**Assignment-2**

1. **Features of Corpus?**

* In Natural Language Processing (NLP), a corpus refers to a large and structured collection of text or spoken language data that is used for various linguistic analysis and machine learning tasks. Here are some key features of a corpus in NLP:
* **Size:** A corpus is typically large, containing a substantial amount of text data. The size can vary significantly depending on the specific use case, ranging from a few gigabytes to terabytes or more.
* **Text Variety:** A good corpus should encompass a wide range of text types and genres, such as news articles, books, social media posts, scientific papers, conversational data, and more. This diversity allows NLP practitioners to handle different language styles and domains.
* **Representative:** A corpus should be representative of the language or domain it aims to model. For example, a medical NLP corpus should include medical texts, while a general-purpose corpus should cover various topics and writing styles.
* **Annotated Data:** Some corpora include annotations or labels, such as part-of-speech tags, named entities, sentiment labels, and more. Annotated corpora are valuable for training and evaluating NLP models.
* **Time Period:** The texts within a corpus may span different time periods. Historical corpora may contain older texts, while contemporary corpora focus on recent language use. Time-based corpora are essential for studying language evolution and change.
* **Metadata:** Corpora often come with metadata, which includes information about the texts, such as publication dates, authors, sources, and more. Metadata can be crucial for analyzing and contextualizing the text data.
* **Open Access:** Some corpora are open access, meaning they are freely available to the research community. Open access corpora promote transparency and collaboration in NLP research.
* **Cleaning and Preprocessing:** Raw text data in a corpus often requires cleaning and preprocessing to remove noise, format text, and perform tasks like tokenization and stemming.

Corpora serve as the foundation for many NLP tasks, including text classification, language modeling, sentiment analysis, and machine translation, among others.

1. **Explain the N-gram model?**

* **N-gram** is a sequence of the N-words in the modeling of NLP.
* Consider an example of the statement for modeling. “I love reading history books and watching documentaries”. In one-gram or unigram, there is a one-word sequence. As for the above statement, in one gram it can be “I”, “love”, “history”, “books”, “and”, “watching”, “documentaries”. In two-gram or the bi-gram, there is the two-word sequence i.e. “I love”, “love reading”, or “history books”. In the three-gram or the tri-gram, there are the three words sequences i.e. “I love reading”, “history books,” or “and watching documentaries”.
* The illustration of the N-gram modeling i.e. for N=1,2,3 is given below in Figure:

|  |
| --- |
| Uni-gram, Bi-gram, and Tri-gram Model |

* **For N-1 words,** the N-gram modeling predicts most occurred words that can follow the sequences. The model is the probabilistic language model which is trained on the collection of the text. This model is useful in applications i.e. speech recognition, and machine translations. A simple model has some limitations that can be improved by smoothing, interpolations, and back off. So, the N-gram language model is about finding probability distributions over the sequences of the word.
* Consider the sentences i.e. "There was heavy rain" and "There was heavy flood". By using experience, it can be said that the first statement is good. The N-gram language model tells that the "heavy rain" occurs more frequently than the "heavy flood". So, the first statement is more likely to occur and it will be then selected by this model. In the one-gram model, the model usually relies on that which word occurs often without pondering the previous words.

1. **Explain Word2vec?**

* **Word2vec** is a combination of models used to represent distributed representations of words in a corpus C. Word2Vec (W2V) is an algorithm that accepts text corpus as an input and outputs a vector representation for each word, as shown in the diagram below:

|  |
| --- |
| https://miro.medium.com/v2/resize:fit:579/1*aaMLVYSbAe086KowQQl6zw.png |

* **There are two flavors of this algorithm namely:** **CBOW** and **Skip-Gram**. Given a set of sentences (also called corpus) the model loops on the words of each sentence and either tries to use the current word w in order to predict its neighbors (i.e., its context), this approach is called “Skip-Gram”, or it uses each of these contexts to predict the current word w, in that case the method is called “Continuous Bag Of Words” (CBOW). To limit the number of words in each context, a parameter called “window size” is used.

|  |
| --- |
| https://miro.medium.com/v2/resize:fit:536/1*vZhxrBkCz-yN_rzZBqSKiA.png |

* The vectors we use to represent words are called neural word embeddings, and representations are strange. One thing describes another, even though those two things are radically different. As Elvis Costello said: “Writing about music is like dancing about architecture.” Word2vec “vectorizes” about words, and by doing so it makes natural language computer-readable. we can start to perform powerful mathematical operations on words to detect their similarities.
* So, a neural word embedding represents a word with numbers. It’s a simple, yet unlikely, translation. Word2vec is similar to an autoencoder, encoding each word in a vector, but rather than training against the input words through reconstruction, as a restricted Boltzmann machine does, word2vec trains words against other words that neighbor them in the input corpus.
* It does so in one of two ways, either using context to predict a target word (a method known as continuous bag of words, or CBOW), or using a word to predict a target context, which is called skip-gram. We use the latter method because it produces more accurate results on large datasets.

|  |
| --- |
| https://miro.medium.com/v2/resize:fit:557/1*99ewFdV861rqY_K14bc69w.png |

* When the feature vector assigned to a word cannot be used to accurately predict that word’s context, the components of the vector are adjusted. Each word’s context in the corpus is the teacher sending error signals back to adjust the feature vector. The vectors of words judged similar by their context are nudged closer together by adjusting the numbers in the vector.

