NLP Assignment-2

1) Explain in detail the techniques of vector space model (VSM) and list the pro’s and con’s

Ans| **Techniques of Vector Space Model (VSM)**

The Vector Space Model (VSM) is a simple algebraic model used for representing text units as vectors of numbers, essential for various machine learning (ML) algorithms dealing with text data. This mathematical model converts text data into a form that ML algorithms can process. Here's a detailed look at the techniques involved:

1.**Text Unit Representation:**

- Text units (characters, phonemes, words, phrases, sentences, paragraphs, documents) are represented as vectors.

- In the simplest form, vectors consist of identifiers like index numbers in a corpus vocabulary.

2. **Similarity Calculation:**

- Cosine Similarity: Measures similarity between two text blobs using the cosine of the angle between their corresponding vectors. Cosine similarity ranges from 1 (identical) to -1 (completely dissimilar), calculated as:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

- Euclidean Distance: Another method used to capture similarity by measuring the distance between vectors in Euclidean space.

Basic Vectorization Approaches

1. One-Hot Encoding:

- Maps each word in the vocabulary to a unique integer ID.

- Each word is represented by a binary vector where only one index (corresponding to the word's ID) is 1, and all others are 0.

- Example: For a corpus vocabulary [dog, bites, man, eats, meat, food], the word "dog" could be represented as [1, 0, 0, 0, 0, 0].

Example:

```
def get_onehot_vector(somestring):
    onehot_encoded = []
    for word in somestring.split():
        temp = [0] * len(vocab)
        if word in vocab:
            temp[vocab[word] - 1] = 1
        onehot_encoded.append(temp)
    return onehot_encoded
get_onehot_vector(processed_docs[1])
# Output: [[0, 0, 1, 0, 0, 0], [0, 1, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0]]
```

Pros:

- Intuitive and Simple: Easy to understand and straightforward to implement.

Cons:

- Sparsity: Results in high-dimensional vectors where most entries are zeros, leading to inefficiency in storage, computation, and learning, often causing overfitting.

- Variable Length: Generates vectors of varying lengths based on the number of words, while most ML algorithms require fixed-length feature vectors.

- No Semantic Similarity: Treats words as atomic units, failing to capture the semantic similarity between words (e.g., "run" and "ran" are as distant as "run" and "apple").

- Out of Vocabulary (OOV) Problem: Cannot handle new words not present in the training corpus without retraining the model and expanding the vocabulary.

2) Explain in detail the techniques of bag of words (BOW) and list the pro's and con's

Ans| **Techniques of Bag of Words (BoW)**

The Bag of Words (BoW) is a classical text representation technique used commonly in Natural Language Processing (NLP), particularly for text classification problems. Here's a detailed explanation of the technique:

1.Text Representation:

- BoW represents text as a collection of words, ignoring the order and context.

- The underlying intuition is that texts belonging to the same class share a unique set of words. Thus, analyzing the words in a text can help identify its class.

2. Word Mapping:

- Similar to one-hot encoding, BoW maps words to unique integer IDs between 1 and |V| (size of the vocabulary).

3. Vector Conversion:

- Each document is converted into a |V|-dimensional vector.

- The i-th component of the vector (i = wid) is the number of times the word w occurs in the document.

4.Example:

- For the toy corpus with vocabulary [dog, bites, man, eats, meat, food]:

  - Document D1 ("dog bites man") is represented as [1, 1, 1, 0, 0, 0] because the words "dog," "bites," and "man" each appear once in D1.

  - Document D4 ("man eats food") is represented as [0, 0, 1, 0, 1, 1].

