Data Analytics And Visualization

ASSIGNMENT-2

Wikipedia Graph Analysis

Ishika Jaiswal (11940510) , Anmol Singhal (11940140)

Part A:

We have labeled 20 links each and written the article links in the Spreadsheet provided.

Screenshots:

1. *Links provided by Anmol (11940140)*

11940140	Arithmetic Mean	1	https://en.wikipedia.org/wiki/Arithmetic_mean
11940140	Bayes Theorem	2	https://en.wikipedia.org/wiki/Bayes%27_theorem
11940140	Complex Numbers	1	https://en.wikipedia.org/wiki/Complex_number
11940140	Coordinate System	1	https://en.wikipedia.org/wiki/Coordinate_system
11940140	Cross Product	1	https://en.wikipedia.org/wiki/Cross_product
11940140	Differentiable Function	2	https://en.m.wikipedia.org/wiki/Differentiable_function
11940140	Harmonic Mean	1	https://en.wikipedia.org/wiki/Harmonic_mean
11940140	Integration By Parts	2	https://en.wikipedia.org/wiki/Integration_by_parts
11940140	Inverse Trignometric Functions	2	https://en.wikipedia.org/wiki/Inverse_trigonometric_functions
11940140	Line (Geometry)	1	https://en.wikipedia.org/wiki/Line_(geometry)
11940140	Mathematical Reasoning	0	https://en.wikipedia.org/wiki/Mathematical_logic
11940140	Parameteric Equation	2	https://en.wikipedia.org/wiki/Parametric_equation
11940140	Product Rule	1	https://en.wikipedia.org/wiki/Product_rule
11940140	Quadratic Equation	1	https://en.wikipedia.org/wiki/Quadratic_equation
11940140	Quotient rule	2	https://en.wikipedia.org/wiki/Quotient_rule
11940140	Sample Space	1	https://en.wikipedia.org/wiki/Sample_space
11940140	Section Formula	1	https://en.wikipedia.org/wiki/Section_formula
11940140	Tangent Half Angle Formula	3	https://en.wikipedia.org/wiki/Tangent_half-angle_formula
11940140	Trignometric identities	3	https://en.wikipedia.org/wiki/List_of_trigonometric_identities
11940140	Weighted Arithmetic Mean	1	https://en.wikipedia.org/wiki/Weighted_arithmetic_mean

2. *Links provided by Ishika (11940510)*

11940510	Calculus	0	https://en.wikipedia.org/wiki/Calculus
11940510	Combinations	2	https://en.wikipedia.org/wiki/Combination
11940510	Differentiation of trigonometric functions	2	https://en.wikipedia.org/wiki/Differentiation_of_trigonometric_functions
11940510	Differentiation rule	1	https://en.wikipedia.org/wiki/Differentiation_rules
11940510	Lebiniz Rule	3	https://en.wikipedia.org/wiki/Leibniz_integral_rule
11940510	limit of a function	0	https://en.wikipedia.org/wiki/Limit_of_a_function
11940510	Locus	1	https://en.wikipedia.org/wiki/Locus_(mathematics)
11940510	Logarithm	1	https://en.wikipedia.org/wiki/Logarithm
11940510	Logarithmic Derivative	1	https://en.wikipedia.org/wiki/Logarithmic_derivative
11940510	Logarithmic Differentiation	2	https://en.wikipedia.org/wiki/Logarithmic_differentiation
11940510	Napier's Constant	2	https://en.wikipedia.org/wiki/E_(mathematical_constant)
11940510	polynomial	1	https://en.wikipedia.org/wiki/Polynomial
11940510	Rolle's Theorem	1	https://en.wikipedia.org/wiki/Rolle%27s_theorem
11940510	Roots of Unity	2	https://en.wikipedia.org/wiki/Root_of_unity
11940510	Secant	1	https://en.wikipedia.org/wiki/Secant_line
11940510	Tangent	1	https://en.wikipedia.org/wiki/Tangent
11940510	Triangular Inequality	1	https://en.wikipedia.org/wiki/Triangle_inequality
11940510	Trignometric functions	1	https://en.wikipedia.org/wiki/Trigonometric_functions

PART B:

To build the Wikipedia Graph, we have traversed the wikipedia website in a BFS Fashion. This allows us to stay close to the root topic. To get children of each node, we scraped all the anchor tags from the links. We filtered the anchor tags based on which is useful or not. We have used multiple root nodes to build the graph. We also saved our graph in a neo4j database.

To calculate the node attributes, we made http requests to the links and scraped the data. From the data, we applied NLP to extract keywords and other NLP features.

