**Assignment 5.2**

**1. Explain the concept of Tokenization.**

**Ans)**

Tokenization refers to a process by which a piece of sensitive data, such as a credit card number, is replaced by a surrogate value known as a token. The sensitive data still generally needs to be stored securely at one centralized location for subsequent reference and requires strong protections around it. The security of a tokenization approach depends on the security of the sensitive values and the algorithm and process used to create the surrogate value and map it back to the original value.

Tokenization is the act of breaking up a sequence of strings into pieces such as words, keywords, phrases, symbols and other elements called tokens. Tokens can be individual words, phrases or even whole sentences. In the process of tokenization, some characters like punctuation marks are discarded. The tokens become the input for another process like parsing and text mining.

Tokenization is used in computer science, where it plays a large part in the process of lexical analysis.

Tokenization relies mostly on simple heuristics in order to separate tokens by following a few steps:

Tokens or words are separated by whitespace, punctuation marks or line breaks

White space or punctuation marks may or may not be included depending on the need

All characters within contiguous strings are part of the token. Tokens can be made up of all alpha characters, alphanumeric characters or numeric characters only.

Tokens themselves can also be separators. For example, in most programming languages, identifiers can be placed together with arithmetic operators without white spaces. Although it seems that this would appear as a single word or token, the grammar of the language actually considers the mathematical operator (a token) as a separator, so even when multiple tokens are bunched up together, they can still be separated via the mathematical operator.

**2. How and when is Gram tokenization is used?**

**Ans)**

**Tokenizing by n-gram**

unnest\_tokens() have been used to tokenize the text by word, or sometimes by sentence, which is useful for the kinds of sentiment and frequency analyses. But we can also use the function to tokenize into consecutive sequences of words of length n, called n-grams.

We do this by adding the token = "ngrams" option to unnest\_tokens(), and setting n to the number of words we wish to capture in each n-gram. When we set nto 2, we are examining pairs of two consecutive words, often called “bigrams”

library(janeaustenr)

austen\_bigrams <- austen\_books() %>%

unnest\_tokens(bigram, text, token = "ngrams", n = 2)

austen\_bigrams %>%

count(bigram, sort = TRUE)

**Filtering n-grams**

As one might expect, a lot of the most common bigrams are pairs of common (uninteresting) words, such as of the and to be: what we call “stop-words” (see Chapter 1). This is a useful time to use tidyr::separate(), which splits a column into multiple based on a delimiter. This lets us separate it into two columns, filter out stop words separately, and then combine the results.

**Analyzing bigrams**

The result of separating bigrams is helpful for exploratory analyses of the text. As a simple example, we might be interested in the most common “streets” mentioned in each book

**3. What is meant by the TFID? Explain in detail.**

Ans)

TF-IDF (term frequency-inverse document frequency) is a statistical measure that evaluates how relevant a word is to a document in a collection of documents.

This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents.

It has many uses, most importantly in automated text analysis, and is very useful for scoring words in machine learning algorithms for Natural Language Processing (NLP).

TF-IDF was invented for document search and information retrieval. It works by increasing proportionally to the number of times a word appears in a document, but is offset by the number of documents that contain the word. So, words that are common in every document, such as this, what, and if, rank low even though they may appear many times, since they don’t mean much to that document in particular.

However, if the word Bug appears many times in a document, while not appearing many times in others, it probably means that it’s very relevant. For example, if what we’re doing is trying to find out which topics some NPS responses belong to, the word Bug would probably end up being tied to the topic Reliability, since most responses containing that word would be about that topic.

How is TF-IDF calculated?

TF-IDF for a word in a document is calculated by multiplying two different metrics:

The term frequency of a word in a document. There are several ways of calculating this frequency, with the simplest being a raw count of instances a word appears in a document. Then, there are ways to adjust the frequency, by length of a document, or by the raw frequency of the most frequent word in a document.

The inverse document frequency of the word across a set of documents. This means, how common or rare a word is in the entire document set. The closer it is to 0, the more common a word is. This metric can be calculated by taking the total number of documents, dividing it by the number of documents that contain a word, and calculating the logarithm.

So, if the word is very common and appears in many documents, this number will approach 0. Otherwise, it will approach 1.