Example:

```
from sklearn.feature_extraction.text import CountVectorizer

count_vect = CountVectorizer()

# Build a BOW representation for the corpus

bow_rep = count_vect.fit_transform(processed_docs)

# Look at the vocabulary mapping
```

```
print("Our vocabulary: ", count_vect.vocabulary_)

# See the BOW rep for the first 2 documents

print("BoW representation for 'dog bites man': ", bow_rep[0].toarray())

print("BoW representation for 'man bites dog': ", bow_rep[1].toarray())

# Get the representation using this vocabulary, for a new text

temp = count_vect.transform(["dog and dog are friends"])

print("BoW representation for 'dog and dog are friends':", temp.toarray())
```

-Binary Representation:

- If frequency isn't needed, initialize `CountVectorizer` with `binary=True` to represent word existence:

```
  count_vect = CountVectorizer(binary=True)

  bow_rep_bin = count_vect.fit_transform(processed_docs)

  temp = count_vect.transform(["dog and dog are friends"])

  print("BoW representation for 'dog and dog are friends':", temp.toarray())
```

Pros:

- Simple and Easy: Straightforward to understand and implement.

- Semantic Similarity: Documents with similar words have vector representations closer to each other in Euclidean space, capturing the semantic similarity. For example, the distance between D1 and D2 is 0, while the distance between D1 and D4 is 2.

- Fixed-Length Encoding: Provides fixed-length encoding for any sentence of arbitrary length.

Cons:

- Sparsity: The vector size increases with vocabulary size, leading to sparsity. This can be managed by limiting the vocabulary to the most frequent words.

- No Semantic Similarity Between Words: Doesn't capture the similarity between different words with the same meaning. For example, "I run," "I ran," and "I ate" will be equally apart in BoW vectors.

- Out of Vocabulary (OOV) Problem: Cannot handle new words not seen in the training corpus.

- Loss of Word Order: The order of words is lost, so texts like D1 ("dog bites man") and D2 ("man bites dog") have the same representation.

3) Explain in detail the techniques of bag of N-gram(BON) and list the pro's and con's

Ans| **Techniques of Bag of N-Grams (BoN)**

The Bag of N-Grams (BoN) approach is an enhancement over the Bag of Words (BoW) technique. It addresses the lack of context and word ordering in previous text representation methods by considering contiguous sequences of words, known as n-grams. Here's a detailed explanation of the technique:

1. N-Gram Formation:

- BoN breaks the text into chunks of n contiguous words (or tokens).

- These chunks are called n-grams, and the approach can capture some context that single-word models (like BoW) cannot.

2. Vocabulary Construction:

- The corpus vocabulary \(V\) is formed by collecting all unique n-grams across the text corpus.

- For example, in a bigram (2-gram) model, the vocabulary might include: {dog bites, bites man, man bites, bites dog, dog eats, eats meat, man eats, eats food}.

3. Vector Representation:

- Each document in the corpus is represented by a vector of length (|V|).

- This vector contains frequency counts of n-grams present in the document, with zeros for n-grams that are not present.

4. Example:

- Using the example corpus with a bigram model, the vectors for documents D1 and D2 would be:

  - D1: [1, 1, 0, 0, 0, 0, 0, 0]

  - D2: [0, 0, 1, 1, 0, 0, 0, 0]

- Each document is represented by an eight-dimensional vector corresponding to the bigrams in the corpus.

5. Code Implementation:

- The following code demonstrates how to create a BoN representation considering 1–3 n-gram word features:

```
from sklearn.feature_extraction.text import CountVectorizer

# n-gram vectorization example with count vectorizer and uni, bi, trigrams

count_vect = CountVectorizer(ngram_range=(1,3))

# Build a BOW representation for the corpus

bow_rep = count_vect.fit_transform(processed_docs)

# Look at the vocabulary mapping

print("Our vocabulary: ", count_vect.vocabulary_)

# Get the representation using this vocabulary, for a new text

temp = count_vect.transform(["dog and dog are friends"])

print("BoW representation for 'dog and dog are friends':", temp.toarray())
```

Pros:

- Captures Context and Word Order: BoN captures some context and word-order information by considering n-grams.

- Semantic Similarity: The resulting vector space captures some semantic similarity, as documents with the same n-grams have vectors closer to each other in Euclidean space.

Cons:

- Increased Dimensionality and Sparsity: As n increases, the dimensionality and sparsity of the vector space increase rapidly.

