NLP ASSIGNMENT 1

1. **Explain One-Hot Encoding**

Answer: One-hot encoding is a technique used to convert categorical data into a numerical format that can be used by machine learning algorithms. It involves creating binary (0 or 1) columns for each unique category in a categorical variable.

Here's a simple explanation:

1. Identify Categorical Variables: Determine which variables in your dataset are categorical (e.g., colors, types, labels).

2. List Unique Categories: For each categorical variable, list all the unique categories it contains (e.g., "Red," "Green," "Blue").

3. Create Binary Columns: For each unique category, create a new binary column. Each of these columns will represent one of the unique categories.

4. Assign Binary Values: For each row in your dataset, assign a value of 1 to the column corresponding to the category present in that row and 0 to all other columns.

Advantages

- No Ordinal Relationships: It prevents the model from assuming any ordinal relationships between categories, which is crucial for nominal data.
- Algorithm Compatibility: Many machine learning algorithms require numerical input, and one-hot encoding provides a way to convert categorical data into a usable format without introducing false ordinal relationships.

Disadvantages

- Increased Dimensionality: It can significantly increase the number of columns, especially if the categorical variable has many unique values, leading to higher computational costs and potential overfitting.
- Sparse Data: The resulting binary vectors are sparse (contain many zeros), which can be inefficient in terms of storage and computation.

Alternatives

- Label Encoding: Assigns each category a unique integer, which can introduce ordinal relationships that may not exist.
- Target Encoding: Replaces each category with the mean of the target variable for that category.
- Binary Encoding: Converts categories to binary code and splits binary digits into separate columns, reducing dimensionality compared to one-hot encoding.

One-hot encoding is commonly used and can be easily implemented using libraries in programming languages like Python.

2. Explain Bag of Words

Answer : Bag of Words (BoW) is a simple and widely used technique for natural language processing (NLP) that transforms text data into numerical features. It's often used in tasks like text classification, sentiment analysis, and information retrieval. Here's an explanation of how it works and its key aspects:

How Bag of Words Works

1. Text Preprocessing:
   - Tokenization: Split the text into individual words (tokens).
   - Normalization: Convert all characters to lowercase to ensure that words like "Cat" and "cat" are treated the same.
   - Removing Stop Words**: Eliminate common words that carry less meaning (e.g., "and," "the," "is").
   - Stemming/Lemmatization: Reduce words to their root form (e.g., "running" to "run").

2. Vocabulary Creation:
   - Create a list of all unique words (vocabulary) from the text corpus (the collection of all documents).

3. Vectorization:
   - For each document, create a vector of fixed length equal to the size of the vocabulary.
   - Each element of the vector represents the frequency (or presence) of a word in the document.
   - Typically, this is done by counting the occurrences of each word from the vocabulary in the document.

Example:

Suppose you have a corpus with the following three documents:
1. "I love machine learning."
2. "Machine learning is fascinating."
3. "I love learning new things."

After preprocessing, the vocabulary might be: ["i", "love", "machine", "learning", "is", "fascinating", "new", "things"]

The BoW representation of each document would be:
1. [1, 1, 1, 1, 0, 0, 0, 0]
2. [0, 0, 1, 1, 1, 1, 0, 0]
3. [1, 1, 0, 1, 0, 0, 1, 1]

Applications:
- Text Classification: Categorizing text into predefined labels (e.g., spam detection, sentiment analysis).
- Information Retrieval: Finding documents that are relevant to a query.

- Document Similarity: Measuring the similarity between documents based on their word content.

Bag of Words is a foundational technique in NLP that represents text data in a simple numerical format, allowing for the application of machine learning algorithms to various text-based tasks.

3. Explain Bag of N-Grams

Answer: Bag of N-Grams is an extension of the Bag of Words (BoW) model in natural language processing (NLP). Instead of treating individual words as the basic units, it considers sequences of words (n-grams) to capture more contextual information. This method is useful for tasks where the relationship between consecutive words carries significant meaning, such as in language modeling, text classification, and sentiment analysis.

How Bag of N-Grams Works:

1. Text Preprocessing:
   - Tokenization: Split the text into individual words (tokens).
   - Normalization: Convert all characters to lowercase and perform other text cleaning steps (removing punctuation, stop words, etc.).

