CS F469

Information Retrieval Project Report

Second Semester 2022-23

Submitted by:

Mayank Gupta (2022H1030099P)
Padshah Rohan Chirag (2022H1030121P)

Instructor-in-Charge: **Dr. Vinti Agarwal**

Problem Statement:

- The objective of the assignment is to identify the medical research proposals which involve studying the SARS-CoV-2 model.

- Develop a machine learning model which can classify the research proposals on the basis of submitted title, keywords and abstract information (AKT)

Steps Followed

Cleaning the Data and Preparing the Graph

- Upon examining our data, we identified the presence of additional columns that needed to be cleaned. To address this, we created a new dataframe that includes the Title, Article Keywords, Article Abstract, and Contextual columns. The Contextual column indicates whether the record is related to COVID or not.

```
In [2]: df.head()
```

Out[2]:

	Unnamed: 0.1	Unnamed: 0	Article title	Article keywords	Article abstract	Contextual
0	0	0	Assessing the impacts of COVID-19 vaccination ...	Affordability;COVID-19 | SARS-CoV-2;Decision-m...	The COVID-19 vaccine supply shortage in 2021 c...	1.0
1	1	1	Association between interleukin-10 gene polymo...	COVID-19;Interleukin-10 gene polymorphisms;SAR...	Polymorphisms in the interleukin-10 (IL10) gen...	1.0
2	2	2	Quality of Life of early-stage breast-cancer p...	Breast;COVID-19;Cancer;EORTC;Oncology;Quality ...	Objectives To describe the Quality of Life (QO...	1.0
3	3	3	The research interest, capacity and culture of...	Barriers;Health Research;Innovation;Motivators...	The UK National Health Service (NHS) is ideall...	0.0
4	4	4	Machine learning prediction for COVID-19 disea...	COVID-19;Classification;Laboratory markers;Mac...	Early prognostication of patients hospitalized...	1.0

- To clean the data, we initially converted all columns to lowercase. Next, we removed stopwords from each column using the nltk package and its stopwords file. We also eliminated punctuation from the data, except for the hyphen ("-"). Through various permutations, we discovered that the hyphen is important, particularly in terms like "COVID-19," so we decided to retain it.

```
[ ]  import string

    df['Article title'] = df['Article title'].str.lower()
    df['Article keywords'] = df['Article keywords'].str.lower()
    df['Article abstract'] = df['Article abstract'].str.lower()

    #If accuracy is bad then try removing on relevant punctuations after assessing the data
    df['Article title'] = df['Article title'].str.replace('[{}]'.format(string.punctuation.replace('-','')), '')
    df['Article abstract'] = df['Article abstract'].str.replace('[{}]'.format(string.punctuation.replace('-','')), '')
    df['Article keywords'] = df['Article keywords'].str.replace('[{}]'.format(string.punctuation.replace('-','')), '')

    df['Article title'] = df['Article title'].apply(lambda x: ' '.join([item for item in x.split() if item not in stop]))
    df['Article keywords'] = df['Article keywords'].apply(lambda x: ' '.join([item for item in str(x).split(';') if item not in stop]))
    df['Article abstract'] = df['Article abstract'].apply(lambda x: ' '.join([item for item in x.split() if item not in stop]))
```

- Train-Test Split: We performed various splits on our data. Initially, we used an 80:20 split without shuffling. During this step, we also created our test data, which excluded the Contextual information. Consequently, our training data consisted of the following: df_train = pd.concat([df_train, df_test_without_contextual, df_unlabeled]).

```
[ ]  from sklearn.model_selection import train_test_split

     df_labeled = df[df['Contextual'].notnull()]
     df_unlabeled = df[df['Contextual'].isnull()]
     df_train, df_test = train_test_split(df_labeled, test_size=0.3, shuffle=False)

     labeled_index_in_train = list(df_train.index.values)
     labeled_index_in_test = list(df_test.index.values)
     unlabeled_index = list(df_unlabeled.index.values)

     train_size = len(df_train) + len(df_test)  + len(df_unlabeled)

     df_test_without_contextual = df_test[['Article title', 'Article keywords', 'Article abstract']]

     df_train = pd.concat([df_train, df_test_without_contextual, df_unlabeled])

     # print(df_train.head())
     # print(df_test.head())

     print(labeled_index_in_train)
     print(labeled_index_in_test)

     print(df_train.info())
     print(df_test.info())
```

- Considering that the majority of our data was unlabeled, with limited labeled data, we opted to employ the TextGCN model. The input for this model is a graph, and to construct the graph, we utilized a matrix of dimensions (n+m)*(n+m), where n represents the number of documents and m denotes the total number of unique words. To populate the matrix, we utilized tf-idf values for relationships between words and documents. For relationships between words, we utilized PMI scores. Additionally, each word and document included a self-loop, resulting in diagonal entries with a value of 1. To represent this matrix, we employed a sparse matrix as follows:

- adj = sp.csr_matrix((weight, (row, col)), shape=(node_size, node_size)) where node_size is the sum of all documents and unique words.