- No Solution for OOV Problem: BoN still does not provide a way to address the out-of-vocabulary (OOV) problem.

4) Explain in detail the techniques of TF-IDE and list the pro's and con's

Ans| **Techniques of TF-IDF**

TF-IDF (Term Frequency–Inverse Document Frequency) is a text representation technique used to quantify the importance of a word in a document relative to the entire corpus. Here’s a detailed explanation of the technique:

1. **Term Frequency (TF):**

- TF measures how often a term \(t \) appears in a document \(d \).

- It is normalized by dividing the number of occurrences of \(t \) by the total number of terms in the document \(d \).

- Formula: $\mathrm{TF}_{t,d} = \frac{\text{Number of occurrences of term } t \text{ in document } d}{\text{Total number of terms in the document } d}$.

2. **Inverse Document Frequency (IDF):**

- IDF measures the importance of a term across the corpus.

- It down-weights terms that are very common across all documents and up-weights terms that are rare.

- Formula: $\mathrm{IDF}_t = \log_e\left(\frac{\text{Total number of documents in the corpus}}{\text{Number of documents with term } t \text{ in them}}\right)$.

3. **TF-IDF Score:**

- The TF-IDF score is the product of TF and IDF.

4. Code Implementation:

- The following code demonstrates how to use TF-IDF to represent text:

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer()

bow_rep_tfidf = tfidf.fit_transform(processed_docs)

print(tfidf.idf_) # IDF for all words in the vocabulary

print(tfidf.get_feature_names()) # All words in the vocabulary.

temp = tfidf.transform(["dog and man are friends"])

