Natural Language Processing

Dr. M. Brindha
Assistant Professor
Department of CSE
NIT, Trichy-15

Introduction

- There is a large volume of unstructured textual data present across all organizations, irrespective of their domain.
- E.g. vast amounts of data in the form of tweets, status updates, comments, hashtags, articles, blogs, wikis, and much more on social media.
- Even retail and e-commerce stores generate a lot of textual data from new product information and metadata with customer reviews and feedback.
- The main challenges associated with textual data are twofold.
- The first challenge deals with effective storage and management of this data.
- The second challenge is with regard to analyzing this data and trying to extract meaningful patterns and useful insights that would be beneficial to the organization.

Introduction

- Usually, textual data is unstructured and does not adhere to any specific predefined data model or schema, which is usually followed by relational databases.
- However, based on the data semantics, it can be stored in either SQL-based database management systems (DBMS) like SQL Server or even NoSQL-based systems like MongoDB and CouchDB, and more recently in information retrieval-based data stores like ElasticSearch and Solr.
- Organizations having enormous amounts of textual datasets often resort to file-based systems like Hadoop where they dump all the data in the Hadoop Distributed File System (HDFS).(Data lake)

Introduction

Analysis

- large number of machine learning and data analysis techniques -most of them are tuned to work with numerical data
- Resort to areas like natural language processing (NLP) and specialized techniques, transformations, and algorithms to analyze text data, or more specifically natural language, which is quite different from programming languages that are easily understood by machines.
- Textual data, being highly unstructured, does not follow or adhere to structured or regular syntax and patterns—hence we cannot directly use mathematical or statistical models to analyze it.

Natural language

- Textual data is unstructured data but it usually belongs to a specific language following specific syntax and semantics.
- Any piece of text data—a simple word, sentence, or document—relates back to some natural language most of the time.
- A natural language is one developed and evolved by humans through natural use and communication , rather than constructed and created artificially, like a computer programming language.
- Human languages like English, Japanese, and Sanskrit are natural languages.

Natural language- The Philosophy of Language

- The philosophy of language mainly deals with the four problems and seeks answers to solve them:

1. The nature of meaning in a language is concerned with the semantics of a language and the nature of meaning itself.

- It describes how structure and syntax in the language pave the way for semantics, or to be more specific, how words, which have their own meanings, are structured together to form meaningful sentences.
- Linguistics is the scientific study of language, a special field that deals with some of these problems.
- Syntax, semantics, grammars, and parse trees are some ways to solve these problems.
- The nature of meaning can be expressed in linguistics between two human beings, notably a sender and a receiver, as what the sender tries to express or communicate when they send a message to a receiver, and what the receiver ends up understanding or deducing from the context of the received message.
- non-linguistic standpoint-body language, prior experiences, and psychological effects are contributors to meaning of language, where each human being perceives or infers meaning in their own way.

Natural language- The Philosophy of Language

2. **The use of language** is more concerned with how language is used as an entity in various scenarios and communication between human beings.

- This includes analyzing speech and the usage of language when speaking, including **the speaker's intent, tone, content and actions involved in expressing a message**.
- This is often termed as a **speech act in linguistics**.
- More advanced concepts such as the **origins of language creation** and human cognitive activities such as **language acquisition** which is responsible for learning and usage of languages are also of prime interest.

Natural language- The Philosophy of Language

3. Language cognition specifically focuses on how the cognitive functions of the human brain are responsible for understanding and interpreting language.

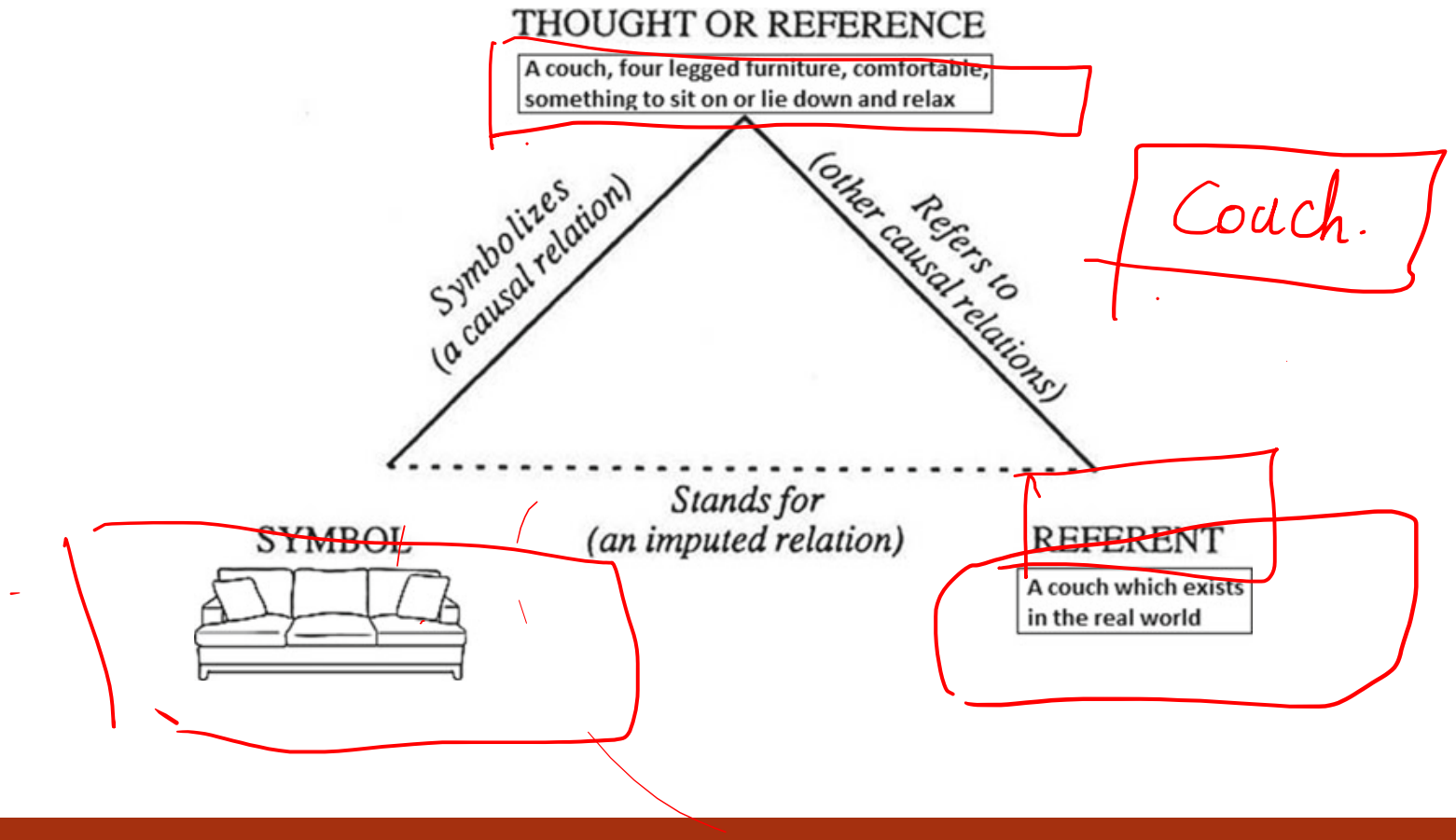
- Considering the example of a typical sender and receiver, there are many actions involved from message communication to interpretation.
- Cognition tries to find out how the mind works in combining and relating specific words into sentences and then into a meaningful message and what is the relation of language to the thought process of the sender and receiver when they use the language to communicate messages.

4. The relationship between language and reality explores the extent of truth of expressions originating from language.

- Usually, philosophers of language try to measure how factual these expressions are and how they relate to certain affairs in our world which are true.

The relationship between language and reality- The Triangle of reference Model

- One of the most popular models is the triangle of reference , which is used to explain how words convey meaning and ideas in the minds of the receiver and how that meaning relates back to a real world entity or fact.



The Triangle of reference Model

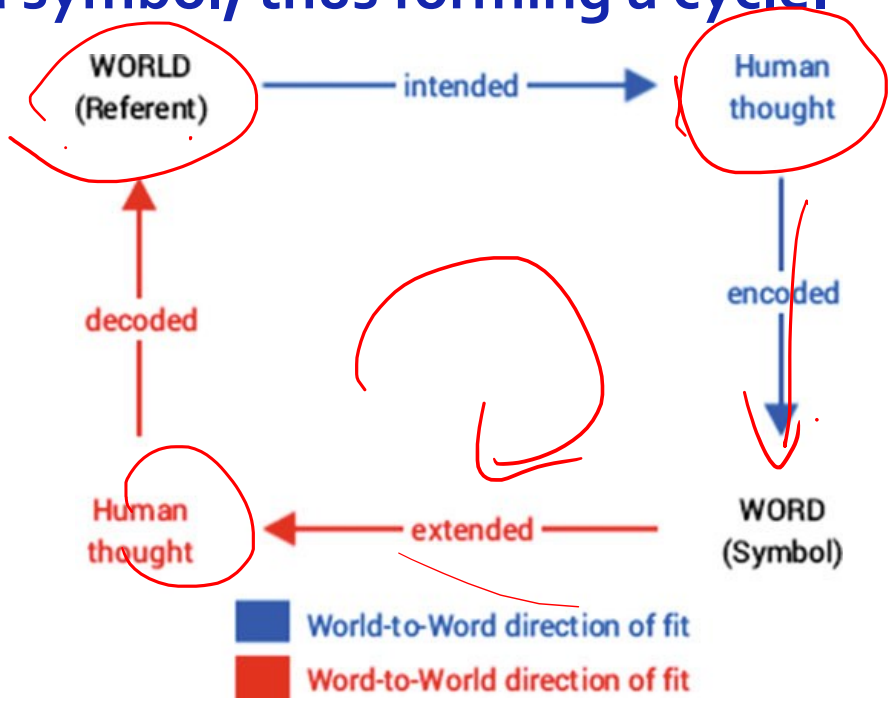
- **The triangle of reference model** is also known as the meaning of meaning model, and it is depicted in Figure with a real example of a couch being perceived by a person which is present in front of him.
- **A symbol** is denoted as a linguistic symbol, like a word or an object that evokes thought in a person's mind.
- In this case, the symbol is the couch, and this evokes thoughts like what is a couch, a piece of furniture that can be used for sitting on or lying down and relaxing, something that gives us comfort .
- These thoughts are known as a **reference** and through this reference the person is able to relate it to something that exists in the real world, termed a **referent**.
- In this case the **referent is the couch** which the person perceives to be present in front of him.

The relationship between language and reality- The direction of fit

- The second way to find out relationships between language and reality is known as the direction of fit.
- The word-to-world direction of fit talks about instances where the usage of language can reflect reality.
- This indicates using words to match or relate to something that is happening or has already happened in the real world.
- An example would be the sentence The Eiffel Tower is really big, which accentuates a fact in reality.
- The other direction of fit, known as world-to-word , talks about instances where the usage of language can change reality.
- An example here would be the sentence I am going to take a swim , where the person I is changing reality by going to take a swim by representing the same in the sentence being communicated.

The relationship between language and reality- The direction of fit

- It is quite clear from the preceding depiction that based on the referent that is perceived from the real world, a person can form a representation in the form of a symbol or word and consequently can communicate the same to another person, which forms a representation of the real world based on the received symbol, thus forming a cycle.

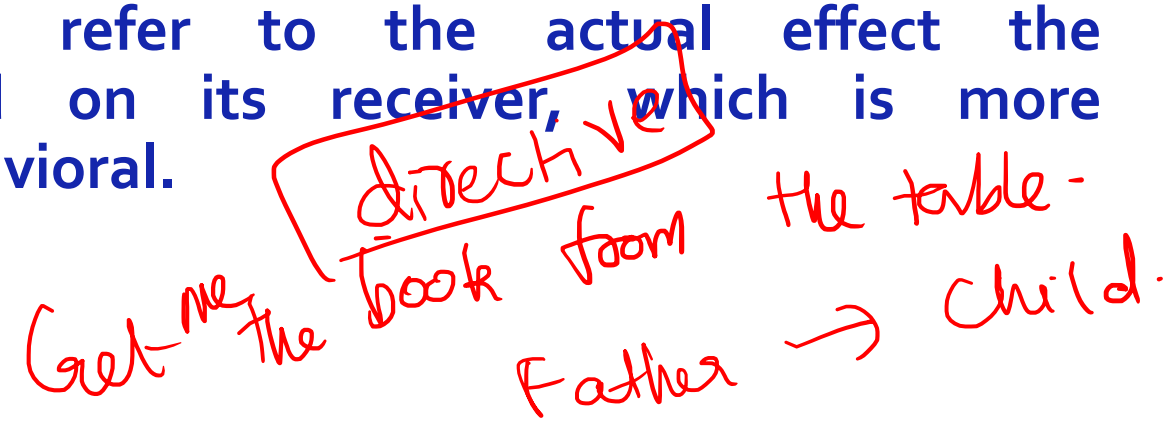


Language Acquisition and Cognitive Learning

- **Language acquisition** is defined as the process by which human beings utilize their cognitive abilities, knowledge, and experience to understand language based on hearing and perception and start using it in terms of words, phrases, and sentences to communicate with other human beings.
- In simple terms, the ability of acquiring and producing languages is language acquisition
- Cognitive abilities along with language-specific knowledge and abilities like syntax, semantics, concepts of parts of speech, and grammar together form language acquisition device that enabled humans to have the ability of language acquisition .

Language Usage

- This include concepts related to speech acts that highlight different ways in which language is used in communication.
- There are three main categories of speech acts: locutionary , illocutionary , and perlocutionary acts.
- **Locutionary acts** are mainly concerned with the actual delivery of the sentence when communicated from one human being to another by speaking it.
- **Illocutionary acts** focus further on the actual semantics and significance of the sentence which was communicated.
- **Perlocutionary acts** refer to the actual effect the communication had on its receiver, which is more psychological or behavioral.



Language Usage

- Get me the book from the table spoken by a father to his child.
- The phrase when spoken by the father forms the locutionary act.
- This significance of this sentence is a directive, which directs the child to get the book from the table and forms an illocutionary act.
- The action the child takes after hearing this, that is, if he brings the book from the table to his father, forms the perlocutionary act.

Language Usage

Five different classes of illocutionary speech acts

1. **Assertives** are speech acts that communicate how things are already existent in the world.

- They are spoken by the sender when he tries to assert a proposition that could be true or false in the real world.
- These assertions could be statements or declarations.
- A simple example would be **The Earth revolves round the Sun.**

2. **Directives** are speech acts that the sender communicates to the receiver asking or directing them to do something.

- This represents a **voluntary act** which the receiver might do in the future after receiving a directive from the sender.
- These directives could be simple requests or even orders or commands. Example: **Get me the book from the table.**

Language Usage

3. Commisives are speech acts that commit the sender or speaker who utters them to some future voluntary act or action.

- Acts like promises, oaths, pledges, and vows represent commisives, and the direction of fit could be either way.
- An example commissive would be I promise to be there tomorrow for the ceremony.

4. Expressives reveal a speaker or sender's disposition and outlook toward a particular proposition communicated through the message.

- These can be various forms of expression or emotion, such as congratulatory, sarcastic, and so on.
- An example expressive would be Congratulations on graduating top of the class .

5. Declarations are powerful speech acts that have the capability to change the reality based on the declared proposition in the message communicated by the speaker\sender.

- The usual direction of fit is world-to-word, but it can go the other way also.
- An example declaration would be I hereby declare him to be guilty of all charges .

Linguistics

- **Linguistics** is defined as the scientific study of language, including form and syntax of language, meaning, and semantics depicted by the usage of language and context of use.

The main distinctive areas of study under linguistics are:

- **Phonetics : This is the study of the acoustic properties of sounds produced by the human vocal tract during speech.**
- **It includes studying the properties of sounds as well as how they are created and by human beings.**
- **The smallest individual unit of human speech in a specific language is called a phoneme.**
- **A more generic term across languages for this unit of speech is phone .**
- **Phonology : This is the study of sound patterns as interpreted in the human mind and used for distinguishing between different phonemes to find out which ones are significant.**
- **The structure, combination, and interpretations of phonemes are studied in detail, usually by taking into account a specific language at a time.**
- **The English language consists of around 45 phonemes.**
- **Phonology usually extends beyond just studying phonemes and includes things like accents, tone, and syllable structures.**

Linguistics

- **Syntax :** This is usually the study of sentences, phrases, words, and their structures.
- It includes researching how words are combined together grammatically to form phrases and sentences.
- Syntactic order of words used in a phrase or a sentence matter because the order can change the meaning entirely.
- **Morphology :** A morpheme is the smallest unit of language that has distinctive meaning.
- This includes things like words, prefixes, suffixes, and so on which have their own distinct meanings.
- Morphology is the study of the structure and meaning of these distinctive units or morphemes in a language.
- **Semantics :** This involves the study of meaning in language and can be further subdivided into lexical and compositional semantics.
  - Lexical semantics : The study of the meanings of words and symbols using morphology and syntax.
  - Compositional semantics : Studying relationships among words and combination of words and understanding the meanings of phrases and sentences and how they are related.

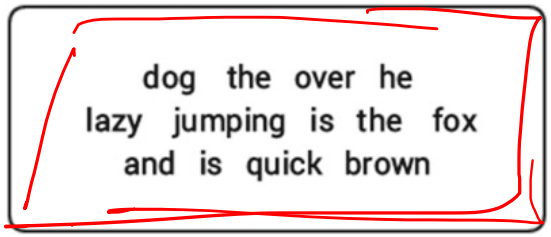
Linguistics

- **Lexicon :** This is the study of properties of words and phrases used in a language and how they build the vocabulary of the language.
- These include what kinds of sounds are associated with meanings for words, the parts of speech words belong to, and their morphological forms.
- **Pragmatics :** This is the study of how both linguistic and nonlinguistic factors like context and scenario might affect the meaning of an expression of a message or an utterance.
- This includes trying to infer whether there are any hidden or indirect meanings in the communication.
- **Discourse analysis :** This analyzes language and exchange of information in the form of sentences across conversations among human beings.
- These conversations could be spoken, written, or even signed.
- **Stylistics :** This is the study of language with a focus on the style of writing, including the tone, accent, dialogue, grammar, and type of voice.
- **Semiotics :** This is the study of signs, symbols, and sign processes and how they communicate meaning.
- Things like analogy, metaphors, and symbolism are covered in this area.

Language Syntax and Structure

- 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 combines into clauses, and clauses get combined into sentences.
- 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.

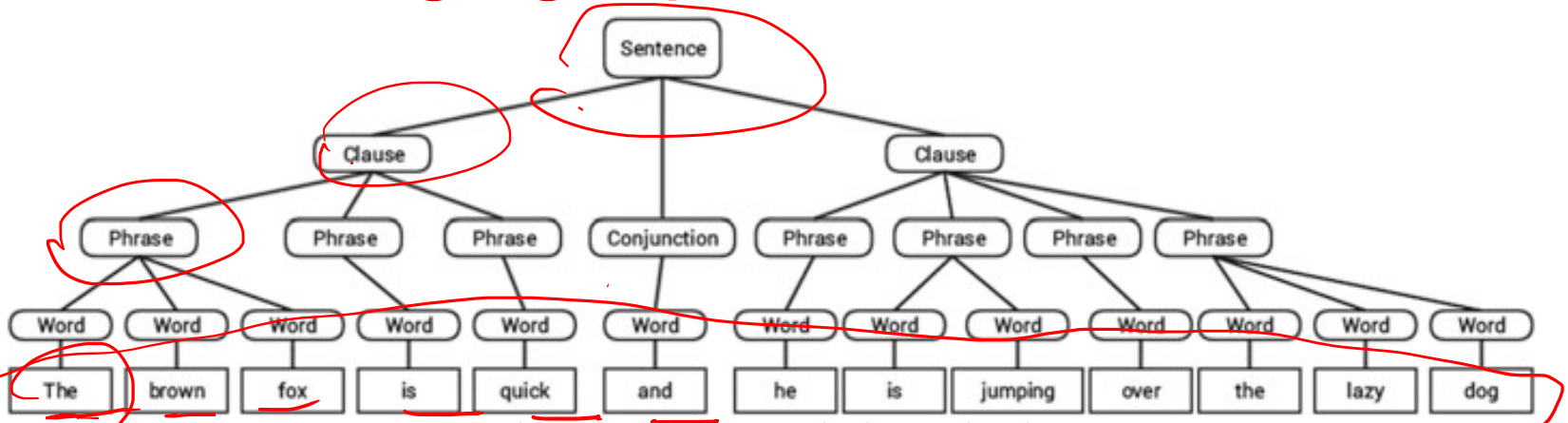
Language Syntax and Structure



A collection of words without any relation or structure

- From the collection of words in the Figure , 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.

Language Syntax and Structure



Structured sentence following the hierarchical syntax

- Hierarchy of sentence → clause → phrase→ word, we can construct the hierarchical sentence tree in Figure using shallow parsing , a technique using for finding out the constituents in a sentence.
- From the hierarchical tree in Figure, we get the sentence The brown fox is quick and he is jumping over the lazy dog .
- 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.