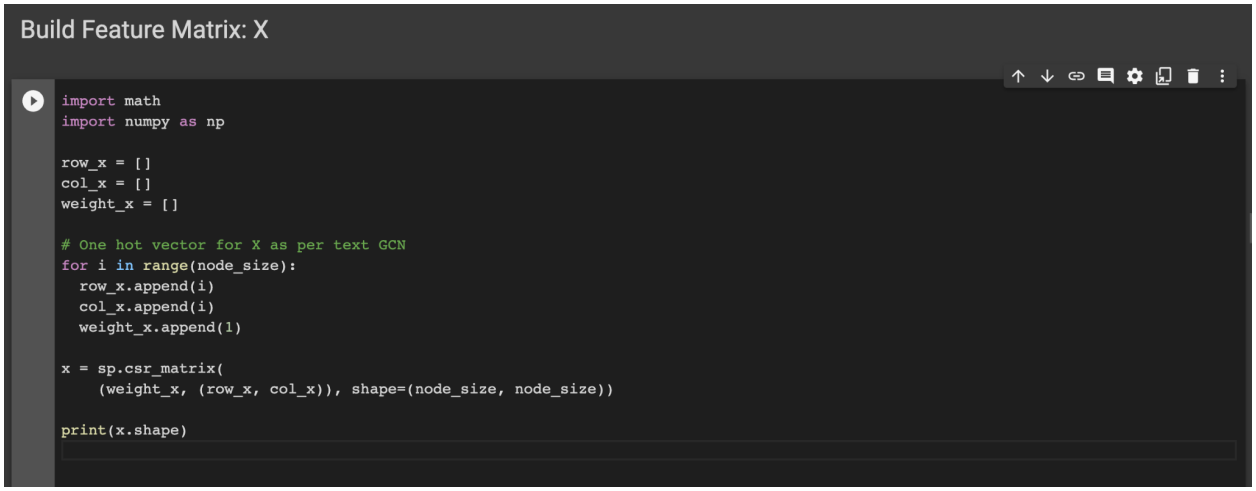
- With this, our graph-building process is complete.

```
import scipy.sparse as sp

#Train size contains all data (train + test both labeled and unlabeled)
print(len(row))
print(len(col))
print(len(weight))
print(node_size)

adj = sp.csr_matrix(
    (weight, (row, col)), shape=(node_size, node_size))

adj = adj + adj.T.multiply(adj.T > adj) - adj.multiply(adj.T > adj)
print(adj)
```

```
11520811
11520811
11520811
85413
  (0, 0)        1.0
  (0, 9298)     3.6525279831908923
  (0, 10758)    4.768761873208821
  (0, 14637)    4.347014332503912
  (0, 14856)    3.5728212405801503
  (0, 15110)    4.137490096366964
  (0, 15357)    9.086249986745132
  (0, 20246)    4.511539008241749
  (0, 20807)    3.502753677963433
  (0, 23773)    3.426767770985511
```

Model Creation

Our Initial Feature matrix consists of one hot vector of size equal to the number of nodes in our graph.

