SOCIAL AND ECONOMIC NETWORK ANALYSIS

Project Report

PRIYADARSAN M (18Z239)

RAJASURIYA C K (18Z240)

ARUL JYOTHI S (18Z206)

AKILAN R (18Z204)

MOHANAPRASANTH T (19Z431)

Assignment Submission in partial fulfilment of the degree

BACHELOR OF ENGINEERING

Branch: COMPUTER SCIENCE AND ENGINEERING

Of Anna University

April 2021

PSG College of Technology

Coimbatore – 641004

1. Problem Statement:

To visualize stack overflow as a social network and do,

- Basic metrics analysis on the graph
- Find the questions which may appear together with the given question based on shortest path
- Tag Prediction

2. Dataset Description:

A dataset of Stack Overflow programming questions. For each question, it includes:

- Question ID
- Creation date
- Closed date, if applicable
- Score
- Number of answers
- Tags

This dataset is ideal for answering questions such as:

- The increase or decrease in questions in each tag over time
- Correlations among tags on questions

This dataset was extracted from the Stack Overflow database. This is all public data.

Link: https://www.kaggle.com/stackoverflow/stacklite

3. Tools Used:

- scikit-learn: Built on NumPy, SciPy and matplotlib which is very useful in predictive data analysis.
- pandas: Open source data analysis and manipulation tool built on Python.
- networkx: Python library for analysing networks and graphs.
- matplotlib: Plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications.
- seaborn: Python data visualization library based on matplotlib which gives high level attractive drawings.
- nltk (Natural Language Tool Kit): Libraries and programs for symbolic and statistical natural language processing for English written in the Python programming language.

4. Challenges Faced:

- Multi-label classification
- Pre-processing the text data in title
- Processing all the nodes and edges

5. Contribution:

Name (Roll number)	**Contribution**
Priyadarsan M (18z239)	Tag Prediction
Rajasuriya C K (18z240)	Extracting the largest connected sub component and shortest path analysis
Arul Jyothi S (18z206)	Tag Prediction
Akilan R (18z204)	Basic level analysis on the graph
Mohanaprasanth T (19z431)	Basic level analysis on the graph

