SAN JOSÉ STATE
UNIVERSITY

Group Project

Clustering Wikipedia Articles Through Topic Modeling

CMPE – 256
Large Scale Analytics

Fall 2019

Group-4

Team members:

Swayam Swaroop Mishra (013725595)
Aashay Mokadam (012724998)
Karthik Munipalle (013854867)

Submitted To:

Prof. Gheorghi Guzun

Table of Contents

Introduction

Wikipedia is a large online encyclopedia with articles covering almost every documented topic in the world. Clustering large set of documents based on the topic they talk about is a useful task to understand the extent of topics and the knowledge base of the articles available on wikipedia. Clustering helps recommend articles based on user interests in topics, summarize large sets of information to reduce the storage overhead of large databases.

Topic modelling is human-like clustering approach of text documents to get extract useful information and provides meaningful summary of the documents compared to other text clustering approaches. Good topic models help process large sources of information in the form of text abundantly available on the internet and help in identifying duplicates, aggregating the knowledge on identical topics there by improving the overall understanding of the information. For example, gathering all the articles based on cancer prevention might help machines and humans identify patterns in the cause of cancer, ideally leading to prevention of such causes. Information pooling becomes easier with topic based clustering.

We want to cluster the articles based on LDA topics and perform graph analysis to analyze the wikipedia articles to identify the best strategy to apply to solve the problem. While one problem is based on text mining, the other method uses a web-mining approach.

Dataset :

We are using the wikipedia articles text data to perform topic modelling using LDA and use the metadata to generate the graph structure of the wikipedia using pyspark script and gensim LDA model. The data is openly available on the internet in https://dumps.wikimedia.org/enwiki/ for english articles. The website contains article dumps for all the wikipedia articles.

LDA Clustering of Wikipedia Articles:

https://dumps.wikimedia.org/enwiki/20191101/enwiki-20191101-pages-articles.xml.bz2

The LDA clustering dataset is an xml file that contains all the raw text from various wikipedia articles as of November. The size of the compressed file is 16GB, after decompression the size is ~60GB**.**

Graph Analysis of Wikipedia Page Links:

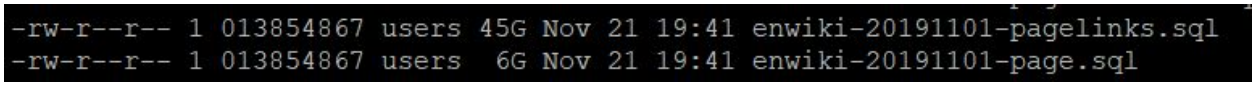
We are using the SQL dump file of the wikipedia article titles and page-links sql files to generate the wikipedia graph structure.

https://dumps.wikimedia.org/enwiki/20191101/enwiki-20191101-pagelinks.sql.gz

https://dumps.wikimedia.org/enwiki/20191101/enwiki-20191101-page.sql.gz

The page-title/info SQL dump is 6GB after decompressing, and the page-links SQL dump is 45GB after decompression.

XML file statistics



Size of the graph dataset

Data Preprocessing :

LDA Clustering:

Our main objective was to convert the articles from the xml format to plain text as well as create a sparse TF-IDF vectors. This is easy to do without decompressing the whole latest wiki dump. Gensim has make_wiki which converts bz2 compressed latest wikipedia articles dump in vector format.

After running the make_wiki on the compressed wiki dump it produced 7 files:

1. **wiki_en_wordids.txt.bz2** : Contains a map of words and their integer ids. The given file is in bz2 compressed before doing LDA modeling we need to decompress.
2. **wiki_en_bow.mm** : contains words counts in Matrix Market Format.
3. **wiki_en_bow.mm.index** : contains index of wiki_en_bow.mm
4. **wiki_en_bow.mm.metadata.cpickle** : contains the metadata of the documents i.e articles and their IDs in a cpickle format.
5. **wiki_en_tfidf.mm** : contains the Tf-IDF of the articles in Market Matrix format. Having, 754782688 non-zeros elements.
6. **wiki_en_tfidf.mm.index** : contains index of wiki_en_tfidf.mm
7. **wiki_en.tfidf_model** : The saved Tf-IDF model which was created during the creating the tf-idf matrix.

In HPC gensim was not able to create **wiki_en_bow.mm.metadata.cpickle file**. So we have to manually create this file by creating a gensim mmcorpus on the wiki dump and create metadata for ~4.7 million documents.

Graph Creation:

In the following cases, we only consider “Main” Wikipedia articles, ie with a “namespace” value of 1.

Getting page-id data:

We parse the **pages.sql** file to extract the page ids and article titles using regular expression search in the entire file in one pass. The resulting (non-redirect) title

and page id map is stored in **wiki_page_data.csv**. Redirected pages are stored in **wiki_page_data_redirects.csv** for use in the next step.