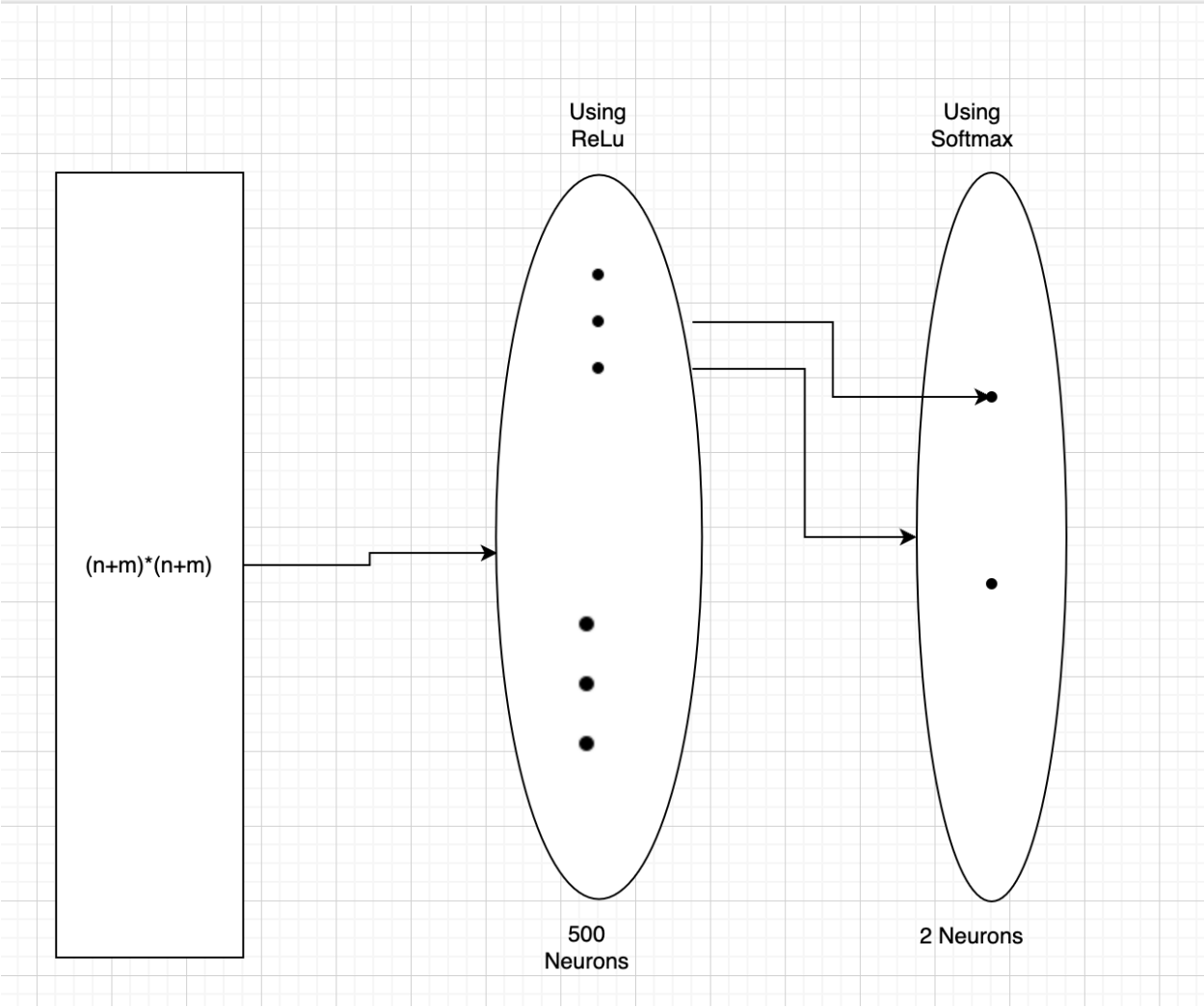


Label Matrix:

- The label matrix consists of a y vector, where:
    - For label=1, the label is [0 1].
    - For label=0, the label is [1 0].
    - For unlabeled and test data, the label is [0 0].

Note:- All the classes are taken from git reference shared in the paper for TextGCN and BertGCN

Architecture



- To enhance this accuracy, we recognized the need to improve our initial feature matrix. As a solution, we incorporated embeddings into the initial feature matrix. We employed BERT to generate embeddings for both documents and words and stored them in a file. However, due to the large size of the files, we faced difficulties processing them.

- Subsequently, we implemented the BERT GCN Model. This model utilizes the BERT embeddings as the initial feature matrix. Each sentence and word obtain their embeddings from BERT, and the GCN model is then applied to this feature matrix.

BERT GCN

- BertGCN constructs a heterogeneous graph over the dataset and represents documents as nodes using BERT representations. By jointly training the BERT and GCN modules within BertGCN, the proposed model can leverage the advantages of both worlds: large-scale pretraining, which takes advantage of the massive amount of raw data, and transductive learning, which jointly learns representations for both training data and unlabeled test data by propagating label influence through graph convolution.