1. **Explain TF-IDF algorithm with Suitable Example?**

* **TF-IDF** stands for Term Frequency-Inverse Document Frequency, is a numerical statistic used in natural language processing (NLP) and information retrieval to measure the importance of a term (word or phrase) within a document relative to a collection of documents, known as a corpus. It is a technique that helps to identify the significance of words or phrases in a document by considering both their frequency in the document and their uniqueness across the corpus. TF-IDF algorithm is made of 2 algorithms multiplied together:
* **Term Frequency:** Term frequency (TF) is how often a word appears in a document, divided by how many words there are.
* **TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)**
* **Inverse document frequency:** Term frequency is how common a word is, inverse document frequency (IDF) is how unique or rare a word is.
* **IDF(t) = log\_e(Total number of documents / Number of documents with term t in it)**
* **Example:** Consider a document containing 100 words wherein the word **apple** appears 5 times.
* The term frequency (i.e., TF) for **apple**is then (5 / 100) = 0.05.
* Now, assume we have 10 million documents and the word apple appears in one thousand of these.
* Then, the inverse document frequency (i.e., IDF) is calculated as log(10,000,000 / 1,000) = 4.
* Thus, the TF-IDF weight is the product of these quantities: 0.05 \* 4 = 0.20.

1. **Explain Point-wise Mutual Information(PMI)?**

* Natural Language Processing domain, there is a popular metric named **pointwise mutual information**, also called the **PMI**. It is a statistical measure to calculate the association between two words in a given corpus. PMI is calculated by comparing the probability of the co-occurrence of two words with their individual probabilities of occurrence. The formula for PMI is as follows: **PMI(x,y) = log(P(x,y) / (P(x) \* P(y)))**
* Consider an example to calculate the PMI between two words in a short Text: **"I love ice cream. Ice cream is delicious."** Let's calculate the PMI between the words "ice" and "cream."
* **Calculate the probability of each word:**
* P("ice"): It appears once out of 8 words, so P("ice") = 1/8.
* P("cream"): It appears twice out of 8 words, so P("cream") = 2/8 = 1/4.
* **Calculate the probability of both words co-occurring:** P("ice" and "cream"): Both words appear together once out of 8 words, so P("ice" and "cream") = 1/8.
* **Plug these values into the PMI formula:**
* PMI("ice", "cream") = log((1/8) / ((1/8) \* (1/4))) = log(4) = 1.3863 (rounded to four decimal places)
* In this example, the PMI between "ice" and "cream" is approximately 1.3863. This positive PMI value indicates that there is a non-random association between the words "ice" and "cream" in the given text, suggesting that they are often found together in this context.

1. **Explain Smoothing techniques?**

* **Smoothing techniques** in Natural Language Processing (NLP) are used to address the problem of zero probabilities or low probabilities that arise when estimating probabilities or frequencies of events, such as word occurrences or n-gram frequencies, from finite data. These techniques are essential for various NLP tasks, including language modeling, machine translation, and speech recognition, where accurate probability estimation is crucial. Smoothing helps prevent zero probabilities, which can lead to poor model generalization and performance. Here are some common smoothing techniques in NLP:
* **Additive Smoothing (Laplace Smoothing):**
* Additive smoothing, also known as Laplace smoothing, is a straightforward technique that adds a small constant (α) to the count of each event in the probability estimation. This effectively smooths out the distribution and prevents zero probabilities. The smoothed probability of an event x is calculated as follows:

|  |
| --- |
|  |

* Count(x) is the observed count of event x.
* N is the total count of events in the data.
* ∣V∣ is the vocabulary size (number of unique events).
* α is the smoothing parameter, typically set to a small positive value, such as 1.