2. N-Gram Generation:
   - N-Grams: Generate contiguous sequences of n words from the text. For example, for a bi-gram (2-gram) model, extract all sequences of two consecutive words.

3. Vocabulary Creation:
   - Create a list of all unique n-grams from the entire corpus of documents.

4. Vectorization:
   - For each document, create a vector of fixed length equal to the size of the n-gram vocabulary.
   - Each element of the vector represents the frequency (or presence) of an n-gram in the document.

Example:

Suppose you have a corpus with the following document:
- "I love machine learning."

For a bi-gram model, the n-grams would be:
- "I love"
- "love machine"
- "machine learning"

If the corpus had more documents, you would collect all unique bi-grams from those documents to form the vocabulary.

The BoN-Gram representation of the document would involve counting the occurrences of each bi-gram in the document. For simplicity, assume the vocabulary is ["I love", "love machine", "machine learning", "learning is", "is fascinating"]:

The vector representation of the document "I love machine learning." would be:
- [1, 1, 1, 0, 0]

Applications
- Text Classification: Categorizing text into predefined labels with improved context understanding.
- Sentiment Analysis: Better capturing sentiment by considering phrases instead of individual words.
- Language Modeling: Predicting the next word in a sequence by considering the previous n-1 words.

Bag of N-Grams enhances the Bag of Words model by capturing sequences of words, providing a more context-aware representation of the text. This method can significantly improve the performance of NLP tasks where word order and context are important.

4. **Explain TF-IDF**

Answer: TF-IDF, which stands for Term Frequency-Inverse Document Frequency, is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents (corpus). It helps to identify words that are important to specific documents but not too common across all documents. This technique is widely used in information retrieval, text mining, and natural language processing (NLP).

How TF-IDF Works
TF-IDF is the product of two metrics: Term Frequency (TF) and Inverse Document Frequency (IDF).

1. Term Frequency (TF):
   - Definition: Measures how frequently a term (word) appears in a document.
   - Purpose: Captures the importance of a term within a single document.

2. Inverse Document Frequency (IDF):
   - Definition: Measures how important a term is within the entire corpus. It helps to downscale terms that appear too frequently across all documents, as they are less informative.
   - Purpose: Highlights terms that are unique to fewer documents and are therefore more informative.

3. TF-IDF Calculation:
   - Result: A higher TF-IDF score indicates that the term is more relevant to the document, considering its frequency within the document and its rarity across the corpus.

Example:
Suppose you have a corpus with three documents:

1. Document 1: "I love machine learning."
2. Document 2: "Machine learning is fascinating."
3. Document 3: "I love learning new things."

Let's calculate the TF-IDF for the term "learning" in Document 1.

1. Term Frequency (TF):
   - The term "learning" appears once in Document 1, which has 4 words.
   - TF("learning", Document 1) = 1/4 = 0.25.

2. Inverse Document Frequency (IDF):
   - "Learning" appears in all three documents.
   - IDF("learning") = log(3/3) = log(1) = 0.

3. TF-IDF:
   - TF-IDF("learning", Document 1) = 0.25 * 0 = 0.

In this case, since "learning" appears in all documents, its IDF is 0, making its TF-IDF score 0, indicating it's not particularly informative across the corpus.

Applications:
- Search Engines: Ranking documents based on the relevance of search queries.
-Text Classification: Features for algorithms to classify documents into categories.
- Information Retrieval: Finding and ranking relevant documents in large text corpora.

TF-IDF is a powerful tool for identifying the most important terms in a document by considering both the frequency of terms within the document and their rarity across the entire corpus, thus providing a robust method for text analysis and retrieval.

5. **What is OOV problem?**

Answer: The OOV (Out-Of-Vocabulary) problem occurs when a text processing system encounters words that were not present in the training dataset or vocabulary used to build the model. This issue is common in natural language processing (NLP) and can significantly affect the performance of various NLP tasks, such as text classification, machine translation, and speech recognition.

Causes of OOV Problem
1. Limited Vocabulary: The training data only includes a finite set of words, which may not cover all possible words in the language or domain.
2. Evolving Language: Language constantly evolves, with new words, slang, and terminologies emerging regularly.
3. Domain-Specific Terms: Different domains or fields may use specialized jargon that is not present in the general training corpus.

