made that involves relations, constants, functions, and constants.

The limitation of PL is that it does not represent any individual entities whereas FOL can easily represent the individual establishment that means if you are writing a single sentence then it can be easily represented in FOL.

PL does not signify or express the generalization, specialization or pattern for example 'QUANTIFIERS' cannot be used in PL but in FOL users can easily use quantifiers as it does express the generalization, specialization, and pattern.

Write the FOL representation for the Natural Language Statement: "There is at least one student who failed the exam".

$\exists x \; student(x) \wedge fail(x, exam)$

Write the Natural Language Statement for the FOL representation: $(\exists x \; player(x) \wedge score(x, goal) \wedge (\forall y \; score(y, goal) \rightarrow x=y))$

There was exactly one player who scored the goal.

Explain the Disjunction, Grouping, and Precedence with respect to regular expression using relevant examples.

Suppose we need to search for texts about pets; perhaps we are particularly interested in cats and dogs. In such a case, we might want to search for either the string cat or the string dog. Since we can't use the square brackets to search for "cat or dog" (why can't we say /[catdog]/?), we need a new operator, the disjunction operator, also called the pipe symbol |. The pattern /cat|dog/ matches either the string cat or the string dog.

Sometimes we need to use this disjunction operator in the midst of a larger sequence. For example, suppose I want to search for information about pet fish for my cousin David. How can I specify both guppy and guppies? We cannot simply say /guppy|ies/, because that would match only the strings guppy and ies. This is because sequences like guppy take precedence over the disjunction operator |.

To make the disjunction operator apply only to a specific pattern, we need to use the parenthesis operators (and). Enclosing a pattern in parentheses makes it act like a single character for the purposes of neighboring operators like the pipe | and the Kleene*. So the pattern /gupp(y|ies)/ would specify that we meant the disjunction only to apply to the suffixes y and ies. The parenthesis operator (is also useful when we are using counters like the Kleene*. Unlike the | operator, the Kleene* operator applies by default only to a single character, not to a whole sequence. Suppose we want to match repeated instances of a string. Perhaps we have a line that has column labels of the form Column 1 Column 2 Column 3. The expression /Column [0-9]+ */ will not match any number of columns; instead, it will match a single column followed by any number of spaces! The star here applies only to the space that precedes it, not to the whole sequence. With the parentheses, we could write the expression /(Column [0-9]+ *)*/ to match the word Column, followed by a number and optional spaces, the whole pattern repeated zero or more times.

This idea that one operator may take precedence over another, requiring us to sometimes use parentheses to specify what we mean, is formalized by the operator hierarchy for regular expressions. The following table gives the order precedence of RE operator precedence, from highest precedence to lowest precedence.

The idea that one operator may take precedence over another, requiring us to sometimes use parentheses to specify what we mean, is formalized by the operator precedence hierarchy for regular expressions.

Parenthesis ()

Counters * + ? {}

Sequences and anchors the ^my end$

Disjunction |

The use of parentheses to store a pattern in memory is called a capture group and it is used in regular expressions for substitutions. A capture group is used (i.e., parentheses surround a pattern), the resulting match is stored in a numbered register. put parentheses (and) around the first pattern and use the number operator \1 in the second pattern to refer back.

s/([0-9]+)/<\1>/

An important use of regular expressions is in substitutions. For example, the substitution operator s/regexp1/pattern/ used in Python and in Unix commands like vim or sed allows a string characterized by a regular expression to be replaced by another string:

s/colour/color/

It is often useful to be able to refer to a particular subpart of the string matching the first pattern. For example, suppose we wanted to put angle brackets around all integers in a text, for example, changing the 35 boxes to the <35> boxes. We'd like a way to refer to the integer we've found so that we can easily add the brackets.

To do this, we put parentheses (and) around the first pattern and use the number operator \1 in the second pattern to refer back. Here's how it looks:

s/([0-9]+)/<\1>/

The parenthesis and number operators can also specify that a certain string or expression must occur twice in the text. For example, suppose we are looking for the pattern "the Xer they were, the Xer they will be", where we want to constrain the two X's to be the same string. We do this by surrounding the first X with the parenthesis operator, and replacing the second X with the number operator \1, as follows:

/the (.*)er they were, the \1er they will be/

Here the \1 will be replaced by whatever string matched the first item in parentheses. So this will match the bigger they were, the bigger they will be but not the bigger they were, the faster they will be. This use of parentheses to store a pattern in memory is called a capture group.

Every time a capture group is used (i.e., parentheses surround a pattern), the reregister resulting match is stored in a numbered register. If you match two different sets of parentheses, \2 means whatever matched the second capture group. Thus /the (.*)er they (.*), the \1er we \2/ will match the faster they ran, the faster we ran but not the faster they ran, the faster we ate. Similarly, the third capture group is stored in \3, the fourth is \4, and so on.

Define the minimum edit distance between two strings.

Compute the edit distance (using insertion cost 1, deletion cost 1, substitution cost 1) of "explanation" to "description". Show your work (using the edit distance grid).

