**Web search and Language based Model**

Web search has become an integral part of our daily lives. We rely on search engines to find answers to our questions, locate businesses, research products, and more. The success of a search engine depends on its ability to retrieve relevant information from a large amount of data available on the web. Information retrieval (IR) is the scientific discipline concerned with designing algorithms and systems for efficient and effective retrieval of information from large collections of data.

Traditionally, information retrieval systems relied on keywords to match user queries with relevant documents. However, with the advent of language models, the approach to web search has shifted to language model-based information retrieval. Language models are statistical models that learn the probability distribution of words in a language, based on their frequency of occurrence in a large corpus of text. These models can be used to assign a score to a query and a document, based on their language model probabilities, and rank the documents in order of relevance.

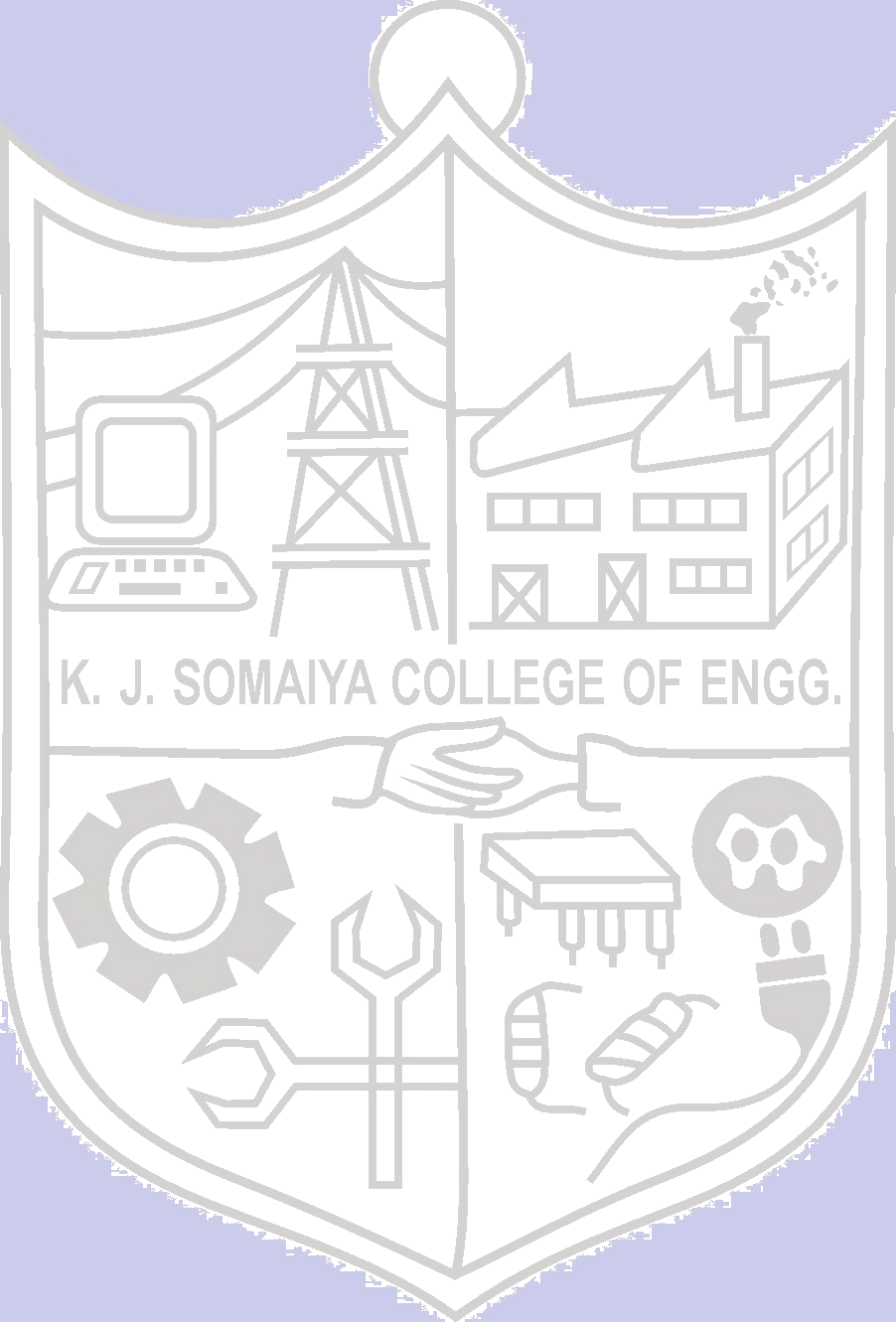
In recent years, neural language models such as BERT and GPT have shown remarkable performance in natural language processing tasks, including information retrieval. These models are trained on large amounts of text data and are able to capture complex relationships between words and sentences. They have been integrated into modern search engines to improve the relevance of search results.

**What are language models and how are they used for web information retrieval?**

Language models are statistical models that learn the probability distribution of words in a language, based on their frequency of occurrence in a large corpus of text. They are commonly used in natural language processing tasks such as language generation, translation, and sentiment analysis. Language models are also increasingly being used in information retrieval, where they help to determine the relevance of documents to a user's query.

In traditional keyword-based approaches to web search, the search engine would match the user's query to documents that contained the same keywords. However, with the introduction of language models, the approach to web search has changed. Instead of simply matching keywords, language models are used to estimate the probability that a document is relevant to a user's query.

The basic idea is that a language model assigns a probability to each word in a document, based on its frequency in the corpus used to train the model. Given a user's query, the model calculates the probability that the query and the document share the same language model. This probability is used as a measure of relevance, and the documents are ranked accordingly. For example, suppose a user enters the query "best pizza in town". A language model-based search engine would estimate the probability that each document containing the word "pizza" is relevant to the user's query, based on the probabilities assigned to the words "best" and "town" as well. The documents with the highest probability would be ranked at the top of the search results.

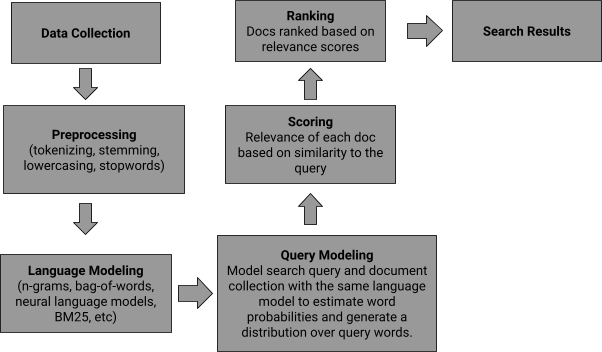
Language models used in web information retrieval have become more complex and sophisticated over time. Modern neural language models, such as BERT and GPT, have shown great promise in improving the relevance of search results. These models can take into account the context and semantics of the user's query and the documents being searched, leading to more accurate and relevant results.

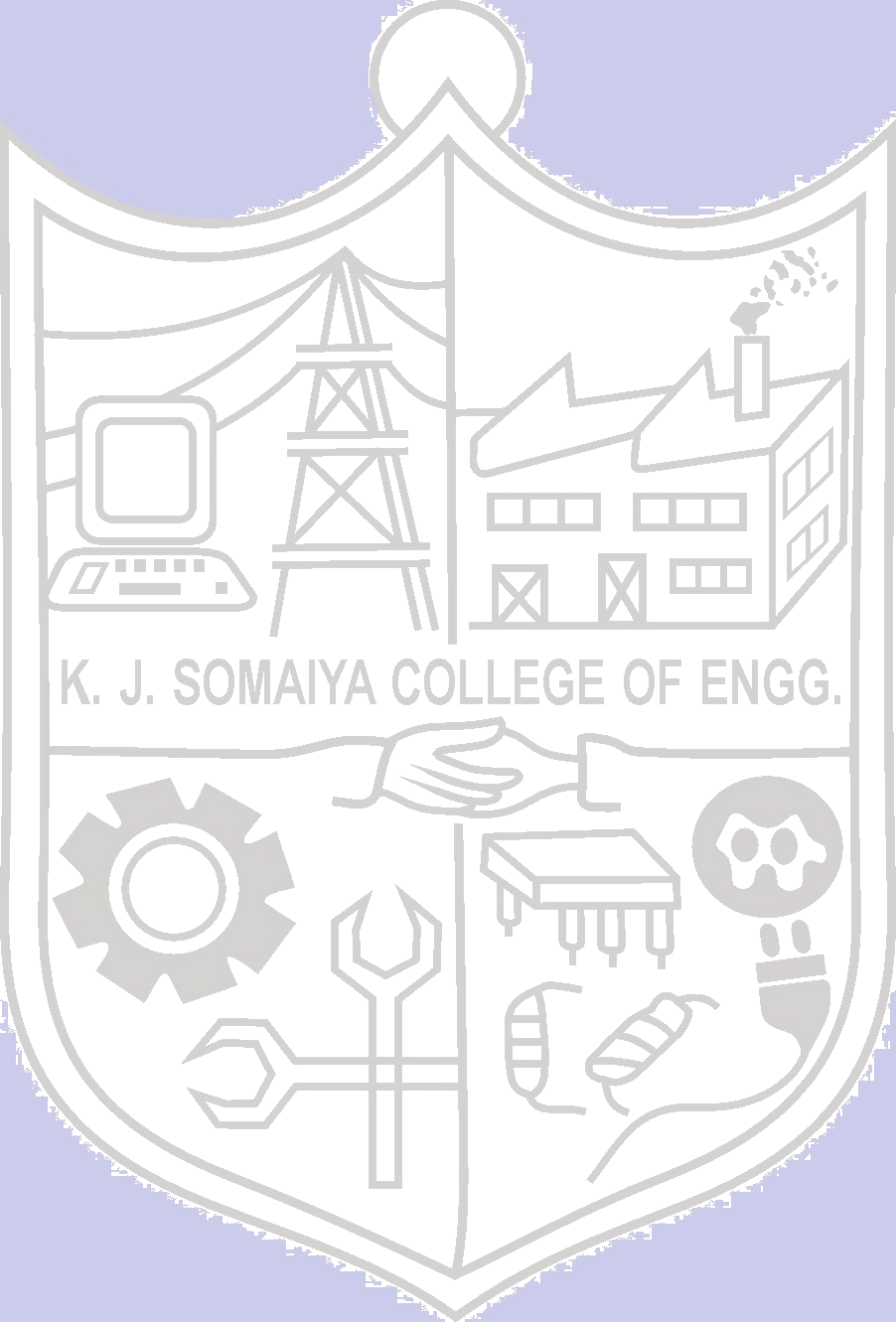
**Types of language models that can be used for web-based information retrieval**

