1. **Describe the NLP pipeline for ranking tickets in a ticketing system by Uber.?**

The information needed to identify the ticket issue and select a solution

After cleaning up the text by removing HTML tags (not shown in the figure), the pre-processing steps consist of tokenization, lowercasing, stop word removal, and lemmatization.

After pre-processing, the ticket text is represented as a collection of words The next step in this pipeline is feature engineering. The bag of words we obtained earlier is fed to two NLP modules—TF-IDF (term frequency and inverse document frequency) and LSI (latent semantic indexing)—which are used to understand the meaning of a text using this bag of words representation.

Uber collects the historical tickets for each solution from their database, forms a bag-of-words vector representation for each solution, and creates a topic model based on these representations. An incoming ticket is then mapped to this topic space of solutions, creating a vector representation for the ticket. Cosine similarity is a common measure of similarity between any two vectors. It is used to create a vector where each element indicates the ticket text’s similarity to one solution. Thus, at the end of this feature engineering step, we end up with a representation indicating the ticket text’s similarity to all possible solutions

In the next stage, modeling, this representation is combined with ticket information and trip data to build a ranking system that shows the three best solutions for the ticket. The matches are then ranked based on a scoring function. The next step in our pipeline is evaluation. How does evaluation work in this context? While the evaluation of model performance itself can be done in terms of an intrinsic evaluation measure such as MRR, the overall effectiveness of this approach is evaluated extrinsically.

1. **3. Use “D1: Dog bites man, D2: Man bites dog, D3: Dog eats meat, and D4: Man eats food” as an input, find their representation using one-hot encoding, bag of words, bag of N-gram, and TF-IDF.?**

**one-hot encoding** :

In one-hot encoding, each word w in the corpus vocabulary is given a unique integer ID word id that is between 1 and |V|, where V is the set of the corpus vocabulary. Each word is then represented by a V-dimensional binary vector of 0s and 1s.

On the positive side, one-hot encoding is intuitive to understand and straightforward to implement.

However, it suffers from a few shortcomings:

• The size of a one-hot vector is directly proportional to size of the vocabulary, and most real-world corpora have large vocabularies

• This representation does not give a fixed-length representation for text

if a text has 10 words, you get a longer representation for it as compared to a text with 5 words.

For most learning algorithms, we need the feature vectors to be of the same length.

• Say we train a model using our toy corpus. At runtime, we get a sentence: “man eats fruits.” The training data didn’t include “fruit” and there’s no way to represent it in our model. This is known as the out of vocabulary (OOV) problem.

**BOW :**

represent the text under consideration as a bag (collection) of words while ignoring the order and context

the advantages of this encoding:

• Like one-hot encoding, BoW is fairly simple to understand and implement.

• With this representation, documents having the same words will have their vector representations closer to each other

• We have a fixed-length encoding for any sentence of arbitrary length.

However, it has its share of disadvantages, too:

• The size of the vector increases with the size of the vocabulary.

• It does not capture the similarity between different words that mean the same thing. Say we have three documents: “I run”, “I ran”, and “I ate”. BoW vectors of all three documents will be equally apart.

• This representation does not have any way to handle out of vocabulary words

• As the name indicates, it is a “bag” of words—word order information is lost in this representation

**Bag of N-Grams** :

It does so by breaking text into chunks of n contiguous words (or tokens).This can help us capture some context, which earlier approaches could not do.

Each chunk is called an n-gram. The corpus vocabulary, V, is then nothing but a collection of all unique n-grams across the text corpus. Then, each document in the corpus is represented by a vector of length |V|

Here are the main pros and cons of BoN:

• It captures some context and word-order information in the form of n-grams.

• Thus, resulting vector space is able to capture some semantic similarity. Documents having the same n-grams will have their vectors closer to each other

• As n increases, dimensionality (and therefore sparsity) only increases rapidly.

• It still provides no way to address the OOV problem.

**TF-IDF :**

it aims to quantify the importance of a given word relative to other words in the document and in the corpus.

If we look back at all the representation schemes we’ve discussed so far, we notice three fundamental drawbacks:

• They’re discrete representations , they treat language units (words, n-grams, etc.) as atomic units. This discreteness hampers their ability to capture relationships between words.

• The feature vectors are high-dimensional representations.

• They cannot handle OOV words

**Describe the pipeline for building a classifier when there is no training data**

1. **Describe the pipeline for building text classification systems.**
   * Collect or create a labeled dataset suitable for the task.
   * Split the dataset into two (training and test) or three parts: training, validation (i.e., development), and test sets, then decide on evaluation metric(s).
   * Transform raw text into feature vectors.
   * Train a classifier using the feature vectors and the corresponding labels from the training set.
   * Using the evaluation metric(s) from Step 2, benchmark the model performance on the test set.
   * Deploy the model to serve the real-world use case and monitor its performance
2. **List the steps for converting training and test data into a format suitable for the neural network.?** 
   * Tokenize the texts and convert them into word index vectors.
   * Pad the text sequences so that all text vectors are of the same length.
   * Map every word index to an embedding vector.
   * Use the output from Step 3 as the input to a neural network architecture