Department of Computer Science and Engineering (Data Science)

Experiment No 6

Name:Akshay Jha **Sap:60009220103** **Batch:D1-1**

Aim: Implement Lesk's algorithm for word sense disambiguation

Theory: -

In Natural Language Processing (NLP), word sense disambiguation (WSD) is the challenge of determining which "sense" (meaning) of a word is activated by its use in a specific context, a process that appears to be mostly unconscious in individuals.

Approaches and Methods to Word Sense Disambiguation (WSD)

Approaches and methods to WSD are classified according to the source of knowledge used in word disambiguation.

Dictionary-based or Knowledge-based Methods

As the name suggests, for disambiguation, these methods primarily rely on dictionaries, treasures and lexical knowledge base. They do not use corpora evidence for disambiguation. The Lesk method is the seminal dictionary-based method introduced by Michael Lesk in 1986. The Lesk definition, on which the Lesk algorithm is based is "measure overlap between sense definitions for all words in context". However, in 2000, Kilgarriff and Rosensweig gave the simplified Lesk definition as "measure overlap between sense definitions of word and current context", which further means identify the correct sense for one word at a time. Here the current context is the set of words in surrounding sentence or paragraph.

Supervised Methods

For disambiguation, machine learning methods make use of sense-annotated corpora to train. These methods assume that the context can provide enough evidence on its own to disambiguate the sense. In these methods, the words knowledge and reasoning are deemed unnecessary. The context is represented as a set of "features" of the words. It includes the information about the surrounding words also. Support vector machine and memory-based learning are the most successful supervised learning approaches to WSD. These methods rely on substantial amount of manually sense-tagged corpora, which is very expensive to create.

Semi-supervised Methods

Due to the lack of training corpus, most of the word sense disambiguation algorithms use semi-supervised learning methods. It is because semi-supervised methods use both labelled as well as unlabeled data. These methods require very small amount of annotated text and large amount of plain unannotated text. The technique that is used by semi supervised methods is bootstrapping from seed data.

Unsupervised Methods

These methods assume that similar senses occur in similar context. That is why the senses can be induced from text by clustering word occurrences by using some measure of similarity of the context. This task is called word sense induction or discrimination. Unsupervised methods have great potential to overcome the knowledge acquisition bottleneck due to non-dependency on manual efforts.

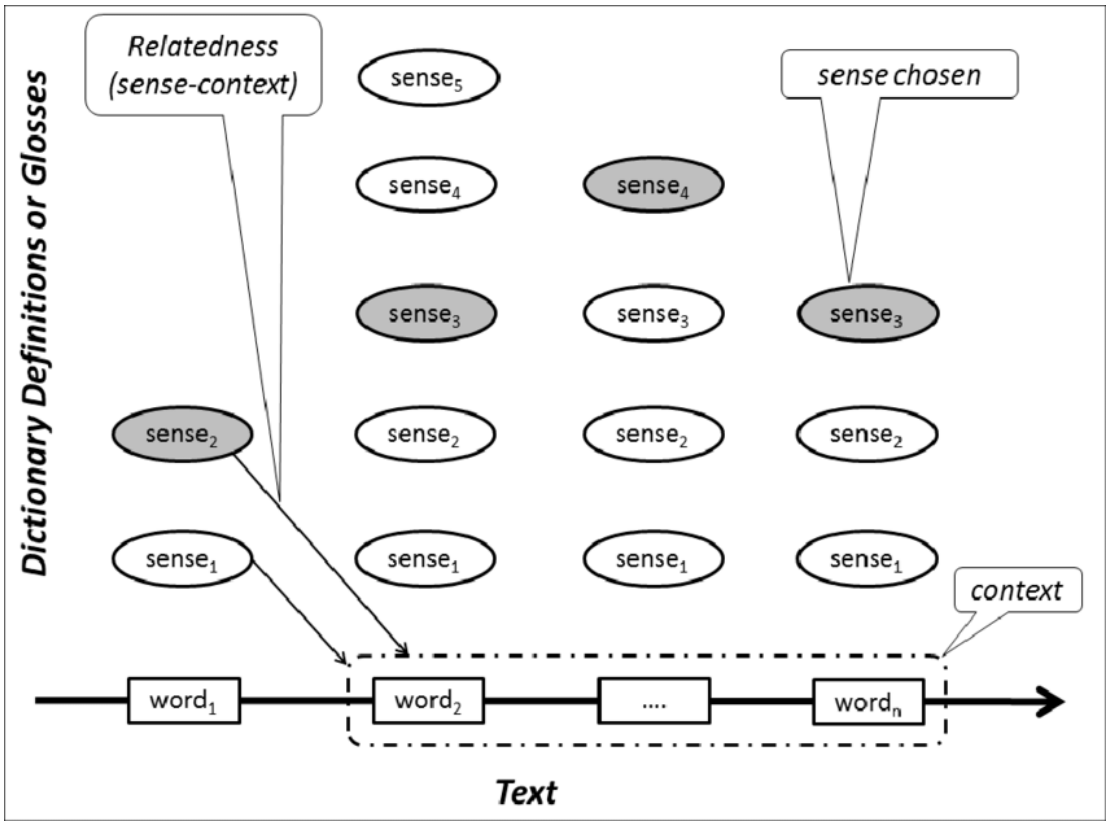
Lesk Algorithm:-Dictionary based approach

