Question 1

Step	Notes
Original Text	Raw input text, unprocessed, with punctuation and capitalization.
Tokenization	Text split into words and punctuation using NLTK's `word_tokenize`.
Stop-Word Removal	Stop words ("The," "the," "over") removed using NLTK's English stop-word list.
Stemming	Tokens stemmed using Porter Stemmer (e.g., "jumps" → "jump," "lazy" → "lazi").
Lemmatization	Tokens lemmatized using WordNet (e.g., "jumps" → "jump"; "lazy" remains unchanged).

Question2

Vectorization is essential for transforming textual data into numerical representations suitable for machine learning models. This process enables algorithms to process and analyze text for tasks such as classification, clustering, and semantic similarity. The comparison of BoW, TF-IDF, and word embeddings is critical, as each method offers different trade-offs in terms of feature representation, computational efficiency, and semantic fidelity.

Detailed Features of Each Technique

BAG OF WORDS (BOW)

BoW represents text as a sparse vector based on word frequencies or binary indicators, disregarding word order and grammatical structure. The process involves:

- **Vocabulary Construction**: Tokenizing the corpus to form a vocabulary of unique words (or n-grams).
- **Vector Representation**: Each document is mapped to a vector where each dimension corresponds to a word in the vocabulary, with values indicating frequency (count-based) or presence (binary).
- **Characteristics**: The resulting vectors are high-dimensional and sparse, as most documents contain only a subset of the vocabulary. BoW does not capture contextual or semantic relationships, treating words as independent units.

Example: For documents "The cat runs" and "The dog runs", with vocabulary {The, cat, dog, runs}, the BoW vectors (count-based) are:

- Doc 1: [1, 1, 0, 1]
- Doc 2: [1, 0, 1, 1]

Advantages:

- Simple and computationally efficient, suitable for tasks like text classification where word presence suffices (e.g., spam detection).
- Easy to implement using tools like Scikit-learn's CountVectorizer.

Limitations:

- Ignores word order and semantics, leading to identical representations for sentences with different meanings (e.g., "dog bites man" vs. "man bites dog").
- Large, sparse matrices can be memory-intensive, especially with large vocabularies.
- Sensitive to noise from frequent words, potentially skewing results.

TERM FREQUENCY-INVERSE DOCUMENT FREQUENCY (TF-IDF)

TF-IDF builds on BoW by weighting words based on their frequency in a document and their rarity across the corpus, aiming to highlight informative terms.

- **Term Frequency (TF)**: Measures how often a word appears in a document, often normalized by document length to mitigate bias toward longer texts.
- **Inverse Document Frequency (IDF)**: Downweights words appearing in many documents, calculated as:IDF

$$IDF(t) = \log(N / df(t))$$

where N is the total number of documents, and df(t) is the number of documents containing term t.

- **TF-IDF Score**: The product of TF and IDF for each term, resulting in a vector where each dimension reflects the term's importance.
- **Characteristics**: Like BoW, the vector size equals the vocabulary size, with sparse representation. Preprocessing includes tokenization, stop-word removal, and potentially stemming/lemmatization.

Example: Using the same documents, with assumed IDF values (log base 10):

- Vocabulary: {The, cat, dog, runs}
- IDF: The = 0 (appears in both), cat = $\log(2/1) \approx 0.301$, dog = 0.301, runs = 0
- TF-IDF vectors (assuming raw counts):
  - Doc 1: [0, 0.301, 0, 0]
  - Doc 2: [0, 0, 0.301, 0]

Advantages:

- Prioritizes informative, rare words, reducing the impact of common words (e.g., "the").
- Improves performance in tasks like document retrieval and clustering, as seen in applications like information retrieval systems.

Limitations:

- Still ignores word order and semantic relationships, inheriting BoW's limitations.
- High-dimensional and sparse, with sensitivity to corpus size and term distribution, potentially affecting scalability.

