Semantics and Pragmatics

Unit-IV

Syed Rameem Zahra
(Assistant Professor)
Department of CSE, NSUT

Semantic Analysis

- Semantic Analysis is a subfield of NLP that attempts to understand the meaning of Natural Language. Semantic Analysis of Natural Language captures the meaning of the given text while taking into account context, logical structuring of sentences and grammar roles.
- Semantic Analysis of Natural Language can be classified into two broad parts:
  - **Lexical Semantic Analysis:** Lexical Semantic Analysis involves understanding the meaning of each word of the text individually. It basically refers to fetching the dictionary meaning that a word in the text is deputed to carry.
  - **Compositional Semantics Analysis:** Although knowing the meaning of each word of the text is essential, it is not sufficient to completely understand the meaning of the text.

Tasks involved in Semantic Analysis

- In order to understand the meaning of a sentence, the following are the major processes involved in Semantic Analysis:
  - **Word Sense Disambiguation:**
    - It involves interpreting the meaning of a word based upon the context of its occurrence in a text.
    - For example, the word ‘Bark’ may mean ‘the sound made by a dog’ or ‘the outermost layer of a tree.’
    - Likewise, the word ‘rock’ may mean ‘a stone‘ or ‘a genre of music‘ – hence, the accurate meaning of the word is highly dependent upon its context and usage in the text.
    - Thus, the ability of a machine to overcome the ambiguity involved in identifying the meaning of a word based on its usage and context is called Word Sense Disambiguation.
  - **Relationship Extraction:**
    - It involves firstly identifying various entities present in the sentence and then extracting the relationships between those entities.

Elements of Semantic Analysis

- **Hyponymy:** Hyponymys refers to a term that is an instance of a generic term. They can be understood by taking class-object as an analogy. For example: ‘Color‘ is a hypernymy while ‘grey‘, ‘blue‘, ‘red‘, etc, are its hyponyms.
- **Homonymy:** Homonymy refers to two or more lexical terms with the same spellings but completely distinct in meaning. For example: ‘Rose‘ might mean ‘the past form of rise‘ or ‘a flower‘, – same spelling but different meanings; hence, ‘rose‘ is a homonymy.
- **Synonymy:** When two or more lexical terms that might be spelt distinctly have the same or similar meaning, they are called Synonymy. For example: (Job, Occupation), (Large, Big), (Stop, Halt).
- **Antonymy**: Antonymy refers to a pair of lexical terms that have contrasting meanings – they are symmetric to a semantic axis. For example: (Day, Night), (Hot, Cold), (Large, Small).
- **Polysemy:** Polysemy refers to lexical terms that have the same spelling but multiple closely related meanings. It differs from homonymy because the meanings of the terms need not be closely related in the case of homonymy. For example: ‘man‘ may mean ‘the human species‘ or ‘a male human‘ or ‘an adult male human‘ – since all these different meanings bear a close association, the lexical term ‘man‘ is a polysemy.
- **Meronomy:** Meronomy refers to a relationship wherein one lexical term is a constituent of some larger entity. For example: ‘Wheel‘ is a meronym of ‘Automobile‘

Basic Units of Semantic System

- **Entity:** An entity refers to a particular unit or individual in specific such as a person or a location. For example Ramesh, Delhi, etc.
- **Concept:** A Concept may be understood as a generalization of entities. It refers to a broad class of individual units. For example Learning Portals, City, Students.
- **Relations:** Relations help establish relationships between various entities and concepts. For example: ‘Ramesh studies in NSUT’, ‘Delhi is a City.’, etc.
- **Predicate:** Predicates represent the verb structures of the sentences.

Approaches to Meaning Representations

- First-order predicate logic (FOPL)
- Semantic Nets
- Frames
- Conceptual dependency (CD)
- Rule-based architecture
- Case Grammar
- Conceptual Graphs

Syntax-Driven Semantic analysis

- The first part of semantic analysis, studying the meaning of individual words is called lexical semantics.
- It includes words, sub-words, affixes (sub-units), compound words and phrases also. All the words, sub-words, etc. are collectively called lexical items.
- In other words, we can say that lexical semantics is the relationship between lexical items, meaning of sentences and syntax of sentence.
- **Classification** of lexical items like words, sub-words, affixes, etc. is performed in lexical semantics.
- **Decomposition** of lexical items like words, sub-words, affixes, etc. is performed in lexical semantics.
- **Differences** as well as similarities between various lexical semantic structures is also analyzed.