Language Syntax and Structure-Words

- Words are the smallest units in a language that are independent and have a meaning of their own.
- It is useful to annotate and tag words and analyze them into their parts of speech (POS) to see the major syntactic categories.

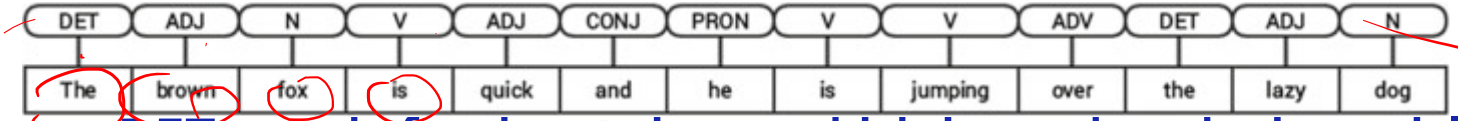
Categories

- ***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* .

Language Syntax and Structure-Words

- ***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* .**
- **Pronouns, prepositions, interjections, conjunctions, determiners, and many others.**
- **Furthermore, 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).**

Language Syntax and Structure-Words



- The tag DET stands for determiner , which is used to depict articles like a , an, the , and so on.
- The tag CONJ indicates conjunction , which is usually used to bind together clauses to form sentences.
- The PRON tag stands for pronoun , which represents words that are used to represent or take the place of a noun.
- The tags N, V, ADJ and ADV are typical open classes and represent words belonging to an open vocabulary.
- **Open classes** are word classes that consist of an infinite set of words and commonly accept the addition of new words to the vocabulary which are invented by people.
- Words are usually added to open classes through processes like morphological derivation , invention based on usage, and creating compound lexemes .
- Some popular nouns added fairly recently include Internet and multimedia.
- **Closed classes** consist of a closed and finite set of words and do not accept new additions. Pronouns are a closed class.

Language Syntax and Structure-Phrases

- In the hierarchy tree, groups of words make up phrases , which form the third level in the syntax tree.
- By principle , phrases are assumed to have at least two or more words, considering the pecking order of words ← phrases ← clauses ← sentences.
- However, a phrase can be a single word or a combination of words based on the syntax and position of the phrase in a clause or sentence.
- For example, the sentence Dessert was good has only three words, and each of them rolls up to three phrases.
- The word dessert is a noun as well as a noun phrase , is depicts a verb as well as a verb phrase , and good represents an adjective as well as an adjective phrase describing the aforementioned dessert.

Categories

- Noun phrase (NP) : These are phrases where a noun acts as the head word. Noun phrases act as a subject or object to a verb.
- Some examples would be dessert , the lazy dog , and the brown fox .

Language Syntax and Structure-Phrases

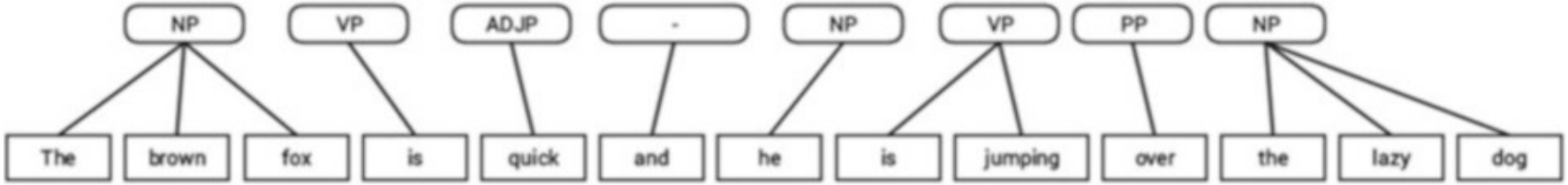
Categories

- **Verb phrase (VP) :** These phrases are lexical units that have a verb acting as the head word. Two forms of verb phrases.
- One form has the verb components as well as other entities such as nouns, adjectives, or adverbs as parts of the object.
- The verb here is known as a finite verb. It acts as a single unit in the hierarchy tree and can function as the root in a clause. This form is prominent in constituency grammars .
- The other form is where the finite verb acts as the root of the entire clause and is prominent in dependency grammars .
- The sentence He has started the engine can be used to illustrate the two types of verb phrases that can be formed. They would be has started the engine and has started based on the two forms.
- **Adjective phrase (ADJP) :** These are phrases with an adjective as the head word. Their main role is to describe or qualify nouns and pronouns in a sentence, and they will be either placed before or after the noun or pronoun.
- The sentence The cat is too quick has an adjective phrase, too quick , qualifying cat , which is a noun phrase.

Language Syntax and Structure-Phrases

Categories

- **Adverb phrase (ADVP) :** These phrases act like adverbs since the adverb acts as the head word in the phrase.
- Adverb phrases are used as modifiers for nouns, verbs, or adverbs themselves by providing further details that describe or qualify them.
- In the sentence, The train should be at the station pretty soon , the adjective phrase pretty soon describes when the train would be arriving.
- **Prepositional phrase (PP) :** These phrases usually contain a preposition as the head word and other lexical components like nouns, pronouns, and so on.
- It acts like an adjective or adverb describing other words or phrases.
- The phrase going up the stairs contains a prepositional phrase up, describing the direction of the stairs.



The brown fox is quick and he is jumping over the lazy dog

Language Syntax and Structure-Clauses

- A clause is a group of words with some relation between them that usually contains a subject and a predicate.
- Sometimes the subject is not present, and the predicate usually has a verb phrase or a verb with an object.
- By default you can classify clauses into two distinct categories : the main clause and the subordinate clause .
- The main clause is also known as an independent clause because it can form a sentence by itself and act as both sentence and clause.
- The subordinate or dependent clause cannot exist just by itself and depends on the main clause for its meaning.

Language Syntax and Structure-Clauses

- **Declarative** : These clauses usually occur quite frequently and denote statements that do not have any specific tone associated with them.
- These are just standard statements, which are declared with a neutral tone and which could be factual or non-factual.
- An example would be Grass is green.
- **Imperative** : These clauses are usually in the form of a request, command, rule, or advice.
- The tone in this case would be a person issuing an order to one or more people to carry out an order, request, or instruction. An example would be Please do not talk in class.
- **Relative** : The simplest interpretation of relative clauses is that they are subordinate clauses and hence dependent on another part of the sentence that usually contains a word, phrase, or even a clause.
- This element usually acts as the antecedent to one of the words from the relative clause and relates to it.
- A simple example would be John just mentioned that he wanted a soda, having the antecedent proper noun John, which was referred to in the relative clause he wanted a soda.

Language Syntax and Structure-Clauses

- **Interrogative : These clauses usually are in the form of questions.**
- **The type of these questions can be either affirmative or negative.**
- **Some examples would be Did you get my mail? and Didn't you go to school?**
- **Exclamative : These clauses are used to express shock, surprise,**

or even compliments.

- **These expressions fall under exclamations , and these clauses often end with an exclamation mark.**
- **An example would be What an amazing race!**

Language Syntax and Structure-Grammar

- Grammar helps in enabling both syntax and structure in language .
- It primarily consists of a set of rules used in determining how to position words, phrases, and clauses when constructing sentences for any natural language.
- Grammar can be subdivided into two main classes—dependency grammars and constituency grammars—based on their representations for linguistic syntax and structure.

Language Syntax and Structure-Grammar

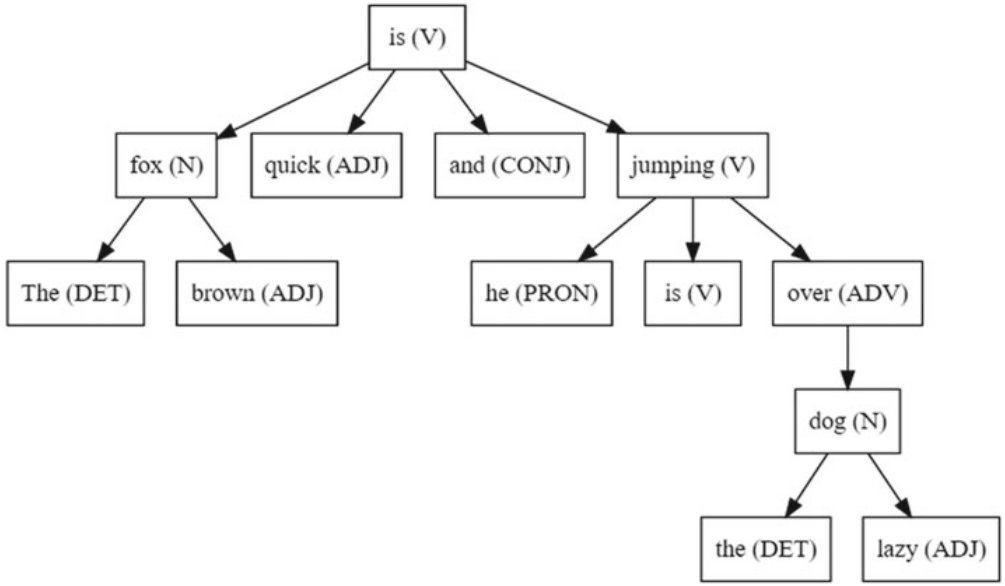
Dependency Grammar

- These grammars do not focus on constituents like words, phrases, and clauses but place more emphasis on words.
- These grammars are also known as word-based grammars.
- Dependencies in this context are labeled word-word relations or links that are usually asymmetrical.
- A word has a relation or depends on another word based on the positioning of the words in the sentence.
- Consequently, dependency grammars assume that further constituents of phrases and clauses are derived from this dependency structure between words.
- The word that has no dependency is called the root of the sentence.
- The verb is taken as the root of the sentence in most cases. All the other words are directly or indirectly linked to the root verb using links , which are the dependencies.

Language Syntax and Structure-Grammar

Dependency Grammar

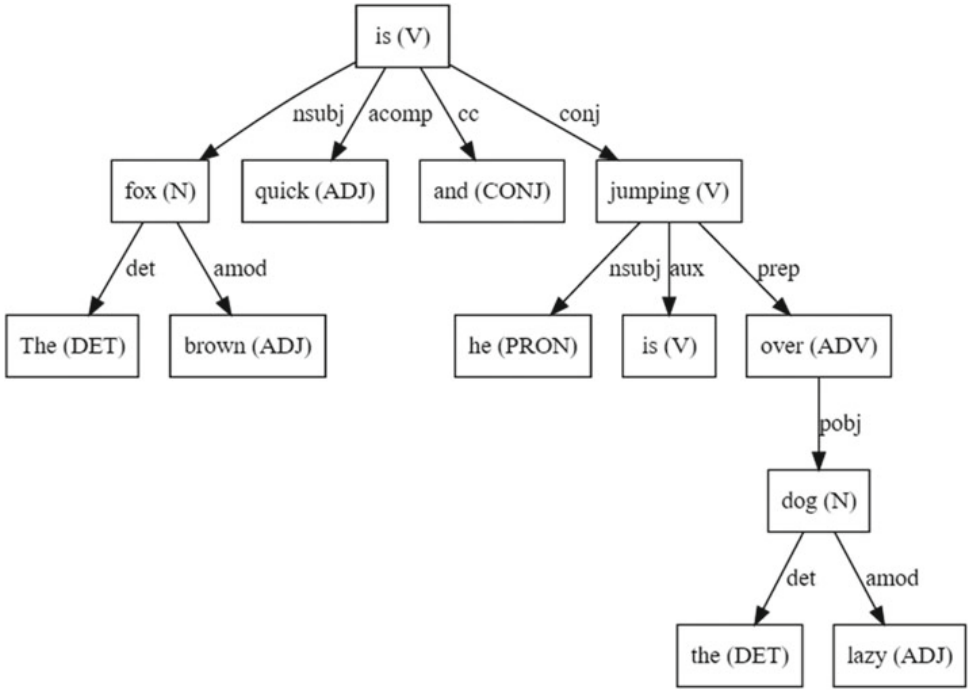
- Dependency grammars always have a one-to-one relationship correspondence for each word in the sentence.
- There are two aspects to this grammar representation.
- One is the syntax or structure of the sentence, and the other is the semantics obtained from the relationships denoted between the words.
- The syntax or structure of the words and their interconnections can be shown using a sentence syntax or parse tree. The dependency tree is a directed acyclic graph (DAG).



Language Syntax and Structure-Grammar

Dependency Grammar

- Each directed edge represents a specific type of meaningful relationship (also known as syntactic function).
- We can also annotate our sentence further showing the specific dependency relationship types between the words .



Language Syntax and Structure-Grammar

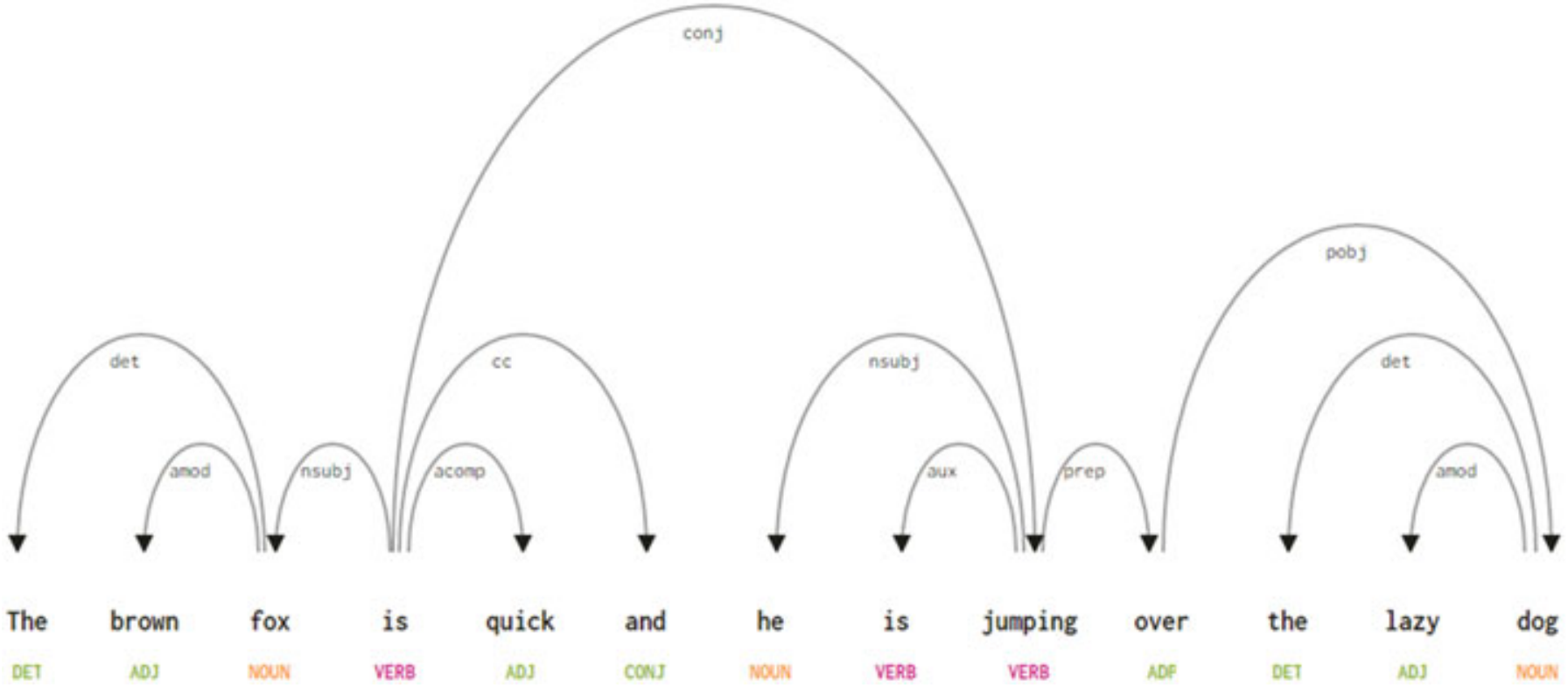
Dependency Grammar

- The dependency tag det denotes the determiner relationship between a nominal head and the determiner. Examples include (*fox* → *the*) and (*dog* → *the*).
- The dependency tag *amod* stands for *adjectival modifier* and stands for any adjective that modifies the meaning of a noun. Examples include (*fox* → *brown*) and (*dog* → *lazy*).
- The dependency tag *nsubj* stands for an entity that acts as a *subject* or agent in a clause. Examples include (*is* → *fox*) and (*jumping* → *he*).
- The dependencies *cc* and *conj* are more to do with linkages related to words connected by *coordinating conjunctions.* Examples include (*is* → *and*) and (*is* → *jumping*).
- The dependency tag *aux* indicates the *auxiliary* or secondary verb in the clause. Example: (*jumping* → *is*).
- The dependency tag *acomp* stands for *adjective complement* and acts as the complement or object to a verb in the sentence. Example: (*is* → *quick*).
- The dependency tag *prep* denotes a *prepositional* modifier, which usually modifies the meaning of a noun, verb, adjective, or preposition. Example: (jumping → over).
- The dependency tag *pobj* is used to denote the *object of a preposition*. Example: (over → dog).

Language Syntax and Structure-Grammar

Dependency Grammar

- Instead of creating a tree with linear orders, it can also represented it with a normal graph because there is no concept of order of words in dependency grammar.



Language Syntax and Structure-Grammar

Constituency Grammar

- Constituency grammars are a class of grammars built upon the principle that a sentence can be represented by several constituents derived from it.
- These grammars can be used to model or represent the internal structure of sentences in terms of a hierarchically ordered structure of their constituents.
- Each and every word usually belongs to a specific lexical category in the case and forms the head word of different phrases.
- These phrases are formed based on rules called *phrase structure rules*. Hence, constituency grammars are also called *phrase structure grammars*.
- *Constituents* are words or groups of words that have specific meaning and can act together as a dependent or independent unit.
- They can also be combined together further to form higher-order structures in a sentence, including phrases and clauses.

Language Syntax and Structure-Grammar

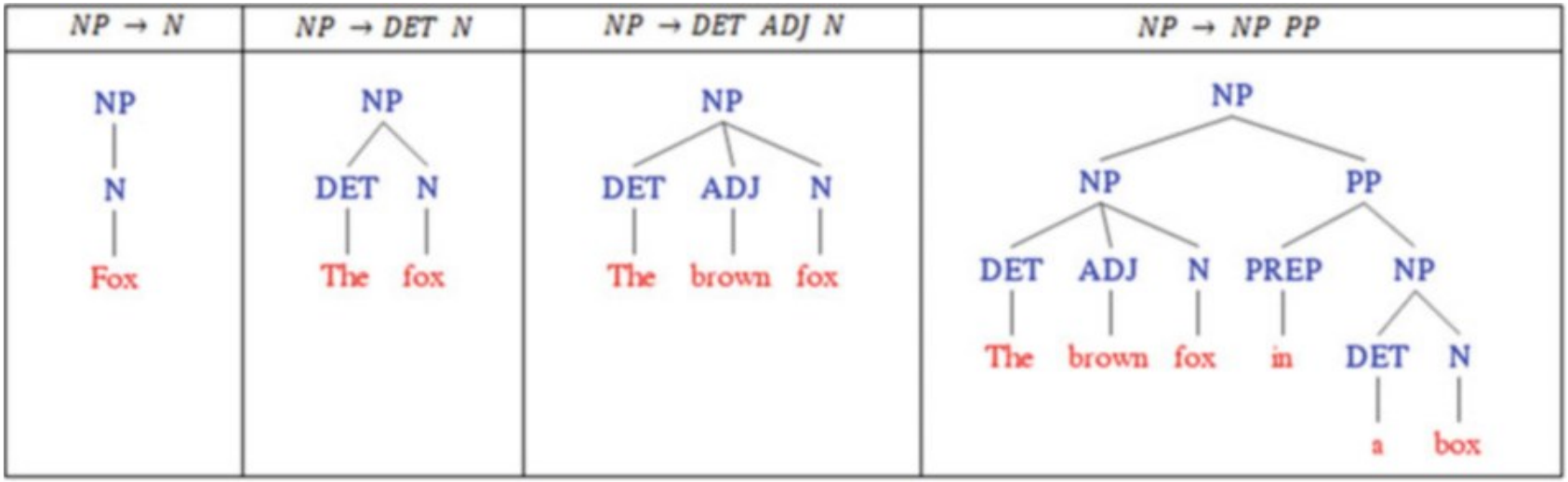
Constituency Grammar

- Phrase structure rules form the core of constituency grammars because they talk about syntax and rules that govern the hierarchy and ordering of the various constituents in the sentences.
- First and foremost, they determine what words are used to construct the phrases or constituents.
- Secondly, these rules determine how we need to order these constituents together.
- If we want to analyze phrase structure, we should be aware of typical schema patterns of the phrase structure rules.
- The generic representation of a phrase structure rule is $S \rightarrow AB$, which depicts that the structure S consists of constituents A and B , and the ordering is A followed by B .
- The phrase structure rule denotes a binary division for a sentence or a clause as $S \rightarrow NP\ VP$ where S is the sentence or clause, and it is divided into the subject, denoted by the noun phrase (NP) and the predicate, denoted by the verb phrase (VP).