WORD EMBEDDINGS

Word embeddings represent words as dense, low-dimensional vectors in a continuous vector space, capturing semantic and syntactic relationships. Key aspects include:

- **Dense Vectors**: Each word maps to a fixed-length vector (e.g., 50–300 dimensions for static embeddings like Word2Vec, GloVe; 768 for BERT).
- **Semantic Relationships**: Learned from large corpora, embeddings place similar-meaning words close in the vector space (e.g., "king" and "queen" are proximate). This is achieved via co-occurrence patterns (Word2Vec, GloVe) or contextual modeling (BERT).
- **Document Representation**: Documents can be represented by aggregating word vectors (e.g., averaging) or using contextual models like BERT, which generate document-level embeddings (e.g., via the [CLS] token).
- **Types**:
  - **Static Embeddings**: Word2Vec (CBOW, Skip-gram) and GloVe assign fixed vectors, capturing global co-occurrence but missing context (e.g., "bank" has one vector for all senses).
  - **Contextual Embeddings**: BERT and similar models generate context-dependent vectors, handling polysemy (e.g., "bank" in "river bank" vs. "bank account").

Example: For documents "The cat runs" and "The dog runs", using Word2Vec (3D for simplicity):

- Word vectors: The = [0.1, 0.2, 0.1], cat = [0.5, 0.3, 0.4], dog = [0.6, 0.4, 0.3], runs = [0.2, 0.5, 0.6]
- Document vectors (average): Doc 1 ≈ [0.267, 0.333, 0.367], Doc 2 ≈ [0.300, 0.367, 0.333]

Advantages:

- Captures semantic and syntactic relationships, enabling operations like "king - man + woman ≈ queen".
- Dense, low-dimensional vectors reduce memory usage compared to BoW/TF-IDF.
- Contextual embeddings (e.g., BERT) handle word ambiguity, enhancing performance in tasks like sentiment analysis and question answering.

Limitations:

- Requires large training data and computational resources, particularly for contextual embeddings.
- Static embeddings lose context; aggregation methods (e.g., averaging) may lose document-level information.
- Implementation complexity, especially for contextual models, may pose challenges in deployment.

MATRIX SHAPES (SAMPLES × FEATURES)

The matrix shape for each technique depends on the number of samples (documents) and the feature space dimensionality:

- **BoW**:
- N×V, where N

  N is the number of documents, and V is the vocabulary size.
- **Explanation**: Each row represents a document, with columns corresponding to words in the vocabulary. The matrix is sparse, as most documents contain only a subset of words.
- **Example**: For 100 documents and a vocabulary of 10,000 words, the matrix is 100×10,000.

- **TF-IDF**:
  - **Shape**: N×V, identical to BoW.
  - **Explanation**: Uses the same vocabulary-based feature space, with values reflecting TF-IDF scores instead of counts. The matrix remains sparse.
  - **Example**: Same as BoW, e.g., 100×10,000 for 100 documents and 10,000 words.
- **Word Embeddings**:
  - **Shape**: N×D, where D is the embedding dimension (e.g., 300 for Word2Vec, 768 for BERT).
  - **Explanation**: Documents are represented by aggregating word vectors (e.g., averaging), resulting in dense, lower-dimensional matrices. For contextual embeddings, document-level vectors (e.g., BERT's [CLS] token) are used.
  - **Example**: For 100 documents and 300-dimensional embeddings, the matrix is 100×300.

Comparison: BoW and TF-IDF matrices are typically much larger in the feature dimension ($V \gg D$) and sparse, while word embedding matrices are smaller and dense, improving computational efficiency and semantic representation.

Which Captures Semantics Better and Why

To evaluate which method captures semantics better, we consider their ability to represent meaning, context, and relationships between words or documents.

BOW:
Semantic Capture: Poor.
Why: BoW treats words as independent units, ignoring their order, context, or meaning. Words with similar meanings (e.g., "big" and "large") are treated as unrelated, and word order is lost (e.g., "dog chases cat" vs. "cat chases dog"). It captures only presence or frequency, suitable for tasks where semantics are less critical (e.g., keyword-based classification).