Word Sense Disambiguation

- Word Sense Disambiguation (WSD) is the task of determining the correct sense of a word in context.
- It is an automatic task: we create a system that automatically disambiguates the senses for us.
- Computationally determining which sense of a word is activated by its use in a particular context.
- E.g. I am going to withdraw money from the **bank**.
- Word sense disambiguation, in natural language processing (NLP), may be defined as the ability to determine which meaning of word is activated by the use of word in a particular context.
- The problem of resolving semantic ambiguity is called WSD.
- Resolving semantic ambiguity is harder than resolving syntactic ambiguity.

Word Sense Disambiguation

- For example, consider the two examples of the distinct sense that exist for the word **“bass”** −
  - I can hear bass sound.
  - He likes to eat grilled bass.
- The occurrence of the word bass clearly denotes the distinct meaning. In first sentence, it means frequency and in second, it means fish.
- Hence, if it would be disambiguated by WSD then the correct meaning to the above sentences can be assigned as follows −
  - I can hear bass/frequency sound.
  - He likes to eat grilled bass/fish.

Evaluation of WSD

- The evaluation of WSD requires the following two inputs −
  - **A Dictionary**
    - The very first input for evaluation of WSD is dictionary, which is used to specify the senses to be disambiguated.
  - **Test Corpus**
    - Another input required by WSD is the high-annotated test corpus that has the target or correct-senses. The test corpora can be of two types :
      - **Lexical sample −** This kind of corpora is used in the system, where it is required to disambiguate a small sample of words.
      - **All-words −** This kind of corpora is used in the system, where it is expected to disambiguate all the words in a piece of running text.

Approaches of WSD

- **Knowledge Based Approaches**
  - WSD using Selectional Preferences (or restrictions)
  - Overlap Based Approaches
    - These approaches rely on knowledge resources like WordNet, Thesaurus etc.
    - May use grammar rules for disambiguation.
    - May use hand coded rules for disambiguation.
- **Machine Learning Based Approaches**
  - Supervised Approaches
  - Semi-supervised Algorithms
  - Unsupervised Algorithms
    - Rely on corpus evidence.
    - Train a model using tagged or untagged corpus.
    - Probabilistic/Statistical models.
- **Hybrid Approaches**
    - Use corpus evidence as well as semantic relations form WordNet

WSD using Selectional Preferences and Arguments

Sense 1

- This airlines **serves** dinner in the evening flight.
- serve (Verb)
  - Agent
  - object – edible