print("Tfidf representation for 'dog and man are friends':\n", temp.toarray())
```

Pros:

- Quantifies Importance: TF-IDF quantifies the importance of words in a document relative to the entire corpus.

- Captures Relevance: It helps in extracting relevant documents from a corpus for a given text query, commonly used in information retrieval systems.

- Improves Weighting: By down-weighting common terms and up-weighting rare ones, it improves the representation over simple frequency counts.

Cons:

- High Dimensionality: TF-IDF vectors are sparse and high-dimensional, which can hamper learning capability and computational efficiency.

- Discrete Representation: It treats words as atomic units, which limits its ability to capture relationships between words.

- OOV Problem: Like other vectorization methods, TF-IDF cannot handle out-of-vocabulary (OOV) words.

5) Explain word embedding and briefly describe the two variables of word embedding

Ans| **Word Embeddings**

Word embeddings are a method of representing words in a continuous vector space where semantically similar words are close together. This approach is based on the distributional hypothesis, which posits that words appearing in similar contexts have similar meanings. Word2vec, developed by Mikolov et al. in 2013, is a notable model for creating such embeddings.

- **Distributional Similarity:** Words that frequently appear in similar contexts should have similar vector representations.

- Low-dimensional, Dense Vectors: Word embeddings typically have dimensions ranging from 50 to 500 and are dense, meaning most values are non-zero.

Word2vec Architectures

1. Continuous Bag of Words (CBOW)

CBOW predicts a center word based on its context words. For example, in the sentence "The quick brown fox jumps over the lazy dog," with a context window size of 2, the context words for "jumps" would be "brown," "fox," "over," and "the." CBOW uses these context words to predict "jumps."

CBOW Model:

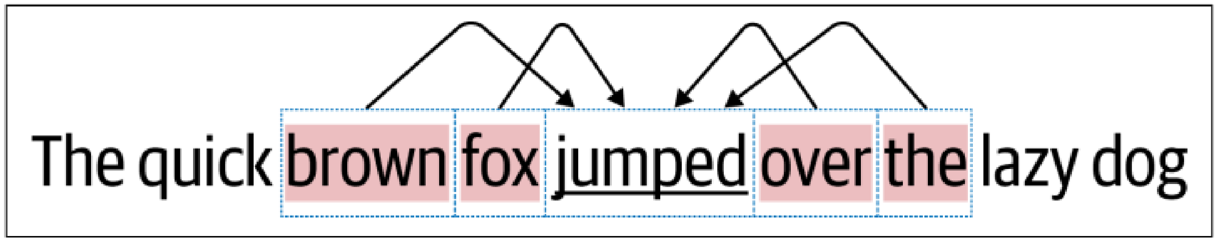


Figure 3-7. CBOW: given the context words, predict the center word

2. SkipGram

SkipGram does the opposite of CBOW. It predicts context words from a given center word. Using the same sentence and context window size, the model predicts "brown," "fox," "over," and "the" given the center word "jumps."

SkipGram Model:

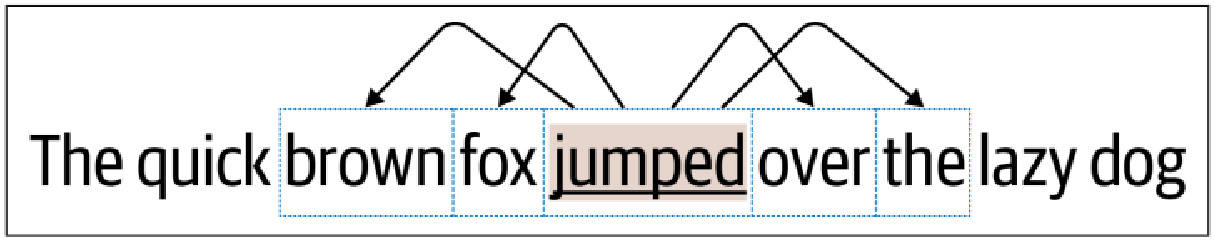


Figure 3-10. SkipGram: given the center word, predict every word in context

Important Variables in Word Embeddings

1. Dimensionality of the Word Vectors

- Definition: The size of the vector space where words are embedded.

- Common Ranges: 50–500 dimensions.

- Impact: Higher dimensions capture more nuances but require more computation.

2. Context Window Size

- Definition: The number of surrounding words considered for predicting the center word (CBOW) or for predicting context words from the center word (SkipGram).

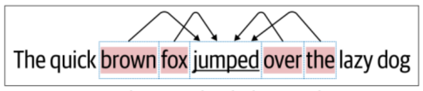
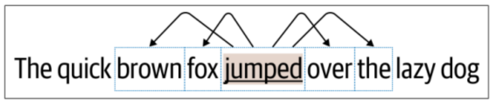
- Impact: Larger window sizes capture broader contexts but may introduce noise; smaller sizes capture more precise, local contexts.

Pre-Trained Word Embeddings

Training word embeddings from scratch is computationally expensive. Pre-trained embeddings, such as Word2vec, GloVe, and fastText, trained on large corpora, are often used to save time and resources while providing robust representations for many applications.

6) What is the key difference between CBOW and Skipgram models in context of word embedding?

Ans|

Aspect	CBOW (Continuous Bag of Words)	SkipGram
Task	Predict the center word from the context words.	Predict the context words from a given center word.
Example	Given context: "brown," "fox," "over," "the" Predicts: "jumps"	Given center word: "jumps" Predicts context: "brown," "fox," "over," "the"