Language Syntax and Structure-Grammar

Constituency Grammar

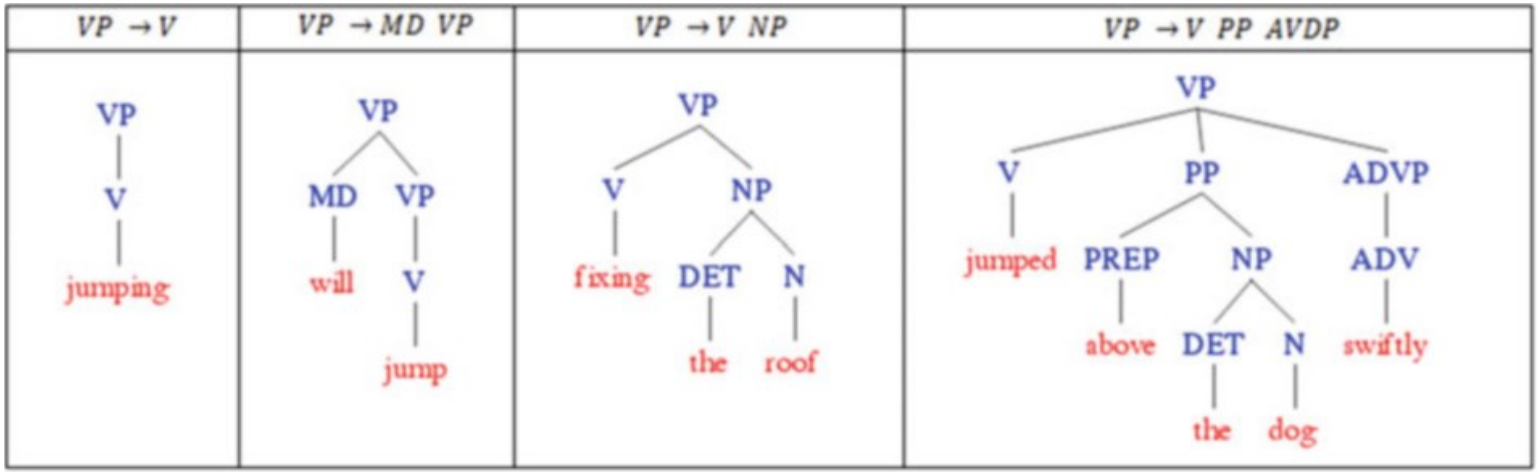
- We can apply more rules to break down each of the constituents further, but the top level of the hierarchy usually starts with a NP and VP.
- The rule for representing a noun phrase is NP → [DET][ADJ]N [PP] , where the square brackets denote that it is optional.
- Usually a noun phrase consists of a noun (N) definitely as the head word and may optionally contain determinants (DET) and adjectives (ADJ) describing the noun, and a prepositional phrase (PP) at the right side in the syntax tree.



Language Syntax and Structure-Grammar

Constituency Grammar

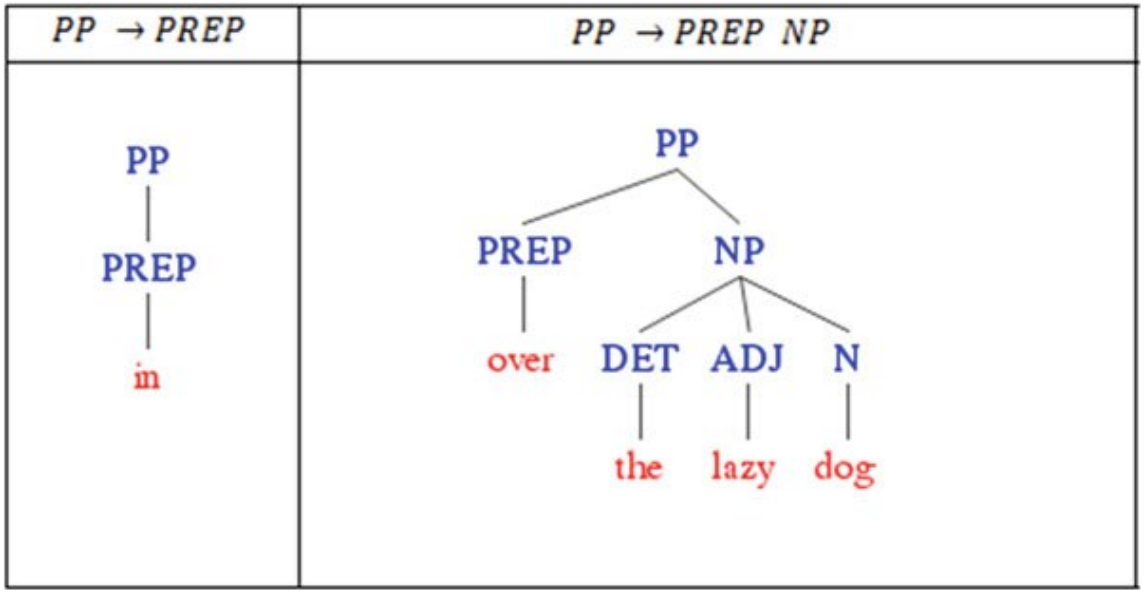
- The rule is of the form VP → V | MD [VP][NP][PP][ADJP][ADVP], where the head word is usually a verb (V) or a modal (MD).
- A modal is itself an auxiliary verb, but we give it a different representation just to distinguish it from a normal verb.
- This is followed by optionally another verb phrase (VP) or noun phrase (NP), prepositional phrase (PP), adjective phrase (ADJP), or adverbial phrase (ADVP).



Language Syntax and Structure-Grammar

Constituency Grammar

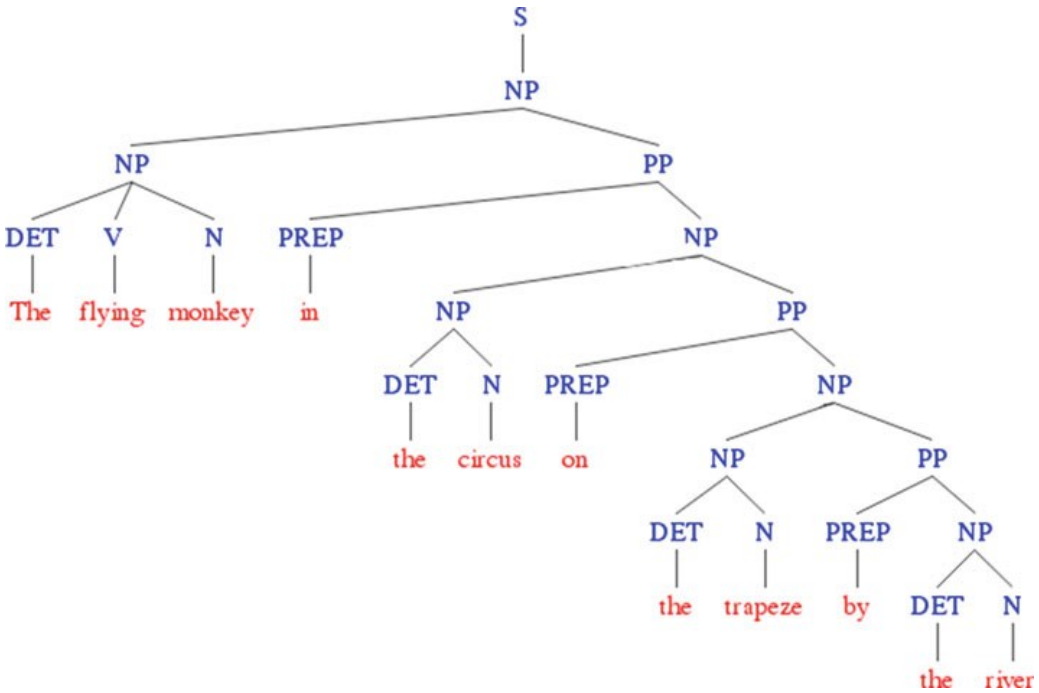
- **Prepositional phrases-** The basic rule has the form PP → PREP [NP], where PREP denotes a preposition, which acts as the head word, and it is optionally followed by a noun phrase (NP)



Language Syntax and Structure-Grammar

Constituency Grammar

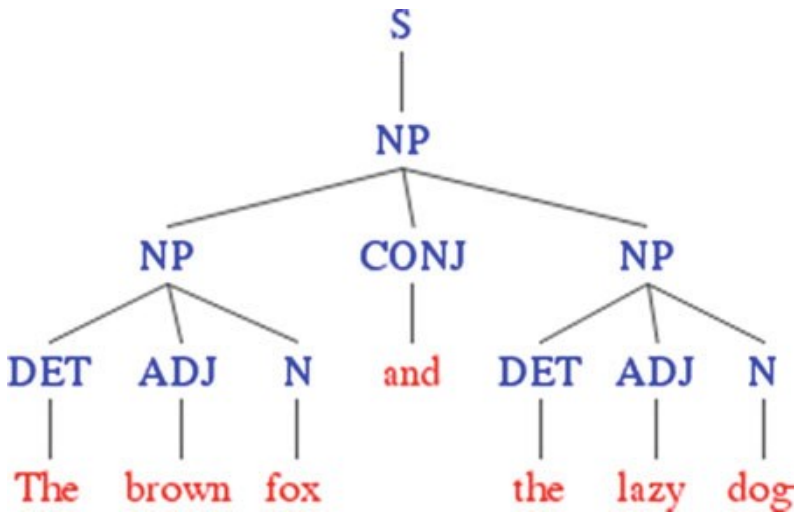
- Recursion is an inherent property of language that allows constituents to be embedded in other constituents, which are depicted by different phrasal categories that appear on both sides of the production rules.
- Recursion lets us create long constituency based syntax trees from sentences.
- A simple example is the representation of the sentence. The flying monkey in the circus on the trapeze by the river depicted by the constituency parse tree.



Language Syntax and Structure-Grammar

Constituency Grammar

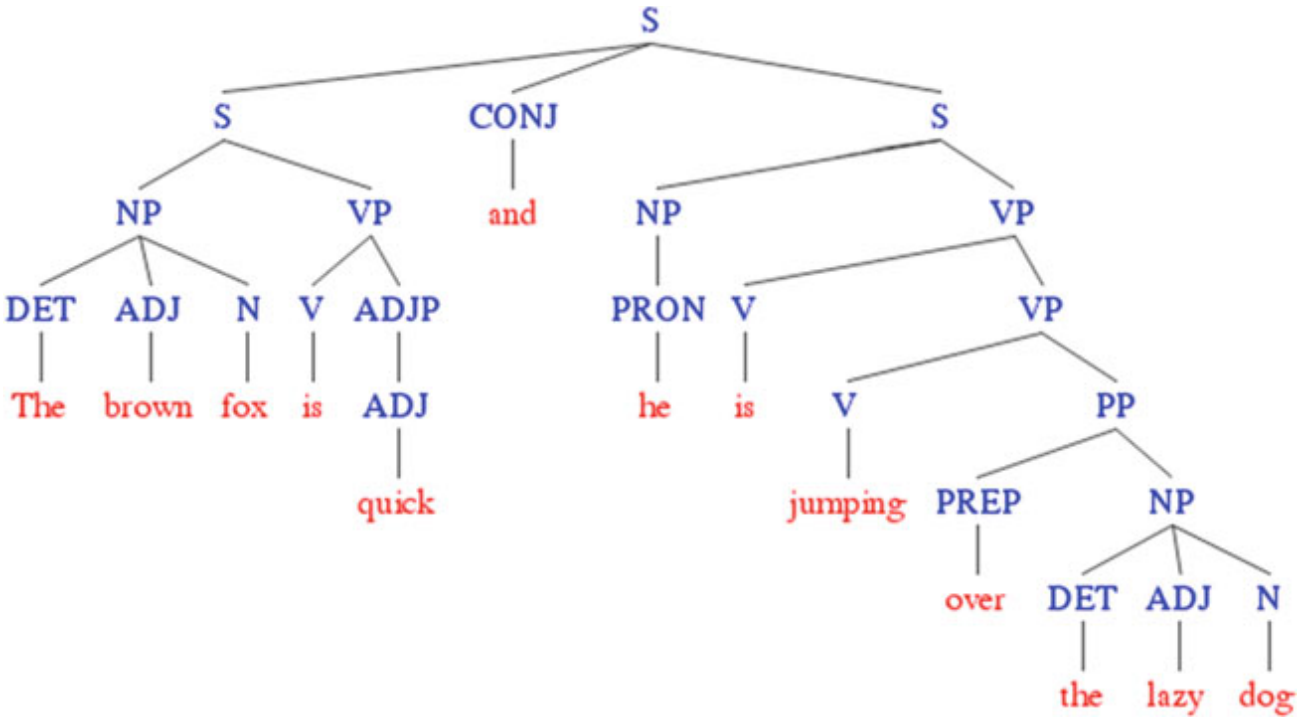
- Conjunctions are used to join clauses and phrases together and form an important part of language syntax.
- Usually words, phrases, and even clauses can be combined together using conjunctions.
- The production rule can be denoted as S → S conj S for all Sε{ S,NP,VP }, where two constituents can be joined together by a conjunction, denoted by conj in the rule.



Language Syntax and Structure-Grammar

Constituency Grammar

- **Constituency grammar for sample sentence**



Language Semantics

- The simplest definition of semantics is the study of meaning.
- Linguistics has its own subfield of linguistic semantics, which deals with the study of meaning in language, the relationships between words, phrases, and symbols, and their indication, meaning, and representation of the knowledge they signify.
- In simple words, semantics is more concerned with the facial expressions, signs, symbols, body language, and knowledge that are transferred when passing messages from one entity to another.

Language Semantics-Lexical Semantic Relations

- Lexical semantics is usually concerned with identifying semantic relations between lexical units in a language and how they are correlated to the syntax and structure of the language.
- Lexical units are usually represented by morphemes, the smallest meaningful and syntactically correct unit of a language.
- Words are inherently a subset of these morphemes.
- Each lexical unit has its own syntax, form, and meaning.
- They also derive meaning from their surrounding lexical units in phrases, clauses, and sentences.
- A lexicon is a complete vocabulary of these lexical units.

Language Semantics-Lexical Semantic Relations

Lemmas and Wordforms

- A lemma is also known as the canonical or citation form for a set of words.
- The lemma is usually the base form of a set of words, known as a lexeme in this context.
- Lemma is the specific base form or head word that represents the lexeme.
- Wordforms are inflected forms of the lemma, which are part of the lexeme and can appear as one of the words from the lexeme in text.
- A simple example would the lexeme {eating, ate, eats}, which contains the wordforms, and their lemma is the word eat .
- These words have specific meaning based on their position among other words in a sentence. This is also known as sense of the word, or wordsense.
- Wordsense gives a concrete representation of the different aspects of a word's meaning.
- Consider the word fair in the following sentences: They are going to the annual fair and I hope the judgement is fair to all.
- Even though the word fair is the same in both the sentences, the meaning changes based on the surrounding words and context.

Language Semantics-Lexical Semantic Relations

Homonyms, Homographs and Homophones

- ***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*.**

Language Semantics-Lexical Semantic Relations

Homonyms, Homographs and Homophones

- ***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.**

Language Semantics-Lexical Semantic Relations

Heteronyms and Heterographs

- **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.

Language Semantics-Lexical Semantic Relations

Polysemes

- Polysemes are words that have the same written form or spelling and different but very relatable meanings.
- While this is very similar to homonymy, the difference is subjective and depends on the context, since these words are relatable to each other.
- A good example is the word bank which can mean

(1) a financial institution,

(2) the bank of the river,

(3) the building that belongs to the financial institution, or

(4) a verb meaning to rely upon .

- These examples use the same word bank and are homonyms. But only (1), (3), and (4) are polysemes representing a common theme (the financial organization representing trust and security).

Language Semantics-Lexical Semantic Relations

Capitonyms

- Capitonyms are words that have the same written form or spelling but have different meanings when capitalized.
- They may or may not have different pronunciations.
- Some examples include the words march (March indicates the month, and march depicts the action of walking) and may (May indicates the month, and may is a modal verb).

Language Semantics-Lexical Semantic Relations

Synonyms and Antonyms

- Synonyms are words that have different pronunciations and spellings but have the same meanings in some or all contexts.
- If two words or lexemes are synonyms, they can be substituted for each other in various contexts, and it signifies them having the same propositional meaning.
- Words that are synonyms are said to be synonymous to each other, and the state of being a synonym is called synonymy.
- Perfect synonymy is, however, almost nonexistent. The reason is that synonymy is more of a relation between senses and contextual meaning rather than just words.
- Consider the synonyms big , huge , and large . They are very relatable and make perfect sense in sentences like That milkshake is really (big/large/huge).
- However, for the sentence Bruce is my big brother , it does not make sense if we substitute big with either huge or large.
- That's because the word big here has a context or sense depicting being grown up or older, and the other two synonyms lack this sense.

Language Semantics-Lexical Semantic Relations

Synonyms and Antonyms

- Antonyms are pairs of words that define a binary opposite relationship.
- These words indicate specific sense and meaning that are completely opposite to each other.
- The state of being an antonym is called antonymy . There are three types of antonyms: graded antonyms , complementary antonyms , and relational antonyms .
- Graded antonyms , as the name suggests, are antonyms with a certain grade or level when measured on a continuous scale, like the pair (fat , skinny).
- Complementary antonyms are word pairs that are opposite in their meaning but cannot be measured on any grade or scale. An example of a complementary antonym pair is (divide , unite).
- Relational antonyms are word pairs that have some relationship between them, and the antonymy is contextual, which is signified by this very relationship. An example of a relational antonym pair is (doctor , patient).

Language Semantics-Lexical Semantic Relations

Hyponyms and Hypernyms

- Hyponyms are words that are usually a subclass of another word. In this case, the hyponyms are generally words with very specific sense and context as compared to the word that is their superclass.
- Hypernyms are the words that act as the superclass to hyponyms and have a more generic sense compared to the hyponyms.
- An example would be the word fruit , which is a hypernym, and the words mango , orange , and pear would be possible hyponyms.
- The relationships depicted between these words are often termed hyponymy and hypernymy .

Representation of Semantics

- Consider the example Get me the book from the table .
- This sentence by nature is a directive, and it directs the listener to do something. Understanding the meaning conveyed by this sentence may involve pragmatics like which specific book? and which specific table? besides the actual deed of getting the book from the table.
- Although the human mind is intuitive, formally representing the meanings and relationships between the various constituents is a challenge—but we can do it using techniques such as propositional logic (PL) and first order logic (FOL).

Representation of Semantics-Propositional logic (PL)

- Propositional logic (PL) , also known as sentential logic or statement logic , is defined as the discipline of logic that is concerned with the study of propositions, statements, and sentences.
- This includes studying logical relationships and properties between propositions and statements, combining multiple propositions to form more complex propositions, and observing how the value of propositions change based on their components and logical operators.
- A proposition or statement is usually declarative and is capable of having a binary truth value that is either true or false.
- A simple example would be the two statements The rocket was faster than the airship and The airship was slower than the rocket , which are distinct but convey the same meaning or proposition.
- However, the terms statement and proposition are often used interchangeably in propositional logic.

Representation of Semantics-Propositional logic (PL)

- The good part about propositional logic is that each proposition has its own truth value , and it is not concerned with further subdividing a proposition into smaller chunks and verifying its logical characteristics.
- Each proposition is considered as an indivisible, whole unit with its own truth value.
- Logical operators may be applied on it and several other propositions.
- Subdividing parts of propositions like clauses or phrases are not considered here.
- To represent the various building blocks of propositional logic, we use several conventions and symbols. Uppercase letters like P and Q are used to denote individual statements or propositions.

Table 1-2. Logical Operators with Their Symbols and Precedence

Sl No.	Operator Symbol	Operator Meaning	Precedence
1	¬	not	Highest
2	∧	and	
3	∨	or	
4	→	if-then	
5	↔	iff (if and only if)	Lowest

Representation of Semantics-Propositional logic (PL)

- Consider the following representations:

P : He is hungry

Q : He will eat a sandwich

- The expression P ^ Q translates to He is hungry and he will eat a sandwich .
- This expresses that the outcome of this operation is itself also a sentence or proposition.
- This is the conjunction operation where P and Q are the conjuncts .
- The outcome of this sentence is True only if both P and Q are True.
- The expression P V Q translates to He is hungry or he will eat a sandwich .
- This expresses that the outcome of this operation is also another proposition formed from the disjunction operation where P and Q are the disjuncts .
- The outcome of this sentence is True if either P or Q or both of them are True .

Representation of Semantics-Propositional logic (PL)

- Consider the following representations:

P : He is hungry

Q : He will eat a sandwich

- The expression P ^ Q translates to He is hungry and he will eat a sandwich .
- This expresses that the outcome of this operation is itself also a sentence or proposition.
- This is the conjunction operation where P and Q are the conjuncts .
- The outcome of this sentence is True only if both P and Q are True.
- The expression P V Q translates to He is hungry or he will eat a sandwich .
- This expresses that the outcome of this operation is also another proposition formed from the disjunction operation where P and Q are the disjuncts .
- The outcome of this sentence is True if either P or Q or both of them are True .