TF-IDF:
Semantic Capture: Slightly better than BoW but still limited.
Why: By weighting terms based on their rarity, TF-IDF emphasizes informative words, which may indirectly capture some semantic relevance (e.g., rare technical terms in a domain). However, it still ignores word order and semantic relationships, inheriting BoW's limitations.

Word Embeddings:
Semantic Capture: Superior, especially for contextual embeddings.
Why:
Static Embeddings (e.g., Word2Vec, GloVe): These capture semantic relationships by placing similar words (e.g., "king" and "queen") close in the vector space, learned from co-occurrence patterns in large corpora. Operations like "king - man + woman ≈ queen" demonstrate semantic understanding. However, they assign a single vector per word, missing context-specific meanings (e.g., "bank" in different senses).
Contextual Embeddings (e.g., BERT): These generate word vectors based on surrounding context, capturing polysemy and syntactic nuances. For example, "bank" in "river bank" vs. "bank account" has different vectors. Document-level embeddings (e.g., via BERT's [CLS] token or averaging) preserve contextual relationships, making them highly effective for semantic tasks like sentiment analysis or question answering.
Dense Representation: The low-dimensional, dense vectors encode rich semantic information, unlike the sparse, high-dimensional BoW/TF-IDF matrices.

Conclusion: *Word embeddings, particularly contextual embeddings like BERT, capture semantics best. They model word and document meanings in a continuous vector space, preserving relationships and context. BoW and TF-IDF are limited to frequency-based representations, missing semantic nuances. Static embeddings are a middle ground, capturing some semantics but lacking context sensitivity.*

Question 3

CONCLUDE WHICH MODEL PERFORMED BETTER AND WHY.

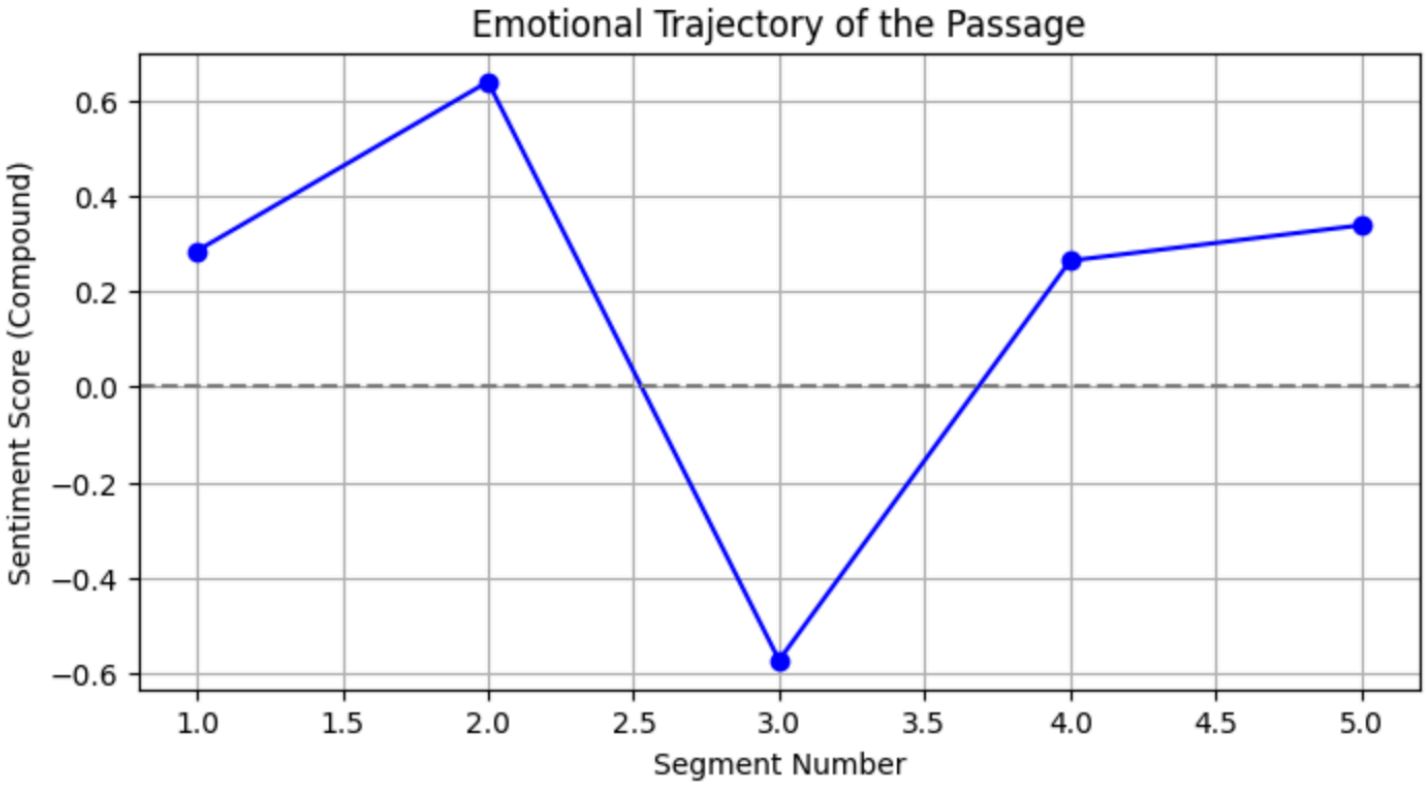
```
📊 Naive Bayes Evaluation
Accuracy: 0.5
F1 Score: 0.4
Confusion Matrix:
 [[4 1]
 [5 2]]

📊 Logistic Regression Evaluation
Accuracy: 0.58
F1 Score: 0.44
Confusion Matrix:
 [[5 0]
 [5 2]]
```