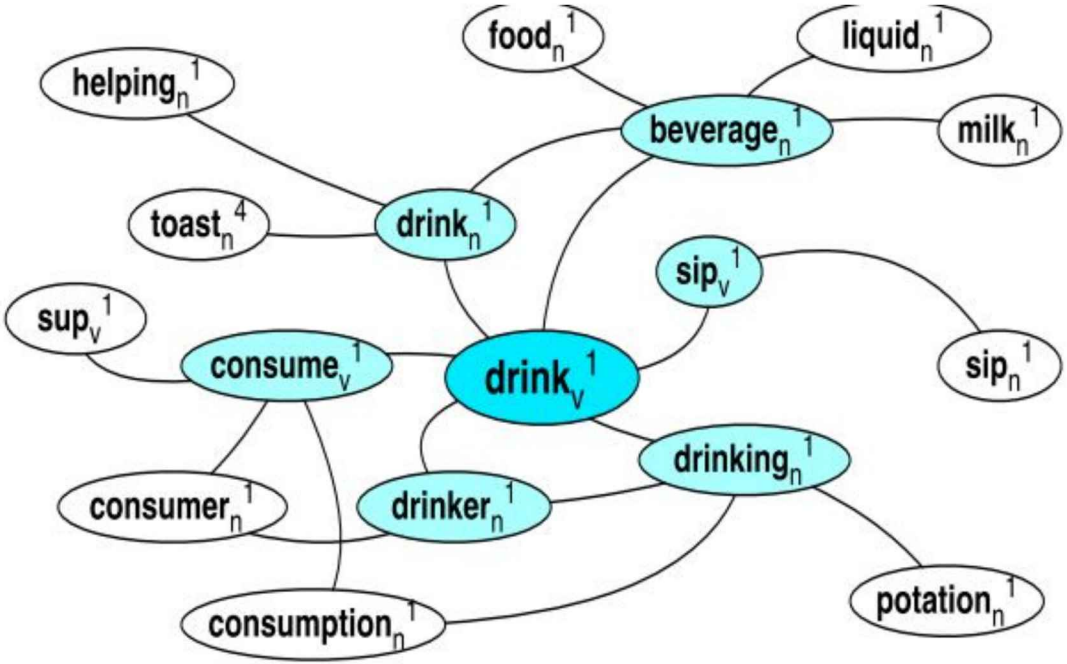
Sense 2

- This airlines **serves** the sector between Agra & Delhi.
- serve (Verb)
  - Agent
  - object – sector

- Requires exhaustive enumeration of:
  - Argument-structure of verbs.
  - Selectional preferences of arguments.
  - Description of properties of words such that meeting the selectional preference criteria can be decided.
- E.g. This flight serves the “**region**” between Mumbai and Delhi
- How do you decide if “**region**” is compatible with “sector”

Graph-based methods

- First, WordNet can be viewed as a graph
  - senses are nodes
  - relations (hypernymy, meronymy) are edges
  - Also add edge between word and unambiguous gloss words



Overlap Based Approaches

- Require a Machine Readable Dictionary (MRD).
- Find the overlap between the features of different senses of an ambiguous word (**sense bag**) and the features of the words in its context (**context bag**).
- These features could be sense definitions, example sentences, hypernyms etc.
- The features could also be given weights.
- The sense which has the maximum overlap is selected as the contextually appropriate sense.

Lesk's Algorithm

- The Lesk algorithm is based on the assumption that words in a given "neighborhood" (section of text) will tend to share a common topic.
- A simplified version of the Lesk algorithm is to compare the dictionary definition of an ambiguous word with the terms contained in its neighborhood.
  - for every sense of the word being disambiguated one should count the number of words that are in both the neighborhood of that word and in the dictionary definition of that sense
  - the sense that is to be chosen is the sense that has the largest number of this count.
- E.g. "On burning **coal** we get **ash**."

Ash

- Sense 1
  Trees of the olive family with pinnate leaves, thin furrowed bark and gray branches.
- Sense 2
  The ***solid*** residue left when ***combustible*** material is thoroughly ***burn***ed or oxidized.
- Sense 3
  To convert into ash

Coal

- Sense 1
  A piece of glowing carbon or ***burn***t wood.
- Sense 2
  charcoal.
- Sense 3
  A black ***solid combustible*** substance formed by the partial decomposition of vegetable matter without free access to air and under the influence of moisture and often increased pressure and temperature that is widely used as a fuel for ***burn***ing

In this case Sense 2 of ash would be the winner sense.

Walker's Algorithm: A Thesaurus Based approach

- **Step 1:** For each sense of the target word find the thesaurus category to which that sense belongs.
- **Step 2:** Calculate the score for each sense by using the context words. A context words will add 1 to the score of the sense if the thesaurus category of the word matches that of the sense.
  - E.g. The money in this **bank** fetches an interest of 8% per annum
  - Target word: **bank**
  - Clue words from the context: **money, interest, annum, fetch**

	Sense1: Finance	Sense2: Location
Money	+1	0
Interest	+1	0
Fetch	0	0
Annum	+1	0
Total	3	0