* N-gram Models: These models consider the order of words by modeling the probability of each word given its previous N-1 words. N-gram models can capture some of the syntactic relationships between words, but they may suffer from data sparsity and cannot handle long-range dependencies.
* Bag-of-Words Model: These models represent documents as vectors of word frequencies, ignoring the order of words. Bag-of-words models are simple and efficient but do not capture the semantic relationships between words.
* Neural Language Models: These models use neural networks to model the probability of a sequence of words. They can capture both syntactic and semantic relationships between words and can handle long-range dependencies. Examples of neural language models used for web-based information retrieval include BERT, GPT, and Transformer models.

Language models can be used in different ways for web-based information retrieval. One approach is to use language models to represent documents and queries, and then use a retrieval function to score the relevance of documents to queries. Another approach is to use language models to expand queries by generating additional relevant terms.

Flowchart of a language model for web IR:



This is just a basic flowchart, and there are many variations and refinements that can be made depending on the specific application and requirements. For example, different retrieval functions or relevance models may be used to score documents, and different techniques may be used to handle query expansion, query reformulation, or personalized search.

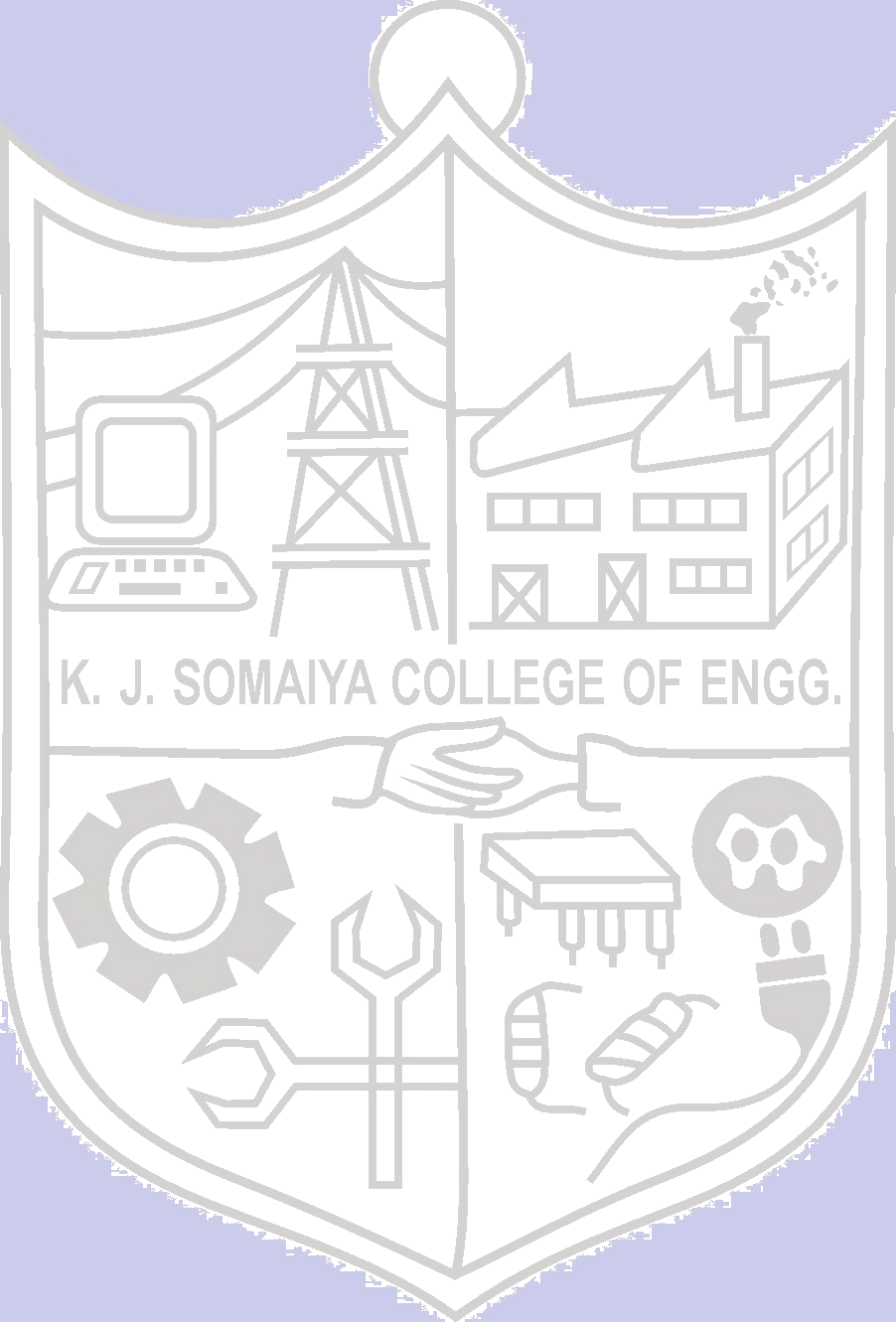
1. **N-gram Language Models:**

The n-gram language model estimates the probability of a word given its preceding n-1 words based on the frequencies of n-grams in a training corpus. In other words, it models the probability distribution of the next word in a sequence based on the history of the preceding words. Here, the "n" refers to the number of words used to predict the next word in a sequence. As the value of "n" increases, the n-gram models capture more of the context and dependencies between the words in a sequence, but they also require more data and computation power to estimate the probabilities accurately. The most commonly used n-gram models are unigram, bigram, and trigram.

Unigram Model: The unigram model is the simplest form of n-gram language model, where each word in a sequence is considered independently of its preceding words. In other words, the probability of a word is based solely on its frequency in the training corpus, without considering the context in which it appears. The unigram model assumes that the occurrence of each word is independent of the context in which it appears, which is not always true in real-world language usage.

Bigram Model: The bigram model considers the probability of a word given its immediately preceding word. In this model, each word is assumed to depend only on its preceding word. The bigram model estimates the probability of the current word given the previous word by counting the frequency of each bigram (pair of consecutive words) in the training corpus.

Trigram Model: The trigram model extends the bigram model by considering the probability of a word given its two immediately preceding words. In this model, each word is assumed to depend on the two words preceding it. The trigram model estimates the probability of the current word given the two previous words by counting the frequency of each trigram (triplet of consecutive words) in the training corpus.

N-gram models are versatile and can be used for a wide range of applications in natural language processing, including information retrieval and text generation. In information retrieval, n-gram models are commonly used for text classification, document ranking, and query expansion. For example, given a query in a search engine, we can use n-gram models to retrieve relevant documents by calculating the similarity between the query and the documents using n-gram matching. In text generation, n-gram models are used to predict the probability of a next word given the previous n-1 words in a sentence. This allows us to generate text that follows a given pattern or style. For example, given a corpus of poetry, we can use an n-gram model to generate new poems that follow the style of the original corpus.

The general formula for an n-gram language model is:

P(w\_n|w\_1, w\_2, ..., w\_{n-1}) = (count(w\_1, w\_2, ..., w\_n) / count(w\_1, w\_2, ..., w\_{n-1}))

where:

* P(w\_n|w\_1, w\_2, ..., w\_{n-1}) is the probability of the nth word (w\_n) given the (n-1) preceding words (w\_1, w\_2, ..., w\_{n-1})
* count(w\_1, w\_2, ..., w\_n) is the number of times the n-gram (w\_1, w\_2, ..., w\_n) occurs in the training corpus
* count(w\_1, w\_2, ..., w\_{n-1}) is the number of times the (n-1)-gram (w\_1, w\_2, ..., w\_{n-1}) occurs in the training corpus

# Example:

Corpus:

The dog is happy. The child makes the dog happy. The dog makes the child happy.