Getting Link Data:

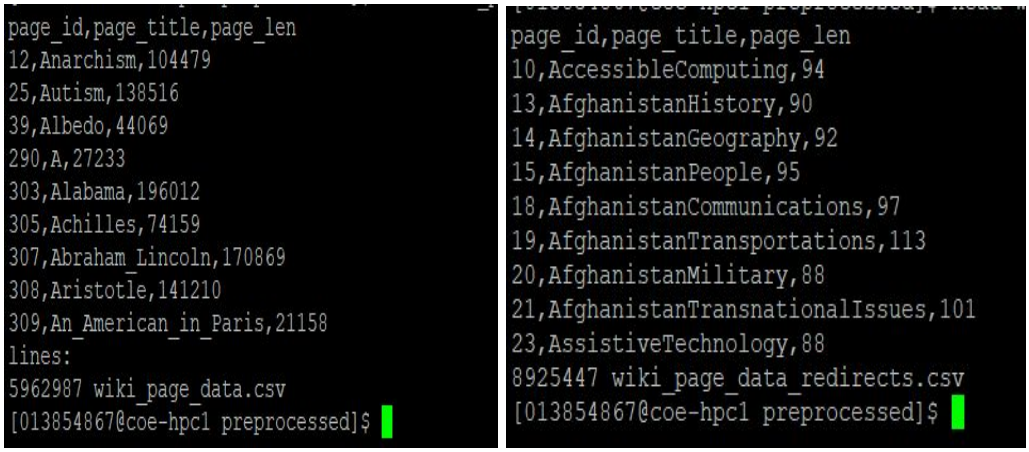
The pagelinks.sql file is parsed to find the links pointing to different pages and self redirects. One challenge here is that a lot of listed links are redirects. Further, the outgoing (destination) pages are listed by title, and not Page ID.

We used an HPC Cluster with a high node count in order to map destination page titles to Page IDs, as well as to resolve the redirect links and map them to their intended destinations. Finally, we remove additional/redundant nodes and redirect links to get graph data in the form of “from_page_id” - “to_page_id” pairs, which are saved into the **adjacency_graph_final_data.csv** file.

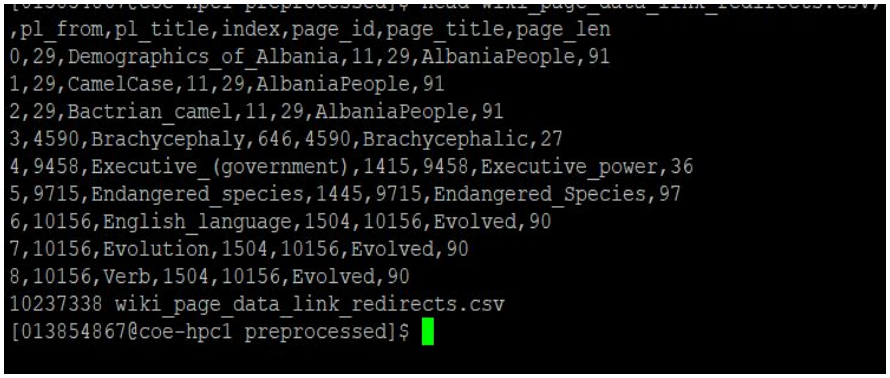
Due to the large number of edges (**~400 million**), the CSV file may occasionally have to be saved in parts, and then later recombined using the **file_concat-graph.sh** script.

Final Generation:

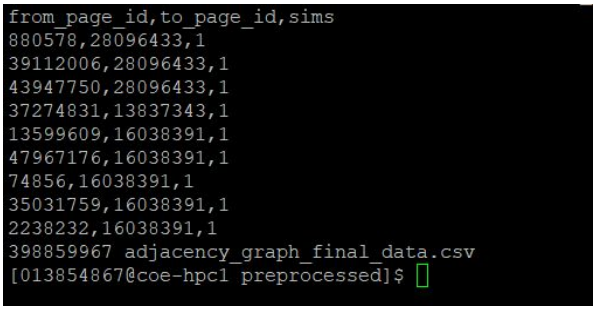
The processing then uses the generated redirect and page-id- title maps to create an edge list by exploding the resulting set of links pointing to a single node. The preprocessing script was written in pyspark to parallelize and create scalable preprocessing script that is efficient for large data.To further speed up the process we are saving the maps generated into csvs which are read from to reduce the parsing overhead on the large data.



pageID-Title map sample(left) and Redirect map sample(right)



Link redirect map sample



Final Graph Data sample

Methodology :

1. Latent Dirichlet allocation (LDA) :-

LDA is a generative statistical model in natural language processing, which explains sets of observations by unobserved groups by which we infer parts of data are similar. Each document is made of different topics and these topics are assigned to it via LDA.