4. Misspellings and Variants: Misspelled words or variations of known words (e.g., British vs. American English) can lead to OOV issues.

Impact of OOV Problem

1. Reduced Accuracy: Models may perform poorly on tasks involving unseen words because they cannot leverage information from these words.
2. Incomplete Understanding: For tasks like text summarization or machine translation, missing or misinterpreting OOV words can lead to incorrect or incomplete results.
3. Increased Uncertainty: Systems may become less confident in their predictions when encountering OOV words, leading to unreliable outputs.

Strategies to Mitigate OOV Problem

1. Subword Tokenization:
   - Byte Pair Encoding (BPE): Breaks words into smaller subword units (e.g., "unhappiness" into "un", "happiness"). This approach allows the model to handle rare and unseen words more effectively.
   - WordPiece: Similar to BPE, used in models like BERT, which breaks words into subword pieces based on frequency in the training data.

2. Character-Level Models: Use character-level representations instead of word-level representations. This way, the model can build words from characters and handle unseen words.

3. Pretrained Embeddings: Use embeddings trained on large, diverse corpora (e.g., Word2Vec, GloVe, FastText) which cover a broader vocabulary. FastText, in particular, uses subword information, making it more robust to OOV words.

4. Data Augmentation: Expand the training dataset to include more diverse vocabulary from various sources, domains, and contexts.

5. Fallback Mechanisms: Implement fallback mechanisms like replacing OOV words with a special token (e.g., "<UNK>") and designing the model to handle these tokens appropriately.

6. Contextual Embeddings: Use models like BERT, GPT, or ELMo that generate context-dependent embeddings, which can infer meanings of OOV words based on surrounding context.

Example:

Suppose a text classification model is trained on movie reviews and the training data includes the word "fantastic" but not "fantabulous." When the model encounters the word "fantabulous" in a new review, it doesn't know how to handle it because it's an OOV word. Strategies like subword tokenization or character-level models can help the model break down "fantabulous" into known subunits or characters, making it possible to infer its meaning.

OOV problem is a significant challenge in NLP, but various strategies, such as subword tokenization, character-level models, and pretrained embeddings, can help mitigate its impact and improve the robustness and accuracy of NLP systems.

6. **What are word embeddings?**

Answer: Word embeddings are numerical representations of words that capture their meanings, semantic relationships, and syntactic properties in a continuous vector space. They are used in natural language processing (NLP) to convert words into a form that can be processed by machine learning algorithms. Unlike traditional one-hot encoding, which represents words as high-dimensional sparse vectors, word embeddings represent words as dense, low-dimensional vectors.

Key Concepts:

1. Dense Vectors: Word embeddings are typically dense vectors of real numbers, where each dimension captures some aspect of the word's meaning or usage.
2. Semantic Similarity: Words with similar meanings are represented by vectors that are close to each other in the embedding space.
3. Dimensionality Reduction: Embeddings reduce the high-dimensional space of one-hot vectors to a lower-dimensional space, making computations more efficient.

Popular Word Embedding Models:

1. Word2Vec: Developed by Google, it includes two main algorithms: Continuous Bag of Words (CBOW) and Skip-gram.
   - CBOW predicts a target word from its context words.
   - Skip-gram predicts context words from a target word.

2. GloVe (Global Vectors for Word Representation): Developed by Stanford, GloVe captures global statistical information by factorizing the word co-occurrence matrix.

3. FastText: Developed by Facebook, it extends Word2Vec by representing words as bags of character n-grams, which allows it to generate embeddings for out-of-vocabulary (OOV) words.

4. BERT (Bidirectional Encoder Representations from Transformers): Unlike static embeddings from Word2Vec and GloVe, BERT generates contextual embeddings, meaning the representation of a word depends on its context within a sentence.

How Word Embeddings Are Learned:

Word embeddings are typically learned using neural network-based models trained on large text corpora. The training process involves adjusting the vector representations of words to minimize a loss function that captures the differences between predicted and actual context words.

Applications of Word Embeddings:

1. Text Classification: Representing text data for classification tasks.
2. Sentiment Analysis: Capturing the sentiment of text by understanding the meaning of words.
3. Machine Translation: Translating text from one language to another by understanding word meanings and relationships.
4. Information Retrieval: Improving search engine results by understanding the context and semantics of queries and documents.