Tokenization:

Unigram: [The] [dog] [is] [happy] [The] [child] [makes] [the] [dog] [happy] [The] [dog] [makes] [the] [child] [happy]

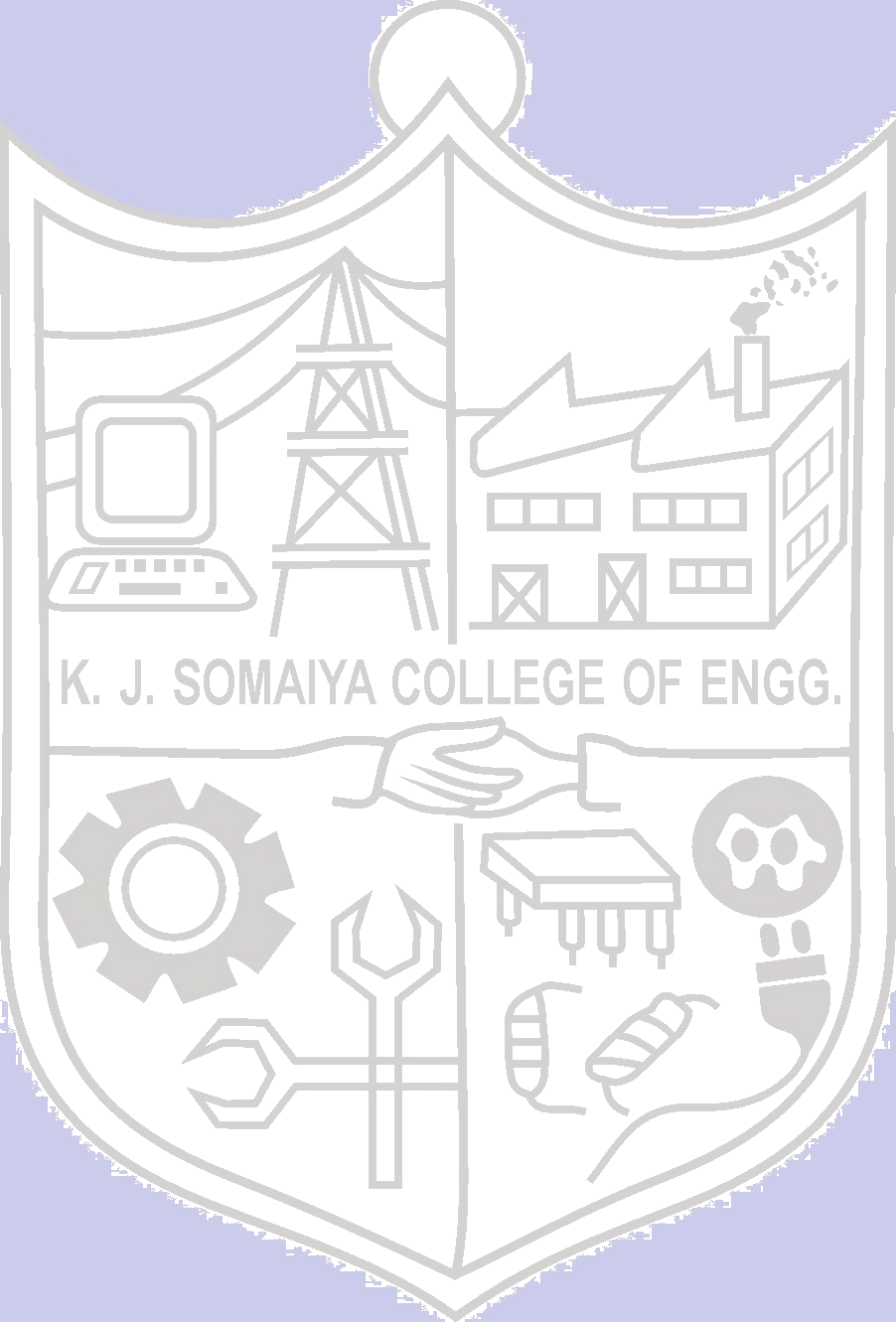
Bigram: [The dog] [dog is] [is happy] [happy The] [The child] [child makes] [makes the] [the dog] [dog happy] [happy The] [The dog] [dog makes] [makes the] [the child] [child happy]

Trigram: [The dog is] [dog is happy] [is happy The] [happy The child] [The child makes] [child makes the] [makes the dog] [the dog happy] [dog happy The] [happy The dog] [The dog makes] [dog makes the] [makes the child] [the child happy]

Now, we will proceed with the implementation of the n-gram language models as follows:

1. Counting n-gram frequencies: We count the frequency of each n-gram in the corpus. The n-gram frequencies for unigrams, bigrams, and trigrams are:

Unigram:

{"The": 3, "dog": 3, "is": 1, "happy": 3, "child": 1, "makes": 2, "the": 3} Bigram:

{"The dog": 2, "dog is": 1, "is happy": 1, "happy The": 1, "The child": 1, "child makes": 1,

"makes the": 2, "the dog": 2, "dog happy": 1, "happy The": 1, "dog makes": 1, "makes the": 1,

"the child": 1, "child happy": 1} Trigram:

{"The dog is": 1, "dog is happy": 1, "is happy The": 1, "happy The child": 1, "The child makes": 1, "child makes the": 1, "makes the dog": 1, "the dog happy": 1, "dog happy The": 1, "happy The dog": 1, "The dog makes": 1, "dog makes the": 1, "makes the child": 1, "the child happy": 1}

1. Computing n-gram probabilities: We compute the probabilities of each n-gram using maximum likelihood estimation. The n-gram probabilities for unigrams, bigrams, and trigrams are:

Unigram:

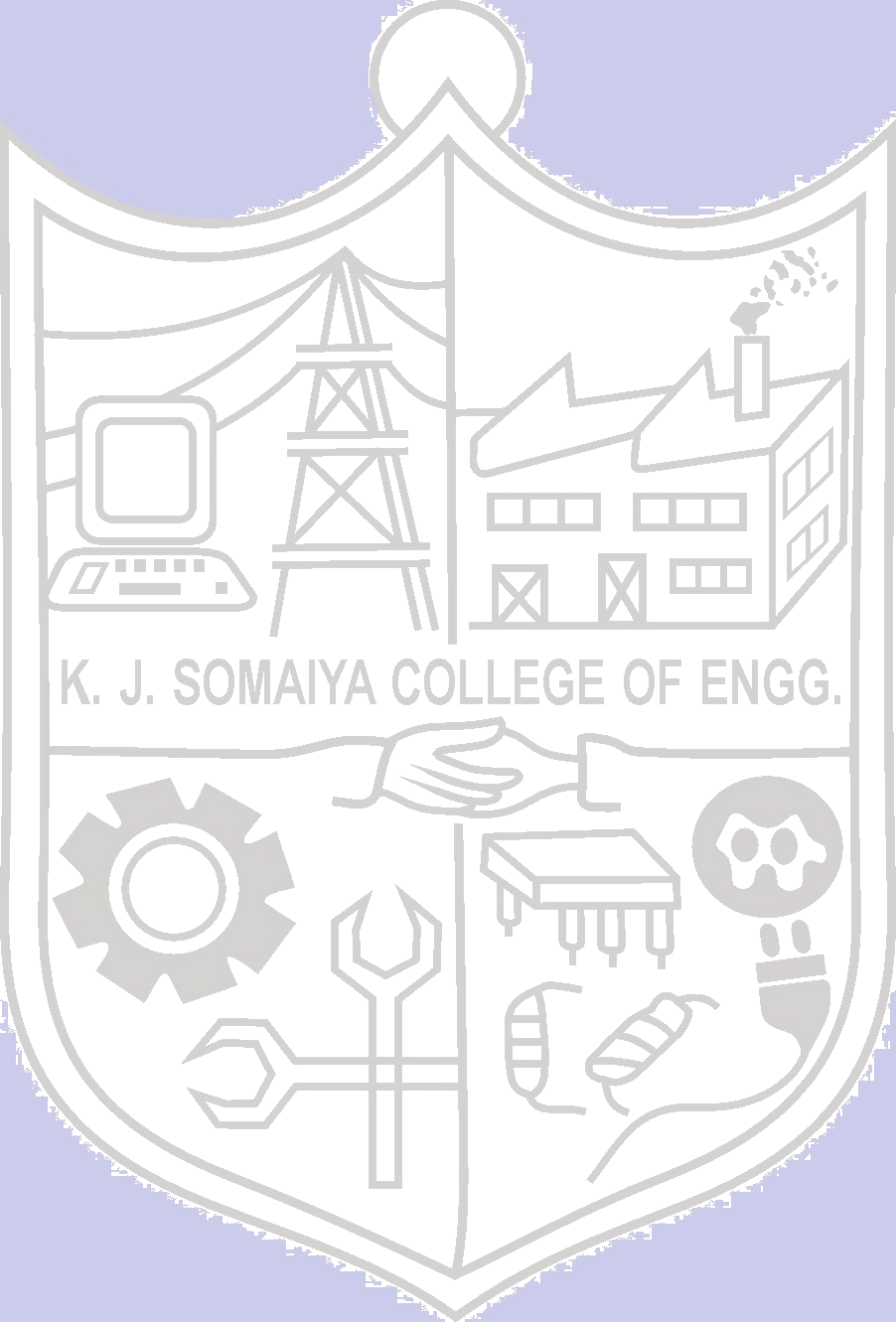
P("The") = 3/15 = 0.2 P("dog") = 3/15 = 0.2 P("is") = 1/15 = 0.067