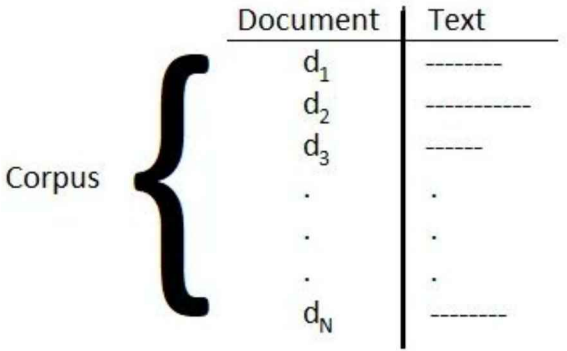
Context words add 1 to the sense when the topic of the word matches that of the sense

Word Similarity: Thesaurus Methods

- Synonymy: a binary relation
  - Two words are either synonymous or not
- Similarity (or distance)
  - Two words are more similar if they share more features of meaning
- Similarity is properly a relation between senses
  - We do not say “The word “**bank**” is not similar to the word “**slope**” “, but we say:
    - Bank^1 is similar to fund^3
    - Bank^2 is similar to slope^5
- But we’ll compute similarity over both words and senses

Converting Text Data to Numeric Data

- Computers only understand numbers
- Once we convert our text into a vector, we can leverage the beauty of Linear Algebra.
- 5 ways of achieving this task:
  - Bag Of Words (BOW)
  - Term Frequency -Inverse Document Frequency (Tf-IDF)
  - Word2Vec (W2V)
  - Average W2V
  - Average TF-IDF
- Note:
  - ***Document-*** *It is nothing but a file that has text data in it. In terms of a dataset , each record or data point can be considered as a document.*
  - ***Corpus-*** *Set of documents is known as corpus. In terms of a dataset , entire data points or whole of the dataset can be considered as a corpus.*



Bag Of Words (BOW)

- In this approach , corresponding to each document , we'll create a vector of dimension d = unique words in the corpus.
- Each cell of a vector would correspond to a unique word and the value in that cell would be the frequency/ number of times that word has appeared in that document.

- E.g.:
- d_1 := "I am fine"
- d_2 := "I am hungry . I am sick"
- d_3 := "Food is fine"
- U = Unique words in the corpus are := {I , am , fine , hungry , sick, food , is}
- number of unique words = n(U) = d = 7

d_1 :=

I	am	fine	hungry	sick	food	is
1	1	1	0	0	0	0

d_2 :=

I	am	fine	hungry	sick	food	is
2	2	0	1	1	0	0

d_3 :=

I	am	fine	hungry	sick	food	is
0	0	1	0	0	1	1

Bag Of Words (BOW)

- Advantages:
  - Easy to code and understand.
  - Can be used to implement a baseline model.
  - Can be used when corpus is small.
- Limitations:
  - Each vector would be a sparse vector.
  - It doesn’t take semantic meaning of the words into consideration.

Term Frequency - Inverse Document Frequency (TF-IDF)

- **Term Frequency** := is defined as the ratio of number of times a word appears in a document to total number of words in that document

$$TF(w_i, d_j) = \frac{\#\text{ of times } w_i \text{ appears in document } d_j}{\text{Total \# of words in document } d_j}$$

- **Inverse Document Frequency** := is defined as log of ratio of number of documents in the corpus to number of documents in which the word has occurred.

$$IDF(w_i, D_c) = \log_e \left(\frac{\text{Number of documents in corpus}}{\text{number of documents in which word } w_i \text{ has occurred} + 1} \right)$$

Data corpus

Term Frequency - Inverse Document Frequency (TF-IDF)

- Higher the IDF value , rarer the word in other documents.
- E.g.:
- d_1 := “I am fine”
- d_2 := “I am hungry . I am sick”
- d_3 := “Food is fine”
- U = Unique words in the corpus are := {I , am , fine , hungry , sick, food , is}
- number of unique words = n(U) = d = 7
- **Step 1 :** Start with empty d dimension vectors
- **Step 2 :** Calculate term frequency of each word in each of the documents
- For e.g
  - in the first document ,
    - TF(“I”,d_1) = TF(“am”,d_1)= TF(“fine”,d_1) =⅓
  - and in second document
    - TF(“I”,d_2) =TF(“am”,d_2) = 2/6
    - TF(“hungry”,d_2) = TF(“sick”,d_2) = ⅙
  - and so on…

d_1 :=

I	am	fine	hungry	sick	food	is
1/3	1/3	1/3	0	0	0	0

d_2 :=

I	am	fine	hungry	sick	food	is
2/6	2/6	0	1/6	1/6	0	0

d_3 :=

I	am	fine	hungry	sick	food	is
0	0	1/3	0	0	1/3	1/3

Term Frequency - Inverse Document Frequency (TF-IDF)