Architecture	Uses context words to predict the center word.	Uses center word to predict context words.
Training Data	Sliding window over text corpus.	Sliding window over text corpus.
Network Structure	Shallow network with single hidden layer.	Shallow network with single hidden layer.
Objective	Learn embeddings by predicting center word from context.	Learn embeddings by predicting context words from center word.
Diagram	 Figure 3-7. CBOW: given the context words, predict the center word	 Figure 3-10. SkipGram: given the center word, predict every word in context

7) What is information extraction (IE) and mention its wide range of real world applications

Ans| **Information Extraction (IE)**

Information extraction (IE) is a natural language processing (NLP) task that involves extracting relevant information from text documents. This information can include key events, people, relationships between entities, and more. The goal of IE is to transform unstructured text into structured data.

Real-World Applications of IE

1. Tagging News and Other Content:

- Extracts important entities from news articles, such as people, organizations, locations, and events. For example, Google News tags articles with relevant entities to help users find specific information.

2. Chatbots:

- Helps chatbots understand user queries by extracting specific information, such as locations and entities, to generate accurate responses. For example, identifying "Eiffel Tower" and "cafe" in the question, “What are the best cafes around the Eiffel Tower?”

3. Social Media Applications:

- Extracts informative excerpts from social media text for decision-making purposes. An example is using Twitter data for traffic updates and disaster relief efforts.

4. Extracting Data from Forms and Receipts:

- Used in banking apps to scan checks and deposit money or in apps that scan bills and receipts. IE, along with optical character recognition (OCR), extracts relevant information from these documents.

8) Write a short note on IE test

Ans| **Information Extraction (IE) Tasks**

Information Extraction (IE) encompasses a range of tasks aimed at extracting knowledge from text. These tasks vary in complexity and provide different types of information:

1. Keyword or Keyphrase Extraction (KPE):

- Identifies important terms or phrases within a text that signify its main topics or themes. For example, recognizing that an article is about “buyback” or “stock price.”

2. Named Entity Recognition (NER):

- Identifies and classifies entities in text into predefined categories such as person names, organizations, locations, etc. For example, recognizing "Apple" as an organization and "Luca Maestri" as a person.

3. Named Entity Disambiguation and Linking:

- Determines the correct identity of an entity among several possible candidates and links it to a unique identifier. For example, recognizing that "Apple" refers to the company Apple, Inc. and not a fruit.

4. Relation Extraction:

- Identifies and classifies relationships between entities in text. For example, extracting the information that Luca Maestri is the finance chief of Apple.

5. Event Extraction:

- Identifies events mentioned in the text and links them to other occurrences of the same event. For example, recognizing and linking articles discussing the same stock buyback event by Apple.

6. Temporal Information Extraction:

- Extracts information about times and dates mentioned in the text, useful for developing applications like calendars and personal assistants.

7. Template Filling:

- Extracts data to fill predefined templates, commonly used in applications like generating weather reports or flight announcements.

9) With a clear diagram illustrate the general pipeline of information extraction and explain each task depicted in the diagram.