5. Named Entity Recognition (NER): Identifying proper nouns and specific entities in text.

Example:
Suppose you have the words "king," "queen," "man," and "woman." A well-trained word embedding model might produce vectors such that the relationship between these words can be captured by vector arithmetic:

- `king` - `man` + `woman` ≈ `queen`

This illustrates that the embeddings capture the gender relationship between "king" and "queen" similar to "man" and "woman."

Word embeddings are a powerful tool in NLP that enable the representation of words in a meaningful, low-dimensional space, capturing semantic relationships and improving the performance of various text processing tasks.

7. Explain Continuous bag of words (CBOW)

Answer: The Continuous Bag of Words (CBOW) model is a neural network-based approach to learning word embeddings, developed as part of the Word2Vec framework by Google. The CBOW model predicts a target word based on its surrounding context words within a fixed-size window. Here's a detailed explanation of how CBOW works:

How CBOW Works:
1. Context Window: Define a window size (k) around the target word. For example, (k = 2), the context window includes the two words before and the two words after the target word.

2. Training Objective: The goal is to predict the target word given the context words. The model maximizes the probability of the target word given the context words.

3. Input and Output:
- Input: The context words within the window.
- Output: The target word in the middle of the context window.

4. Architecture:
- Input Layer: Represents the context words as one-hot encoded vectors.
- Hidden Layer: A single hidden layer where the word embeddings are learned. The weights between the input and hidden layer represent the word embeddings.
- Output Layer: A softmax layer that predicts the target word based on the context words.

Steps in CBOW:
1. One-Hot Encoding: Each context word is represented as a one-hot encoded vector. If the vocabulary size is (V), each one-hot vector has a dimension of (V), with only one element set to 1 and the rest set to 0.

2. Embedding Layer: The one-hot vectors are multiplied by a weight matrix (W) (size (V times N), where (N) is the embedding dimension) to obtain dense vector representations (embeddings) of the context words. The embeddings of the context words are averaged to produce a single vector.

3. Hidden Layer: The averaged context vector is passed through the hidden layer. This layer captures the relationships between words based on their context.

4. Output Layer: The hidden layer output is multiplied by another weight matrix (W') (size (N times V)) and passed through a softmax function to produce a probability distribution over the vocabulary. The model predicts the target word as the word with the highest probability.

5. Training: The model is trained using stochastic gradient descent (SGD) or another optimization algorithm to minimize the cross-entropy loss between the predicted and actual target words.

Example:
Consider the sentence: "The quick brown fox jumps over the lazy dog."

If the context window size is 2, for the target word "brown", the context words are "the" and "quick" (before) and "fox" and "jumps" (after).

1. One-hot encode the context words: "the", "quick", "fox", "jumps".
2. Convert these one-hot vectors to embeddings.
3. Average the embeddings of the context words.
4. Pass the average through the hidden layer.
5. Predict the target word "brown" using the softmax output layer.
6. Update the weights based on the prediction error.

Applications:
- Word Embeddings: Learning dense vector representations of words that capture semantic meanings.
- Text Classification: Providing word embeddings as features for classification tasks.
- Semantic Analysis: Understanding word relationships and meanings in text.

The CBOW model is a neural network approach to word embedding learning that predicts a target word based on its surrounding context words, making it efficient and effective for capturing semantic information from large text corpora.

8. Explain SkipGram

Answer: The Skip-gram model is another neural network-based approach to learning word embeddings, also developed as part of the Word2Vec framework by Google. Unlike the Continuous Bag of Words (CBOW) model, which predicts a target word from its context, the Skip-gram model does the opposite: it predicts the context words given a target word. This method is particularly effective for capturing the semantic relationships between words, even for rare words in the corpus.

How Skip-gram Works:

1. Context Window: Define a window size (k) around the target word. For example, (k = 2), the context window includes the two words before and the two words after the target word.

2. Training Objective: The goal is to maximize the probability of the context words given the target word. The model learns to predict context words from a given target word.

3. Input and Output:
   - Input: The target word.
   - Output: The context words within the window around the target word.

4. Architecture:
   - Input Layer: Represents the target word as a one-hot encoded vector.
   - Hidden Layer: A single hidden layer where the word embeddings are learned. The weights between the input and hidden layer represent the word embeddings.
   - Output Layer: A softmax layer that predicts the probability distribution over the vocabulary for each context word.

