We are implementing text classification using Supervised and Semi-supervised learning using Graph Convolution Network (GCN) on Un-directed graph. Instead of using already available graph dataset, we are reading text files, prepossessed them and generating unique dictionary of words.

We are creating single text graph which consists of nodes and edges. The words in documents and document itself will consider as nodes of corporus, in such a way there are two types of nodes, Word node and Document node. Edges in graph are representing connection between words with document and words with words. If word appears in a document, then there will be a edge which is called as Document-to-word edge. One word can belong to multiple documents. We are also maintaining co-occurrence of words and showing their relationship in graph as edge. We called such edges as Word-to-Word edge. We are generating weights of edges using two techniques. Document-to-word edge’s weights are generated using Term Frequency-Inverse Document Frequency (TF-IDF) while Word-to-word edge’s weights are generated using Point-wise Mutual Information (PMI). In TF-IDF, term frequency is the number of times the word appear in the document and Inverse Document Frequency of word shows how that word is significant in whole corpus. The Point-wise Mutual Information is a popular measure which shows word association.

**Dataset**

We are using Kaggle dataset: **7 topic data for text classification**

There are files which belongs to 7 different categories of course work such as Account, Biology, Physics, Math’s, Geography, Computer Science, History. The content of files collected from 11th or 12th grade school books and from other resources as well. Its 5 MB zip file, which as 3,142 files. The course category wise number of files are as follows:

|  |  |
| --- | --- |
| Category of course | Number of files in particular category |
| Accounting | 284 |
| Biology | 635 |
| Geography | 98 |
| Physics | 769 |
| Maths | 214 |
| History | 500 |
| Computer Science | 642 |
| Total Files | 3,142 |

Above-described dataset is available on following link to download.

<https://www.kaggle.com/datasets/deepak711/4-subject-data-text-classification>

**Creation of Graph**

The Networkx library is used to generate a text corpus. A python is providing NetworkX package for creation, manipulation and study structure of graphs. The graph creation process starts with creation of empty graph using Graph class from NetworkX library as follows:

>>> import networkx as nx

>>> G = nx.Graph()

Currently it doesn’t have nodes and edges. We need to add nodes and connections between them. We can add node in Graph as single node, array of nodes and as dictionary of nodes as follows:

>>> G.add\_node(1)

>>> G.add\_nodes\_from([2,3])

>>> G.add\_nodes\_from([

(4, {“color”:”red”}),

(5,{“color”:”blue”})

])

We can add node in graph as integer, word and as image as well. In our case, we are adding all unique words and all document’s identity into empty graph.

We just added nodes but we need to add connections between those nodes i.e. edges. We can add edges in graph as follows:

>>> G.add\_edge(1,2)

>>> G.add\_edge([(1,2),(1,3)])

We can also add edges in graph with its respective weights as follows:

G.add\_edges\_from([(1,2,{“weight”:8})])

Graph details of current dataset is as follows:

|  |  |
| --- | --- |
| Total Number of Document Nodes | 106 |
| Total Number of Word Nodes | 4737 |
| Total Number of Nodes (Document Nodes + Word Nodes) | 4843 |
| Total Number of Document-to-word Edges | 12422 |
| Total Number of Word-toward Edges | 57055 |
| Total Number of Edges | 69477 |

|  |  |
| --- | --- |
| Total Number of Document Nodes | 560 |
| Total Number of Word Nodes | 14097 |
| Total Number of Nodes (Document Nodes + Word Nodes) | 14657 |
| Total Number of Document-to-word Edges | 70344 |
| Total Number of Word-toward Edges | 292289 |
| Total Number of Edges | 362633 |

**Weights of edges**

Once we have created graph from text files, next step is to calculate weight of edges. There are two measures we are using to calculate weights of edges: TF-IDF (Term Frequency Inverse Document Frequency) and PMI (Pointwise Mutual Information).

We are using TF-IDF values as weights of Document-to-word edges and PMI values as weights of Word-to-word. TF-IDF shows how much that word is important in a document amongst collection of documents. While PMI shows correlation between words within defined sliding window.

We calculate TF-IDF as follows:

We calculate PMI as follows:

Text, letter

Description automatically generated

Where

P( i, j) : Probability of word i and j together

P(i) : Probability of word i

P(j) : Probability of word j

#W( i, j) : Number of sliding windows in a corpus that contain both word i and j

#W(i) : Number of sliding windows in a corpus that contain word i

#W : Total number of sliding window in a corpus

A positive value of PMI shows that there is a high semantic relation between words in a corpus. On the other hand, a negative PMI value shows less or no semantic relation in the corpus. So therefore, we are considering only those edges between words who has positive PMI values.

**Creating Adjacency Matrix**

We are generating adjacency matrix of a weighted graph as follows:

Text, letter

Description automatically generated

We can fetch adjacency matrix from graph using networkX library as follows:

A = nx.to\_numpy\_matrix(G, weight="weight")

In this way, we will get adjacency matrix of a weighted graph with, TF-IDF value of a word in a document and PMI value of co-occurrence of words.