Ans| General Pipeline for Information Extraction (IE)

The general pipeline for Information Extraction (IE) involves several steps of Natural Language Processing (NLP), each contributing to the overall extraction process. The following diagram and explanations illustrate the typical NLP pipeline for IE tasks:

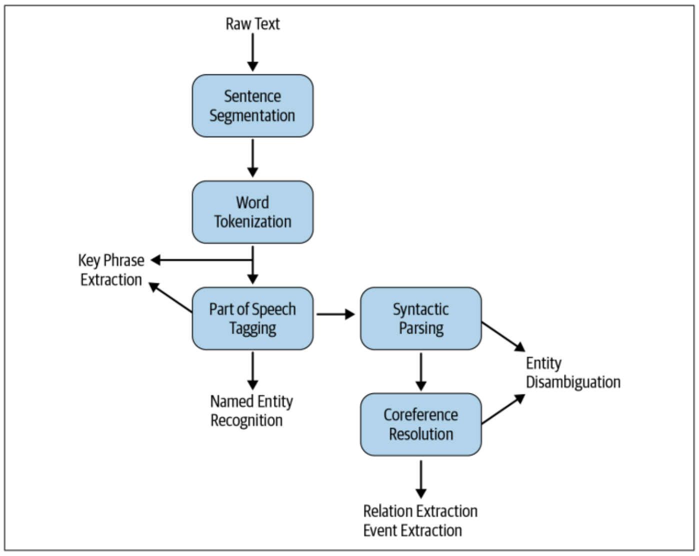


Figure 5-3. IE pipeline illustrating NLP processing needed for some IE tasks

1. Raw Text:

- The input text from which information needs to be extracted.

2. Sentence Segmentation:

- Splits the text into individual sentences. This helps in processing the text at a finer granularity.

3. Word Tokenization:

- Breaks down each sentence into individual tokens (words, punctuation, etc.). This is essential for subsequent NLP tasks.

4. Part of Speech (POS) Tagging:

- Assigns part-of-speech tags (e.g., noun, verb, adjective) to each token. This step is crucial for understanding the grammatical structure of the text.

- Key Phrase Extraction: This can be performed after POS tagging to identify important words and phrases.

5. Syntactic Parsing:

- Analyzes the syntactic structure of the text. This includes tasks like dependency parsing, which identifies relationships between tokens.

6. Coreference Resolution:

- Identifies when different expressions refer to the same entity (e.g., "Albert Einstein," "Einstein," "he"). This step is crucial for entity disambiguation.

- Relation Extraction: Identifies relationships between recognized entities.

- Event Extraction: Identifies and links occurrences of specific events mentioned in the text.

7. Named Entity Recognition (NER):

- Identifies and categorizes entities like names of people, organizations, locations, dates, etc. Requires deeper NLP processing including POS tagging and parsing.

Explanation of Each Task

1. Key Phrase Extraction (KPE):

- Extracts significant phrases from text that summarize its main topics. Minimal NLP processing, such as POS tagging, may be required.

2. Named Entity Recognition (NER):

- Identifies and categorizes entities such as persons, organizations, locations, dates, and other specialized entities in the text.

3. Relation Extraction:

- Finds and classifies relationships between identified entities in the text. Builds on top of NER and coreference resolution.

4. Event Extraction:

- Identifies events mentioned in the text and links them to related articles or mentions over time. May involve complex NLP processing including temporal information extraction and coreference resolution.

5. Entity Disambiguation:

- Clarifies which specific entity is being referred to in cases where multiple entities share the same name.

10) What is KPE? With a proper code snippet illustrate implementation of KPE

Ans| **Keyphrase Extraction (KPE)**

Keyphrase Extraction (KPE) is an Information Extraction (IE) task focused on identifying significant words and phrases from a text document that capture its main topics. It is useful for various downstream NLP tasks such as search/information retrieval, automatic document tagging, recommendation systems, and text summarization.

KPE can be performed using supervised learning, which requires labeled datasets, or unsupervised learning, which does not. Unsupervised approaches are more popular due to their domain agnosticism and lower data requirements. These methods often use graph-based algorithms, where words and phrases are represented as nodes in a weighted graph, and the importance of each node is determined based on its connections.

Implementation of KPE

The Python library `textacy`, built on top of the well-known library `spaCy`, provides implementations for common graph-based keyword and phrase extraction algorithms. Here is a code snippet illustrating KPE using `textacy` with the algorithms TextRank and SGRank:

```
from textacy import *

import textacy.ke
```

```
# Read the text from a file

mytext = open("nlphistory.txt").read()

# Load the spaCy language model

en = textacy.load_spacy_lang("en_core_web_sm", disable=("parser",))

# Create a spaCy document from the text

doc = textacy.make_spacy_doc(mytext, lang=en)

# Extract keyphrases using TextRank

textrank_output = [kps for kps, weights in textacy.ke.textrank(doc, normalize="lemma",
topn=5)]

print("Textrank output: ", textrank_output)

# Extract keyphrases using SGRank

sgrank_output = [kps for kps, weights in textacy.ke.sgrank(doc, n_keyterms=5)]

print("SGRank output: ", sgrank_output)
```

11) What are NER and how a NER system is built?

Ans| **Named Entity Recognition (NER)**

Named Entity Recognition (NER) is an Information Extraction (IE) task focused on identifying entities within a text. Entities typically include names of persons, locations, organizations, and specialized strings such as monetary expressions, dates, products, and more. NER is crucial for various NLP applications like search engines, machine translation, and more, as it helps in identifying and categorizing these entities for further processing.

How a NER System is Built

There are multiple approaches to building an NER system:

1. Gazetteer-Based Approach:

- Maintain a collection of relevant names (e.g., person names, locations).

- Perform a lookup to identify named entities.

- This approach is simple but has limitations in handling new names and variations.

2. Rule-Based NER:

- Use predefined patterns based on word tokens and POS tags.
- Example: A pattern like "NNP was born" can indicate that "NNP" (proper noun) is a person.
- Libraries like Stanford NLP's RegexNER and spaCy's EntityRuler facilitate rule-based NER.

3. Machine Learning-Based NER:

- Train a model to predict named entities in unseen text.
- NER is treated as a sequence labeling problem, where the context is important.