The minimum edit distance between two strings is defined as the minimum number of editing operations (operations like insertion, deletion, substitution) needed to transform one string into another. Much of natural language processing is concerned with measuring how similar two strings are. For example in spelling correction, the user typed some erroneous string—let's say graffe–and we want to know what the user meant. The user probably intended a word that is similar to graffe. Among candidate similar words, the word giraffe, which differs by only one letter from graffe, seems intuitively to be more similar. Another example comes from coreference, the task of deciding whether two strings such as the following refer to the same entity:

Stanford President Marc Tessier-Lavigne

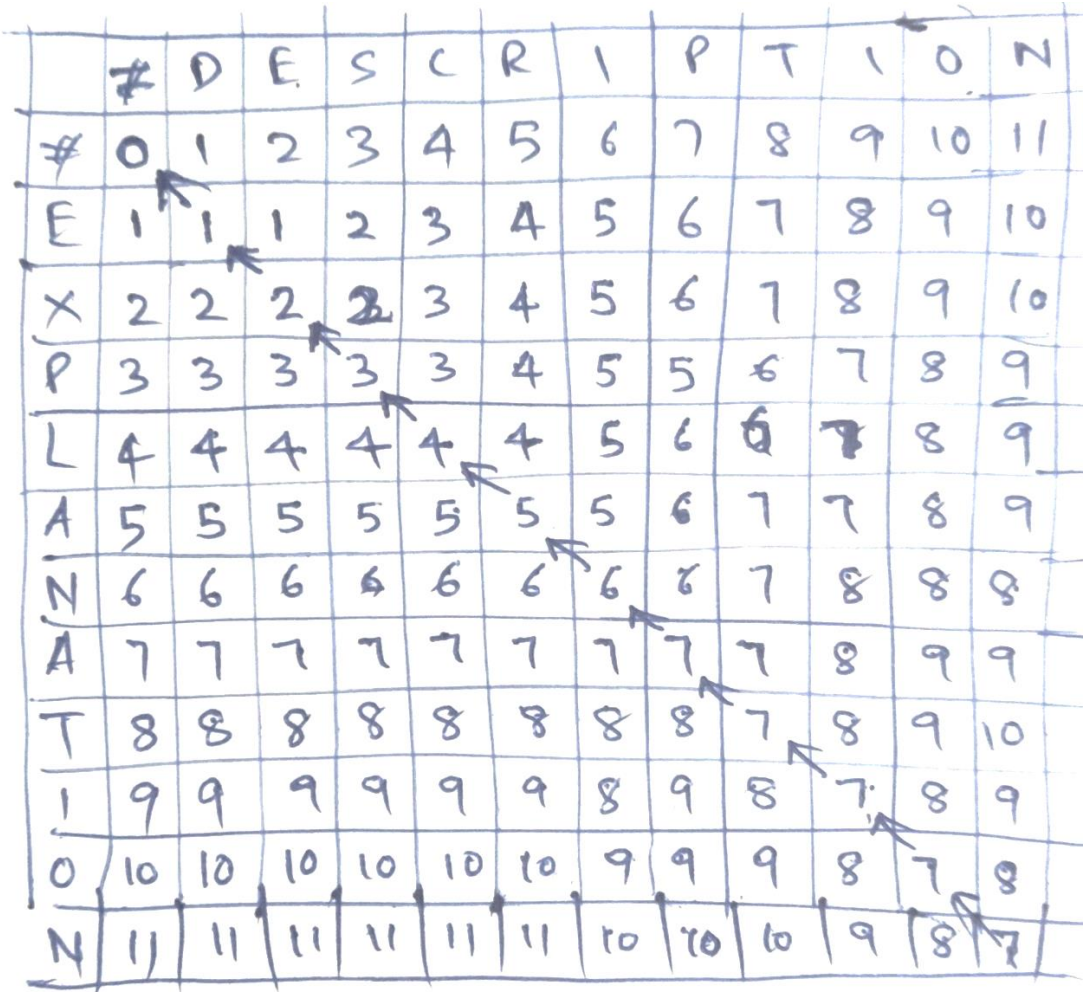
Stanford University President Marc Tessier-Lavigne

Again, the fact that these two strings are very similar (differing by only one word) seems like useful evidence for deciding that they might be coreferential.

The gap between *intention* and *execution*, for example, is 5 (delete an i, substitute e for n, substitute x for t, insert c, substitute u for n). It's much easier to see this by looking at the most important visualization for string distances, an alignment between the two strings, Given two sequences, an alignment is a correspondence between substrings of the two sequences. Thus, we say I aligns with the empty string, N with E, and so on. Beneath the aligned strings is another representation; a series of symbols expressing an operation list for converting the top string into the bottom string: d for deletion, s for substitution, i for insertion.

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

Fig. representing the minimum edit distance between two strings as an alignment. The final row gives the operation list for converting the top string into the bottom string: d for deletion, s for substitution, i for insertion.



Hence by backtrace using edit distance grid the edit distance = 7