**Graph Convolution Network (GCN) details**

We are creating two layers of GCN network using Pytorch framework.

**Splitting Graph Dataset**

Currently we are splitting graph dataset into two section, training dataset and test dataset (In future will think about validation dataset as well).

We are splitting nodes labels rather than actually splitting graph dataset. This method is known as **Transudative setting.** At first, we are randomly selecting 20% dataset node labels as test dataset and rest of node labels will use for training model.

**Loss function and regularization**

We are calculating error using cross entropy loss function as follows:

loss\_fun = torch.nn.CrossEntropyLoss()

l1\_crit = nn.L1Loss(size\_average=False)

loss = loss\_fun(predicted\_class,actual\_class)

for param in model.parameters():

reg\_loss += l1\_crit(param, target=torch.zeros\_like(param))

loss\_train += regularization\_factor \* reg\_loss

**Results**

We are providing bunch of output files as follows:

1. Line graph of Training loss per epochs on training dataset
2. Line graph of Testing loss on test dataset
3. Line graph of Training accuracy per epochs on training dataset
4. Line graph of Testing accuracy on test dataset
5. Combined line graph of loss for training and test dataset
6. Combined line graph of accuracy for training and test dataset
7. Output.txt file which keep track of all epochs and respective accuracy and loss of training and test dataset.
8. Bar chart for class label distribution of training dataset
9. Bar chart for class label distribution of test dataset
10. Histogram for TF-IDF
11. Histogram for PMI
12. TSNE for first layer before training for training dataset
13. TSNE for first layer before training for test dataset
14. TSNE for first layer after training for training dataset
15. TSNE for first layer after training for test dataset
16. Bar chart of Precision of all categories for training dataset
17. Bar chart of Precision of all categories for test dataset
18. Bar chart of Recall of all categories for training dataset
19. Bar chart of Recall of all categories for test dataset
20. Bar chart of F1 score of all categories for training dataset
21. Bar chart of F1 score of all categories for test dataset
22. Heatmap of confusion matrix of training dataset
23. Heatmap of confusion matrix of test dataset
24. DatasetDetails.csv file
25. GraphDetails.csv file
26. NeuralNetworkDetails.csv file
27. Trained model finalized\_model.h5
28. Prediction.txt file

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| No of files | Total no of unique words | Total no of edges | First Hidden layer size | Second Hidden layer size | Total no of epochs | Learning rate | Dataset Size | | Loss | | Accuracy | |
| Training Dataset | Test Dataset | Training Dataset | Test Dataset | Training Dataset | Test Dataset |
| 106 | 4843 | 69477 | 4843X300 | 300X120 | 1000 | 0.0001 | 85 | 21 | 1.147 | 1.33 | 0.741 | 0.523 |
| 106 | 4843 | 69477 | 4843X300 | 300X120 | 1000 | 0.0001 | 85 | 21 | 1.189 | 1.601 | 0.882 | 0.761 |
| **106** | **4843** | **69477** | **4843X300** | **300X120** | **1000** | **0.0001** | **85** | **21** | **1.112** | **1.315** | **0.905** | **0.761** |
| **106** | **4843** | **69477** | **4843X250** | **250X50** | **1000** | **0.0001** | **85** | **21** | **1.548** | **1.704** | **0.811** | **0.714** |
| 559 | 14646 | 362390 | 14646X300 | 300X120 | 1000 | 0.0001 | 447 | 112 | 1.663 | 1.723 | 0.449 | 0.366 |
| 560 | 14097 | 362633 | 14097X250 | 250X100 | 1000 | 0.0001 | 448 | 112 | 1.689 | 1.654 | 0.4308 | 0.4375 |
| 560 | 14097 | 362633 | 14097X250 | 250X100 | 1000 | 0.0001 | 448 | 112 | 1.710 | 1.725 | 0.4196 | 0.375 |
| 560 | 14097 | 362633 | 14097X330 | 250X130 | 1000 | 0.0001 | 448 | 112 | 1.6805 | 1.691 | 0.4397 | 0.375 |
| **560** | **14097** | **362633** | **14097X350** | **350X150** | **2000** | **0.0001** | **448** | **112** | **1.159** | **1.259** | **0.810** | **0.6071** |
| **560** | **14097** | **362633** | **14097X400** | **400X200** | **2500** | **0.0001** | **448** | **112** | **1.041** | **1.080** | **0.8214** | **0.6517** |
| **560** | **14097** | **362633** | **14097X400** | **400X200** | **2500** | **0.0001** | **448** | **112** | **0.977** | **0.999** | **0.8973** | **0.6785** |
| **560** | **14097** | **362633** | **14097X400** | **400X200** | **3500** | **0.0001** | **448** | **112** | **0.709** | **0.702** | **0.9821** | **0.8303** |
| **3140** | **35280** | **1606812** | **35280X400** | **400X200** | **3500** | **0.0001** | **2511** | **629** |  |  |  |  |