- We are using Roberta based pre trained BERT GCN Model. The prediction will be done by linear interpolation between pretrained Bert model and GCN model.

- RoBERTa Encoding: A pre-trained RoBERTa model is used to encode the tokens and generate contextualized word embeddings. RoBERTa incorporates advanced techniques such as dynamic masking, larger training corpus, and longer training duration to enhance language understanding.

- The X feature matrix will be populated with the embeddings of the pretrained model and the size of X will become => (n+m) X 128

- This model works in 3 steps:
    - Update the document embeddings of documents from a pretrained model.Initially all the features will be 0. After this function call only the features of documents will be updated keeping the word features as all 0's.

```
# Training
def update_feature():
    global model, g, doc_mask
    # no gradient needed, uses a large batchsize to speed up the process
    dataloader = Data.DataLoader(
        Data.TensorDataset(g.ndata['input_ids'][doc_mask], g.ndata['attention_mask'][doc_mask]),
        batch_size=256 #1024
    )
    with th.no_grad():
        model = model.to(gpu)
        model.eval()
        cls_list = []
        for i, batch in enumerate(dataloader):
            input_ids, attention_mask = [x.to(gpu) for x in batch]
            output = model.bert_model(input_ids=input_ids, attention_mask=attention_mask)[0][:, 0]
            cls_list.append(output.cpu())
        cls_feat = th.cat(cls_list, axis=0)
    g = g.to(cpu)
    g.ndata['cls_feats'][doc_mask] = cls_feat
    return g
```

    - Train the model where the loss is computed only on labeled train data. For this purpose a **mask variable** is used which is a boolean array. It will only consider the datapoints with value 1.

```
def train_step(engine, batch):
    global model, g, optimizer
    model.train()
    model = model.to(gpu)
    g = g.to(gpu)
    optimizer.zero_grad()
    (idx, ) = [x.to(gpu) for x in batch]
    optimizer.zero_grad()
    train_mask = g.ndata['train'][idx].type(th.BoolTensor)
    y_pred = model(g, idx)[train_mask]
    y_true = g.ndata['label_train'][idx][train_mask]
    loss = F.nll_loss(y_pred, y_true)
    loss.backward()
    optimizer.step()
    g.ndata['cls_feats'].detach_()
    train_loss = loss.item()
    with th.no_grad():
        if train_mask.sum() > 0:
            y_true = y_true.detach().cpu()
            y_pred = y_pred.argmax(axis=1).detach().cpu()
            train_acc = accuracy_score(y_true, y_pred)
        else:
            train_acc = 1
    return train_loss, train_acc
```

- Last step after the training is to test the accuracy of the model against the test data.

```
evaluator.run(idx_loader_test)
metrics = evaluator.state.metrics
test_acc, test_nll = metrics["acc"], metrics["nll"]

print(
        "Test acc: {:.4f} loss: {:.4f}"
        .format(test_acc, test_nll)
    )
```

Result

- By utilizing the TextGCN model, we achieved an accuracy of 58% on 80-20 split of labeled data.

- By utilizing the BertGCN model, we achieved an accuracy of 71% on 80-20 split of labeled data.

Conclusion

- Performance Comparison: The BertGCN model outperformed the TextGCN model in accuracy. With an 80-20 split of labeled data, the BertGCN model achieved an accuracy of 71%, while the TextGCN model achieved an accuracy of 58%. This indicates that

incorporating the BERT embeddings into the GCN model resulted in improved predictive capabilities.

- Importance of pre trained Models: Leveraging pre-trained models like BERT and Roberta proved beneficial. These models, trained on large amounts of text data, provided valuable language understanding and feature extraction capabilities, enhancing the performance of the downstream GCN model.

- Future Directions: Further improvements can be explored by fine-tuning the BERT component or experimenting with different graph construction techniques. Additionally, expanding the labeled dataset or incorporating semi-supervised learning approaches can potentially enhance the accuracy and generalization of the model.

- Overall, the project highlights the effectiveness of combining BERT embeddings with a Graph Convolutional Network, showcasing improved performance in text classification tasks compared to using GCNs alone. The achieved accuracy of 71% demonstrates the potential impact of this approach in practical applications.

Future Work

- We can provide the word embeddings to the BertGCN model which are right now consider as 0 using techniques like word2vec which might enhance the accuracy of the model.

References

- https://github.com/ZeroRin/BertGCN
- https://github.com/yao8839836/text_gcn