```
def sent2feats(sentence):
    feats = []
    sen_tags = pos_tag(sentence)
    for i in range(len(sentence)):
        word = sentence[i]
        wordfeats = {'tag': sen_tags[i][1]}
        wordfeats["prevTag"] = sen_tags[i-1][1] if i > 0 else "<S>"
        wordfeats["nextTag"] = sen_tags[i+1][1] if i < len(sentence) - 1 else "</S>"
        feats.append(wordfeats)
    return feats


def train_seq(X_train, Y_train, X_dev, Y_dev):
    crf = CRF(algorithm='lbfgs', c1=0.1, c2=10, max_iterations=50)
    crf.fit(X_train, Y_train)
    y_pred = crf.predict(X_dev)
    print(metrics.flat_f1_score(Y_dev, y_pred, average='weighted'))
```

Using Pre-Trained NER Models

- Libraries: Stanford NER, spaCy, AllenNLP

```
import spacy

nlp = spacy.load("en_core_web_lg")

text = "On Tuesday, Apple announced its plans for another major chunk of the money: It will buy back a further $75 billion in stock."

doc = nlp(text)

for ent in doc.ents:

    print(ent.text, "\t", ent.label_)
```

Enhancing NER Systems

1. Active Learning:

- Start with a pre-trained model and improve it using tools like Prodigy.
- Manually tag sentences and correct predictions to retrain the model.

2. Combination Approaches:

- Use a mix of machine learning models, gazetteers, and pattern matching heuristics for better performance.

12) Write a short note on named entity disambiguate and linking.

Ans| **Named Entity Disambiguation and Linking**

Named Entity Disambiguation (NED)is the NLP task of assigning a unique identity to entities mentioned in the text. This is crucial for resolving ambiguities when the same entity name refers to different real-world entities. For example, in the sentence "Lincoln drives a Lincoln Aviator and lives on Lincoln Way," the three mentions of "Lincoln" refer to a person, a vehicle, and a location, respectively. NED ensures each mention is correctly linked to its corresponding entity.

Named Entity Linking (NEL) combines Named Entity Recognition (NER) and NED to connect entities mentioned in text to their real-world counterparts, such as Wikipedia pages. NEL is essential for various applications like question answering and constructing knowledge bases, such as the Google Knowledge Graph.

Building an NEL system involves several steps:

1. Contextual Identification: Just like NER, NEL relies on contextual information to identify entities and their spans.

2. Parsing: This includes identifying linguistic elements like subjects, verbs, and objects.

3. Coreference Resolution: Linking multiple references to the same entity (e.g., Albert Einstein, Einstein) to a single entry in a knowledge base.

NEL is typically modeled as a supervised machine learning problem and evaluated using precision, recall, and F1 scores. State-of-the-art NEL systems often use advanced neural network architectures. Due to the complexity and need for large annotated datasets and encyclopedic resources, it is more common to use off-the-shelf services from providers like IBM Watson and Microsoft Azure rather than developing in-house systems.

13) What is Relation Extraction(RE)? Explain the RE approach in brief.

Ans| Relation Extraction (RE)

Relation Extraction (RE) is an Information Extraction (IE) task that involves identifying entities and the relationships between them from text documents. This task is crucial for building a knowledge base that connects various entities like people, organizations, and events based on textual content. Such knowledge bases can be used for applications like financial analysis, improving search systems, and developing question-answering systems.

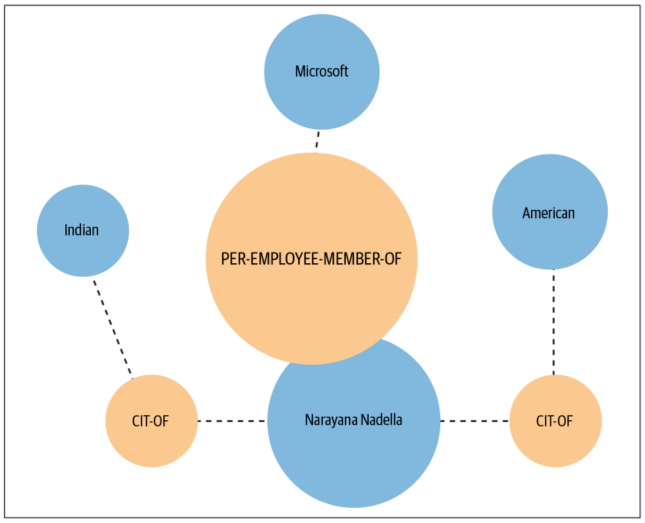


Figure 5-10. Relation extraction demo

Approaches to RE

Hand-Built Patterns: Early methods used handwritten patterns and regular expressions to capture specific relationships. These patterns can achieve high precision but often lack coverage and are challenging to create for all possible relations within a domain.

Supervised Learning: RE is often treated as a supervised classification problem. The process is typically divided into two steps:

1. Binary Classification: Determine whether two entities in a text are related.

2. Multiclass Classification: If they are related, identify the specific type of relationship.

This approach uses features such as:

- Handcrafted features

- Contextual features (e.g., words around the entities)

- Syntactic structures (e.g., noun phrase-verb phrase-noun phrase patterns)

Neural models may utilize various embedding representations followed by architectures like recurrent neural networks.

Domain Specificity and Bootstrapping: Supervised and pattern-based approaches are generally domain-specific and require large amounts of annotated data, which can be challenging to obtain. Bootstrapping can be used to generalize from a small set of seed patterns by learning new patterns based on sentences extracted using these seeds.

Distant Supervision: This method uses large databases like Wikipedia and Freebase to collect examples of many relations, creating a large dataset of relations. These datasets are then used in a supervised RE approach.

Unsupervised RE (Open IE): This approach aims to extract relations from the web without relying on training data or predefined lists of relations. It identifies relations in the form of <verb, argument1, argument2> tuples. However, the challenge lies in mapping these tuples to standardized relations in a database.

14) In context of IE distinguish between KPE and NER.

Ans| **Key Phrase Extraction (KPE) vs. Named Entity Recognition (NER)**

Key Phrase Extraction (KPE) and Named Entity Recognition (NER) are both Information Extraction (IE) tasks, but they serve different purposes and operate in distinct ways.

Key Phrase Extraction (KPE):

- **Purpose:** KPE aims to identify and extract important phrases or keywords from a text. These key phrases are typically informative and represent the main topics or concepts discussed in the text.

- **Functionality:** KPE focuses on extracting significant terms or phrases without necessarily categorizing them into predefined entity types. It is more about summarizing the content by identifying phrases that convey essential information.

- **Example:** In a news article about a scientific discovery, KPE might extract phrases like "quantum computing breakthrough" or "new algorithm."

Named Entity Recognition (NER):

- **Purpose:** NER aims to identify and classify named entities mentioned in the text into predefined categories such as persons, organizations, locations, dates, etc.

- **Functionality:** NER involves recognizing specific types of entities and tagging them accordingly. It goes beyond merely identifying significant phrases by categorizing them into specific, well-defined entity types.

- **Example:** In the same news article, NER would recognize and classify "IBM" as an organization, "John Doe" as a person, and "New York" as a location.

15) Consider the training corpus with 4 sentences or documents VSM, BOW,TF-IDF modeling technique to find the word to vector text representation

S1/D1 – The house jack built

S2/D2 – The malt

S3/D3 – Jack lay house built

S4/D4 – ate malt

16) Consider the given training corpus apply bigram modeling technique to find the probability of each bigram formed

This is the house that jack built.

This is the malt

That lay in the house that jack built

That ate the malt

That lay in the house that jack built