Steps in Skip-gram:

1. One-Hot Encoding: The target word is represented as a one-hot encoded vector. If the vocabulary size is (V), the one-hot vector has a dimension of (V), with only one element set to 1 and the rest set to 0.

2. Embedding Layer: The one-hot vector is multiplied by a weight matrix (W) (size (V times N), where (N) is the embedding dimension) to obtain a dense vector representation (embedding) of the target word.

3. Hidden Layer: The resulting dense vector is passed through the hidden layer, which captures the relationships between the target word and its context words.

4. Output Layer: The hidden layer output is multiplied by another weight matrix (W') (size (N times V)) and passed through a softmax function to produce a probability distribution over the vocabulary. The model predicts the context words as the words with the highest probabilities.

5. Training: The model is trained using stochastic gradient descent (SGD) or another optimization algorithm to minimize the cross-entropy loss between the predicted and actual context words.

Example:

Consider the sentence: "The quick brown fox jumps over the lazy dog."

If the context window size is 2, for the target word "fox", the context words are "quick" and "brown" (before) and "jumps" and "over" (after).

1. One-hot encode the target word "fox".
2. Convert this one-hot vector to an embedding.
3. Pass the embedding through the hidden layer.

4. Predict the context words "quick", "brown", "jumps", and "over" using the softmax output layer.
5. Update the weights based on the prediction error for each context word.

Applications:
- Word Embeddings: Learning dense vector representations of words that capture semantic meanings.
- Text Classification: Providing word embeddings as features for classification tasks.
- Semantic Analysis: Understanding word relationships and meanings in text.

The Skip-gram model is a neural network approach to word embedding learning that predicts context words given a target word. This method is effective for capturing the semantic and syntactic relationships between words, making it valuable for a variety of NLP tasks.

9. Explain Glove Embeddings.

Answer: GloVe (Global Vectors for Word Representation) embeddings are a type of word embedding model designed to capture global statistical information about word co-occurrences in a corpus of text. Developed by researchers at Stanford University, GloVe differs from models like Word2Vec (CBOW and Skip-gram) by leveraging both global corpus statistics and local context window-based statistics to generate word embeddings.

Conceptual Explanation of GloVe Embeddings:
GloVe embeddings are designed to capture the semantic relationships between words by leveraging the global statistical information of their co-occurrences across a corpus of text. Here's how GloVe embeddings work conceptually:

1. Word Co-occurrence Matrix:
   - GloVe begins by constructing a global word-word co-occurrence matrix based on the entire corpus. This matrix (X) captures how frequently words appear together within a fixed context window.
   - Each element (X_{ij} \) in the matrix represents the number of times word (j) appears in the context of word (i).

2. Objective:
   - The main objective of GloVe is to learn word embeddings (vector representations of words) such that certain relationships between words are preserved. These relationships are typically derived from the logarithms of the co-occurrence probabilities of word pairs.
   - The model aims to minimize the difference between the dot product of word embeddings and the logarithm of their co-occurrence counts in the corpus.

3. Training Process:
   - Initialization: Initialize word embeddings randomly or using pre-trained vectors.
   - Optimization: Use optimization algorithms (such as gradient descent) to iteratively adjust word embeddings based on the objective function. This function ensures that words that often co-occur in similar contexts have similar vector representations.

- Weighting: GloVe uses a weighting function to balance the influence of frequently occurring word pairs (like "the" and "and") versus less common pairs (like "dog" and "cat"). This helps in focusing more on informative co-occurrence patterns.

4. Output:
- After training, each word in the vocabulary is represented by a dense vector (embedding) of fixed dimensions (e.g., 100, 200 dimensions).
- These embeddings capture semantic relationships such as similarity and analogy (e.g., vector("king") - vector("man") + vector("woman") ≈ vector("queen")).

Applications of GloVe Embeddings:
- Word Similarity: Computing similarity between words based on vector distances.
- Text Classification: Providing features for classifiers by representing words as dense vectors.
- Language Modeling: Improving the performance of models that predict the next word in a sequence.

GloVe embeddings are powerful tools in NLP for capturing semantic relationships between words by leveraging global statistical information from word co-occurrences in a corpus. These embeddings have become widely used for various tasks due to their effectiveness and efficiency in learning dense vector representations of words.