Annexure – I:

```
#Import required packages
import pandas as pd
import networkx as nx
from matplotlib.pyplot import figure
import matplotlib.pyplot as plt
import seaborn as sns
#Read the tags dataset and considering the first 1500 nodes
df1 = pd.read_csv('question_tags.csv')
df1 = df1.iloc[:1500]
G = nx.Graph()
#Constructing the graph
G = nx.from_pandas_edgelist(df1, 'Id', 'Tag')
#Plotting the graph
figure(figsize=(50, 50))
Gr = nx.draw(G, with_labels=True)
Gr
#Finding the degree centrality
degreeCentrality = nx.degree_centrality(G)
tag = []
centrality = []
for key, value in degreeCentrality.items():
    tag.append(key)
    centrality.append(value)
cent = pd.DataFrame()
cent['Tags'] = tag
cent['centrality'] = centrality
cent = cent.sort_values(by='centrality', ascending=False)
plt.figure(figsize=(15, 10))
_ = sns.barplot(x='centrality', y='Tags', data=cent[:10], orient='h')
_ = plt.xlabel('Degree Centrality')
_ = plt.ylabel('Tag')
_ = plt.title('Top 10 Degree Centrality Scores in StackOverflow tag network')
plt.show()
#Calculating betweenness centrality and plotting the ones with high value
```

```
between = nx.betweenness_centrality(G)
tag = []
betweenness = []
for key, value in between.items():
    tag.append(key)
    betweenness.append(value)
bet = pd.DataFrame()
bet['Tag'] = tag
bet['betweenness'] = betweenness
bet = bet.sort_values(by='betweenness', ascending=False)
plt.figure(figsize=(15, 10))
_ = sns.barplot(x='betweenness', y='Tag', data=bet[:10], orient='h')
_ = plt.xlabel('Betweenness Centrality')
_ = plt.ylabel('Correspondent')
_ = plt.title('Top 10 Betweenness Centrality Scores in StackOverflow Tag network')
plt.show()
#Density of the graph
print(nx.density(G))
#Calculating eigenvector centrality and plotting the ones with high value
eigen = nx.eigenvector_centrality(G, max_iter=600)
tag = []
eigenvect = []
for key, value in eigen.items():
    tag.append(key)
    eigenvect.append(value)
ev = pd.DataFrame()
ev['Tag'] = tag
ev['EigenVector'] = eigenvect
ev = ev.sort_values(by='EigenVector', ascending=False)
plt.figure(figsize=(15, 10))
_ = sns.barplot(x='EigenVector', y='Tag', data=ev[:10], orient='h')
_ = plt.xlabel('Eigen Vector Centrality')
_ = plt.ylabel('Correspondent')
_ = plt.title('Top 10 EigenVector Centrality Scores in StackOverflow Tag network')
plt.show()
#Calculating page rank for the nodes and plotting them
pg_rank = nx.pagerank(G)
tag = []
pr = []
for key, value in pg_rank.items():
    tag.append(key)
    pr.append(value)
pagerank = pd.DataFrame()
pagerank['Tag'] = tag
pagerank['PageRank'] = pr
pagerank = pagerank.sort_values(by='PageRank', ascending=False)
```

```
plt.figure(figsize=(15, 10))
_ = sns.barplot(x='PageRank', y='Tag', data=pagerank[:10], orient='h')
_ = plt.xlabel('PageRank')
_ = plt.ylabel('Correspondent')
_ = plt.title('Top 10 Pagerank Scores in StackOverflow Tag network')
plt.show()
#Extracting the largest connected sub component present
cur_graph = G
if not nx.is_connected(cur_graph):
    # get a list of unconnected networks
    sub_graphs = (G.subgraph(c) for c in nx.connected_components(G))
    sub_graphs = list(sub_graphs)
    main_graph = sub_graphs[0]
    for sg in sub_graphs:
        if len(sg.nodes()) > len(main_graph.nodes()):
            main_graph = sg
    cur_graph = main_graph
#Plotting the sub graph
figure(figsize=(30, 20))
Grsub = nx.draw(cur_graph, with_labels=True)
Grsub
#Calculating edge betweenness of the edges
edge_bet = nx.edge_betweenness(cur_graph)
tag = []
eb = []
for key, value in edge_bet.items():
    tag.append(key)
    eb.append(value)
edgebetween = pd.DataFrame()
edgebetween['Tag'] = tag
edgebetween['EdgeBetweenness'] = eb
edgebetween = edgebetween.sort_values(by='EdgeBetweenness', ascending=False)
plt.figure(figsize=(15, 10))
_ = sns.barplot(x='EdgeBetweenness', y='Tag', data=edgebetween[:10], orient='h')
_ = plt.xlabel('EdgeBetweenness')
_ = plt.ylabel('Correspondent')
_ = plt.title('Top 10 EdgeBetween Scores in StackOverflow Tag network')
plt.show()
#Import for community detection
from networkx.algorithms import community
#Generating communities using Girvan-Newman algorithm
communities_generator = community.girvan_newman(cur_graph)
tl = list(list(c) for c in next(communities_generator))
#First level community
tll=[]
for i in tl:
```

```
    for j in i:
        tll.append(j)
    break
tll
#Top level community present in the sub graph
clr_comm = nx.Graph()
clr_comm = cur_graph
clrs=[]
for i in clr_comm.nodes:
    if i in tll:
        clrs.append('red')
    else:
        clrs.append('blue')
plt.figure(figsize=(40,40))
nx.draw_networkx(clr_comm, with_labels=False,node_color=clrs)
plt.show()
#Find the shortest path between the given id and every other question
path_list=[]
node_num = int(input("Enter the question id: "))
for i in colored_graph.nodes:
    if(type(i)==int):
        if(i!=node_num):
            path = nx.shortest_path(colored_graph, node_num, i)
            path_list.append(path)
def len_func(l):
    return len(l)
#Sort the shortest paths based on their lengths
path_list.sort(key=len_func)
#Print the top 6 nodes which are nearer to the given question id
print("The questions with id which are most likely to appear along with", node_num)
node_list=[]
count=0
for i in path_list:
    if count>5:
        break
    length = len(i)
    print(i[length-1])
    node_list.append(i[length-1])
    count+=1
#Read the csv files as dataframes
df = pd.read_csv("Questions.csv", encoding="ISO-8859-1")
tags = pd.read_csv("Tags.csv", encoding="ISO-8859-1", dtype={'Tag': str})
grouped_tags = tags.groupby("Id")['Tag'].apply(lambda tags: ' '.join(tags))
grouped_tags.reset_index()
df = df.merge(grouped_tags_final, on='Id')
#Import nltk packages for data pre-processing
```

```
import nltk
from nltk.tokenize import ToktokTokenizer
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import stopwords
new_df.dropna(subset=['Tags'], inplace=True)
#Initialize the pre-processors
lemma=WordNetLemmatizer()
stop_words = set(stopwords.words("english"))
token=ToktokTokenizer()
def lemitizeWords(text):
    words=token.tokenize(text)
    listLemma=[]
    for w in words:
        x=lemma.lemmatize(w, pos="v")
        listLemma.append(x)
    return ' '.join(map(str, listLemma))
def stopWordsRemove(text):
    stop_words = set(stopwords.words("english"))
    words=token.tokenize(text)
    filtered = [w for w in words if not w in stop_words]
    return ' '.join(map(str, filtered))
#Cleaning the title of questions
new_df['Title'] = new_df['Title'].apply(lambda x: str(x))
new_df['Title'] = new_df['Title'].apply(lambda x: lemitizeWords(x))
new_df['Title'] = new_df['Title'].apply(lambda x: stopWordsRemove(x))
#Import scikit learn packages for prediction
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import SGDClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import hamming_loss, accuracy_score, precision_score, recall_score
from sklearn.metrics import f1_score
#Count vectorize the tags
vectorizer = CountVectorizer(tokenizer = lambda x: x.split(), binary=True)
multilabel_y = vectorizer.fit_transform(new_df['Tags'])
#Train test split
x_train=new_df.head(58360)
x_test=new_df.tail(new_df.shape[0] - 58360)
y_train = multilabel_y[0:58360,:]
y_test = multilabel_y[58360:new_df.shape[0],:]
#Tfidf vectorizer for text data (title)
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tf = TfidfVectorizer(min_df=0.00009, max_features=200000, smooth_idf=True, norm
="l2", \ tokenizer = lambda x: x.split(), sublinear_tf=False, ngram_range=(1,2))
x_train_multilabel = vectorizer_tf.fit_transform(x_train_final)
x_test_multilabel = vectorizer_tf.transform(x_test_final)
#One vs Rest classifier for multi-label classification
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=0.00001, penalty='l1'), n_jobs=-1)
classifier.fit(x_train_multilabel, y_train)
predictions = classifier.predict (x_test_multilabel)
#Metrics score
print("Hamming loss ",hamming_loss(y_test,predictions))
#Since it is tag prediction, the precision and recall score plays a major role
precision = precision_score(y_test, predictions, average='micro')
recall = recall_score(y_test, predictions, average='micro')
f1 = f1_score(y_test, predictions, average='micro')
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))
```