The code is clearly written and well explained using comments in the notebook provided.

We had to limit the size of our graph as in some later steps in the assignment, a larger graph was causing the RAM Crash on our collab. (This was happening in the SVD Decomposition Step.)

Since we had to limit the size of our graph, we selected 50 root nodes, and set the max size of our graph to 500 nodes. These parameters can be changed in the code easily.

The diameter of our graph was found to be 10.

We also limited the number of new children per node, while traversing in bfs fashion to 5. This allowed more and more nodes to be explored for further links rather than only 1 node fetching all the new nodes in the graph.

NLP Features:

For NLP, we have used NLTK and Spacy libraries. Both of them can be used to extract keywords. Spacy is found to be more efficient in terms of time.

For every node, we have calculated Two Types of NLP related features:

1. **<u>TF-IDF vectors for our nodes:</u>** A binary vector can be implemented for each node based on which word is present or not. But this gives us very limited information. Other information like frequency, importance can be implemented using TF-IDF. Our TF-IDF vector is basically the binary vector but in place of 1's, we replace 1 by the TF-IDF score of the keyword. This part is completely manually implemented. To calculate TF, we have used the Augmented Variation of Term Frequency. This has been explained in the code using comments.

2. **<u>Weighted Average NLP Embeddings:</u>**
    a. **<u>Manually Implemented Embeddings:</u>** In this part, we have manually implemented the NLP Embeddings for our words using the Co-Occurrence matrix. First, we implemented the Co-Occurrence Matrix. Then, we applied SVD Decomposition on the Co-Occurrence Matrix using the scipy library. From this, we had to decrease the size of embeddings. To calculate the ideal size of embeddings, we use binary search to find a size which helps us maintain at least 99% approximation factor. From this, we calculated the nlp-embeddings for words. Now, to calculate the average nlp embeddings for nodes, we took a **tf-idf weighted** average of the embeddings of all keywords for a node.
    b. **<u>Word2Vec Embeddings:</u>** In this part, we used the Word2Vec Library to calculate the NLP Embeddings for words. From these embeddings, we again took a tf-idf weighted average for all keywords corresponding to a node.

For our nodes, we have also calculated the **node2vec** embeddings which are dependent not on the NLP Features, but the structure of the graph and random walks in our graph.

This part is well-commented and clearly explained in the notebook provided.

PART C:

Along with the above mentioned features for our nodes, we have also calculated features based on several centrality metrics and clustering coefficient.

The following concepts have been used: Degree Centrality, Closeness Centrality, Betweenness Centrality, Page Rank and Clustering Coefficient.

Manual Implementations have been done and implementation using the networkX library is also shown.

This part is well-commented and clearly explained in the notebook provided.

PART D:

We have developed 3 node classification models

1. **Label Propagation - Manual Implementation:** We have manually implemented the label propagation which propagates the labels in a BFS Fashion. The algorithm goes on for various iterations. The nodes for which we have ties, try to acquire labels till the last iteration. In the last iteration, we do Tie-Breaking based on similarities of a node with its neighbors.
2. **Label Propagation - Using Sklearn Library:** We have also implemented the Label Propagation Algorithm using the Sklearn Library as taught in the Tutorial.
3. **Artificial Neural Network:** We had some features corresponding to every node, and for our root-nodes we had the difficulty labels. Based on this, we trained our ANN and then used it to label the remaining nodes in the graph.

This part is well-commented and clearly explained in the notebook provided.

PART E:

We have implemented 2 types of Article Ordering

1. A person comes and says, "Hey, I have read these links and here is my labeling for these, suggest me some links to read after these"

In that case, we have implemented a recommendation system as taught in class. We had the data of all the students and their labels. So, we used that to find the most similar persons to our user and then used their ratings to order articles for the user to read.

Note: This is like a movie recommendation system

2. User comes and says, "Hey, I want to read this link. In what order should I study for this?"

For this, we need to find relevant other links, and print in an order such that the user can read the links in that order.

For this purpose, we need an idea of prerequisite. i.e. for a link, we need to be able to find prerequisites. But in our graph, edges do not necessarily mean that one link is prerequisite to the other.

So, we will simply order the relevant topics in order of difficulty.

Note that, here we will use the difficulty labeling that we did for all nodes using ANN.

To check relevancy, all topics within a fixed distance in our wiki graph will be considered relevant for a topic.

This means, if link A can be reached from link B in steps less than the fixed distance then A is relevant for B and B is also relevant for A.

We have also printed the BFS traversal of our created graph.

This part is well-commented and clearly explained in the notebook provided.