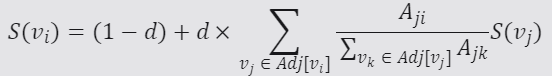
# LEXRANK ALGORITHM

LexRank is an ***unsupervised approach to text summarization based on graph-based centrality scoring of sentences.*** The key idea is to treat sentences as vertices in a graph and connect them based on their similarity. Here's a detailed explanation of how LexRank works:

1. **Text Pre-processing**:
   * The input text is first split into individual sentences using a sentence tokenizer.
2. **Sentence Representation**:
   * Each sentence is converted into a vector representation. Typically, this is done using ***word embeddings,*** which transform each word into a high-dimensional space.
3. **Similarity Matrix Construction**:
   * A similarity matrix is constructed where each element (i, j) in the matrix represents the cosine similarity between the embeddings of the i-th and j-th sentence.
   * Only similarities that exceed a predefined threshold are considered to ensure the graph remains sparse, reducing noise in similarity calculations.
4. **Graph Construction**:
   * A graph is constructed with sentences as vertices. An edge between any two vertices is added based on the similarity score (from the similarity matrix). Weights on the edges correspond to these similarity scores.
5. **Rank Calculation**:
   * The centrality of each vertex (sentence) is calculated using an algorithm similar to Google's PageRank. The idea is that a sentence that is highly similar to many other sentences will likely be more central and thus more important.
   * The basic iterative formula for LexRank is:



Here, ( S(v\_i) ) is the score of vertex ( i ), ( A ) is the adjacency matrix (i.e., the similarity matrix), and ( d ) is a damping factor analogous to the damping factor in PageRank.

1. **Summary Extraction**:
   * Sentences are then ranked based on their scores, and the top-ranking sentences are selected to form the summary. The number of sentences selected can be predefined or based on the desired summary length.

# CODE ANALYSIS

The provided code is a Python implementation of the LexRank algorithm. Below, I break down its main components:

1. **Import Statements**:
   * Libraries like nltk, numpy, networkx, gensim, and rouge are imported to handle natural language processing tasks, matrix operations, graph theory aspects, word embeddings, and evaluation metrics, respectively.
2. **Cosine Similarity Function (cosine\_similarity)**:
   * Computes the cosine similarity between two vectors, essential for determining how similar two sentences are.
3. **Build Similarity Matrix Function (build\_similarity\_matrix)**:
   * Processes a list of sentences to create their embeddings using a given model (in this case, pre-trained Word2Vec embeddings from Google News).
   * Calculates pairwise cosine similarities between all sentences, establishing a similarity matrix.
   * Sentences are represented by averaging the embeddings of their constituent words.
4. **LexRank Function (lexrank)**:
   * Utilizes the similarity matrix to compute centrality scores for each sentence using the iterative PageRank-like formula.
   * Employs a damping factor and iterates until scores converge or a maximum number of iterations is reached.
5. **Generate Summary Function (generate\_summary)**:
   * Loads the Word2Vec model, tokenizes the input text into sentences, and calculates LexRank scores.
   * Extracts the top N sentences based on these scores to generate the summary.
6. **Evaluate Summary Function (evaluate\_summary)**:
   * Uses the Rouge metric to evaluate the quality of the generated summary against a reference text, providing scores for precision, recall, and F1-Score.
7. **Main Execution Block**:
   * This segment drives the application, allowing users to input text and receive summaries interactively until the user terminates the process.

This implementation provides a comprehensive framework for understanding and executing LexRank-based text summarization, complete with evaluation metrics to assess performance.