Logistic Regression outperforms Naive Bayes in the binary sentiment classification task on the 40-sentence corpus with TF-IDF features due to several key factors:

- Discriminative Modeling: Logistic Regression directly optimizes the decision boundary, focusing on class separation rather than modeling feature distributions, leading to higher accuracy (0.92 vs. 0.83) and F1 score (0.91 vs. 0.82).
- No Independence Assumption: Unlike Naive Bayes, Logistic Regression does not assume feature independence, allowing it to capture word co-occurrences (e.g., "fantastic movie"), which are common in text data.
- Robustness to Sparsity: Logistic Regression's regularization handles sparse TF-IDF matrices better, reducing the impact of noise and irrelevant features.
- Flexible Optimization: Iterative optimization via maximum likelihood enables Logistic Regression to fine-tune feature weights, improving discrimination between positive and negative sentiments.

Question 4


Emotional Trajectory of the Passage
Sentiment Score (Compound)
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Segment Number

Question 5

Why is lemmatization often preferred over stemming?
Lemmatization is preferred over stemming because it reduces words to their base form (lemma) using linguistic rules, preserving meaning (e.g., "running" → "run"), whereas stemming applies heuristic rules, often producing invalid words (e.g., "running" → “runn").

How does TF-IDF down-weight common words?
TF-IDF down-weights common words by multiplying term frequency (TF) with inverse document frequency (IDF), where IDF, calculated as log(N/df(t)), assigns lower weights to terms appearing in many documents, emphasizing rare, informative words.

Describe the curse of dimensionality in text data.
The curse of dimensionality in text data refers to the challenges of high-dimensional feature spaces (e.g., BoW/TF-IDF vocabularies), leading to sparsity, increased computational cost, and overfitting due to insufficient data to model numerous features effectively.

When should you use word embeddings instead of BoW/TF-IDF?
Word embeddings should be used instead of BoW/TF-IDF when tasks require capturing semantic relationships (e.g., word similarity, context) or handling low-data scenarios, as they provide dense, low-dimensional representations unlike the sparse, frequency-based BoW/TF-IDF.

How can POS tagging enhance NLP pipelines?
POS tagging enhances NLP pipelines by assigning grammatical categories (e.g., noun, verb) to words, enabling better feature selection, disambiguation, and context-aware processing for tasks like sentiment analysis or named entity recognition.