Representation of Semantics-Propositional logic (PL)

- The expression P → Q translates to If he is hungry, then he will eat a sandwich .
- This is the implication operation which determines that P is the premise or antecedent and Q is the consequent .
- It is just like a rule stating that Q will occur only if P has already occurred or is True .
- The expression P↔ Q translates to He will eat a sandwich if and only if he is hungry which is basically a combination of the expressions If he is hungry then he will eat a sandwich (P → Q) and If he will eat a sandwich, he is hungry (Q → P).
- This is the biconditional or equivalence operation that will evaluate to True if and only if the two implication operations described evaluate to True .
- The expression ¬P translates to He is not hungry , which depicts the negation operation and will evaluate to True if and only if P evaluates to False .

Representation of Semantics-Propositional logic (PL)

- A simple example: The statements P : We will play football , Q : The stadium is open , and R : It will rain today can be combined and represented as Q^ ¬R ⟶ P to depict
- the complex proposition If the stadium is open and it does not rain today, then we will play football .
- The semantics of the truth value or outcome of the final proposition can be evaluated based on the truth value of the individual propositions and the operators.

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$	$P \leftrightarrow Q$
False	*False*	*True*	*False*	*False*	*True*	*True*
False	*True*	*True*	*False*	*True*	*True*	*False*
True	*False*	*False*	*False*	*True*	*False*	*False*
True	*True*	*False*	*True*	*True*	*True*	*True*

Representation of Semantics-First Order logic (FOL)

- First order logic (FOL), also known popularly as predicate logic and first order predicate calculus , is defined as a collection of well-defined formal systems which is used extensively in deduction, inference, and representation of knowledge.
- FOL allows us to use quantifiers and variables in sentences, which enable us to overcome some of the limitations with propositional logic.
- If we are to consider the pros and cons of propositional logic (PL), considering the points in its favor, PL is declarative and allows us to easily represent facts using a well-formed syntax.
- PL also allows complex representations like conjunctive, disjunctive, and negated knowledge representations.
- This by nature makes PL compositional wherein a composite or complex proposition is built from the simple propositions that are its components along with logical connectives.
- However, there are several areas where PL is lacking. It is definitely not easy to represent facts in PL because for each possible atomic fact, we will need a unique symbolic representation. Hence, due to this limitation, PL has very limited expressive power.

Representation of Semantics-First Order logic (FOL)

- Objects : These are specific entities or terms with individual unique identities like people, animals, and so on.
- Relations : These are also known as predicates and usually hold among objects or sets of objects and express some form of relationship or connection, like is_man , is_brother , is_mortal .
- Relations typically correspond to verbs.
- Functions : These are a subset of relations where there is always only one output value or object for some given input. Examples would be height , weight , age_of .
- Properties : These are specific attributes of objects that help in distinguishing them from other objects, like round, huge, and so on.
- Connectives : These are the logical connectives that are similar to the ones in PL, which include not (¬), and (^), or (V), implies (→), and iff (if and only if ↔).

Representation of Semantics-First Order logic (FOL)

- Quantifiers : These include two types of quantifiers: universal (∀), which stands for "for all" or "all," and existential (∃), which stands for "there exists" or "exists." They are used for quantifying entities in a logical or mathematical expression.
- Constant symbols : These are used to represent concrete entities or objects in the world. Examples would be John , King , Red , and 7 .
- Variable symbols : These are used to represent variables like x , y , and z .
- Function symbols : These are used to map functions to outcomes. Examples would be, age_of(John) = 25 or color_of(Tree) = Green.
- Predicate symbols : These map specific entities and a relation or function between them to a truth value based on the outcome.
- Examples would be color(sky, blue) = True.

Representation of Semantics-First Order logic (FOL)

- Objects are represented by various terms , which could be either a function , variable , or constant based on the different components depicted previously.
- These terms do not need to be defined and do not return values. Various propositions are usually constructed using predicates and terms with the help of predicate symbols.
- An n-ary predicate is constructed from a function over n-terms which have either a True or False outcome.
- An atomic sentence can be represented by an n-ary predicate, and the outcome is True or False depending on the semantics of the sentence—that is, if the objects represented by the terms have the correct relation among themselves as specified by the predicate.
- A complex sentence or statement is formed using several atomic sentences and logical connectives.
- A quantified sentence adds the quantifiers mentioned earlier to sentences.

Representation of Semantics-First Order logic (FOL)

- Quantifiers are one advantage FOL has over PL, since they enable us to represent statements about entire sets of objects without needing to represent and enumerate each object by a different name.
- The universal quantifier (∀) asserts that a specific relation or predicate is True for all values associated with a specific variable.
- The representation ∀ x F(x) indicates that F holds for all values of x in the domain associated with x .
- An example would be ∀x cat(x) → animal(x) , which indicates that all cats are animals.
- Universal quantifiers are usually used with the implies (→) connective to form rules and statements.
- An important thing to remember is that universal quantifiers are almost never used in statements to indicate some relation for every entity in the world using the conjunction (^)connective.
- An example would be the representation ∀ x dog(x) ^ eats_meat(x) , which actually means that every entity in the world is a dog and they eat meat, which sounds kind of absurd!

Representation of Semantics-First Order logic (FOL)

- The existential quantifier (∃) asserts that a specific relation or predicate holds True for at least some value associated with a specific variable.
- The representation, ∃ x F(x) indicates that F holds for some value of x in the domain associated with x .
- An example would be ∃ x student(x) ^ pass_exam(x) , which indicates that there is at least one student who has passed the exam.
- This quantifier gives FOL a lot of power since we can make statements about objects or entities without specifically naming them.
- Existential quantifiers are usually used with the conjunction ^ connective to form rules and statements.

Representation of Semantics-First Order logic (FOL)

Table 1-3. Representation of Natural Language Statements Using First Order Logic

Sl No.	FOL Representation	Natural Language Statement
1	¬ eats(John, fish)	John does not eat fish
2	is_hot(pie) ∧ is_delicious(pie)	The pie is hot and delicious
3	is_hot(pie) ∨ is_delicious(pie)	The pie is either hot or delicious
4	study(John, exam) → pass(John, exam)	If John studies for the exam, he will pass the exam
5	∀x student(x) → pass(x, exam)	All students passed the exam
6	∃x student(x) ∧ fail(x, exam)	There is at least one student who failed the exam
7	(∃x student(x) ∧ fail(x, exam) ∧ (∀y fail(y, exam) → x=y))	There was exactly one student who failed the exam
8	∀x (spider(x) ∧ black_widow(x)) → poisonous(x)	All black widow spiders are poisonous

Natural Language Processing

- NLP is defined as a specialized field of computer science and engineering and artificial intelligence with roots in computational linguistics.
- It is primarily concerned with designing and building applications and systems that enable interaction between machines and natural languages evolved for use by humans.
- This also makes NLP related to the area of Human-Computer Interaction (HCI) .
- NLP techniques enable computers to process and understand natural human language and utilize it further to provide useful output.

Natural Language Processing-Applications

Machine translation

- It is defined as the technique that helps in providing syntactic, grammatical, and semantically correct translation between any two pair of languages.
- On a simple level, machine translation is the translation of natural language carried out by a machine.
- By default, the basic building blocks for the machine translation process involve simple substitution of words from one language to another, but in that case we ignore things like grammar and phrasal structure consistency.
- Hence, more sophisticated techniques have evolved over a period of time, including combining large resources of text corpora along with statistical and linguistic techniques.
- One of the most popular machine translation systems is Google Translate.
- Over time, machine translation systems are getting better providing translations in real time as you speak or write into the application.

Natural Language Processing-Applications

Speech Recognition Systems

- Most difficult application for NLP.
- This test is defined as a test of intelligence for a computer.
- A question is posed to a computer and a human, and the test is passed if it is impossible to say which of the answers given was given by the human.
- Over time, a lot of progress has been made in this area by using techniques like speech synthesis, analysis, syntactic parsing, and contextual reasoning.
- But one chief limitation for speech recognition systems still remains: They are very domain specific and will not work if the user strays even a little bit from the expected scripted inputs needed by the system.
- Speech-recognition systems are now found in many places, from desktop computers to mobile phones to virtual assistance systems.

Natural Language Processing-Applications

Speech Recognition Systems

- Most difficult application for NLP.
- This test is defined as a test of intelligence for a computer.
- A question is posed to a computer and a human, and the test is passed if it is impossible to say which of the answers given was given by the human.
- Over time, a lot of progress has been made in this area by using techniques like speech synthesis, analysis, syntactic parsing, and contextual reasoning.
- But one chief limitation for speech recognition systems still remains: They are very domain specific and will not work if the user strays even a little bit from the expected scripted inputs needed by the system.
- Speech-recognition systems are now found in many places, from desktop computers to mobile phones to virtual assistance systems.

Natural Language Processing-Applications

Question Answering Systems

- Question Answering Systems (QAS) are built upon the principle of Question Answering, based on using techniques from NLP and information retrieval (IR).
- QAS is primarily concerned with building robust and scalable systems that provide answers to questions given by users in natural language form.
- Imagine being in a foreign country, asking a question to your personalized assistant in your phone in pure natural language, and getting a similar response from it.
- This is the ideal state toward which researchers and technologists are working.
- Some success in this field has been achieved with personalized assistants like Siri and Cortana, but their scope is still limited because they understand only a subset of key clauses and phrases in the entire human natural language.

Natural Language Processing-Applications

Question Answering Systems

- To build a successful QAS, you need a huge knowledgebase consisting of data about various domains.
- Efficient querying systems into this knowledgebase would be leveraged by the QAS to provide answers to questions in natural language form.
- Creating and maintaining a queryable vast knowledgebase is extremely difficult—hence, you find the rise of QAS in niche domains like food, healthcare, e-commerce, and so on.
- Chatbots are one emerging trend that makes extensive use of QAS.

Natural Language Processing-Applications

Contextual Recognition and Resolution

- This covers a wide area in understanding natural language and includes both syntactic and semantic-based reasoning.
- Word sense disambiguation is a popular application, where we want to find out the contextual sense of a word in a given sentence.
- Consider the word book . It can mean an object containing knowledge and information when used as a noun, and it can also mean to reserve a seat or a table when used as a verb.

Natural Language Processing-Applications

Contextual Recognition and Resolution

- Coreference resolution is another problem in linguistics NLP is trying to address.
- By definition, coreference is said to occur when two or more terms or expressions in a body of text refer to the same entity. Then they are said to have the same referent .
- Consider John just told me that he is going to the exam hall .
- In this sentence, the pronoun he has the referent John.
- Resolving such pronouns is a part of coreference resolution, and it becomes challenging once we have multiple referents in a body of text.
- For example, John just talked with Jim. He told me we have a surprise test tomorrow .
- In this body of text, the pronoun he could refer to either John or Jim , thus making pinpointing the exact referent difficult.

Natural Language Processing-Applications

Text Summarization

- The main aim of text summarization is to take a corpus of text documents—which could be a collection of texts, paragraphs, or sentences—and reducing the content appropriately to create a summary that retains the key points of the collection.
- Summarization can be carried out by looking at the various documents and trying to find out the keywords, phrases, and sentences that have an important prominence in the whole collection.
- Two main types of techniques for text summarization include extraction-based summarization and abstraction-based summarization .
- With the advent of huge amounts of text and unstructured data, the need for text summarization in getting to valuable insights quickly is in great demand.

Natural Language Processing-Applications

Text Summarization

- Text-summarization systems usually perform two main types of operations.
- The first is generic summarization , which tries to provide a generic summary of the collection of documents under analysis.
- The second type of operation is query-based summarization, which provides query-relevant text summaries where the corpus is filtered further based on specific queries, relevant keywords and phrases are extracted relevant to the query, and the summary is constructed.

Natural Language Processing-Applications

Text Categorization

- The main aim of text categorization is identifying to which category or class a specific document should be placed based on the contents of the document.
- This is one of the most popular applications of NLP and machine learning because with the right data, it is extremely simple to understand the principles behind its internals and implement a working text categorization system.
- Both supervised and unsupervised machine learning techniques can be used in solving this problem, and sometimes a combination of both is used.
- This has helped build a lot of successful and practical applications, including spam filters and news article categorization.

Text Analytics

- With the advent of huge amounts of computing power, unstructured data, and success with machine learning and statistical analysis techniques, it wasn't long before text analytics started garnering a lot of attention.
- However, text analytics poses some challenges compared to regular analytical methods.
- Free-flowing text is highly unstructured and rarely follows any specific pattern—like weather data or structured attributes in relational databases.
- Hence, standard statistical methods aren't helpful when applied out of the box on unstructured text data.

Text Analytics

- Text analytics , also known as text mining , is the methodology and process followed to derive quality and actionable information and insights from textual data.
- This involves using NLP, information retrieval, and machine learning techniques to parse unstructured text data into more structured forms and deriving patterns and insights from this data that would be helpful for the end user.
- Text analytics comprises a collection of machine learning, linguistic, and statistical techniques that are used to model and extract information from text primarily for analysis needs, including business intelligence, exploratory, descriptive, and predictive analysis.

Text Analytics

- Some of the main techniques and operations in text analytics:
  - Text classification
  - Text clustering
  - Text summarization
  - Sentiment analysis
  - Entity extraction and recognition
  - Similarity analysis and relation modeling

Text Analytics

- Doing text analytics is sometimes a more involved process than normal statistical analysis or machine learning.
- Before applying any learning technique or algorithm, you have to convert the unstructured text data into a format acceptable by those algorithms.
- By definition, a body of text under analysis is often a document, and by applying various techniques we usually convert this document to a vector of words, which is a numeric array whose values are specific weights for each word that could either be its frequency, its occurrence, or various other depictions.
- Often the text needs to be cleaned and processed to remove noisy terms and data, called **text pre-processing .**

Text Analytics

- Once we have the data in a machine-readable and understandable format, we can apply relevant algorithms based on the problem to be solved at hand.
- The applications of text analytics are manifold. Some of the most popular ones include the following:
  - Spam detection
  - News articles categorization
  - Social media analysis and monitoring
  - Bio-medical
  - Security intelligence
  - Marketing and CRM
  - Sentiment analysis
  - Ad placements
  - Chatbots
  - Virtual assistants

Text Processing and Understanding Text

- Popular text pre-processing techniques
  - Tokenization
  - Tagging
  - Chunking
  - Stemming
  - Lemmatization
- Some basic operations much of the time, such as dealing with misspelled text, removing stopwords, and handling other irrelevant components based on the problem to be solved.

Text Processing -Text Tokenization

- Tokens are independent and minimal textual components that have some definite syntax and semantics.
- A paragraph of text or a text document has several components including sentences that can be further broken down into clauses, phrases, and words.
- The most popular tokenization techniques include sentence and word tokenization, which are used to break down a text corpus into sentences, and each sentence into words.
- Thus, tokenization can be defined as the process of breaking down or splitting textual data into smaller meaningful components called tokens.

Text Processing -Text Tokenization

Sentence Tokenization

- Sentence tokenization is the process of splitting a text corpus into sentences that act as the first level of tokens which the corpus is comprised of.
- This is also known as sentence segmentation , because we try to segment the text into meaningful sentences.
- Any text corpus is a body of text where each paragraph comprises several sentences.
- There are various ways of performing sentence tokenization. Basic techniques include looking for specific delimiters between sentences, such as a period (.) or a newline character (\n), and sometimes even a semi-colon (;).
- NLTK framework provides various interfaces for performing sentence tokenization.
  - sent_tokenize, PunktSentenceTokenizer, RegexpTokenizer, Pre-trained sentence tokenization models

Text Processing -Text Tokenization

Word Tokenization

- Word tokenization is the process of splitting or segmenting sentences into their constituent words.
- A sentence is a collection of words, and with tokenization we essentially split a sentence into a list of words that can be used to reconstruct the sentence.
- Word tokenization is very important in many processes, especially in cleaning and normalizing text where operations like stemming and lemmatization work on each individual word based on its respective stems and lemma.
- Similar to sentence tokenization, nltk provides various useful interfaces for word tokenization,
  - word_tokenize
  - TreebankWordTokenizer
  - RegexpTokenizer
  - Inherited tokenizers from RegexpTokenizer

Text Processing -Text Normalization or Wrangling

- Text normalization is defined as a process that consists of a series of steps that should be followed to wrangle, clean, and standardize textual data into a form that could be consumed by other NLP and analytics systems and applications as input.
- Often tokenization itself also is a part of text normalization.
- Besides tokenization, various other techniques include cleaning text, case conversion, correcting spellings, removing stopwords and other unnecessary terms, stemming, and lemmatization.
- Text normalization is also often called text cleansing or wrangling .

Text Processing -Text Normalization or Wrangling

Cleaning Text

- Lot of extraneous and unnecessary tokens and characters should be removed before performing any further operations like tokenization or other normalization techniques.
- This includes extracting out meaningful text from data sources like HTML data, which consists of unnecessary HTML tags, or even data from XML and JSON feeds.
- Clean_html() from nltk or even the BeautifulSoup library to parse HTML data.

Tokenizing Text

- Tokenize text before or after removing unnecessary characters and symbols from the data.
- This choice depends on the problem trying to solve and the data dealt with.

Text Processing -Text Normalization or Wrangling

Removing Special Characters

- One important task in text normalization involves removing unnecessary and special characters.
- These may be special symbols or even punctuation that occurs in sentences. This step is often performed before or after tokenization.
- The main reason for doing so is because often punctuation or special characters do not have much significance when we analyze the text and utilize it for extracting features or information based on NLP and ML.

Tokenizing Text

- Tokenize text before or after removing unnecessary characters and symbols from the data.
- This choice depends on the problem trying to solve and the data dealt with.

Text Processing -Text Normalization or Wrangling

Expanding Contractions

- Contractions are shortened version of words or syllables.
- They exist in either written or spoken forms. Shortened versions of existing words are created by removing specific letters and sounds.
- In case of English contractions, they are often created by removing one of the vowels from the word.
- Examples would be is not to isn't and will not to won't , where you can notice the apostrophe being used to denote the contraction and some of the vowels and other letters being removed.
- Various forms of contractions exist that are tied down to the type of auxiliary verbs that give us normal contractions, negated contractions, and other special colloquial contractions, some of which may not involve auxiliaries.

Text Processing -Text Normalization or Wrangling

Expanding Contractions

- Contractions do pose a problem for NLP and text analytics because, we have a special apostrophe character in the word.
- Plus we have two or more words represented by a contraction, and this opens a whole new can of worms when we try to tokenize this or even standardize the words.
- Hence, there should be some definite process by which we can deal with contractions when processing text.
- Ideally, you can have a proper mapping for contractions and their corresponding expansions and then use it to expand all the contractions in your text.

Text Processing -Text Normalization or Wrangling

Case Conversions

- Modify the case of words or sentences to make things easier, like matching specific words or tokens.
- Usually there are two types of case conversion operations that are used a lot.
- These are lowercase and uppercase conversions, where a body of text is converted completely to lowercase or uppercase.
- There are other forms also, such as sentence case or proper case.
- Lowercase is a form where all the letters of the text are small letters, and in uppercase they are all capitalized.

Text Processing -Text Normalization or Wrangling

Removing Stopwords

- Stopwords , sometimes written stop words , are words that have little or no significance.
- They are usually removed from text during processing so as to retain words having maximum significance and context.
- Stopwords are usually words that end up occurring the most if you aggregated any corpus of text based on singular tokens and checked their frequencies.
- Words like a, the , me , and so on are stopwords.
- There is no universal or exhaustive list of stopwords. Each domain or language may have its own set of stopwords.

Text Processing -Text Normalization or Wrangling

Correcting Words

- The definition of incorrect here covers words that have spelling mistakes as well as words with several letters repeated that do not contribute much to its overall significance.
- To illustrate some examples, the word finally could be mistakenly written as fianlly , or someone expressing intense emotion could write it as finallllyyyyyy .
- The main objective here would be to standardize different forms of these words to the correct form so that we do not end up losing vital information from different tokens in the text.

Text Processing -Text Normalization or Wrangling

Correcting Words-Correcting Repeating Characters

- The first step in our algorithm would be to identify repeated characters in a word using a regex pattern and then use a substitution to remove the characters one by one.
- Consider the word finalllyyy from the earlier example.
- The pattern r'(\w*)(\w)\2(\w*)' can be used to identify characters that occur twice among other characters in the word, and in each step we will try to eliminate one of the repeated characters using a substitution for the match by utilizing the regex match groups (groups 1, 2, and 3) using the pattern r'\1\2\3' and then keep iterating through this process till no repeated characters remain.
- One repeated character is removed at each stage until we end up with the word finaly in the end.
- However, semantically this word is incorrect—the correct word was finally , which we obtained in step 3.
- Utilize the WordNet corpus to check for valid words at each stage and terminate the loop once it is obtained.