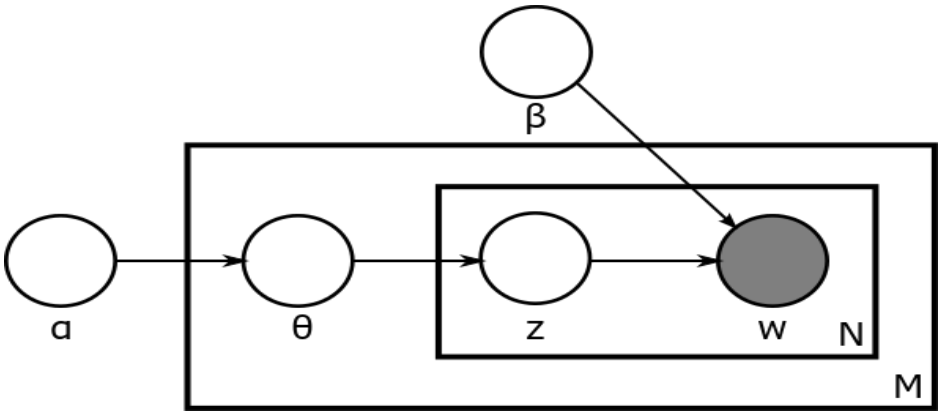


Plate notation is used to represent probabilistic graphical models. The boxes or plates represents replicates and these entities are repeated. In the figure documents are represented by outer plate where as repeated words location in a document are represented by inner plate.

The given variables are as follows:

M → number of documents

N → number of words in a document (document *i* has N_i words)

$\boldsymbol{\alpha}$ → parameter of the Dirichlet prior on the per-document topic distributions

$\boldsymbol{\beta}$ → parameter of the Dirichlet prior on the per-topic word distribution

$\boldsymbol{\Theta}_i$ → topic distribution for document *i*

$\boldsymbol{\varphi}_k$ → word distribution for topic *k*

$\mathbf{z}_{ij}$ → topic for the *j*-th word in document *i*

$\mathbf{w}_{ij}$ → specific word.

w_{ij} are observable variables when W is grayed out and others are latent variables.

Modeling:

- In our model after running the gensim make_wiki script on the latest wikipedia articles we get 7 files as explained in the LDA data preprocessing.
- The wiki_en_wordids.txt obtained after decompressing the wiki_en_wordids.txt.bz2 is used to create id to word dictionary using gensim corpora dictionary.
- Similarly wiki_en_tfidf.mm was used to create the a tf-idf gensim MmCorpus.
- The above two files were given as input to the LDA modeling, with 1 pass where it will iterate once over the whole tf-idf and create 100 topics and 3 pass where it will iterate 3 times over the whole tf-idf and create 100 topics.
- At the end of the lda modeling we got 100 topics and each topic were assigned words and how much they weight to form that topic.

2. K-Means Clustering of Document-Topic matrix:

k-Means Clustering is the most popular clustering algorithm that finds use far and wide in the field of machine learning. It is an iterative clustering algorithm, which uses some defined distance metric in order to cluster items based on proximity.

First, a “k” number of random cluster heads are elected. Following this, all points that lie closest to each of the “k” heads are allocated to their respective clusters. Then, a new cluster head is calculated using the updated membership of that cluster (usually the centroid), for each of the “k” clusters. This constitutes the first iteration, and the 2 previous steps (allocation of points to a cluster, and recomputing cluster heads) are iteratively repeated until convergence, ie there is little to no difference in cluster allocation for all clusters.

For our purposes, we use the PySpark KMeans package in order to cluster topical data obtained from LDA.

Here, the 4.7 million **documents** are the items, and their scores for each of the 100 **topics** are the features. Hence, we feed a matrix of dimensions [4700000 x 100] into our clustering algorithm.

The experiment has been run thrice, with:

- K = 100 clusters
- K = 50 clusters
- K = 35 clusters

3. Graph Analysis - Power Iteration Clustering

Another perspective of the clustering problem is to consider every article as node in a large network of wikipedia articles. The links between each article will form the edges between the networks. This results in a large graph with **4.7 million nodes** and **398 million edges**. The resultant graph is stored as a text-file of edge lists, each line representing one edge. Clustering a large graph of this magnitude requires scalable clustering algorithm that parallelizes the process of clustering. Therefore, we use Power Iteration Clustering(PIC) for clustering the large wikipedia graph.