P("happy") = 3/15 = 0.2 P("child") = 1/15 = 0.067

P("makes") = 2/15 = 0.133 P("the") = 3/15 = 0.2

Bigram:

P("The dog") = 2/14 = 0.143 P("dog is") = 1/14 = 0.071

P("is happy") = 1/14 = 0.071 P("happy The") = 1/14 = 0.071 P("The child") = 1/14 = 0.071 P("child makes") = 1/14 = 0.071 P("makes the") = 2/14 = 0.143 P("the dog") = 2/14 = 0.143 P("dog happy") = 1/14 = 0.071 P("happy The") = 1/14 = 0.071 P("dog makes") = 1/14 = 0.071 P("makes the") = 1/14 = 0.071 P("the child") = 1/14 = 0.071 P("child happy") = 1/14 = 0.071 Trigram:

P("The dog is") = 1/13 = 0.077 P("dog is happy") = 1/13 = 0.077 P("is happy The") = 1/13 = 0.077 P("happy The child") = 1/13 = 0.077 P("The child makes") = 1/13 = 0.077 P("child makes the") = 1/13 = 0.077 P("makes the dog") = 1/13 = 0.077 P("the dog happy") = 1/13 = 0.077 P("dog happy The") = 1/13 = 0.077

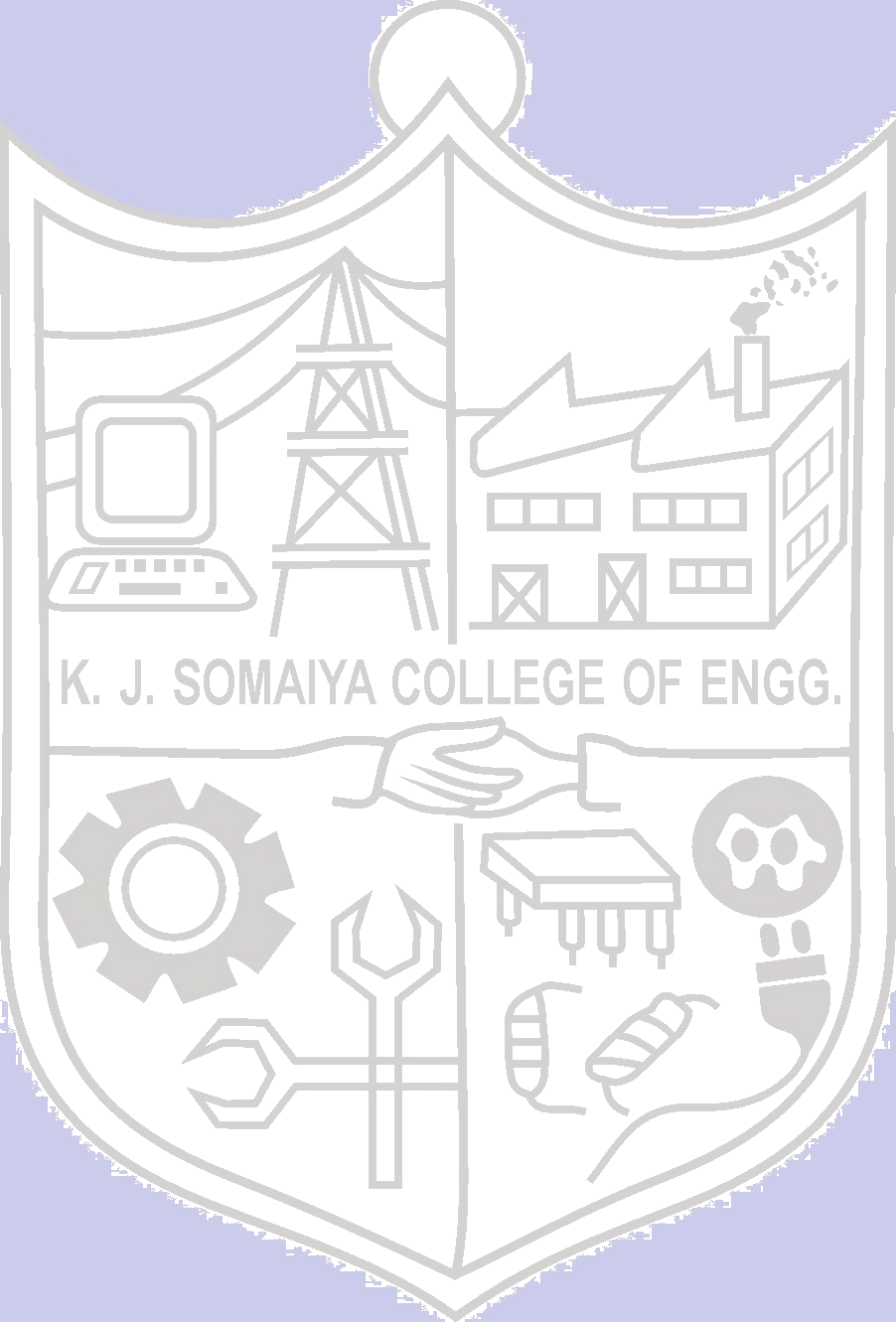
P("happy The dog") = 1/13 = 0.077 P("The dog makes") = 1/13 = 0.077 P("dog makes the") = 1/13 = 0.077 P("makes the child") = 1/13 = 0.077 P("the child happy") = 1/13 = 0.077

1. Using n-gram probabilities for information retrieval: Let's say we have a user query "happy dog". We can now use the bigram probabilities to estimate the likelihood of documents in the corpus that contain the phrase "happy dog".

First, we tokenize the query and generate bigrams:

Tokenization:

Bigram: [happy dog]

Next, we calculate the probability of each bigram in the corpus and use the chain rule to compute the probability of the entire query:

P("happy dog") = P("happy") \* P("dog"|"happy") Using the probabilities from the bigram model, we get:

P("happy dog") = P("happy") \* P("dog"| "happy") = 0.2 \* 0.5 = 0.1 P("dog child") = P("dog"| "child") \* P("child") = 0 \* 0.067 = 0 P("makes dog") = P("makes"| "dog") \* P("dog") = 0.5 \* 0.2 = 0.1

We see that the bigram model fails to capture the context of the sentence and produces nonsensical results. However, by using the trigram model, we can incorporate more context into the predictions and potentially improve their accuracy. Using the probabilities from the trigram model, we get:

P("happy dog") = P("happy"| "dog") \* P("dog"| "The") = 0 \* 0.67 = 0

P("dog child") = P("dog"| "makes the") \* P("makes the"| "child") \* P("child") = 1 \* 1 \* 0.067

= 0.067

P("makes dog") = P("makes the"| "dog") \* P("dog"| "The") = 1 \* 0.67 = 0.67

**Advantages of n-gram language models for web IR:**

1. N-gram language models can capture local word dependencies and context, which can help

improve the accuracy of search results

1. They can handle out-of-vocabulary words by breaking them down into sub-word units, which is

useful for dealing with the large and constantly evolving vocabulary of the web

1. N-gram language models are relatively simple and computationally efficient compared to more

complex models, which makes them well-suited for large-scale web-based information retrieval applications