Text Processing -Text Normalization or Wrangling

Correcting Words- Correcting Spellings

- Another problem we face is incorrect or wrong spellings that occur due to human error, or even machine-based errors you may have seen thanks to features like auto-correcting text.
- There are various ways of dealing with incorrect spellings where the final objective is to have tokens of text with the correct spelling.
- The main objective is that, given a word, we need to find the most likely word that is the correct form of that word.
- The approach we would follow is to generate a set of candidate words that are near to our input word and select the most likely word from this set as the correct word.
- We use a corpus of correct English words in this context to identify the correct word based on its frequency in the corpus from our final set of candidates with the nearest distance to our input word.
- This distance, which measures how near or far a word is from our input word, is also called edit distance .

Text Processing -Text Normalization or Wrangling

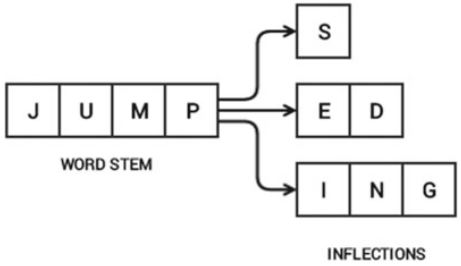
Stemming

- Understanding the process of stemming requires understanding what word stems represent.
- Morphemes, smallest independent unit, consist of units that are stems and affixes.
- Affixes are units like prefixes, suffixes, and so on, which are attached to a word stem to change its meaning or create a new word altogether.
- Word stems are also often known as the base form of a word, and we can create new words by attaching affixes to them in a process known as inflection .
- The reverse of this is obtaining the base form of a word from its inflected form, and this is known as stemming .

Text Processing -Text Normalization or Wrangling

Stemming

- Consider the word JUMP . You can add affixes to it and form new words like JUMPS , JUMPED , and JUMPING .
- In this case, the base word JUMP is the word stem. If we were to carry out stemming on any of its three inflected forms, we would get back the base form.
- This is illustrated in Figure.



- The figure shows how the word stem is present in all its inflections since it forms the base on which each inflection is built upon using affixes.
- Stemming helps us in standardizing words to their base stem irrespective of their inflections, which helps many applications like classifying or clustering text, and even in information retrieval.
- The nltk package has several implementations for stemmers. These stemmers are implemented in the stem module, which inherits the StemmerI interface in the nltk.stem. api module.

Text Processing -Text Normalization or Wrangling

Lemmatization

- The process of lemmatization is very similar to stemming—you remove word affixes to get to a base form of the word. But in this case, this base form is also known as the root word , but not the root stem .
- The difference is that the root stem may not always be a lexicographically correct word; that is, it may not be present in the dictionary. The root word, also known as the lemma , will always be present in the dictionary.
- The lemmatization process is considerably slower than stemming because an additional step is involved where the root form or lemma is formed by removing the affix from the word if and only if the lemma is present in the dictionary.
- The nltk package has a robust lemmatization module that uses WordNet and the word's syntax and semantics, like part of speech and context, to get the root word or lemma.

Understanding Text Syntax and Structure

- Parts of speech (POS) tagging
- Shallow parsing
- Dependency-based parsing
- Constituency-based parsing

- Necessary Dependencies
  - The nltk library, preferably version 3.1 or 3.2.1
  - The spacy library
  - The pattern library
  - The Stanford parser
  - Graphviz and necessary libraries for the same

Text Processing -Text Normalization or Wrangling

Parts of Speech (POS) Tagging

- Parts of speech (POS) are specific lexical categories to which words are assigned based on their syntactic context and role.
- The process of classifying and labeling POS tags for words called parts of speech tagging or POS tagging.
- POS tags are used to annotate words and depict their POS, which is really helpful when we need to use the same annotated text later in NLP-based applications because we can filter by specific parts of speech and utilize that information to perform specific analysis, such as narrowing down upon nouns and seeing which ones are the most prominent, word sense disambiguation, and grammar analysis.
- The Penn Treebank notation -POS tagging.
- **Recommended POS Tagger**: The method is using nltk 's recommended pos_tag() function, which is actually based on the Penn Treebank.

Text Processing -Text Normalization or Wrangling

Parts of Speech (POS) Tagging

Table 3-1. Parts of Speech Tags

Sl No.	TAG	DESCRIPTION	EXAMPLE(S)
1	CC	Coordinating Conjunction	*and, or*
2	CD	Cardinal Number	*five, one, 2*
3	DT	Determiner	*a, the*
4	EX	Existential *there*	*there were two cars*
5	FW	Foreign Word	*d'hoevre, mais*
6	IN	Preposition/ Subordinating Conjunction	*of, in, on, that*
7	JJ	Adjective	*quick, lazy*
8	JJR	Adjective, comparative	*quicker, lazier*
9	JJS	Adjective, superlative	*quickest, laziest*
10	LS	List item marker	*2)*
11	MD	Verb, modal	*could, should*
12	NN	Noun, singular or mass	*fox, dog*
13	NNS	Noun, plural	*foxes, dogs*
14	NNP	Noun, proper singular	*John, Alice*
15	NNPS	Noun, proper plural	*Vikings, Indians, Germans*
16	PDT	Predeterminer	*both the cats*
17	POS	Possessive ending	*boss's*
18	PRP	Pronoun, personal	*me, you*
19	PRP$	Pronoun, possessive	*our, my, your*
20	RB	Adverb	*naturally, extremely, hardly*
21	RBR	Adverb, comparative	*better*
22	RBS	Adverb, superlative	*best*
23	RP	Adverb, particle	*about, up*
24	SYM	Symbol	*%, $*
25	TO	Infinitival to	*how to, what to do*
26	UH	Interjection	*oh, gosh, wow*

Table 3-1. (continued)

Sl No.	TAG	DESCRIPTION	EXAMPLE(S)
27	VB	Verb, base form	*run, give*
28	VBD	Verb, past tense	*ran, gave*
29	VBG	Verb, gerund/ present participle	*running, giving*
30	VBN	Verb, past participle	*given*
31	VBP	Verb, non-3rd person singular present	*I think, I take*
32	VBZ	Verb, 3rd person singular present	*he thinks, he takes*
33	WDT	Wh-determiner	*which, whatever*
34	WP	Wh-pronoun, personal	*who, what*
35	WP$	Wh-pronoun, possessive	*whose*
36	WRB	Wh-adverb	*where, when*
37	NP	Noun Phrase	*the brown fox*
38	PP	Prepositional Phrase	*in between, over the dog*
39	VP	Verb Phrase	*was jumping*
40	ADJP	Adjective Phrase	*warm and snug*
41	ADVP	Adverb Phrase	*also*
42	SBAR	Subordinating Conjunction	*whether or not*
43	PRT	Particle	*up*
44	INTJ	Interjection	*hello*
45	PNP	Prepositional Noun Phrase	*over the dog, as of today*
46	-SBJ	Sentence Subject	*the fox jumped over the dog*
47	-OBJ	Sentence Object	*the fox jumped over the dog*

Text Processing -Text Normalization or Wrangling

Parts of Speech (POS) Tagging-Building Your Own POS Taggers

- Data preparation : Usually consists of pre-processing the data before extracting features and training
- Feature extraction : The process of extracting useful features from raw data that are used to train machine learning models
- Features : Various useful attributes of the data (examples could be age, weight, and so on for personal data)
- Training data : A set of data points used to train a model
- Testing/validation data : A set of data points on which a pretrained model is tested and evaluated to see how well it performs
- Model : Built using a combination of data/features and a machine learning algorithm that could be supervised or unsupervised
- Accuracy : How well the model predicts something (detailed evaluation metrics like precision, recall, and F1-score)

Text Processing -Text Normalization or Wrangling

Parts of Speech (POS) Tagging-Building Your Own POS Taggers

- **Treebank corpus in nltk**

```
from nltk.corpus import treebank
data = treebank.tagged_sents()
train_data = data[:3500]
test_data = data[3500:]
```

```
# get a look at what each data point looks like
In [17]: print train_data[0]
 [(u'Pierre', u'NNP'), (u'Vinken', u'NNP'), (u',', u','), (u'61', u'CD'),
(u'years', u'NNS'), (u'old', u'JJ'), (u',', u','), (u'will', u'MD'),
(u'join', u'VB'), (u'the', u'DT'), (u'board', u'NN'), (u'as', u'IN'), (u'a',
u'DT'), (u'nonexecutive', u'JJ'), (u'director', u'NN'), (u'Nov.', u'NNP'),
(u'29', u'CD'), (u'.', u'.')]

# remember tokens is obtained after tokenizing our sentence
tokens = nltk.word_tokenize(sentence)
In [18]: print tokens
['The', 'brown', 'fox', 'is', 'quick', 'and', 'he', 'is', 'jumping', 'over',
'the', 'lazy', 'dog']


from nltk.tag import DefaultTagger
dt = DefaultTagger('NN')

# accuracy on test data
In [24]: print dt.evaluate(test_data)
0.145415819537
# tagging our sample sentence
In [25]: print dt.tag(tokens)
 [('The', 'NN'), ('brown', 'NN'), ('fox', 'NN'), ('is', 'NN'), ('quick',
'NN'), ('and', 'NN'), ('he', 'NN'), ('is', 'NN'), ('jumping', 'NN'),
('over', 'NN'), ('the', 'NN'), ('lazy', 'NN'), ('dog', 'NN')]
```

Text Processing -Text Normalization or Wrangling

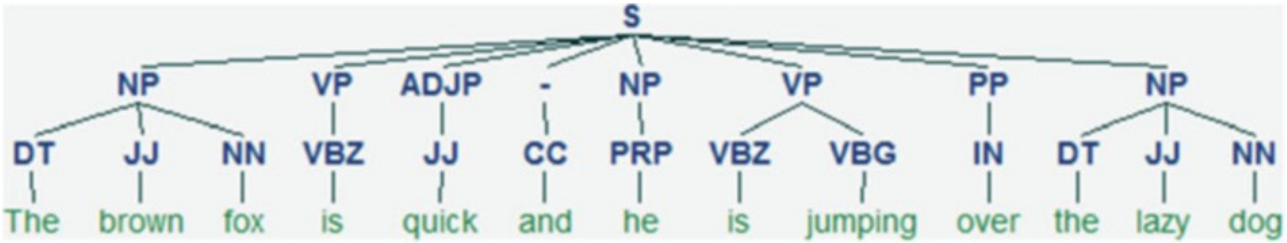
Parts of Speech (POS) Tagging-Building Your Own POS Taggers

```
from nltk.tag import RegexpTagger
# define regex tag patterns
patterns = [
        (r'.*ing$', 'VBG'),               # gerunds
        (r'.*ed$', 'VBD'),                # simple past
        (r'.*es$', 'VBZ'),                # 3rd singular present
        (r'.*ould$', 'MD'),               # modals
        (r'.*\'s$', 'NN$'),               # possessive nouns
        (r'.*s$', 'NNS'),                 # plural nouns
        (r'^-?[0-9]+(.[0-9]+)?$', 'CD'),  # cardinal numbers
        (r'.*', 'NN')                     # nouns (default) ... ]
rt = RegexpTagger(patterns)

# accuracy on test data
In [27]: print rt.evaluate(test_data)
0.240391131765

# tagging our sample sentence

In [28]: print rt.tag(tokens)
 [('The', 'NN'), ('brown', 'NN'), ('fox', 'NN'), ('is', 'NNS'), ('quick',
'NN'), ('and', 'NN'), ('he', 'NN'), ('is', 'NNS'), ('jumping', 'VBG'),
('over', 'NN'), ('the', 'NN'), ('lazy', 'NN'), ('dog', 'NN')]
```

Text Processing -Text Normalization or Wrangling

Shallow Parsing

- Shallow parsing , also known as light parsing or chunking , is a technique of analyzing the structure of a sentence to break it down into its smallest constituents (which are tokens such as words) and group them together into higher-level phrases.
- In shallow parsing, there is more focus on identifying these phrases or chunks rather than diving into further details of the internal syntax and relations inside each chunk.
- The main objective of shallow parsing is to obtain semantically meaningful phrases and observe relations among them.
- Recommended Shallow Parser: pattern package is used to create a shallow parser to extract meaningful chunks out of sentences.



Text Processing -Text Normalization or Wrangling

Shallow Parsing-Building own Shallow Parser

```
# train the shallow parser
ntc = NGramTagChunker(train_data)

# test parser performance on test data
In [114]: print ntc.evaluate(test_data)
ChunkParse score:
    IOB Accuracy:  99.6%
    Precision:     98.4%
    Recall:       100.0%
    F-Measure:     99.2%

# parse our sample sentence
In [115]: tree = ntc.parse(tagged_sentence)
     ...: print tree
(S
  (NP The/DT brown/JJ fox/NN)
  is/VBZ
  (NP quick/JJ)
  and/CC
  (NP he/PRP)
  is/VBZ
  jumping/VBG
  over/IN
  (NP the/DT lazy/JJ dog/NN))
```

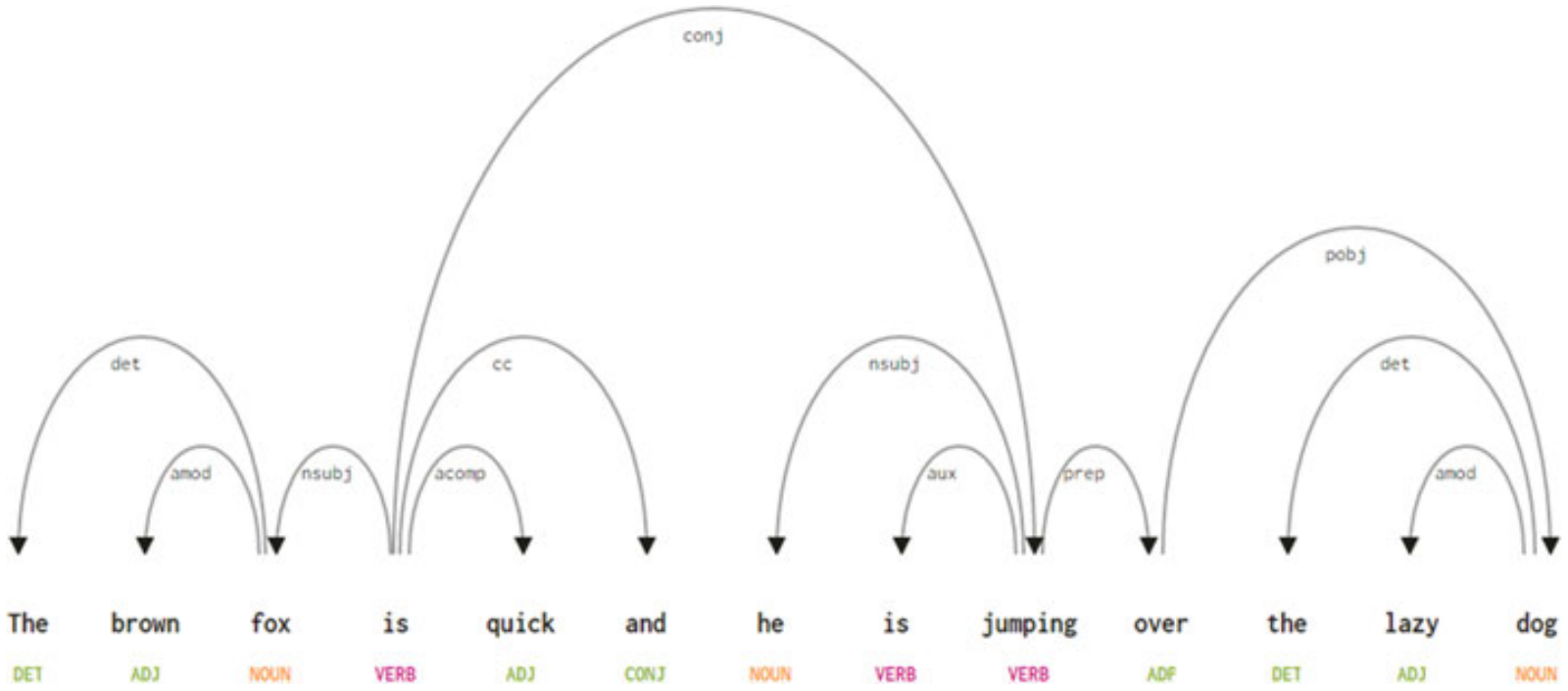
Text Processing -Text Normalization or Wrangling

Dependency-based Parsing

- In dependency-based parsing, we try to use dependency-based grammars to analyze and infer both structure and semantic dependencies and relationships between tokens in a sentence.
- Dependency-based grammars help us in annotating sentences with dependency tags that are one-to-one mappings between tokens signifying dependencies between them.
- A dependency grammar-based parse tree representation is a labelled and directed tree or graph , to be more precise.
- The nodes are always the lexical tokens, and the labelled edges show dependency relationships between the heads and their dependents.
- The labels on the edges indicate the grammatical role of the dependent.

Text Processing -Text Normalization or Wrangling

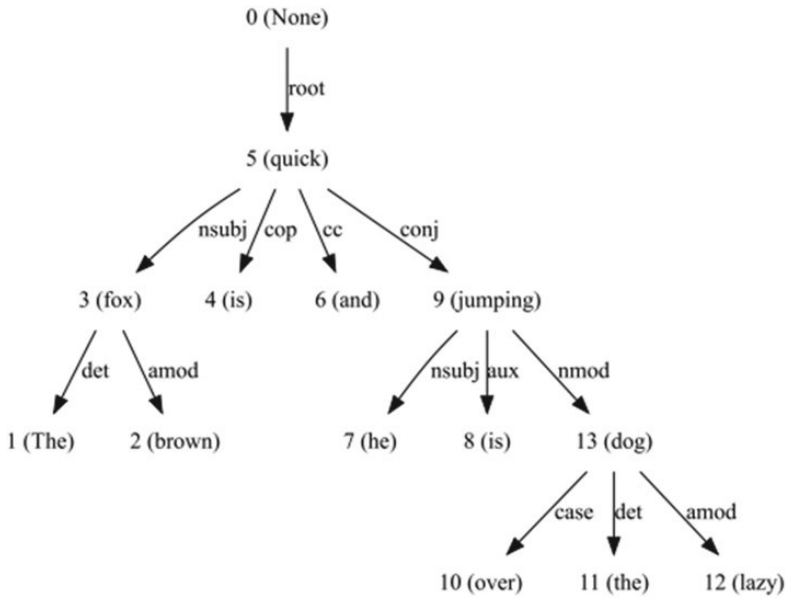
Dependency-based Parsing



- **Recommended Dependency Parsers: Use spacy to analyze our sample sentence and generate each token and its dependencies. Next, use nltk and the Stanford Parser to generate the dependency tree for the sample sentence.**

Text Processing -Text Normalization or Wrangling

Dependency-based Parsing



- **Recommended Dependency Parsers: A side note would be that you will need graphviz installed to generate the annotated dependency tree shown in Figure.**

Text Processing -Text Normalization or Wrangling

Dependency-based Parsing-Building own Parser

- First leverage nltk's DependencyGrammar class to generate production rules from a user input grammar.
- Then use ProjectiveDependencyParser , a projective, production rule-based dependency parser to perform the dependency based parsing.

```
import nltk
tokens = nltk.word_tokenize(sentence)

dependency_rules = """
'fox' -> 'The' | 'brown'
'quick' -> 'fox' | 'is' | 'and' | 'jumping'
'jumping' -> 'he' | 'is' | 'dog'
'dog' -> 'over' | 'the' | 'lazy'
"""
dependency_grammar = nltk.grammar.DependencyGrammar.fromstring(dependency_
rules)
```

```
# print production rules
In [143]: print dependency_grammar
Dependency grammar with 12 productions
  'fox' -> 'The'
  'fox' -> 'brown'
  'quick' -> 'fox'
  'quick' -> 'is'
  'quick' -> 'and'
  'quick' -> 'jumping'
  'jumping' -> 'he'
  'jumping' -> 'is'
  'jumping' -> 'dog'
  'dog' -> 'over'
  'dog' -> 'the'
  'dog' -> 'lazy'

# build dependency parser
dp = nltk.ProjectiveDependencyParser(dependency_grammar)

# parse our sample sentence
res = [item for item in dp.parse(tokens)]
tree = res[0]

# print dependency parse tree
In [145]: print tree
 (quick (fox The brown) is and (jumping he is (dog over the lazy)))
```

Text Processing -Text Normalization or Wrangling

Constituency-based Parsing

- Constituent-based grammars are used to analyze and determine the constituents a sentence is usually composed of.
- Besides determining the constituents, another important objective is to find out the internal structure of these constituents and see how they link to each other.
- There are usually several rules for different types of phrases based on the type of components they can contain, and we can use them to build parse trees.
- In general, a constituency-based grammar helps specify how we can break a sentence into various constituents.
- Once that is done, it further helps in breaking down those constituents into further subdivisions, and this process repeats till we reach the level of individual tokens or words.
- These grammars have various production rules and usually a context-free grammar (CFG) or phrase structured grammar is sufficient for this.

