DSAI SET 4

EURON

1.What is text preprocessing in NLP, and why is it important?

Text preprocessing is a crucial step in Natural Language Processing (NLP) that involves cleaning and transforming raw text data into a format suitable for analysis and modeling. It ensures that the text is structured and meaningful for machine learning algorithms. Key steps include:

1. **Lowercasing** – Converts text to lowercase to maintain uniformity.
2. **Removing special characters and punctuations** – Cleans unnecessary symbols.
3. **Tokenization** – Splits text into meaningful units (words, subwords, or sentences).
4. **Stopword Removal** – Eliminates common but uninformative words (e.g., "the", "is").
5. **Stemming and Lemmatization** – Reduces words to their root forms to normalize variations.
6. **Normalization** – Converts numbers, contractions, and other variations to a standard format.
7. **Handling missing data** – Addresses gaps in text data.

Preprocessing improves model efficiency, reduces noise, and enhances accuracy by standardizing input data.

---

2. What is the difference between stemming and lemmatization?

Both stemming and lemmatization reduce words to their base forms, but they differ in their approaches:

Aspect	Stemming	Lemmatization
Definition	A rule-based process of removing suffixes to obtain the root form of a word.	A linguistically-informed approach that reduces words to their base form using vocabulary and morphological analysis.
Output	Produces root words that may not be meaningful (e.g., "running" → "run", "troubled" → "troubl").	Returns valid dictionary words (e.g., "running" → "run", "troubled" → "trouble").
Algorithm	Uses heuristic rules (e.g., Porter Stemmer, Snowball Stemmer).	Uses dictionaries, POS tagging, and word sense disambiguation (e.g., WordNet Lemmatizer).
Accuracy	Less accurate, faster.	More accurate, computationally expensive.

Lemmatization is preferred for NLP tasks where semantic meaning is essential, while stemming is useful for simple text mining.

---

3. Explain the Bag-of-Words (BoW) model and its limitations.

Bag-of-Words (BoW) is a fundamental technique for text representation in NLP. It converts text into a numerical feature vector based on word occurrence.

Steps:

1. **Tokenization:** Split text into words.
2. **Vocabulary Building:** Create a set of unique words from the corpus.
3. **Vectorization:** Convert each document into a vector representing word counts.

Example:

Corpus:

- "I love NLP."
- "NLP is amazing."

Vocabulary = {I, love, NLP, is, amazing}

Sentence	I	love	NLP	is	amazing
"I love NLP."	1	1	1	0	0
"NLP is amazing."	0	0	1	1	1

Limitations:

1. **Ignores word order** – "NLP is good" and "Good is NLP" have the same vector.
2. **Sparsity** – Large vocabularies lead to high-dimensional sparse matrices.
3. **No semantic understanding** – Cannot capture meaning beyond word frequency.
4. **Unable to handle OOV (Out-of-Vocabulary) words** – New words are not represented.

Alternative models like TF-IDF, Word2Vec, and BERT address these limitations.

---

4. How does TF-IDF work, and how is it different from BoW?

TF-IDF (Term Frequency-Inverse Document Frequency) is an improvement over BoW that assigns importance to words based on their frequency and relevance.

Formula:

$$TF = \frac{\text{Number of times a word appears in a document}}{\text{Total words in the document}}$$

$$IDF = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents containing the word}}\right)$$

$$TF - IDF = TF \times IDF$$

Differences from BoW:

Feature	BoW	TF-IDF
Representation	Frequency count of words.	Weighted score based on term frequency and document rarity.
Handling Common Words	Treats all words equally.	Reduces importance of frequent words.
Semantic Awareness	Lacks context understanding.	Partially addresses this by discounting common words.
Sparsity	High-dimensional sparse matrix.	Still sparse but reduces noise.

TF-IDF is effective in search engines and document classification.

---

5. What are word embeddings, and why are they useful in NLP?

Word embeddings are dense vector representations of words, capturing semantic meaning based on context. Unlike BoW and TF-IDF, they preserve relationships between words.

Types:

1. **Word2Vec** – Uses Skip-gram and CBOW (Continuous Bag-of-Words).
2. **GloVe (Global Vectors for Word Representation)** – Captures co-occurrence statistics.
3. **FastText** – Improves upon Word2Vec by handling subwords.

Benefits:

- Captures word similarity (e.g., "king" and "queen" are close in vector space).
- Supports transfer learning.
- Reduces feature space dimensionality.

Example: "King" - "Man" + "Woman" ≈ "Queen" (Vector arithmetic in embeddings).

---

6. How does Word2Vec work? Explain CBOW and Skip-gram.

Word2Vec is a neural network-based approach for learning word embeddings.

Two Models:

1. **CBOW (Continuous Bag-of-Words)**
    - Predicts a target word from its surrounding context words.
    - Example: Given ["the", "cat", "on", "the"], predict "mat".
    - Faster, better for small datasets.
2. **Skip-gram**
    - Given a word, predicts surrounding words.
    - Example: Given "cat", predict ["the", "on", "mat"].
    - Works well on large datasets, handles rare words better.

Both models use Negative Sampling or Hierarchical Softmax for efficient training.

---

7. What is the difference between Word2Vec, GloVe, and FastText?

Feature	Word2Vec	GloVe	FastText
Approach	Uses neural networks (CBOW/Skip-gram).	Uses matrix factorization of word co-occurrence.	Extends Word2Vec with subword information.
Training	Based on local context.	Captures global corpus statistics.	Uses character n-grams.
OOV Handling	Poor	Poor	Good (learns from subwords).
Performance	Good for contextual tasks.	Good for capturing overall distribution.	Excellent for morphologically rich languages.

FastText is preferred for languages with complex word structures.

---

8. What are contextual embeddings, and how do they differ from static embeddings?

Contextual embeddings (e.g., BERT, ELMo) generate dynamic word representations based on surrounding words, unlike static embeddings (Word2Vec, GloVe).

Example:

- Static: "bank" has the same vector in "river bank" and "bank loan".
- Contextual: "bank" has different vectors in each sentence.

BERT (Bidirectional Encoder Representations from Transformers) is a powerful contextual model.

---

9. How does BERT capture context differently from previous models?

BERT uses a **Transformer-based** bidirectional approach, unlike traditional left-to-right or right-to-left models.

- Uses **Masked Language Modeling (MLM)** for bidirectional learning.
- Employs **Next Sentence Prediction (NSP)** for sentence-level tasks.
- Handles polysemy and contextual meaning.

This makes BERT superior for tasks like Q&A, sentiment analysis, and text classification.

---

10. What is sequence modeling, and where is it used?

Sequence modeling captures dependencies across sequences (e.g., text, speech).

Examples:

- RNNs, LSTMs, GRUs for time-series predictions.
- Transformer-based models for NLP tasks.

Applications include chatbots, speech recognition, and machine translation.