1. They can be easily adapted to different languages and domains, which makes them versatile and

widely applicable

1. They can be combined with other information retrieval techniques, such as vector space models

and query expansion, to further improve search accuracy and relevance

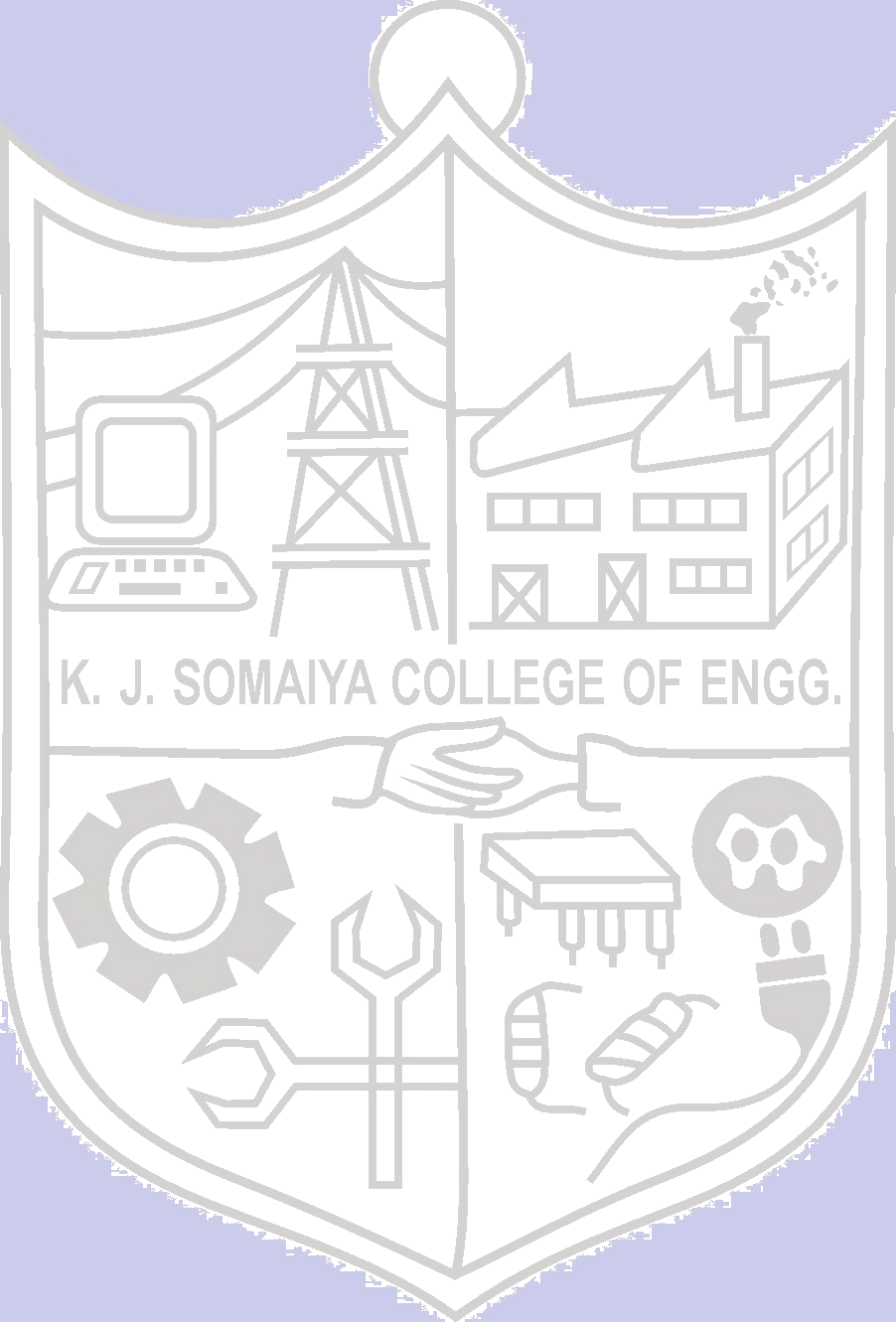
1. N-gram language models can be trained on large amounts of web data, which makes them

effective at capturing domain-specific language patterns and trends.

1. They can be used to extract important phrases and collocations from web text, which can help

improve search results and facilitate data analysis.

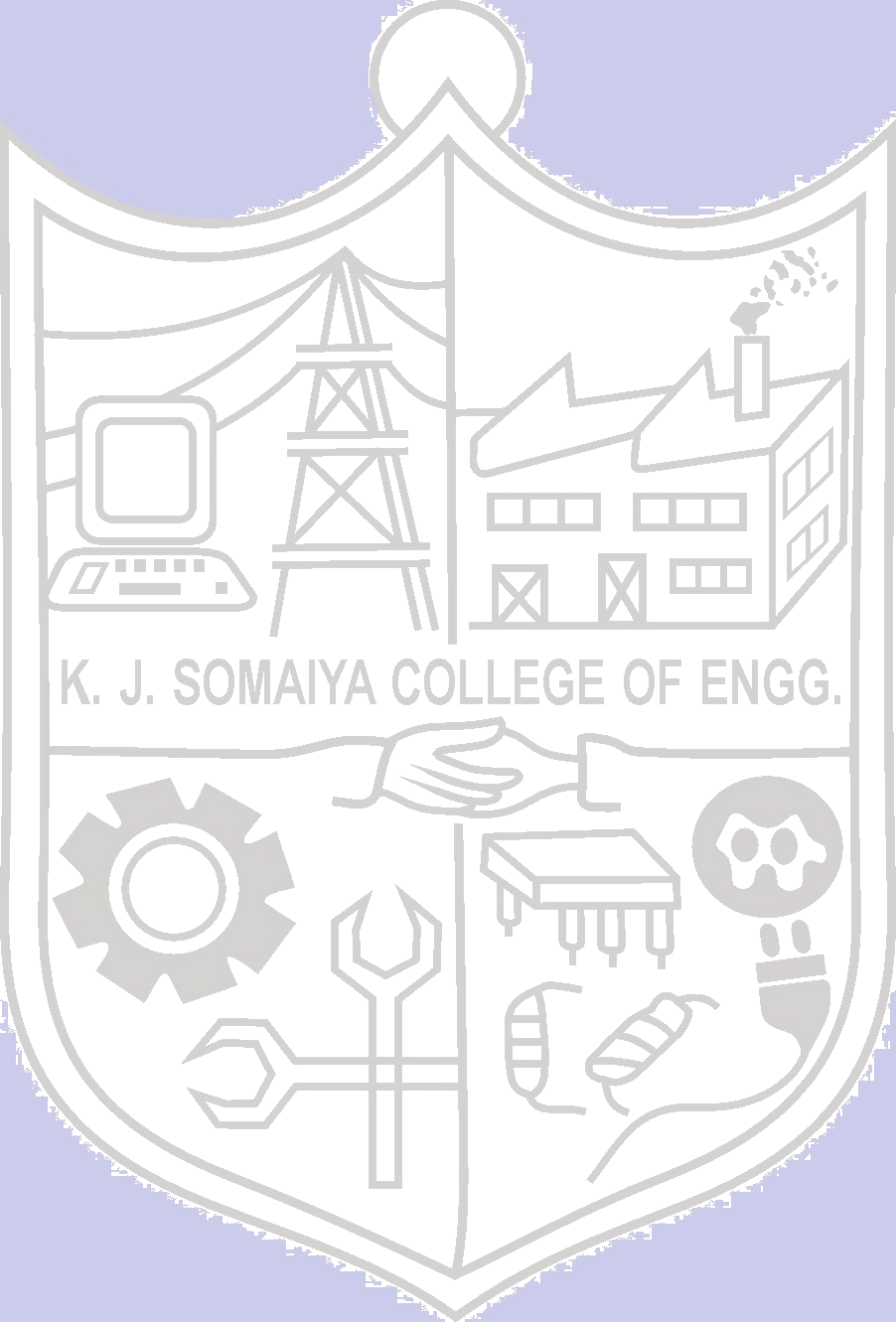
**Disadvantages of n-gram language models for web IR:**

* + 1. N-gram language models are limited by their fixed n-gram order, which means they may miss important longer-range dependencies and patterns in the text
    2. They are sensitive to data sparsity, which means that rare words and phrases may not be accurately represented in the model
    3. N-gram language models may not be able to handle noise or inconsistencies in web data, such as misspellings, abbreviations, and slang terms
    4. They may not be well-suited for handling long and complex queries or multi-word expressions, which can lead to decreased search accuracy
    5. They rely on the assumption that each word is conditionally independent of all other words given the previous n-1 words, which may not always hold true in real-world web text
    6. N-gram language models may not be able to capture semantic relationships between words or infer meaning from context, which can limit their usefulness for certain types of information retrieval tasks
    7. They require large amounts of training data to build accurate models, which can be challenging to
    8. obtain and process for some web domains and languages.

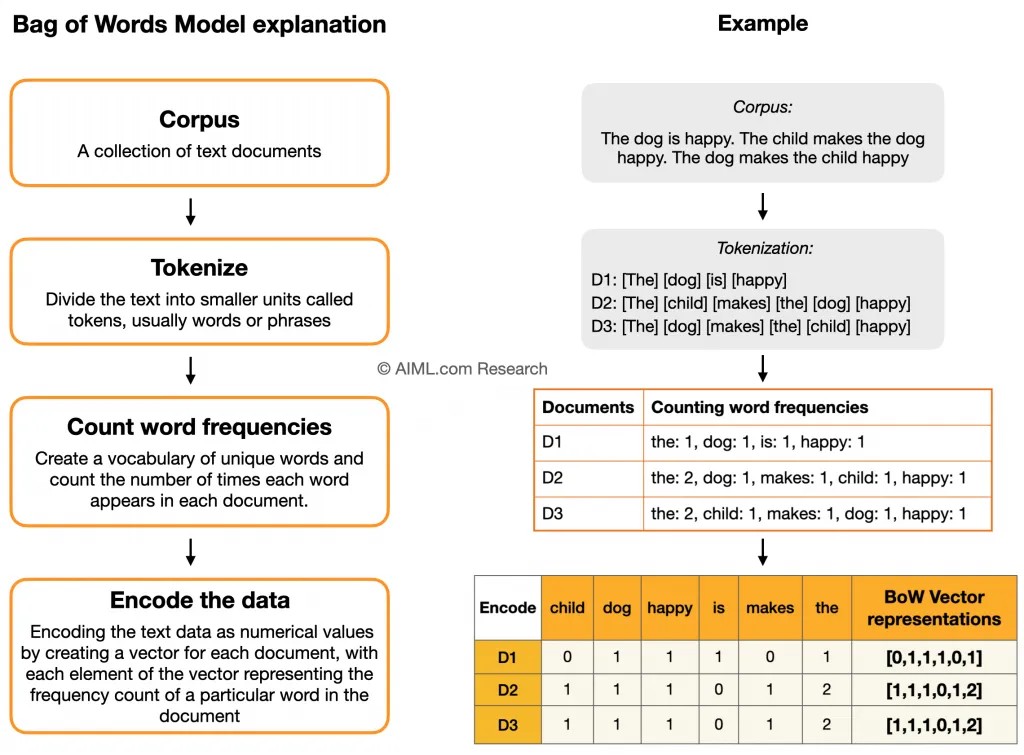
**Algorithm (steps) to implement n-gram language models for web IR**