Power Iteration Clustering is a scalable graph clustering algorithm which uses power iterations to calculate cluster coefficients values of each node in the graph. After deriving these values the data is pipelined into K-means clustering to cluster the nodes based on cluster distinctive values of the nodes. PIC uses the data matrix with edge list and their corresponding similarities as weight of the edge. The input graph matrix of PIC is an affinity matrix, that is the nodes(i,j) as rows and columns with the similarity of i,j as the values in the matrix. The algorithm proposes to stop the power iterations before all the nodes approximate to same global value. This approach reduces the number of iterations over the large network, therefore efficient for clustering large networks. The intermediate results obtained during the power iterations will contain distinct values of the algorithm is stopped at the right number of iterations. The resultant values are clustered using K-means clustering, this step is scalable compared to traditional spectral or agglomerative clustering of graphs. Below snippet shows the PIC algorithm.(*Power Iteration Clustering*,Frank Lin and William W Cohen).

Algorithm 1 The PIC algorithm

Input: A row-normalized affinity matrix W and the number of clusters k

Pick an initial vector $\mathbf{v}^0$

repeat

Set $\mathbf{v}^{t+1} \leftarrow \frac{W\mathbf{v}^t}{\|W\mathbf{v}^t\|_1}$ and $\boldsymbol{\delta}^{t+1} \leftarrow |\mathbf{v}^{t+1} - \mathbf{v}^t|$.

Increment t

until $|\boldsymbol{\delta}^t - \boldsymbol{\delta}^{t-1}| \simeq \mathbf{0}$

Use k-means to cluster points on $\mathbf{v}^t$

Output: Clusters $C_1, C_2, ..., C_k$

Power Iteration Clustering Algorithm

We are using the power iteration clustering available in the pyspark.mllib.clustering library to perform power iteration clustering on the resulting graph. The idea is to cluster with less iterations as possible. We are running the clustering on the basic run to generate 100 clusters with max-iterations value set to 15 power iterations. The model converged in 15 minutes yielding reasonable results. The other max_iteration values we tried are 20 and 25 power iterations. The model with 25 power iterations generated good results when compared to 15 and 20 power iterations. The resulting clusters were much more proportionate compared to other values of power iterations. The run-time of the clustering algorithm was faster than the traditional graph clustering methods like agglomerative or spectral graph clustering algorithms with clusters converging in 2-3 hours on the large dataset containing 4.7 million nodes.

Results and Analysis :

Silhouette score of k-Means Clusters:

A clustering evaluation score that tells how well an object fits into its cluster. It's values range between -1 and 1. Higher the value better the object fits into the cluster. If the score is negative or close to -1 it means that the number of clusters are not ideal.

Our results are as follows:

1 LDA Pass

- Silhouette Score for k-Means with 100 Centers : **0.6783790554519792**
- Silhouette Score for k-Means with 50 Centers : **0.6807962496000666**
- Silhouette Score for k-Means with 35 Centers : **0.6829596873431214**

3 LDA Passes

- Silhouette Score for k-Means with 100 Centers : **0.6778574255021571**
- Silhouette Score for k-Means with 50 Centers : **0.6807410707165481**
- Silhouette Score for k-Means with 35 Centers : **0.680850892899637**

Homogeneity:

This metric gauges the purity of each cluster with regard to the classes of their members. In order to measure the similarity in Power-Iteration Clustering, and k-Means Clustering of LDA Topical Data, we calculate the following metrics on the clusters obtained through Power Iteration Clustering (25 iterations), and assign expected class labels based on k-Means clusters:

- Homogeneity Score: 0.016603222803089825
- V Measure Score: 0.018949665276818004
- Normalized Mutual Information Score: 0.019141785196960935
- Adjusted Mutual Information Score: 0.015570843802447377

From these metrics, it is evident that the two clustering methods have vastly different cluster allocations. This could have two possible reasons:

1. The sizes and shapes of clusters could be widely different
2. The inter-linkage of Wikipedia articles are done greedily (linking to another page whenever possible), without much regard for the semantic information/topic that the text on the page deals with.
3. The number of passes did not differ the cluster properties by much, there is not much topic drift between 1 pass and 3 pass of LDA.

Contribution :

Features	Aashay	Karthik	Swayam
Preprocessing	Major preprocessing	Graph preprocessing for PIC	LDA preprocessing
Modeling	K-means on document-topic matrix	PIC modeling	LDA Modeling
Evaluation	Cluster evaluation of LDA and PIC	Basic PIC evaluation	Basic LDA model testing

Github link:

https://github.com/Swayam595/Clustering_Wiki_Articles_Through_Topic_Modeling

Reference:

[1] https://radimrehurek.com/gensim/wiki.html

[2] https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation

[3] http://www.cs.cmu.edu/~frank/papers/icml2010-pic-final.pdf