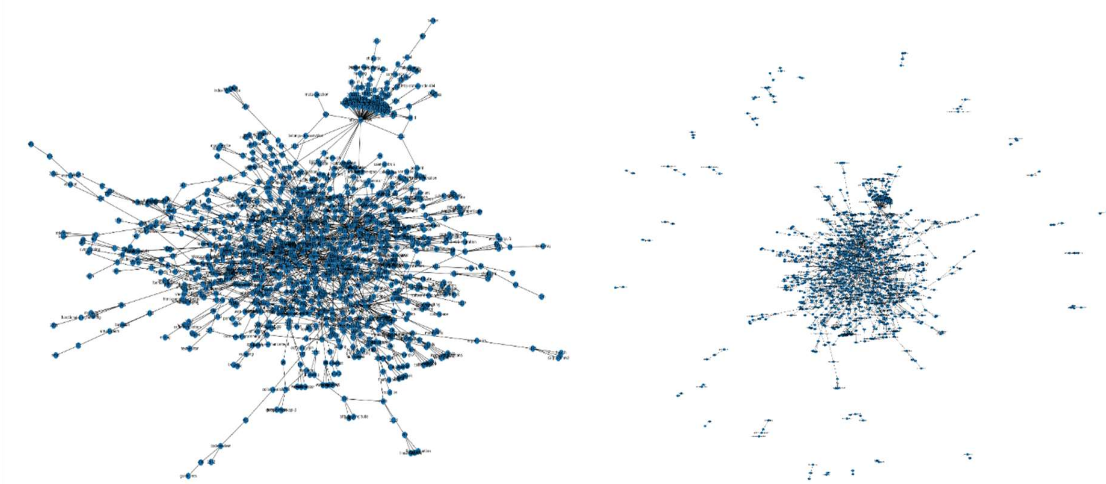
Annexure- II:

```
The questions with id which are most likely to appear along with 4
8
9
11
16
38
39
```

Figure 1: Questions which may appear as similar questions in the feed

```
Hamming loss  0.0001798999064993907
Precision: 0.7528, Recall: 0.2574, F1-measure: 0.3836
```

Figure 2: Performance metrics score of tag prediction



References:

[1] Pandas - https://pandas.pydata.org/docs/reference/index.html

[2] Networkx - https://networkx.org/documentation/stable/reference/algorithms/index.html

[3] Nltk - https://www.nltk.org/

[4] Text pre-processing - https://medium.com/@datamonsters/text-preprocessing-in-python-steps-tools-and-examples-bf025f872908

[5] Tag prediction basics - https://www.kaggle.com/miljan/predicting-tags-for-stackoverflow

[6] Count Vectorizer - https://www.educative.io/edpresso/countvectorizer-in-python

[7] SGD Classifier - https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

[8] TF-IDF Vectorizer - https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html