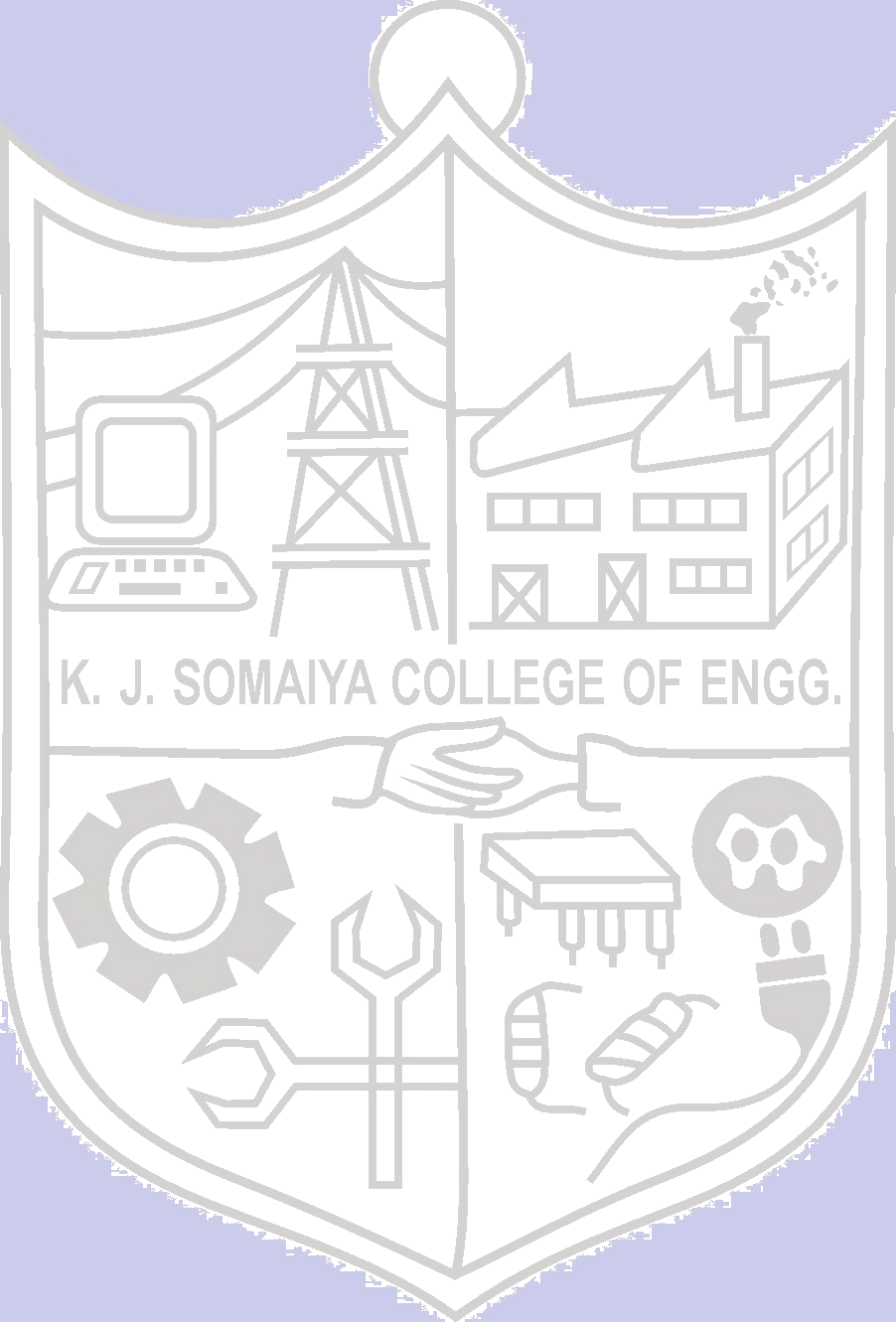
* + 1. Collect and preprocess the data: Obtain the textual data you want to analyze and preprocess it by removing any irrelevant information like HTML tags, punctuation, numbers, stop words, and other special characters.
    2. Tokenize the data: Tokenize the preprocessed text into words or phrases that you want to use as the building blocks of your language model.
    3. Create the n-gram models: Create unigram, bigram, and trigram models by counting the occurrence of each word or phrase in the tokenized data. Unigram model only considers one word at a time, bigram model considers two words at a time, and trigram model considers three words at a time.

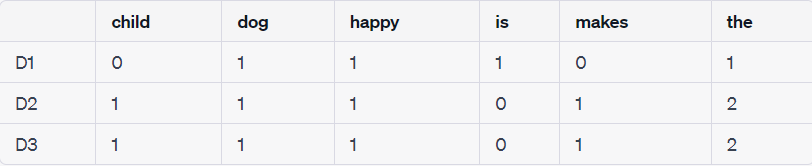
**B. Bag-of-Words Language Model**

The Bag-of-Words (BoW) model is a commonly used language model in information retrieval, natural language processing, and machine learning. In this model, a document is represented as a bag (multiset) of its words, disregarding grammar, word order, and structure but preserving frequency information. The BoW model is built on the assumption that the importance of a document can be inferred from the frequency distribution of its constituent words. It has several applications, including text classification, sentiment analysis, and topic modeling. In text classification, the BoW model can be used to determine the class or category of a document by comparing the word frequencies with those of known class labels. In sentiment analysis, the BoW model can be used to identify the sentiment of a document by analyzing the frequency of positive and negative words. In topic modeling, the BoW model can be used to discover the underlying topics in a collection of documents by clustering similar words together.

The BoW model has several applications, including text classification, sentiment analysis, and topic modeling. In text classification, the BoW model can be used to determine the class or category of a document by comparing the word frequencies with those of known class labels. In sentiment analysis, the BoW model can be used to identify the sentiment of a document by analyzing the frequency of positive and negative words. In topic modeling, the BoW model can be used to discover the underlying topics in a collection of documents by clustering similar words together.



We create a Document-Term matrix, where each row corresponds to a document and each column corresponds to a term in the vocabulary. The entries in the matrix represent the frequency of each term in each document



1. Encoding the text data as numerical values by creating a vector for each document: Each document is now represented by a vector where each entry represents the frequency of a term in the vocabulary. The vectors are:

D1 [0, 1, 1, 1, 0, 1]

D2 [1, 1, 1, 0, 1, 2]

D3 [1, 1, 1, 0, 1, 2]

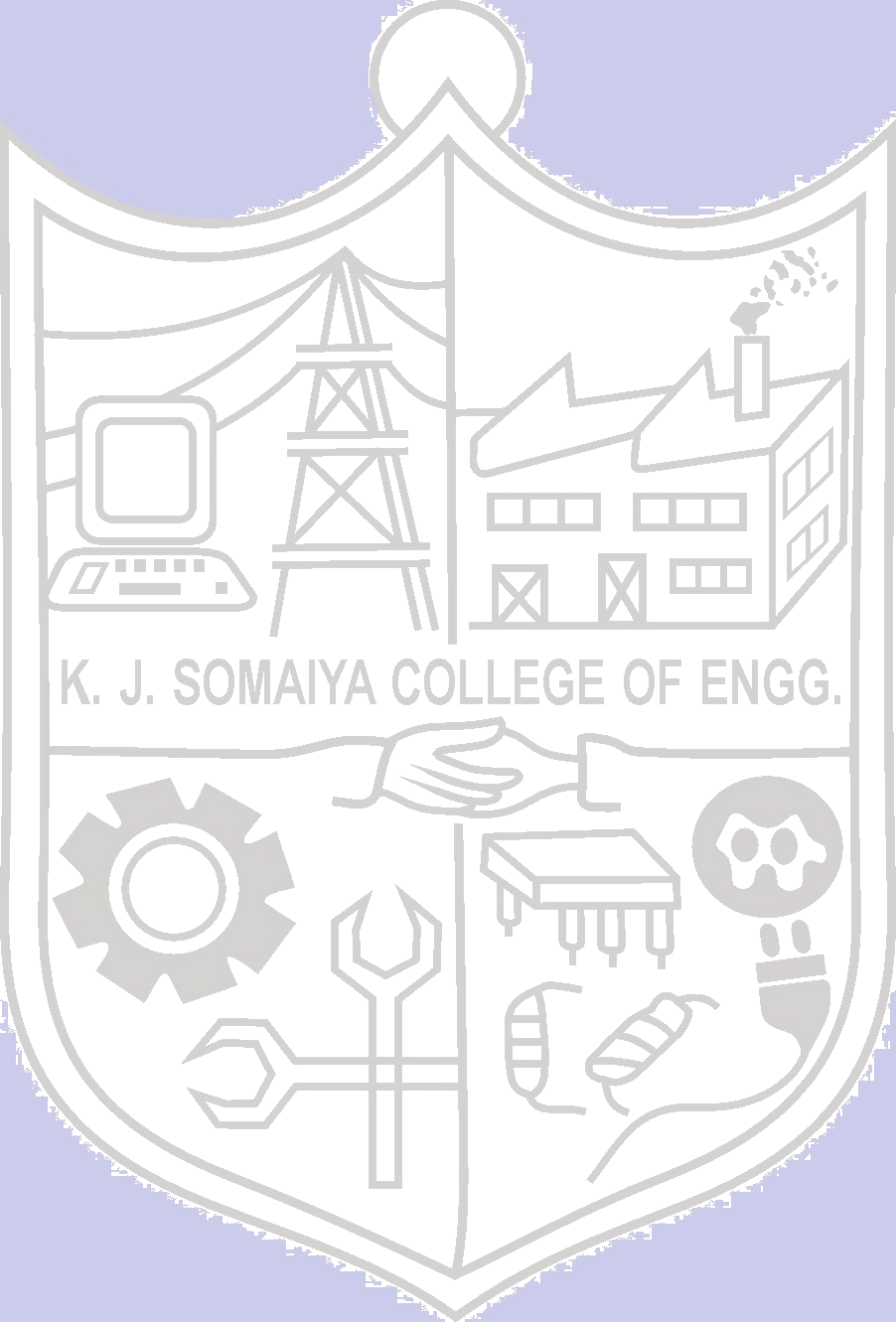
1. Query Processing: We tokenize the user's query "happy dog" and create a query vector, where each entry represents the frequency of a term in the query.