Text Processing -Text Normalization or Wrangling

Constituency-based Parsing

- Once we have a set of grammar rules, a constituency parser can be built that will process input sentences according to these rules and help in building a parse tree.
- The parser is what brings the grammar to life and can be said to be a procedural interpretation of the grammar.
- There are various types of parsing algorithms, including the following:
  - Recursive Descent parsing
  - Shift Reduce parsing
  - Chart parsing
  - Bottom-up parsing
  - Top-down parsing
  - PCFG parsing

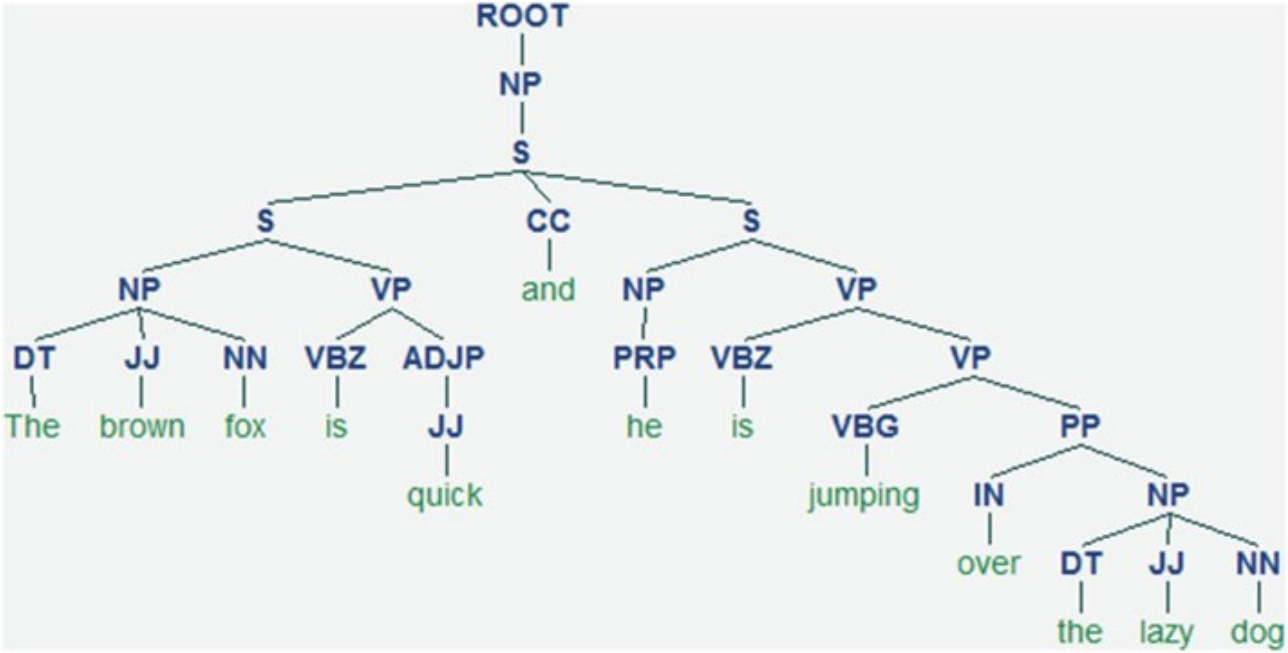
Text Processing -Text Normalization or Wrangling

Constituency-based Parsing

- **Recursive Descent parsing** usually follows a top-down parsing approach and it reads in tokens from the input sentence and tries to match them with the terminals from the grammar production rules.
- It keeps looking ahead by one token and advances the input read pointer each time it gets a match.
- **Shift Reduce parsing** follows a bottom-up parsing approach where it finds sequences of tokens (words/phrases) that correspond to the righthand side of grammar productions and then replaces it with the lefthand side for that rule.
- This process continues until the whole sentence is reduced to give us a parse tree.
- **Chart parsing** uses dynamic programming ,which stores intermediate results and reuses them when needed to get significant efficiency gains.
- In this case, chart parsers store partial solutions and look them up when needed to get to the complete solution.

Text Processing -Text Normalization or Wrangling

Constituency-based Parsing



- **Recommended Constituency Parsers: Use nltk and the StanfordParser to generate parse trees.**

Text Processing -Text Normalization or Wrangling

Constituency-based Parsing-Building own Parser

- To build your own CFG, you can use the nltk.CFG.fromstring function to feed in your own production rules and then use parsers like ChartParser or RecursiveDescentParser , both of which belong to the nltk package.
- We will use nltk's ViterbiParser here to train a parser on the treebank corpus that provides annotated parse trees for each sentence in the corpus.

```
import nltk
from nltk.grammar import Nonterminal
from nltk.corpus import treebank

# get training data
training_set = treebank.parsed_sents()

# view a sample training sentence
In [161]: print training_set[1]
(S
  (NP-SBJ (NNP Mr.) (NNP Vinken))
  (VP
    (VBZ is)
    (NP-PRD
      (NP (NN chairman))
      (PP
        (IN of)
        (NP
          (NP (NNP Elsevier) (NNP N.V.))
          (, ,)
          (NP (DT the) (NNP Dutch) (VBG publishing) (NN group))))))
  (. .))
```

```
# build the parser
viterbi_parser = nltk.ViterbiParser(treebank_grammar)

# get sample sentence tokens
tokens = nltk.word_tokenize(sentence)

# get parse tree
In [170]: result = list(viterbi_parser.parse(tokens))
Traceback (most recent call last):
  File "<ipython-input-170-c2cdab3cd56c>", line 1, in <module>
    result = list(viterbi_parser.parse(tokens))
  File "C:\Anaconda2\lib\site-packages\nltk\parse\viterbi.py", line 112, in
parse
    self._grammar.check_coverage(tokens)
ValueError: Grammar does not cover some of the input words: u"'brown',
'fox', 'lazy', 'dog'".
```

Feature Engineering

- Features are unique, measurable attributes or properties for each observation or data point in a dataset.
- Features are usually numeric in nature and can be absolute numeric values or categorical features that can be encoded as binary features for each category in the list using a process called one-hot encoding.
- The process of extracting and selecting features is both art and science, and this process is called feature extraction or feature engineering.
- Usually extracted features are fed into ML algorithms for learning patterns that can be applied on future new data points for getting insights.
- These algorithms usually expect features in the form of numeric vectors because each algorithm is at heart a mathematical operation of optimization and minimizing loss and error when it tries to learn patterns from data points and observations.
- So, with textual data there is the added challenge of figuring out how to transform textual data and extract numeric features from it.

Feature Engineering-Techniques

- The Vector Space Model is a concept and model that is very useful in case we are dealing with textual data and is very popular in information retrieval and document ranking.
- The Vector Space Model, also known as the Term Vector Model, is defined as a mathematical and algebraic model for transforming and representing text documents as numeric vectors of specific terms that form the vector dimensions.
- Mathematically this can be defines as follows. Say we have a document D in a document vector space VS.
- The number of dimensions or columns for each document will be the total number of distinct terms or words for all documents in the vector space.
- So, the vector space can be denoted

$$VS = \{W_1, W_2, \ldots, W_n\}$$

where there are n distinct words across all documents.

Feature Engineering-Techniques

- Now we can represent document D in this vector space as

$$D = \{w_{D1}, w_{D2}, \ldots, w_{Dn}\}$$

where wDn denotes the weight for word n in document D.

- This weight is a numeric value and can represent anything, ranging from the frequency of that word in the document, to the average frequency of occurrence, or even to the TF-IDF weight.

- Feature-extraction techniques:
  - Bag of Words model
  - TF-IDF model
  - Advanced word vectorization models

Required packages: nltk, gensim, and scikit-learn libraries, which you can install using pip.

Feature Engineering-Techniques

Bag of Words

- The Bag of Words model is one of the simplest yet most powerful techniques to extract features from text documents.
- The essence of this model is to convert text documents into vectors such that each document is converted into a vector that represents the frequency of all the distinct words that are present in the document vector space for that specific document.
- Considering our sample vector from the previous mathematical notation for D, the weight for each word is equal to its frequency of occurrence in that document.
- we can even create the same model for individual word occurrences as well as occurrences for n-grams, which would make it an n-gram Bag of Words model such that frequency of distinct n-grams in each document would also be considered.

Feature Engineering-Techniques

Bag of Words

```
from sklearn.feature_extraction.text import CountVectorizer

def bow_extractor(corpus, ngram_range=(1,1)):

    vectorizer = CountVectorizer(min_df=1, ngram_range=ngram_range)
    features = vectorizer.fit_transform(corpus)
    return vectorizer, features
```

```
CORPUS = [
'the sky is blue',
'sky is blue and sky is beautiful',
'the beautiful sky is so blue',
'i love blue cheese'
]

new_doc = ['loving this blue sky today']
```

```
# build bow vectorizer and get features
In [371]: bow_vectorizer, bow_features = bow_extractor(CORPUS)
     ...: features = bow_features.todense()
     ...: print features
[[0 0 1 0 1 0 1 0 1]
 [1 1 1 0 2 0 2 0 0]
 [0 1 1 0 1 0 1 1 1]
 [0 0 1 1 0 1 0 0 0]]
```

```
In [379]: display_features(features, feature_names)
   and  beautiful  blue  cheese  is  love  sky  so  the
0    0          0     1       0   1     0    1   0    1
1    1          1     1       0   2     0    2   0    0
2    0          1     1       0   1     0    1   1    1
3    0          0     1       1   0     1    0   0    0

In [380]: display_features(new_doc_features, feature_names)
   and  beautiful  blue  cheese  is  love  sky  so  the
0    0          0     1       0   0     0    1   0    0
```

Feature Engineering-Techniques

TF-IDF Model

- The Bag of Words model is good, but the vectors are completely based on absolute frequencies of word occurrences.
- This has some potential problems where words that may tend to occur a lot across all documents in the corpus will have higher frequencies and will tend to overshadow other words that may not occur as frequently but may be more interesting and effective as features to identify specific categories for the documents.
- This is where TF-IDF comes into the picture. TF-IDF stands for Term Frequency-Inverse Document Frequency, a combination of two metrics: term frequency and inverse document frequency.
- This technique was originally developed as a metric for ranking functions for showing search engine results based on user queries and has come to be a part of information retrieval and text feature extraction now.

Feature Engineering-Techniques

TF-IDF Model

- TF-IDF is the product of two metrics and can be represented as tfidf =tf ′idf , where term frequency (tf) and inverse-document frequency (idf) represent the two metrics.
- Term frequency denoted by tf is what we had computed in the Bag of Words model.
- Term frequency in any document vector is denoted by the raw frequency value of that term in a particular document. Mathematically it can be represented as $tf(w, D) = f_{w_D}$,
- where fwD denotes frequency for word w in document D, which becomes the term frequency (tf).
- There are various other representations and computations for term frequency, such as converting frequency to a binary feature where 1 means the term has occurred in the document and 0 means it has not.
- Sometimes you can also normalize the absolute raw frequency using logarithms or averaging the frequency.

Feature Engineering-Techniques

TF-IDF Model

- Inverse document frequency denoted by idf is the inverse of the document frequency for each term.
- It is computed by dividing the total number of documents in our corpus by the document frequency for each term and then applying logarithmic scaling on the result.
- In our implementation we will be adding 1 to the document frequency for each term just to indicate that we also have one more document in our corpus that essentially has every term in the vocabulary.
- This is to prevent potential division-by-zero errors and smoothen the inverse document frequencies.
- We also add 1 to the result of our idf computation to avoid ignoring terms completely that might have zero idf.
- Mathematically our implementation for idf can be represented by

Feature Engineering-Techniques

TF-IDF Model

- Mathematically our implementation for idf can be represented by

$$idf(t) = 1 + \log\frac{C}{1 + df(t)}$$

- where idf(t) represents the idf for the term t, C represents the count of the total number of documents in our corpus, and df(t) represents the frequency of the number of documents in which the term t is present.
- Thus the term frequency-inverse document frequency can be computed by multiplying the above two measures together.
- The final TF-IDF metric we will be using is a normalized version of the tfidf matrix we get from the product of tf and idf.
- We will normalize the tfidf matrix by dividing it with the L2 norm of the matrix, also known as the Euclidean norm, which is the square root of the sum of the square of each term's tfidf weight.
- final tfidf feature vector $tfidf = \frac{tfidf}{\|tfidf\|}$, where $\|tfidf\|$ represents the Euclidean L2 norm for the tfidf matrix.

Feature Engineering-Techniques

TF-IDF Model-Code Snippet

```
from sklearn.feature_extraction.text import TfidfTransformer

def tfidf_transformer(bow_matrix):

    transformer = TfidfTransformer(norm='l2',
                                   smooth_idf=True,
                                   use_idf=True)
    tfidf_matrix = transformer.fit_transform(bow_matrix)
    return transformer, tfidf_matrix
```

```
    import numpy as np
    from feature_extractors import tfidf_transformer
    feature_names = bow_vectorizer.get_feature_names()

    # build tfidf transformer and show train corpus tfidf features
    In [388]: tfidf_trans, tdidf_features = tfidf_transformer(bow_featur
         ...: features = np.round(tdidf_features.todense(), 2)
         ...: display_features(features, feature_names)
        and  beautiful  blue  cheese    is  love   sky    so   the
0  0.00       0.00  0.40    0.00  0.49  0.00  0.49  0.00  0.60
1  0.44       0.35  0.23    0.00  0.56  0.00  0.56  0.00  0.00
2  0.00       0.43  0.29    0.00  0.35  0.00  0.35  0.55  0.43
3  0.00       0.00  0.35    0.66  0.00  0.66  0.00  0.00  0.00

    # show tfidf features for new_doc using built tfidf transformer
    In [389]: nd_tfidf = tfidf_trans.transform(new_doc_features)
         ...: nd_features = np.round(nd_tfidf.todense(), 2)
         ...: display_features(nd_features, feature_names)
       and  beautiful  blue  cheese   is  love   sky   so  the
0  0.0        0.0  0.63     0.0  0.0   0.0  0.77  0.0  0.0
```

```
# compute tfidf feature matrix
tfidf = tf * idf

# show tfidf feature matrix
In [410]: display_features(np.round(tfidf, 2), feature_names)
    and  beautiful  blue  cheese    is  love   sky    so   the
0  0.00       0.00  1.0     0.00  1.22  0.00  1.22  0.00  1.51
1  1.92       1.51  1.0     0.00  2.45  0.00  2.45  0.00  0.00
2  0.00       1.51  1.0     0.00  1.22  0.00  1.22  1.92  1.51
3  0.00       0.00  1.0     1.92  0.00  1.92  0.00  0.00  0.00
```

```
# compute L2 norms
norms = norm(tfidf, axis=1)

# print norms for each document
In [412]: print np.round(norms, 2)
[ 2.5   4.35  3.5   2.89]

# compute normalized tfidf
norm_tfidf = tfidf / norms[:, None]

# show final tfidf feature matrix
In [415]: display_features(np.round(norm_tfidf, 2), feature_names)
    and  beautiful  blue  cheese    is  love   sky    so   the
0  0.00       0.00  0.40    0.00  0.49  0.00  0.49  0.00  0.60
1  0.44       0.35  0.23    0.00  0.56  0.00  0.56  0.00  0.00
2  0.00       0.43  0.29    0.00  0.35  0.00  0.35  0.55  0.43
3  0.00       0.00  0.35    0.66  0.00  0.66  0.00  0.00  0.00
```

Feature Engineering-Techniques

Advanced Word Vectorization Models

- There are various approaches to creating more advanced word vectorization models for extracting features from text data.
- A couple of them will be discussed that use Google's popular word2vec algorithm.
- The word2vec model, is a neural network–based implementation that learns distributed vector representations of words based on continuous Bag of Words and skip-gram–based architectures.
- The word2vec framework is much faster than other neural network–based implementations and does not require manual labels to create meaningful representations among words.
- We will be using the gensim library in our implementation, which is Python implementation for word2vec that provides several high-level interfaces for easily building these models.

Feature Engineering-Techniques

Advanced Word Vectorization Models

- The basic idea is to provide a corpus of documents as input and get feature vectors for them as output.
- Internally, it constructs a vocabulary based on the input text documents and learns vector representations for words based on various techniques mentioned earlier, and once this is complete, it builds a model that can be used to extract word vectors for each word in a document.
- Using various techniques like average weighting or tfidf weighting, we can compute the averaged vector representation of a document using its word vectors.

Feature Engineering-Techniques

Advanced Word Vectorization Models-Parameters

- **size:** This parameter is used to set the size or dimension for the word vectors and can range from tens to thousands.
- **window**: This parameter is used to set the context or window size. which specifies the length of the window of words that should be considered for the algorithm to take into account as context when training.
- **min_count**: This parameter specifies the minimum word count needed across the corpus for the word to be considered in the vocabulary.
- This helps in removing very specific words that may not have much significance because they occur very rarely in the documents.
- **sample:** This parameter is used to downsample effects of occurrence of frequent words. Values between 0.01 and 0.0001 are usually ideal.

Feature Engineering-Techniques

Advanced Word Vectorization Models

- Once we build a model, we will define and implement two techniques of combining word vectors together in text documents based on certain weighing schemes.
- Averaged word vectors
- TF-IDF weighted word vectors
- Ilustrating Feature-extraction process by building our word2vec model on our sample training corpus before going into further implementations.

```
# tokenize corpora
TOKENIZED_CORPUS = [nltk.word_tokenize(sentence)
                    for sentence in CORPUS]
tokenized_new_doc = [nltk.word_tokenize(sentence)
                    for sentence in new_doc]

# build the word2vec model on our training corpus
model = gensim.models.Word2Vec(TOKENIZED_CORPUS, size=10, window=10,
                               min_count=2, sample=1e-3)
```

Feature Engineering-Techniques

Advanced Word Vectorization Models-Averaged word vectors

```
In [430]: print model['sky']
[ 0.01608407 -0.04819566  0.04227461 -0.03011346  0.0254148   0.01728328
  0.0155535   0.00774884 -0.02752112  0.01646519]

In [431]: print model['blue']
[-0.0472235   0.01662185 -0.01221706 -0.04724348 -0.04384995  0.00193343
 -0.03163504 -0.03423524  0.02661656  0.03033725]
```

Vector representation of each word

- Each word vector is of length 10 based on the size parameter specified earlier.
- But when we deal with sentences and text documents, they are of unequal length, and we must carry out some form of combining and aggregation operations to make sure the number of dimensions of the final feature vectors are the same, regardless of the length of the text document, number of words, and so on.

Feature Engineering-Techniques

Advanced Word Vectorization Models-Averaged word vectors

- An average weighted word vectorization scheme, where for each text document we will extract all the tokens of the text document, and for each token in the document we will capture the subsequent word vector if present in the vocabulary.
- We will sum up all the word vectors and divide the result by the total number of words matched in the vocabulary to get a final resulting averaged word vector representation for the text document.
- This can be mathematically represented using the equation

$$AWV(D) = \frac{\sum_{1}^{n} wv(w)}{n}$$

- where AVW(D) is the averaged word vector representation for document D, containing words w1, w2, …, wn, and wv(w) is the word vector representation for the word w.

Feature Engineering-Techniques

Advanced Word Vectorization Models-Averaged word vectors

```
model := the word2vec model we built
vocabulary := unique_words(model)
document := [words]
matched_word_count := 0
vector := []

for word in words:
        if word in vocabulary:
                vector := vector + model[word]
                matched_word_count :=  matched_word_count + 1

averaged_word_vector := vector / matched_word_count
```

```
# get averaged word vectors for our training CORPUS
In [445]: avg_word_vec_features = averaged_word_vectorizer(corpus=TOKENIZED_
CORPUS,
     ...:                                                  model=model,
     ...:                                                  num_features=10)
     ...: print np.round(avg_word_vec_features, 3)
[[ 0.006 -0.01   0.015 -0.014  0.004 -0.006 -0.024 -0.007 -0.001  0.   ]
 [-0.008 -0.01   0.021 -0.019 -0.002 -0.002 -0.011  0.002  0.003 -0.001]
 [-0.003 -0.007  0.008 -0.02  -0.001 -0.004 -0.014 -0.015  0.002 -0.01 ]
 [-0.047  0.017 -0.012 -0.047 -0.044  0.002 -0.032 -0.034  0.027  0.03 ]]

# get averaged word vectors for our test new_doc
In [447]: nd_avg_word_vec_features = averaged_word_
vectorizer(corpus=tokenized_new_doc,
     ...:                                                  model=model,
     ...:                                                  num_
                                                           features=10)
     ...: print np.round(nd_avg_word_vec_features, 3)
[[-0.016 -0.016  0.015 -0.039 -0.009  0.01  -0.008 -0.013  0.     0.023]]
```

```
import numpy as np

# define function to average word vectors for a text document
def average_word_vectors(words, model, vocabulary, num_features):

    feature_vector = np.zeros((num_features,),dtype="float64")
    nwords = 0.

    for word in words:
        if word in vocabulary:
            nwords = nwords + 1.
            feature_vector = np.add(feature_vector, model[word])
```

Feature Engineering-Techniques

Advanced Word Vectorization Models-TF-IDF Weighted Averaged Word Vectors

- Our previous vectorizer simply sums up all the word vectors pertaining to any document based on the words in the model vocabulary and calculates a simple average by dividing with the count of matched words.
- Introduce a new and novel technique of weighing each matched word vector with the word TF-TDF score and summing up all the word vectors for a document and dividing it by the sum of all the TF-IDF weights of the matched words in the document.
- This would basically give us a TF-IDF weighted averaged word vector for each document.

$$TWA(D) = \frac{\sum_{1}^{n} wv(w) * tfidf(w)}{n}$$

- where TWA(D) is the TF-IDF weighted averaged word vector representation for document D, containing wordsw1, w2, …, wn, where wv(w) is the word vector representation and tfidf(w) is the TF-IDF weight for the wordw.