Lesk Algorithm is a way of Word Sense Disambiguation. The Lesk algorithm is a dictionary-based approach that is considered seminal. It is founded on the idea that words used in a text are related to one another, and that this relationship can be seen in the definitions of the words and their meanings. The pair of dictionary senses having the highest word overlap in their dictionary meanings are used to disambiguate two (or more) terms. Michael E. Lesk introduced the Lesk algorithm in 1986 as a classic approach for word sense disambiguation in Natural Language Processing. The Lesk algorithm assumes that words in a given "neighborhood" (a portion of text) will have a similar theme. The dictionary definition of an uncertain word is compared to the terms in its neighborhood in a simplified version of the Lesk algorithm.

Basic Lesk Algorithm implementation involves the following steps:

- Count the number of words in the neighborhood of the word and in the dictionary definition of that sense for each sense of the word being disambiguated.
- The sense to be picked is the one with the greatest number of items in this count.

Basically, the context is chosen from meaning of the nearest words. Following is the simplified pictorial representation of the same...



Advantages and Disadvantages

There are numerous advantages to Lesk's algorithm, the primary being that its simplicity makes it easy to implement, applicable in a variety of different contexts, and thus easily generalizable. Lesk notes that the algorithm does not depend on global information, meaning that since the same word could be referenced many times throughout a text but change each time, the meaning of a word is only derived from the collection of immediate supporting words in its context window, rather than from the entire text itself. Despite its simplicity and power, the biggest drawback to Lesk's original algorithm is its performance — its accuracy was proposed by Lesk to be only around 50–70%, and has been shown to be much lower when experimentally validated against sense-tagged texts (Viveros-Jiménez, 2013). The algorithm also notably suffers from low recall, in that it cannot provide a corresponding contextual definition for many words simply because there is either no overlap to be found between dictionary definitions at all, or that several definitions have the same number of overlaps. Furthermore, Lesk leaves several questions unanswered.

Lab Experiment to be performed in this session: -

Step 1: Import Library's

Step2: Perform Tokenization

Step3: Take the ambiguous word and find the semantic of a word within various context.