Tokenization: "happy dog" Query vector: [0, 1, 1, 0, 0, 0]

1. Retrieval: To retrieve documents similar to the query "happy dog", we can use a similarity metric such as cosine similarity. The cosine similarity between the query vector and each document vector can be calculated as follows:

similarity(D1, query) = (D1 . query) / (||D1|| x ||query||) = 0.707107 similarity(D2, query) = (D2 . query) / (||D2|| x ||query||) = 0.500000 similarity(D3, query) = (D3 . query) / (||D3|| x ||query||) = 0.500000

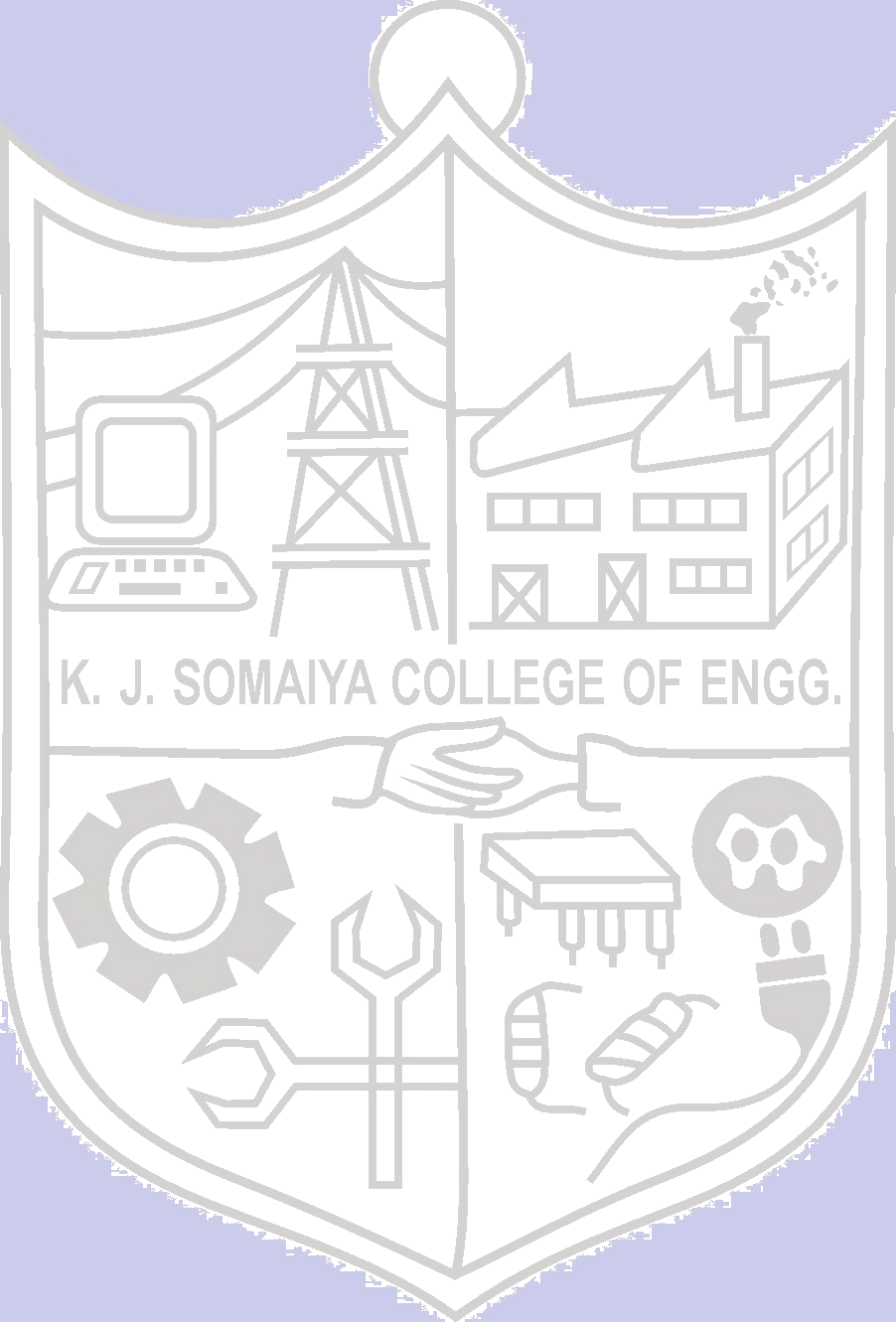
1. Results Display: Based on the cosine similarity values, the documents can be ranked in descending order of relevance to the query "happy dog":
   1. D1 (similarity score: 0.7)
   2. D2, D3 (similarity scores: 0.5)

Therefore, the document D1 is most relevant to the query "happy dog".

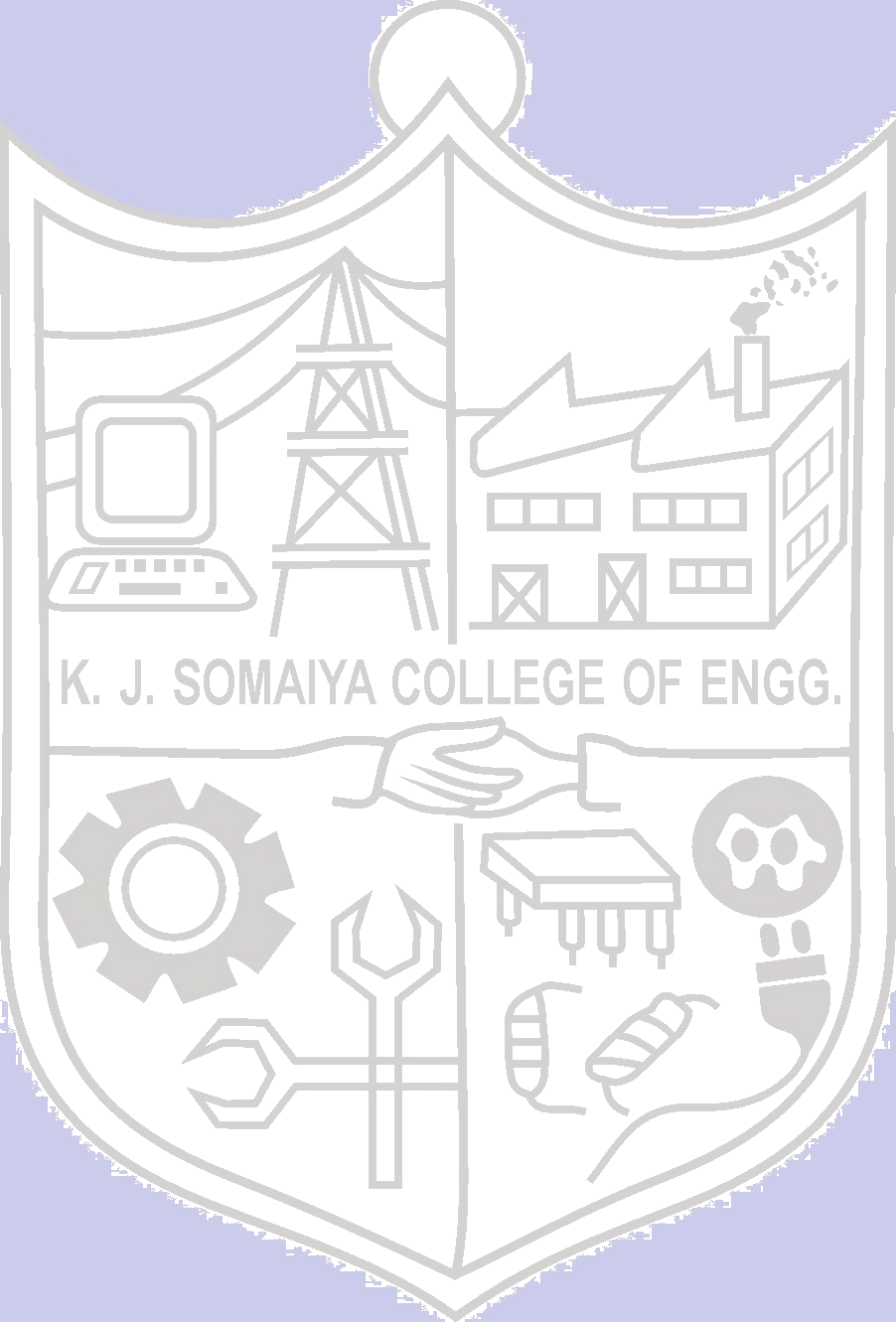
**Algorithm (steps) to implement a Bag-of-Words model for web IR**