Feature Engineering-Techniques

Advanced Word Vectorization Models-TF-IDF Weighted Averaged Word Vectors

```
model := the word2vec model we built
vocabulary := unique_words(model)
document := [words]
tfidfs := [tfidf(word) for each word in words]
matched_word_wts := 0
vector := []

for word in words:
        if word in vocabulary:
                word_vector := model[word]
                weighted_word_vector := tfidfs[word] x word_vector
                vector := vector + weighted_word_vector
                matched_word_wts :=  matched_word_wts + tfidfs[word]

tfidf_wtd_avgd_word_vector := vector / matched_word_wts
```

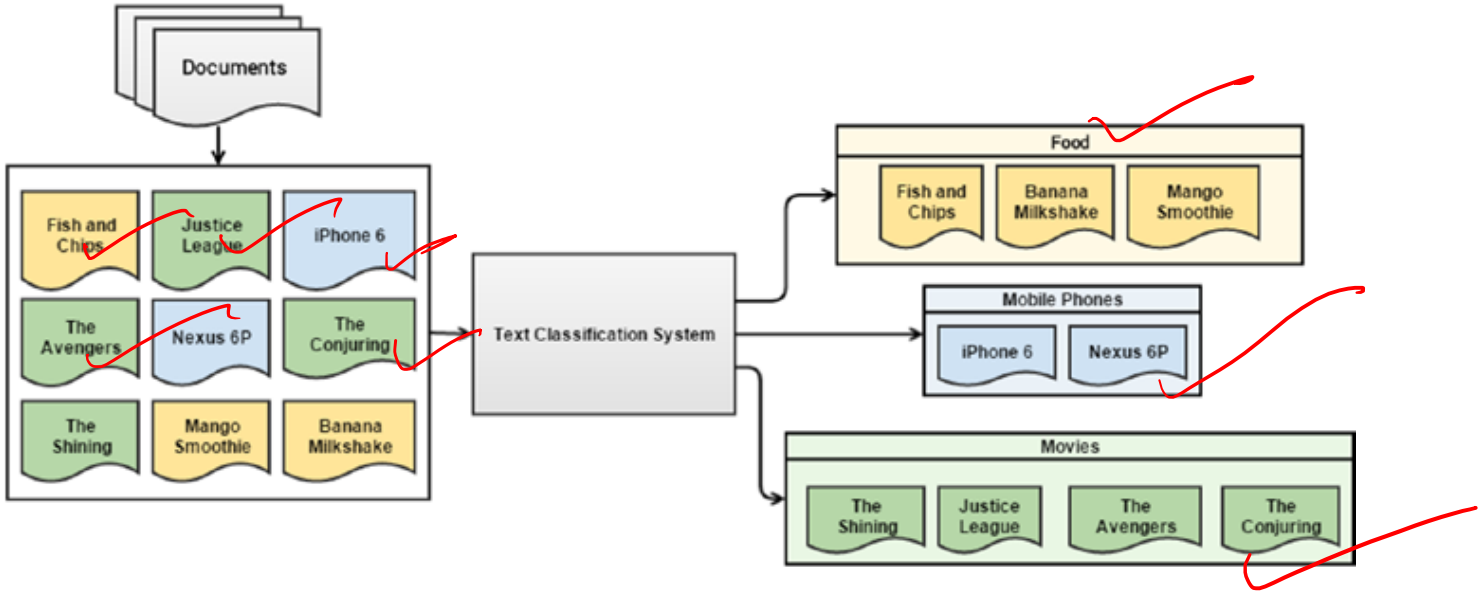
```
     ...: print np.round(wt_tfidf_word_vec_features, 3)
[[ 0.011 -0.011  0.014 -0.011  0.007 -0.007 -0.024 -0.008 -0.004 -0.004]
 [ 0.    -0.014  0.028 -0.014  0.004 -0.003 -0.012  0.011 -0.001 -0.002]
 [-0.001 -0.008  0.007 -0.019  0.001 -0.004 -0.012 -0.018  0.001 -0.014]
 [-0.047  0.017 -0.012 -0.047 -0.044  0.002 -0.032 -0.034  0.027  0.03 ]]

# compute avgd word vector for test new_doc
In [454]: nd_wt_tfidf_word_vec_features = tfidf_weighted_averaged_word_
vectorizer(corpus=tokenized_new_doc, tfidf_vectors=nd_tfidf, tfidf_
vocabulary=vocab, model=model, num_features=10)
     ...: print np.round(nd_wt_tfidf_word_vec_features, 3)
[[-0.012 -0.019  0.018 -0.038 -0.006  0.01  -0.006 -0.011 -0.003  0.023]]
```

```
# define function to compute tfidf weighted averaged word vector for a document
def tfidf_wtd_avg_word_vectors(words, tfidf_vector, tfidf_vocabulary, model,
num_features):

    word_tfidfs = [tfidf_vector[0, tfidf_vocabulary.get(word)]
                   if tfidf_vocabulary.get(word)
                   else 0 for word in words]
    word_tfidf_map = {word:tfidf_val for word, tfidf_val in zip(words, word_
    tfidfs)}

    feature_vector = np.zeros((num_features,),dtype="float64")
```

Text Classification

- Text or document classification is the process of assigning text documents into one or more classes or categories, assuming that we have a predefined set of classes.
- Documents here are textual documents, and each document can contain a sentence or even a paragraph of words.
- A text classification system would successfully be able to classify each document to its correct class(es) based on inherent properties of the document.
- Mathematically, given some description and attributes d for a document D, where $d \in D$, and a set of predefined classes or categories, $C=\{c1, c2, c3,...cn\}$.
- The actual document D can have many inherent properties and attributes that lead it to being an entity in a high-dimensional space.
- Using a subset of that space with a limit set of descriptions and features depicted by d, we should be able to successfully assign the original document D to its correct class Cx using a text classification system T. This can be represented by $T \rightarrow Cx$.

Text Classification



- We can see there are several documents representing products which can be assigned to various categories of food, mobile phones, and movies.
- Initially, these documents are all present together, just as a text corpus has various documents in it.
- Once it goes through a text classification system, represented as a black box here, we can see that each document is assigned to one specific class or category we had defined previously.

Text Classification

- Here the documents are just represented by their names, but in real data, they can contain much more, including descriptions of each product, specific attributes such as movie genre, product specifications, constituents, and many more properties that can be used as features in the text classification system to make document identification and classification easier.
- Types:
  - Content-based classification
  - Request-based classification
- **Content based classification** is the type of text classification where priorities or weights are given to specific subjects or topics in the text content that would help determine the class of the document.
- A conceptual example would be that a book with more than 30 percent of its content about food preparations can be classified under cooking/recipes.
- **Request-based classification** is influenced by user requests and is targeted towards specific user groups and audiences.
- This type of classification is governed by specific policies and ideals

Automated Text Classification

- To automate text classification, several ML techniques are used.
- There are mainly two types of ML techniques that are relevant to solving this problem:
  - Supervised machine learning
  - Unsupervised machine learning
    - Reinforcement learning and semi-supervised learning are also used.

Automated Text Classification- Unsupervised Learning

- Unsupervised learning refers to specific ML techniques or algorithms that do not require any pre-labelled training data samples to build a model.
- We usually have a collection of data points, which could be text or numeric, depending on the problem we are trying to solve.
- We extract features from each of the data points using a process known as feature extraction and then feed the feature set for each data point into our algorithm.
- We are trying to extract meaningful patterns from the data, such as trying to group together similar data points using techniques like clustering or summarizing documents based on topic models.
- This is extremely useful in text document categorization and is also called document clustering, where we cluster documents into groups purely based on their features, similarity, and attributes, without training any model on previously labelled data.

Automated Text Classification- Supervised Learning

- Supervised learning refers to specific ML techniques or algorithms that are trained on pre-labelled data samples known as training data.
- Features or attributes are extracted from this data using feature extraction, and for each data point we will have its own feature set and corresponding class/label.
- The algorithm learns various patterns for each type of class from the training data.
- Once this process is complete, we have a trained model.
- This model can then be used to predict the class for future test data samples once we feed their features to the model.
- Thus the machine has actually learned, based on previous training data samples, how to predict the class for new unseen data samples.

Automated Text Classification- Supervised Learning

Types

- **Classification:** The process of supervised learning is referred to as classification when the outcomes to be predicted are distinct categories, thus the outcome variable is a categorical variable in this case.
- Examples would be news categories or movie genres.

- **Regression:** Supervised learning algorithms are known as regression algorithms when the outcome we want to predict is a continuous numeric variable.
- Examples would be house prices or people's weights.

ML-based Text Classification

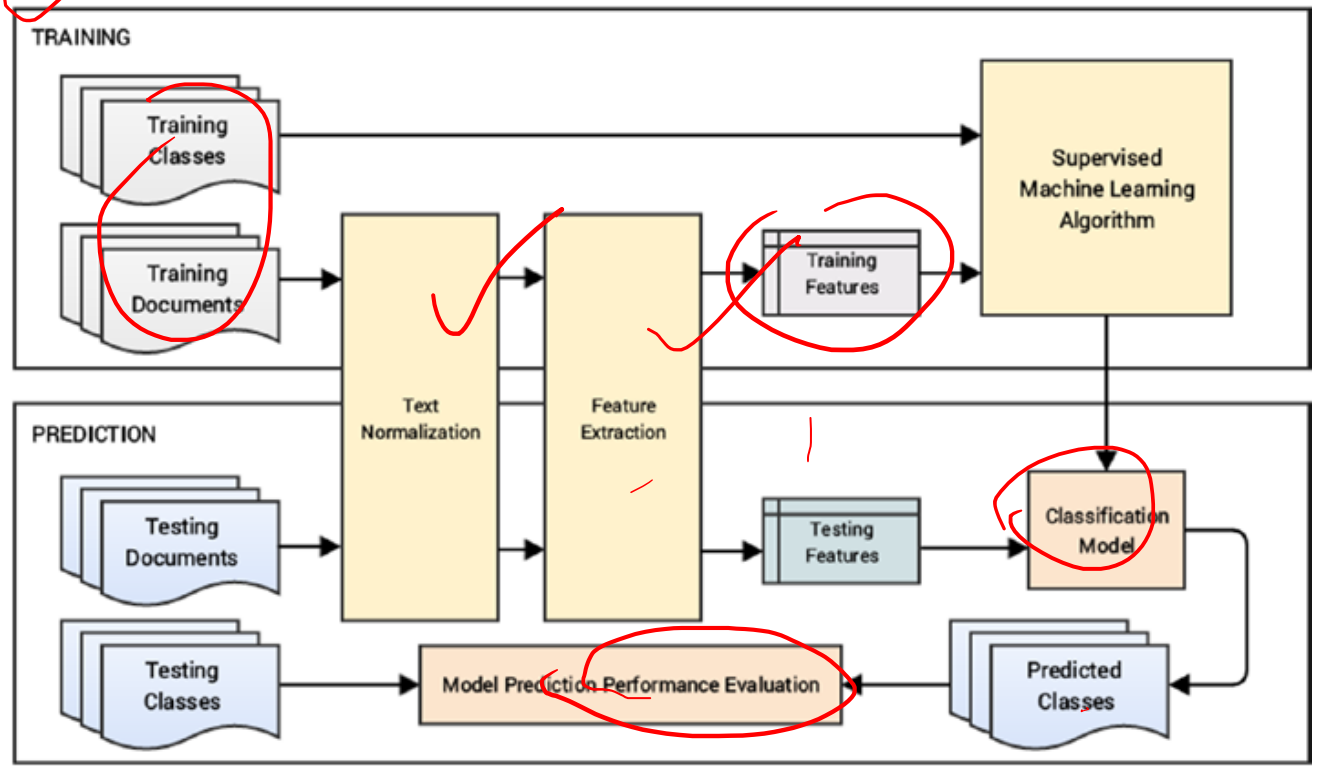
- We have a training set of documents labelled with their corresponding class or category.
- This can be represented by TS, which is a set of paired documents and labels, TS = $TS = \{(d_1, c_1), (d_2, c_2), \ldots, (d_n, c_n)\}$ where d1, d2, …, dn is the list of text documents, and their corresponding labels are c1, c2, …, cn such that $c_x \in C = \{c1, c2, \ldots, cn\}$
- where cx denotes the class label for document x and C denotes the set of all possible distinct classes, any of which can be the class or classes for each document.
- Assuming we have our training set, we can define a supervised learning algorithm F such that when it is trained on our training dataset TS, we build a classification model or classifier γ such that we can say that F (TS) =g .
- Thus the supervised learning algorithm F takes the input set of (document, class) pairs TS and gives us the trained classifier γ, which is our model. This process is known as the **training process.**
- This model can then take a new, previously unseen document ND and predict its class C_{ND} such that $c_{ND} \in C$. This process is known as the **prediction process** and can be represented by $\gamma : TD \rightarrow c_{ND}$

ML-based Text Classification

- There are a few types of text classification based on the number of classes to predict and the nature of predictions.
- These types of classification are based on the dataset, the number of classes/categories pertaining to that dataset, and the number of classes that can be predicted on any data point:
- **Binary classification** is when the total number of distinct classes or categories is two in number and any prediction can contain either one of those classes.
- **Multi-class classification**, also known as multinomial classification, refers to a problem where the total number of classes is more than two, and each prediction gives one class or category that can belong to any of those classes.
- This is an extension of the binary classification problem where the total number of classes is more than two.
- **Multi-label classification** refers to problems where each prediction can yield more than one outcome/predicted class for any data point.

Text Classification Blueprint

- The following main steps outline a typical workflow for a text classification system
  1. Prepare train and test datasets
  2. Text normalization
  3. Feature extraction
  4. Model training
  5. Model prediction and evaluation
  6. Model deployment



Text Classification Blueprint

- There are two main boxes for Training and Prediction, which are the two main processes involved in building a text classifier.
- In general, the dataset we have is usually divided into two or three splits called the training, validation (optional), and testing datasets, respectively.
- There is an overlap of the Text Normalization and Feature Extraction modules in Figure for both processes, indicating that no matter which document we want to classify and predict its class, it must go through the same series of transformations in both the training and prediction process.
- Each document is first pre-processed and normalized, and then specific features pertaining to the document are extracted.
- These processes are always uniform in both the training and prediction processes to make sure that our classification model performs consistently in its predictions.

Text Classification Blueprint

- In the Training process, each document has its own corresponding class/category that was manually labeled or curated beforehand.
- These training text documents are processed and normalized in the Text Normalization module, giving us clean and standardized training text documents.
- They are then passed to the Feature Extraction module where different types of feature-extraction techniques are used to extract meaningful features from the processed text documents.
- These features are usually numeric arrays or vectors because standard ML algorithms work on numeric vectors.
- Once we have our features, we select a supervised ML algorithm and train our model.

Text Classification Blueprint

- Training the model involves feeding the feature vectors for the documents and the corresponding labels such that the algorithm is able to learn various patterns corresponding to each class/category and can reuse this learned knowledge to predict classes for future new documents.
- Often an optional validation dataset is used to evaluate the performance of the classification algorithm to make sure it generalizes well with the data during training.
- A combination of these features and the ML algorithm yields a Classification Model, which is the end stage of the Training process.
- Often this model is tuned using various model parameters with a process called hyperparameter tuning to build a better performing optimal model.

Text Classification Blueprint

- The Prediction process shown in the figure involves trying to either predict classes for new documents or evaluating how predictions are working on testing data.
- The test dataset documents go through the same process of normalization and feature extraction, and the test document features are passed to the trained Classification Model, which predicts the possible class for each of the documents based on previously learned patterns.
- If you have the true class labels for the documents that were manually labelled, you can evaluate the prediction performance of the model by comparing the true labels and the predicted labels using various metrics like accuracy.
- This would give an idea of how well the model performs its predictions for new documents.

Classification Models

- Classification models are supervised machine learning algorithms that are used to classify, categorize, or label data points based on what it has observed in the past.
- Each classification algorithm is a supervised learning algorithm so it requires training data.
- This training data consists of a set of training observations where each observation is a pair that consists of an input data point, which is usually a feature vector like we observed earlier, and a corresponding output outcome for that input observation.
- There are three stages that classification algorithms go through during the modeling phase:
  - Training
  - Evaluation
  - Tuning

Classification Models

- **Evaluation** involves trying to test the prediction performance of our model to see how well it has trained and learned on the training dataset.
- For this, we use a validation dataset and test the performance of our model by predicting on that dataset and testing our predictions against the actual class labels, also called the **ground truth**.
- We also often use **cross validation** where the data is divided into folds and a chunk of it is used for training and the remaining is used to validate the trained model.
- A point to remember is that we also tune the model based on the validation results to get to an optimal configuration, which yields the maximum accuracy and minimum error.
- We also evaluate our model against a **holdout or test dataset**, but we never tune our model against that dataset because that would lead to the model being biased or overfit against very specific features from the test dataset.
- The holdout or test dataset is somewhat of a representative sample of what new real data samples might look like, for which the model will generate predictions and how it might perform on these new data samples.

Classification Models

Algorithms

- Multinomial Naïve Bayes
- Logistic regression
- Support vector machines
- Random forest
- Gradient boosting machine

- The last two models mentioned in this list are ensemble techniques, which use a collection or ensemble of models to learn and predict outcomes, including random forests and gradient boosting.

Multinomial Naïve Bayes

- Special case of the popular Naïve Bayes algorithm used specifically for prediction and classification tasks where we have more than two classes.
- The Naïve Bayes algorithm is a supervised learning algorithm that puts into action the very popular Bayes theorem.
- However there is a "naïve" assumption here that each feature is completely independent of the others.
- Nevertheless, this algorithm still works remarkably well in many use cases related to classification, including multi-class document classification, spam filtering, and so on.
- They can train really fast compared to other classifiers and work well even when we do not have sufficient training data.
- Models often do not perform well when they have a lot of features and this phenomenon is known as curse of dimensionality.
- Naïve Bayes takes care of this problem by decoupling the class variable related conditional feature distributions, thus leading to each distribution being independently estimated as a single dimension distribution.

Naïve Bayes

Mathematically, given a response class variable y and a set of n features in the form of a feature vector, {x1, x2, … , xn} and using the Bayes theorem, we can denote the probability of the occurrence of y given the features as follows:

$$P(y|x_1, x_2, \ldots, x_n) = \frac{P(y) \times P(x_1, x_2, \ldots, x_n|y)}{P(x_1, x_2, \ldots, x_n)}$$

under the assumption that

$$P(x_i|y, x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|y)$$

and for all *i* we can represent this as follows:

$$P(y|x_1, x_2, \ldots, x_n) = \frac{P(y) \times \prod_{i=1}^{n} P(x_i|y)}{P(x_1, x_2, \ldots, x_n)}$$

where *i* ranges from 1 to **n**. In simple terms, this can be written as follows:

$$posterior = \frac{prior \times likelihood}{evidence}$$

and now since $P(x_1, x_2, \ldots, x_n)$ is constant, the model can be expressed as follows:

$$P(y|x_1, x_2, \ldots, x_n) \propto P(y) \times \prod_{i=1}^{n} P(x_i|y)$$

Multinomial Naïve Bayes

- **Multinomial Naïve Bayes** is an extension of the algorithm for predicting and classifying data points, **where the number of distinct classes or outcomes are more than two.**
- In this case, the feature vectors are usually assumed to be word counts from the bag of words model, but TF-IDF-based weights also work.
- One limitation is that **negative weight based features can't be fed** into this algorithm.
- This distribution can be represented as py = {py1, py2, … , pyn} for each class label y and the total number of features is n which could be represented as the total vocabulary of distinct words or terms in text analytics.
- From the equation, pyi = P(xi|y) represents the probability of feature i in any observation sample that has an outcome or class y.

Multinomial Naïve Bayes

- The parameter py can be estimated with a smoothened version of maximum likelihood estimation (with relative frequency of occurrences) and represented as follows:

$$\hat{p}_{yi} = \frac{F_{yi} + \alpha}{F_y + \alpha n}$$

- Where $F_{yi} = \sum_{x \in TD} x_i$ is the frequency of occurrence for the feature i in a sample for class label y in our training dataset TD and $F_y = \sum_{i=1}^{|TD|} F_{yi}$ is the total frequency of all features for the class label y.
- $\alpha \geq 0$, which accounts for the features that are not present in the learning data points and helps get rid of zero probability related issues. There are some specific settings for this parameter, which are used quite often.
- The value of $\alpha = 1$ is known as Laplace smoothing and $\alpha < 1$ is known as Lidstone smoothing.
- The Scikit-Learn library provides an excellent implementation for Multinomial Naïve Bayes in the class MultinomialNB.