- **Step 3:** Calculate IDF values of each of the unique word in the corpus
- IDF("I",D_c) = IDF("am",D_c) =IDF("fine",D_c) = $\log_e\left(\frac{3}{2+1}\right) = 0$
- IDF("hungry",D_c) =IDF("sick",D_c) = IDF("food",D_c) = IDF("is",D_c)= $\log_e\left(\frac{3}{2}\right) = 0.405$
- **Step 4 :** Multiply each value in the vector (containing TF values) with the corresponding IDF values of the word

d_1 :=

I	am	fine	hungry	sick	food	is
0	0	0	0	0	0	0

d_2 :=

I	am	fine	hungry	sick	food	is
0	0	0	.067	.067	0	0

d_3 :=

I	am	fine	hungry	sick	food	is
0	0	0	0	0	.135	.135

CATEGORY	APPROACH	⊕ PROS	⊖ CONS	ALGORITHM	AREAS
Edit based Similarities	compare two strings by counting the minimum number of operations required to transform one string into the other	+ Simplicity + Good for short strings	– Inefficient for longer strings – Computationally expensive – Doesn't take into account the semantic meaning	**Hamming** **Levenshtein** **Damerau-Levenshtein** **Jaro-Winkler** **Strcmp95** **Needleman-Wunsch** **Gotoh** **Smith-Waterman** **MLIPNS**	**Spelling correction** **Duplication search** **DNA analysis** **Correction systems for OCR** **Measuring the linguistic distance of the languages**
Token based similarities	compare two strings by looking at units (tokens) of the strings (for example, words, n-grams etc)	+ Computational efficience + Applicable for long texts + Can take into account the semantic meaning	– May be not very good for single words or short phrases from several words – Magnitude of the vectors doesn't play any role in comparison for some algorithms	**Jaccard index** **Sørensen-Dice coefficient** **Tversky index** **Overlap coefficient** **Tanimoto distance** **Cosine similarity** **Bag distance**	**Computer science** **Text mining** **Many general NLP problems** **Bioinformatics** **Informatiion retieval**
Sequence based	compare two strings by looking at different subsequences of the strings	+ Simplicity + Good for short strings	– Inefficient for longer strings – Computationally expensive – Doesn't take into account the semantic meaning	**Longest common subsequence similarity** **Longest common substring similarity** **Ratcliff-Obershelp similarity**	**Computer science** **Bioinformatics** **Version control systems**
Phonetic	compare two strings by comparing how do they sound in pronunciation	+ Allows to analyze another side of the language: pronunciation + Good for short strings	– Inefficient for longer strings – Doesn't take into account the semantic meaning – Needs special preprocessing of the strings – Applicable only for very specific tasks	**MRA** **Soundex** **Metaphone** **NYSIIS** **Editex**	**Names retrieval** **Database software**
Simple	compare two string by looking at some simple measurements of each string	+ The simpliest algorithms + Good for short strings + Computational efficience	– Inefficient for longer strings – Doesn't take into account the semantic meaning – Very primitive – Applicable only for very specific tasks	**Prefix similarity** **Postfix similarity** **Length distance** **Identity similarity**	**Computer science**
Hybrid	combines edit based approach with the token based approach	+ Can take into account the semantic meaning of the word in a text + Can be used for texts of any length + Performs better in texts with misspellings + Flexibility: you can pick any edit based similarity measure	– Can be computationally inefficient for long texts (depends on the edit similarity) – It is not symmetrical	**Monge-Elkan**	**General NLP problems**

Similarity Distances for Natural Language Processing

- **Longest Common Substring (LCS):**
- Given two strings, s1 of length n1 and s2 of length n2, it simply considers the length of the longest string (or strings) which is substring of both s1 and s2.
- Applications: data deduplication and plagiarism detection.