Name:Akshay Jha

Sap:60009220103

Batch:D1-1

```
# Step 1: Import Libraries
import nltk
from nltk.wsd import lesk
from nltk.tokenize import word_tokenize
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
from nltk import pos_tag
from collections import Counter

# Download necessary datasets
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('omw-1.4')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

```
True
```

```
# Initialize lemmatizer
lemmatizer = WordNetLemmatizer()

# Function to get WordNet POS tag
def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
```

```
    else:
        return wordnet.NOUN  # Default to noun

# Step 2: Perform Tokenization, POS Tagging, and Lemmatization with
Context Expansion
def preprocess_sentence(sentence, window=3):
    tokens = word_tokenize(sentence)
    pos_tags = pos_tag(tokens)
    lemmatized_tokens = [lemmatizer.lemmatize(word.lower(),
get_wordnet_pos(tag)) for word, tag in pos_tags]
    expanded_context = []
    for i, word in enumerate(lemmatized_tokens):
        left_context = lemmatized_tokens[max(0, i - window):i]
        right_context = lemmatized_tokens[i + 1:i + 1 + window]
        expanded_context.extend(left_context + [word] + right_context)
    return expanded_context

# Step 3: Apply Improved Lesk Algorithm with Context Prioritization
def improved_lesk(sentence, ambiguous_word):
    tokens = preprocess_sentence(sentence)
    best_sense = None
    sense_frequencies = Counter()
    financial_keywords = {"money", "loan", "deposit", "fund",
"approve", "finance", "transaction", "account", "banking", "credit",
"debit", "cash", "withdraw"}
    savings_keywords = {"piggy", "coins", "box", "save", "home",
"container"}
    is_financial_context = any(word in tokens for word in
financial_keywords)
    is_savings_context = any(word in tokens for word in
savings_keywords)
    for sense in wordnet.synsets(ambiguous_word, pos=wordnet.NOUN):  #
Focus only on noun senses
        gloss_words = set(word_tokenize(sense.definition().lower()))
        example_words = set()
        for example in sense.examples():
            example_words.update(word_tokenize(example.lower()))
        # Combine gloss and example words
        signature = gloss_words.union(example_words)
        overlap = len(signature.intersection(tokens))
        # Prioritize financial-related senses in financial contexts
        if is_financial_context:
            if "bank.n.02" in sense.name():
                overlap += 30  # Stronger boost for financial
institution meaning
            elif "savings_bank.n.02" in sense.name():
                overlap = -100  # Forcefully exclude 'money box'
meaning in financial contexts
            elif "depository_financial_institution.n.01" in
sense.name():
```

```
                overlap += 50  # Strongest boost for financial
institution
        # Prioritize savings-related senses in savings contexts
        if is_savings_context:
            if "savings_bank.n.02" in sense.name():
                overlap += 50  # Strong boost for piggy bank meaning
            elif "depository_financial_institution.n.01" in
sense.name():
                overlap -= 20  # Slightly demote financial institution
in savings context
        sense_frequencies[sense] = overlap
    # Select the sense with highest overlap, breaking ties by depth
    if sense_frequencies:
        best_sense = max(sense_frequencies, key=lambda sense:
(sense_frequencies[sense], sense.min_depth()))
    return best_sense

# Example Usage
sentences = [
    "He sat on the bank of the river.",
    "The bank approved the loan quickly.",
    "As a child, I used to save my coins in a piggy bank."
]

ambiguous_word = "bank"

for sentence in sentences:
    sense = improved_lesk(sentence, ambiguous_word)
    if sense:
        print(f"Sentence: {sentence}")
        print(f"Best Sense: {sense.name()}")
        print(f"Definition: {sense.definition()}\n")
    else:
        print(f"No sense found for: {sentence}\n")
```

```
Sentence: He sat on the bank of the river.
Best Sense: bank.n.01
Definition: sloping land (especially the slope beside a body of water)

Sentence: The bank approved the loan quickly.
Best Sense: depository_financial_institution.n.01
Definition: a financial institution that accepts deposits and channels
the money into lending activities

Sentence: As a child, I used to save my coins in a piggy bank.
Best Sense: savings_bank.n.02
Definition: a container (usually with a slot in the top) for keeping
money at home
```