Logistic Regression

- The logistic regression model is a statistical model also known as the logit or logistic model since it uses the logistic (popularly also known as sigmoid) mathematical function to estimate the parameter values.
- These are the coefficients of all our features such that the overall loss is minimized when predicting the outcome
- It doesnot focus on errors but more about maximizing the likelihood of the predicted values to the observed values using Maximum-Likelihood Estimation (MLE).
- Considering a binary classification problem of predicting two classes, a 0 or a 1, in the logistic model, the log-odds (the logarithm of the odds) for the class/category labelled as 1 are basically the equation of the linear regression model (linear combination of one or more independent features, which can be categorical or continuous).
- However, we need to predict discrete classes or categories.
- Thus, the corresponding probability of the class labeled 1 can vary between 0 and 1, depicting the confidence of the prediction.
- The function that helps us convert the log-odds to probability is the logistic function.

Logistic Regression

- A standard multiple linear regression model, depicted as follows: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$ such that {x1, x2, … , xn} are our features and we are trying to estimate the coefficients, {β1, β2, … , βn}.
- To predict the categorical classes, represent this as the log-odds, as follows using the logit of the probability p.

$$logodds = logit(p) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

- This means that if p is the probability of predicting a specific class, $\frac{p}{1-p}$ the odds of that is, which is basically the ratio of the favorable outcomes to the unfavorable outcomes.
- Logit of p is basically the log-odds $logit(p) = \log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$
- Class probability values $p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n)}}$
- In Scikit-Learn, the LogisticRegression model can be leveraged to use the logistic regression model for classification.
- The solvers implemented in the LogisticRegression class are "liblinear", "newton-cg", "lbfgs", "sag", and "saga"

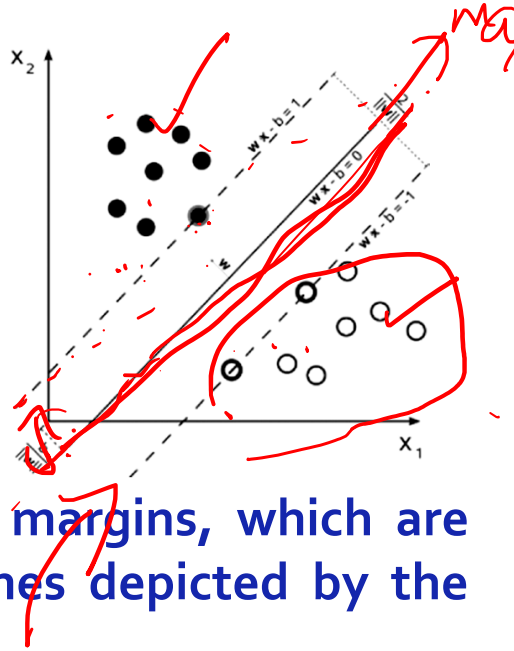
Support Vector Machines

- In machine learning, support vector machines, known popularly as SVMs, are supervised learning algorithms.
- They are used for classification, regression, novelty and anomaly, and outlier detection.
- Considering a binary classification problem, if we have training data such that each data point or observation belongs to a specific class, the SVM algorithm can be trained based on this data such that it can assign future data points into one of the two classes.
- This algorithm represents the training data samples as points in space such that points belonging to either class can be separated by a wide gap between them (hyperplane) and the new data points to be predicted are assigned classes based on which side of this hyperplane they fall into.
- This process is for a typical linear classification process.
- However, SVM can also perform non-linear classification by an interesting approach known as a kernel trick, where kernel functions are used to operate on high-dimensional feature spaces that are non-linear separable.

Support Vector Machines

- The SVM algorithm takes in a set of training data points and constructs a hyperplane of a collection of hyperplanes for a high-dimensional feature space.
- The larger the margins of the hyperplane, the better the separation. This leads to lower generalization errors of the classifier.
- Considering a training dataset of n data points $(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)$ such that the class variable yi ∈ {−1, 1} where each value indicates the class corresponding to the point $\vec{x}_i$.
- Each data point $\vec{x}_i$ is a feature vector. The objective of the SVM algorithm is to find the max-margin hyperplane which separates the set of data points having class label of yi = 1 from the set of data points having class label yi = − 1 so that the distance between the hyperplane and sample data points from either class nearest to it is maximized.
- These sample data points are known as the support vectors.

Support Vector Machines



- Figure shows the hyperplane and the support vectors.
- The hyperplane can be defined as the set of points $\vec{x}_i$ which satisfy $\vec{w}\cdot\vec{x}+b=0$ where $\vec{w}$ is the normal vector to the hyperplane and $\frac{b}{\|\vec{w}\|}$ gives us the offset of the hyperplane from the origin to the support vectors highlighted in Figure.
- When the data is linearly separable we have hard margins, which are basically represented by the two parallel hyperplanes depicted by the dotted lines.
- This helps in separating the data points belonging to the two different classes. This is done by taking into account that the distance between them is as large as possible.
- The region bounded by these two hyperplanes forms the margin with the max-margin hyperplane being in the middle. These hyperplanes have the equations $\vec{w}\cdot\vec{x}+b=1$ and $\vec{w}\cdot\vec{x}+b=-1$

Support Vector Machines

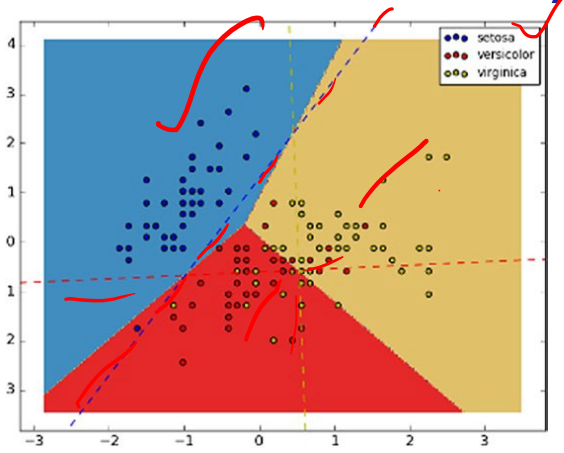
- The data points that are not linearly separable, for which we can use the hinge loss function, which can be represented as

$$\max\left(0, 1 - y_i\left(\vec{w} \cdot \vec{x}_i + b\right)\right)$$

- The Scikit-Learn implementation of SVM can be found in SVC, LinearSVC, or SGDClassifier, where we use the hinge loss function (set by default) to optimize and build the model.
- This loss function helps us get the soft margins and is often known as a soft-margin SVM.
- We can also use different kernel functions to convert the existing feature space into an even higher dimensional feature space, where the data can be separated linearly.
- This is popularly known as the kernel trick in SVM.

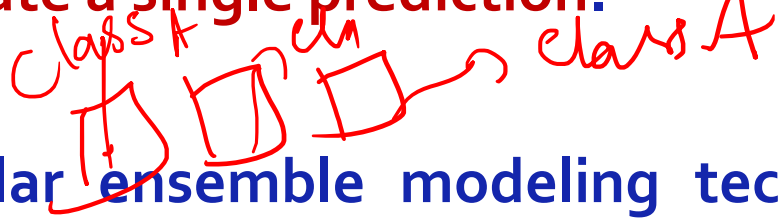
Support Vector Machines

- For a multi-class classification problem, if we have n classes, for each class a binary classifier is trained and learned that helps is separating between each class and the other n-1 classes.
- During prediction, the scores (distances to hyperplanes) for each classifier are computed and the maximum score is chosen for selecting the class label.
- The stochastic gradient descent is often used for minimizing the loss function in SVM algorithms.
- Figure shows how three classifiers are trained in total for a three-class SVM problem over the very popular iris dataset.



Ensemble Models

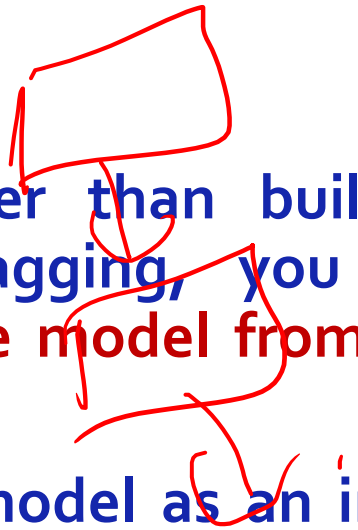
- Ensemble models are essentially models or meta-estimators that are literally made up of other models or estimators.
- These sub-models are models that are simple estimators and may not be able to make accurate predictions to the extent of what you get when you combine several of these estimators.
- In the case of random forest, the sub-models are decision trees. Typically, random forests train many decision trees and combine them to generate a single prediction.



Two categories

- Bagging: A very popular ensemble modeling technique. In bagging, you take subsets of the data (bootstrap samples typically) and train a model on each subset in parallel.
- Then the subsets are allowed to simultaneously vote on the outcome and the final outcome is usually an average aggregation.
- The random forest model is perhaps the most popular example of a bagging model.

Ensemble Models



Boosting:

- Another ensemble technique where, rather than building multiple models in parallel like with bagging, you use sequential modeling and try to improve one model from the mistakes of the previous model!
- Boosting typically uses the output of one model as an input into the next in a form of sequential processing.
- Gradient boosting machines is one of the most popular boosting models.

Random Forest

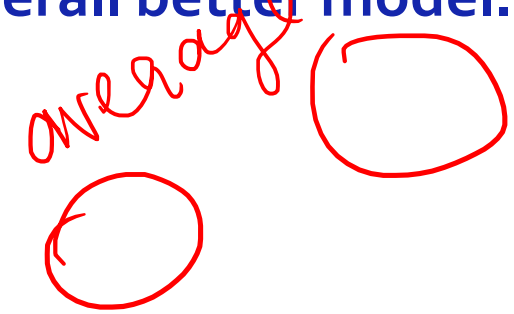
- Decision trees are a family of supervised machine learning algorithms that can represent and interpret sets of rules automatically from the underlying data.
- They use metrics like information gain and gini-index to build the tree.
- However, a major drawback of decision trees is that since they are non-parametric, the more data there is, greater the depth of the tree.
- We can end up with really huge and deep trees that are prone to overfitting.
- The model might work really well on training data, but instead of learning, it just memorizes all the training samples and builds very specific rules to them.
- Hence, it performs really poorly on the test data. Random forests try to tackle this problem.

Random Forest

- Random forest is a meta-estimator or an ensemble model that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.
- The sub-sample size is always the same as the original input sample size, but the samples are drawn with replacement (bootstrap samples).
- In random forests, all the trees are trained in parallel (bagging model/bootstrap aggregation).
- Besides this, each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set.
- Also, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features.
- Instead, the split that is picked is the best split among a random subset of the features.

Random Forest

- Thus the randomness introduced in a random forest is both due to random sampling of data and random selection of features when splitting nodes in each tree.
- Hence, due to this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random decision tree).
- However, due to averaging, the overall variance of the model decreases significantly as compared to the increase in bias and hence it gives us an overall better model.



Random Forest

- When building a random forest, you can set specific model parameters for both the base decision trees and the overall forest.
- For the trees, you usually have the same parameters as a normal decision tree model like the tree depth, number of leaves, number of features in each split, samples per leaf, criteria for the node splits, information gain, and gini impurity.
- For the forest, you can tune the total number of trees needed, the number of features to be used per tree, and so on.

Gradient Boosting Machines

- Gradient boosting machines, popularly known as GBMs, can be used for regression and classification.
- Typically, GBMs builds an additive model in a forward stage-wise sequential fashion; they allow for the optimization of arbitrary differentiable loss functions.
- GBMs can usually work on any combination of models (weak learners) and loss functions.
- Scikit-Learn uses GBRTs (Gradient Boosted Regression Trees), which are generalized boosting models that can be applied to arbitrary differentiable loss functions.
- The beauty of this model is that is accurate and can be used for both regression and classification problems.
- GBRT considers additive models $F(x)=\sum_{m=1}^{M}\gamma_m h_m(x)$

where hm(x) can be defined as the base models or weak learners—in this case the decision trees.

Evaluating Classification Models

- Training, tuning, and building models are an important part of the whole analytics lifecycle, but it's even more important to know how well these models are performing.
- Performance of classification models is usually based on how well they are predicting outcomes for new data points.
- Usually this performance is measured against a test or holdout dataset, which consists of data points that were not used to influence or train the classifier in any way.
- This test dataset has several observations and their corresponding labels. We extract features in the same way as when training the model.
- These features are fed to the already trained model and we obtain predictions for each data point.
- These predictions are then matched against the actual labels to see how well or how accurately the model has predicted.
- There are several metrics to determine a model's prediction performance.

Evaluating Classification Models

- Accuracy
- Precision
- Recall
- F1-score
- Wisconsin Diagnostic Breast Cancer dataset.
- This dataset has 30 attributes or features and a corresponding label for each data point (breast mass) depicting if it has cancer (malignant: label value 1) or no cancer (benign: label value 0).
- Building a basic logistic regression model and evaluating it.
- we have 398 observations in our train dataset and 171 observations in our test dataset.
- It leverages the Scikit-Learn metrics module

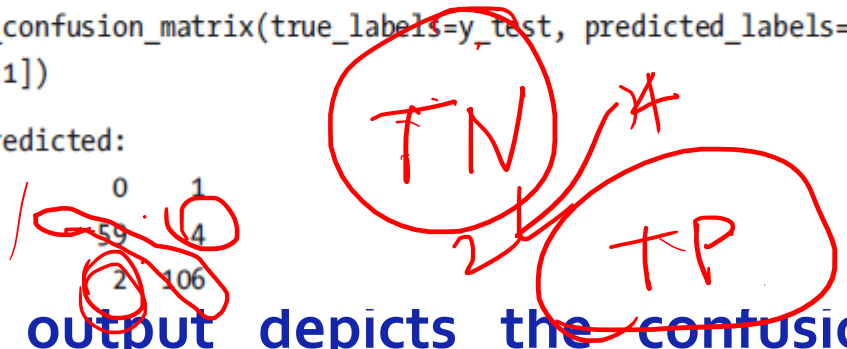
Evaluating Classification Models- Confusion Matrix

- A confusion matrix is one of the most popular ways to evaluate a classification model.
- Although the matrix by itself is not a metric, the matrix representation can be used to define a variety of metrics, all of which become important in some specific case or scenario.
- A confusion matrix can be created for both a binary classification as well as a multi-class classification model.
- A confusion matrix is created by comparing the predicted class label of a data point with its actual class label.
- This comparison is repeated for the whole dataset and the results of this comparison are compiled in a matrix or tabular format. This resultant matrix is our confusion matrix.
- logistic regression model on our breast cancer dataset and look at the confusion matrix for the model predictions on the test dataset.

Evaluating Classification Models- Confusion Matrix

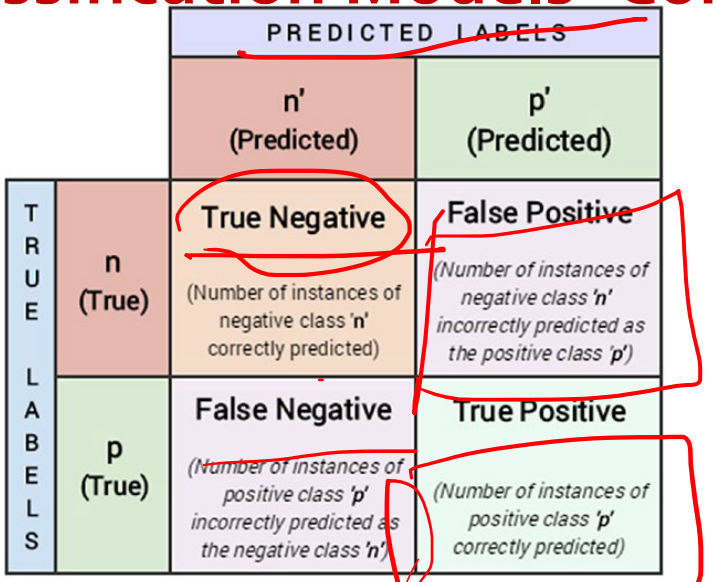
```
from sklearn import linear_model
# train and build the model
logistic = linear_model.LogisticRegression()
logistic.fit(X_train,y_train)

# predict on test data and view confusion matrix
import model_evaluation_utils as meu

y_pred = logistic.predict(X_test)
meu.display_confusion_matrix(true_labels=y_test, predicted_labels=y_pred,
classes=[0, 1])
```

```
          Predicted:
                0    1
Actual: 0      59    4
        1       2  106
```



- The preceding output depicts the confusion matrix with necessary annotations.
- We can see that out of 63 observations with label 0 (benign), our model has correctly predicted 59 observations.
- Similarly, out of 108 observations with label 1 (malignant), our model has correctly predicted 106 observations

Evaluating Classification Models- Confusion Matrix




```
positive_class = 1
TP = 106
FP = 4
TN = 59
FN = 2
```

- Each column in the confusion matrix represents classified instance counts based on predictions from the model and each row of the matrix represents instance counts based on the actual/true class labels.
- This structure can also be reversed, i.e. predictions depicted by rows and true labels by columns.
- In a typical binary classification problem, we usually have a class label that's defined as the positive class,

Evaluating Classification Models- Confusion Matrix

- True Positive (TP): This is the count of the total number of instances from the positive class where the true class label was equal to the predicted class label, i.e., the total instances where we correctly predicted the positive class label with our model.
- False Positive (FP): This is the count of the total number of instances from the negative class where our model misclassified them by predicting them as positive. Hence, the name, "false" positive.
- True Negative (FN): This is the count of the total number of instances from the negative class, where the true class label was equal to the predicted class label, i.e., the total instances where we correctly predicted the negative class label with our model.
- False Negative (FN): This is the count of the total number of instances from the positive class where our model misclassified them by predicting them as negative. Hence the name, "false" negative.

Evaluating Classification Models- Performance Metrics

Accuracy

- Accuracy is one of the most popular measures of classifier performance.
- It is defined as the overall proportion of correct predictions of the model.
- The formula for computing accuracy from the confusion matrix is as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

```
fw_acc = round(meu.metrics.accuracy_score(y_true=y_test, y_pred=y_pred), 5)
mc_acc = round((TP + TN) / (TP + TN + FP + FN), 5)
print('Framework Accuracy:', fw_acc)
print('Manually Computed Accuracy:', mc_acc)
```

```
Framework Accuracy: 0.96491
Manually Computed Accuracy: 0.96491
```

Evaluating Classification Models- Performance Metrics

Precision

- Precision, also known as positive predictive value, is another metric that can be derived from the confusion matrix.
- It is defined as the number of predictions made that are actually correct or relevant out of all the predictions based on the positive class.

$$Precision = \frac{TP}{TP + FP}$$

- A model with high precision will identify a higher fraction of positive classes as compared to a model with a lower precision.

```
fw_prec = round(meu.metrics.precision_score(y_true=y_test, y_pred=y_pred), 5)
mc_prec = round((TP) / (TP + FP), 5)
print('Framework Precision:', fw_prec)
print('Manually Computed Precision:', mc_prec)
```

```
Framework Precision: 0.96364
Manually Computed Precision: 0.96364
```

Evaluating Classification Models- Performance Metrics

Recall

- Recall, also known as sensitivity, is a measure of a model to identify the percentage of relevant data points.
- It is defined as the number of instances of the positive class that were correctly predicted.
- This is also known as hit rate, coverage, or sensitivity.

$$Recall = \frac{TP}{TP + FN}$$

- Recall becomes an important measure of classifier performance when we want to catch the most number of instances of a particular class even when it increases our false positives. For example, consider the case of bank fraud.
- A model with high recall will give us higher number of potential fraud cases. But it will also help us raise alarm for most of the suspicious cases.

```
fw_rec = round(meu.metrics.recall_score(y_true=y_test, y_pred=y_pred), 5)
mc_rec = round((TP) / (TP + FN), 5)
print('Framework Recall:', fw_rec)
print('Manually Computed Recall:', mc_rec)

Framework Recall: 0.98148
Manually Computed Recall: 0.98148
```

Evaluating Classification Models- Performance Metrics

F1 Score

- There are some cases in which we want a balanced optimization of both precision and recall.
- The F1-score is the harmonic mean of precision and recall and helps us optimize a classifier for balanced precision and recall performance.

$$F1\,Score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

```
fw_f1 = round(meu.metrics.f1_score(y_true=y_test, y_pred=y_pred), 5)
mc_f1 = round((2*mc_prec*mc_rec) / (mc_prec+mc_rec), 5)
print('Framework F1-Score:', fw_f1)
print('Manually Computed F1-Score:', mc_f1)
```

```
Framework F1-Score: 0.97248
Manually Computed F1-Score: 0.97248
```

Build our text classifiers

- Traditional feature representation (BOW, TF-IDF) and classification models
- Advanced feature representation (Word2Vec, GloVe, FastText) and classification models

Thank You!!!