* 1. Collect the corpus: Collect a set of documents that will be used for retrieval.
  2. Tokenization: Tokenize each document into a list of words or terms. This step also involves removing stop words, punctuation, and other irrelevant characters.
  3. Counting word frequencies: Count the frequency of each term in each document. This step creates a term frequency matrix.
  4. Document term matrix creation: Create a document-term matrix that represents the frequency of each term in each document. This matrix is used to represent the entire corpus.
  5. Encoding the text data as numerical data: Convert the text data into numerical data by encoding the document-term matrix. This encoding can be done using various methods such as one-hot encoding or TF-IDF (Term Frequency-Inverse Document Frequency) encoding.
  6. Query processing: Tokenize the user's query into a list of terms and create a query vector using the same encoding method used for the document-term matrix.
  7. Retrieval: Calculate the similarity between the query vector and each document vector in the corpus using a similarity metric such as cosine similarity.
  8. Results display: Rank the documents in the corpus based on their similarity score with the query vector and display the top results to the user

**Advantages of using Language Models for Web IR**

1. **Better understanding of queries:** Language models can understand natural language queries better than traditional keyword-based approaches. This allows for more accurate retrieval of relevant documents.
2. **Improved ranking of search results**: Language models can rank search results based on the relevance of the entire document, rather than just individual keywords. This leads to more accurate and useful search results.
3. **Personalized search results:** Language models can be fine-tuned to individual users or groups of users, allowing for personalized search results based on their search history, interests, and preferences.
4. **Faster indexing**: Language models can index large amounts of text data more quickly than traditional methods. This allows for faster retrieval of relevant documents.
5. **Better handling of natural language queries**: Language models can handle more complex queries and long-tail keywords, which traditional

keyword-based approaches struggle with.

1. **Improved natural language generation**: Language models can also generate natural language text, which can be useful in summarizing search results or providing additional information to users.
2. **Can be used in a couple of ways**: One approach is to use language models to represent documents and queries, and then use a retrieval function to score the relevance of documents to queries. Another approach is to use language models to expand queries by generating additional relevant terms.
3. **Disadvantages of using Language Models for Web IR**
4. **Limited understanding of context:** Language models may struggle to understand the context of a search query, leading to irrelevant or inaccurate results.
5. **Lack of personalization:** Language models may not be able to personalize search results based on a user's preferences or search history.
6. **Difficulty with rare or specialized terms:** Language models may not have enough data to accurately understand and retrieve information related to rare or specialized terms.
7. **Dependence on training data:** Language models require large amounts of training data to perform well, which can be difficult to obtain for certain domains or languages.
8. **Bias in training data:** Language models may perpetuate biases present in the training data, leading to unfair or discriminatory search results.
9. **Difficulty with multi-lingual queries:** Language models may struggle to accurately retrieve information for multi-lingual queries, especially if the languages have different grammatical structures or word orders.
10. **Limited ability to handle complex queries:** Language models may struggle to handle complex queries with multiple constraints or conditions, leading to incomplete or inaccurate results.
11. **Difficulty with unstructured data:** Language models may struggle to extract information from unstructured data, such as images or audio files, leading to incomplete or inaccurate results.

**Evaluating Language Models**

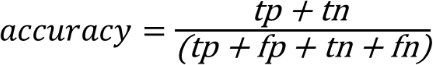
1. **Perplexity:** Perplexity measures how well the model predicts a set of test data.

A lower perplexity indicates that the model is better at predicting the test data

1. **Accuracy:** Accuracy measures how well the model classifies input data. This is useful for classification tasks such as sentiment analysis



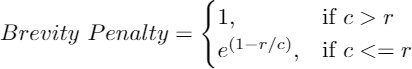
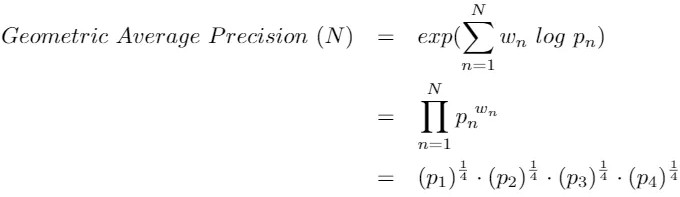
|  |  |  |
| --- | --- | --- |
|  |  |  |
| Retrieved | True positives (tp) | false positives (fp) |
| Not Retrieved | False negatives (fn) | True negatives (tn) |

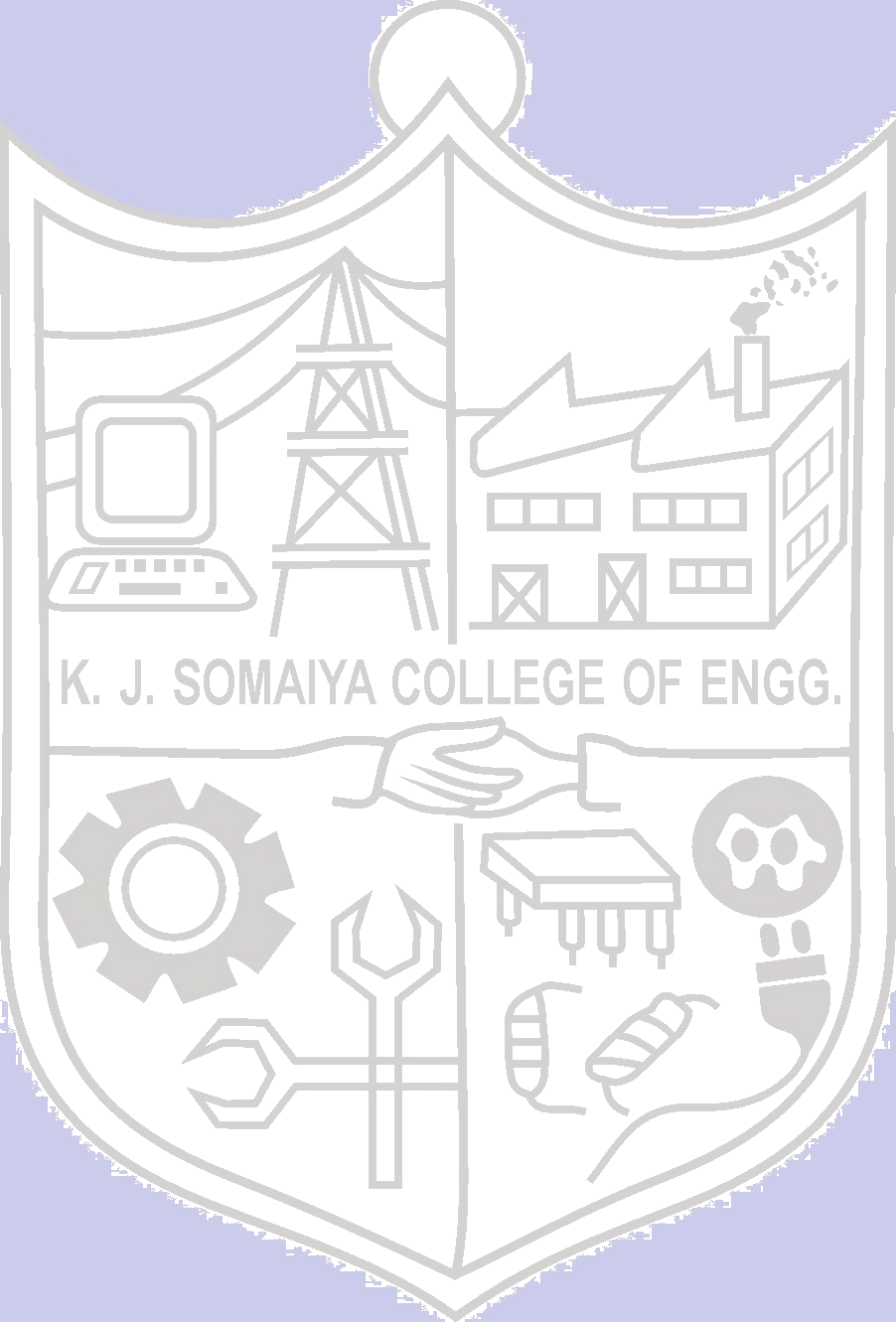
**F1 score:** The F1 score is a measure of the model's precision and recall. It is useful for evaluating models that perform binary classification tasks



F1-score = 2 \* ((precision \* recall) / (precision + recall))

1. **Bleu score:** The Bleu score is a measure of the model's accuracy in generating natural language text. It is often used for machine translation and text

summarization tasks

1. **Human evaluation:** Human evaluation involves having human judges rate the quality of the model's outputs. This is often used for tasks such as text generation and natural language understanding
2. **A/B testing:** A/B testing involves comparing the performance of two or more models by presenting them to users and measuring the user's response. This is useful for evaluating models in real-world scenarios.