LCS('Lydia', 'Lydoko') = 3

		L	Y	D	I	A
	0	0	0	0	0	0
L	0	1	0	0	0	0
Y	0	0	2	0	0	0
D	0	0	0	3	0	0
O	0	0	0	0	0	0
K	0	0	0	0	0	0
O	0	0	0	0	0	0

Similarity Distances for Natural Language Processing

- **Levenshtein Edit Distance (V. Levenshtein, 1965):**
- It quantifies how dissimilar two text units are to one another by computing the minimum number of single-character edits (replacement, deletion and insertion operations) required to convert text unit 1 into text unit 2.
- For example, the Levenshtein distance between "kitten" and "sitting" is 3, since the following 3 edits change one into the other, and there is no way to do it with fewer than 3 edits:
  - kitten → sitten (substitution of "s" for "k"),
  - sitten → sittin (substitution of "i" for "e"),
  - sittin → sitting (insertion of "g" at the end).

Similarity Distances for Natural Language Processing

- **Hamming Distance (R. Hamming, 1950):**
- Hamming distance between two equal size strings measures the minimum number of replacements required to change one string into the other.
- Mathematically, this can be written as follows:

$$\text{Hamming distance:} \quad D_{hamming}(x, y) = \sum_{i=1}^{n} 1_{x_i \neq y_i}$$

Similarity Distances for Natural Language Processing

- **Cosine Similarity:**
- The cosine similarity of two text units simply computes the cosine of the angle formed by the two vectors representing the text units, i.e. the inner product of Euclidian space of the normalized vectors.
- When close to 1, the two units are close in the chosen vector space, when close to -1, the two units are far apart.

$$similarity(w_1, w_2) = cos(w_1, w_2) = \frac{w1.w2}{\|w1\|.\|w2\|}$$

Similarity Distances for Natural Language Processing

- **Euclidean distance** is a token-based similarity distance.
- Given two points in Rn, the Euclidean distance metric is simply the length of a line segment between these two points.
- It is also often referred as the l2-norm, the l2-distance or the Pythagorean metric and can be expressed as:

$$\text{Euclidean distance:} \quad D_{euclidean}(x, y) = \|x - y\|_2 = \sqrt[2]{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Machine Learning based approach

- If we have data that has been hand-labelled with correct word senses, we can used a supervised learning approach and learn from it!
  - We need to extract features and train a classifier
  - The output of training is an automatic system capable of assigning sense labels to unlabelled words in a context.
- Two variants of WSD task:
  - **Lexical Sample task** (we need labelled corpora for individual senses)
    - Small pre--selected set of target words (ex difficulty)
    - And inventory of senses for each word
    - Supervised machine learning: train a classifier for each word
  - **All-words task** (each word in each sentence is labelled with a sense)
    - Every word in an entire text
    - A lexicon with senses for each word
- So in ML based approach, we need:
  - the tag set (“sense inventory”)
  - the training corpus
  - A set of features extracted from the training corpus
  - A classifier

Supervised Approaches

- **Naïve Bayes**
  - Suffers from data sparseness.
  - Since the scores are a product of probabilities, some weak features might pull down the overall score for a sense.
  - A large number of parameters need to be trained.
- **Decision Lists**
  - A word-specific classifier. A separate classifier needs to be trained for each word.
  - Uses the single most predictive feature which eliminates the drawback of Naïve Bayes.
- **Exemplar Based K-NN**
  - A word-specific classifier.
  - Will not work for unknown words which do not appear in the corpus.
  - Uses a diverse set of features (including morphological and noun-subject-verb pairs)
- **SVM**
  - A word-sense specific classifier.
  - Gives the highest improvement over the baseline accuracy.
  - Uses a diverse set of features.
- **HMM**
  - Significant in lieu of the fact that a fine distinction between the various senses of a word is not needed in tasks like MT.
  - A broad coverage classifier as the same knowledge sources can be used for all words belonging to super sense.
  - Even though the polysemy was reduced significantly there was not a comparable significant improvement in the performance