|  |
| --- |
|  |

* **Add-One Smoothing (Lidstone Smoothing):** Add-One smoothing, a variation of additive smoothing, sets α to 1. It is a simple and widely used method. The smoothed probability is calculated as:
* **Good-Turing Smoothing:** Good-Turing smoothing is a more sophisticated technique that estimates probabilities based on the relative frequencies of different events. It aims to predict the probability of unseen events by adjusting the probability of observed events. The idea is to estimate how many times an event occurs exactly once in the data (i.e., the number of events with a frequency of 1), and then use this estimate to adjust the probabilities. Good-Turing smoothing can work well when dealing with rare events.
* **Kneser-Ney Smoothing:** Kneser-Ney smoothing is a more complex technique that models the probability of a word based on its position in a phrase or n-gram. It uses a back-off mechanism, where the probability of an n-gram is based on the probability of the (n-1)-gram and the probability of a shorter context. Kneser-Ney smoothing often performs well in language modeling tasks.
* **Interpolation Smoothing:** Interpolation smoothing combines lower-order n-gram models with higher-order models to estimate probabilities. It assigns weight to each n-gram model and combines their probabilities to estimate the probability of an n-gram. Interpolation smoothing can capture both short-range and long-range dependencies in language.

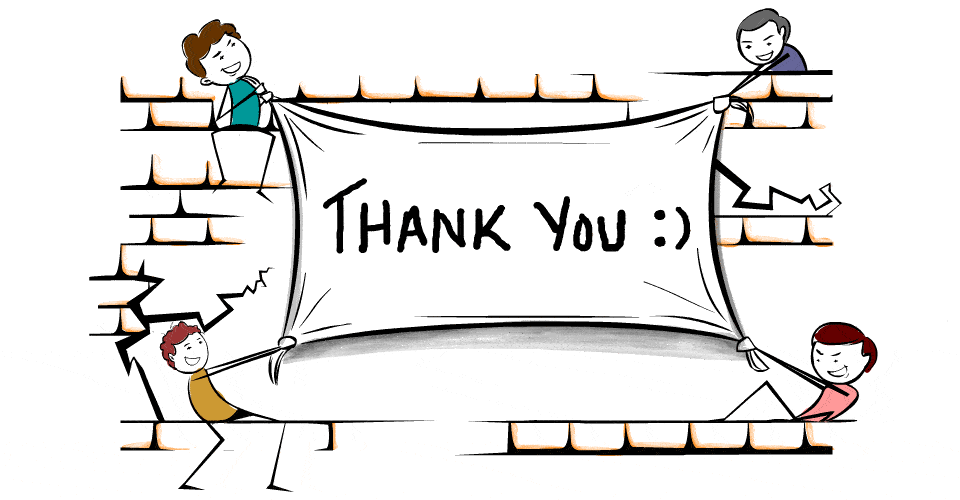
1. **Explain Relevance ranking algorithms and Types?**

|  |
| --- |
|  |

* **Relevance ranking** is a core problem of Information Retrieval which plays a fundamental role in various real world applications, such as search engines. Given a query and a set of candidate text documents, relevance ranking algorithms determine how relevant each text document is for the given query.
* **Retrieval:** It involves searching for documents or items from a large dataset (such as a database, corpus, or collection of web pages) that are relevant to a user's query or information need. Retrieval can be based on various criteria, such as keyword matching, similarity to a query, or relevance to specific user preferences.
* **Ranking:** Ranking is the process of ordering or sorting the retrieved documents or items based on their relevance to the user's query. After retrieval, documents that are deemed more relevant are assigned higher ranks, while less relevant documents receive lower ranks.
* **Re-ranking:** Re-ranking aims to improve the precision and relevance of the top-ranked results, ensuring that the most relevant documents are presented at the very top of the list. Re-ranking can be performed using additional information or features not considered in the initial ranking, such as user behavior, user profiles, query expansion, or more complex machine learning models.
* **Ranking algorithms can be divided into two categories:**

1. **Deterministic ranking algorithms:** A deterministic ranking algorithm is one in which the order of the items in the ranked list is fixed and does not change, regardless of the input data. An example of a deterministic ranking algorithm is the rank-by-feature algorithm. In this algorithm, each item is assigned a rank based on its feature value. The item with the highest feature value is assigned a rank of 1, and the item with the lowest feature value is assigned a rank of N, where N is the number of items in the dataset.
2. **Probabilistic ranking algorithms:** In a probabilistic ranking algorithm, the order of the items in the ranked list may vary, depending on the input data. An example of a probabilistic ranking algorithm is the rank-by-confidence algorithm. In this algorithm, each item is assigned a rank based on its confidence value. The item with the highest confidence value is assigned a rank of 1, and the item with the lowest confidence value is assigned a rank of N, where N is the number of items in the dataset.

**(PPT Give you more understanding than PDF)** The material for the PDF has been compiled from various sources such as books, tutorials (offline and online), lecture notes, several resources available on Internet. The information contained in this PDF is for general information and education purpose only. While we endeavor to keep the information up to date and correct, **we make no representation of any kind about the completeness and accuracy of the material.** The information shared through this PDF material should be used for educational purpose only.

****