I. Introduction

For the term project entitled heterogeneous data, the goal was to combine both graph and text data processing methods. The data that was provided is a data dump from the Data Science StackExchange community. For graph processing, Neo4j was used which is an ACID-compliant transactional database with native graph storage and processing. Afterwards, node2vec was employed to create embeddings on the text data located in different nodes of the graph. These embeddings were then stored in the graph nodes as well. Finally, word2vec models from genism were used to draw conclusions from the data.
After analyzing node2vec on neo4j I looked at two more hypotheses. The first looks at whether Users with higher reputations have a higher reputation/answer score.

II. Neo4j

In assignment 3 I used Neo4j to graph relationships in data taken from Reddit where one subreddit could be linked to another subreddit in order to explore sentiments between these different subreddits. In initially exploring the data provided for this project I decided that Neo4j would be a perfect fit. I created two different models of the data inside of Neo4j.

The first model I made used three of the XML files from the stack exchange data science data. The posts, users and tags files were converted to CSV through the use of a script that was found online. A script was also used to populate neo4j with the resulting data. This script was altered to work with the data and also with the current version of neo4j. Data is imported into neo4j via the bin/neo4j-import tool. This is something that I recently learned to do. An example of a query done on this data is shown in figure 1. Figure 2 is a visual example of an all shortest paths query done on this graph.

	u.displayname	u.reputation	u.location	answers
0	n1k31t4	8181	Europa	117
1	Neil Slater	18824	Durham, United Kingdom	36
2	Juan Esteban de la Calle	1504	Bogotá, Colombia	24
3	Jan van der Vegt	6995	Amsterdam	23
4	Aditya	1577	West Bengal, India	22
..	...	...	...	...
455	Christian Sauer	437	Düren, Germany	1
456	MCP_infiltrator	966	New York, United States	1
457	Christopher Louden	1120	San Antonio, TX	1
458	Stanpol	572	Norway	1
459	adesantos	483	Madrid	1

Figure 1: MATCH (u:User)-[:POSTED]->(answer)<-[:PARENT_OF]-()-[:HAS_TAG]-(:Tag {tagId:'python'}) WHERE u.displayname contains 't' RETURN u.displayname,u.reputation,u.location, count(distinct answer) AS answers ORDER BY answers DESC;

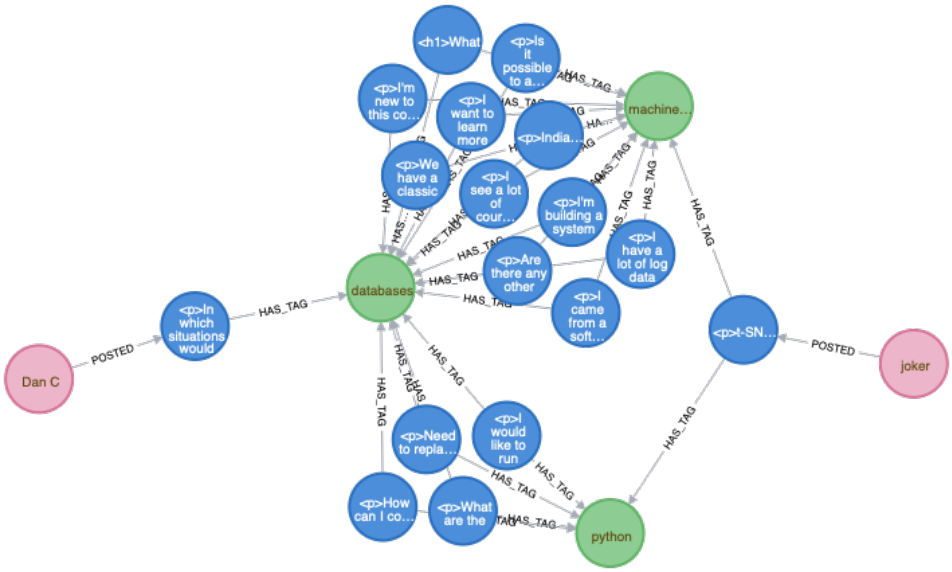

Figure 2: All shortest paths between the users 'joker' and 'Dan C'. This query was done on the complicated graph that was not used for node2vec.

The second graph that I made was a simplified version using only the tag and content data in order to run node2vec. This graph was created via Python code using the same techniques that were learned in completing assignment 3. The steps used to create this graph are explained in section IV, modelling. Figure 3 is an example of what this graph looks like with only 2 node types.

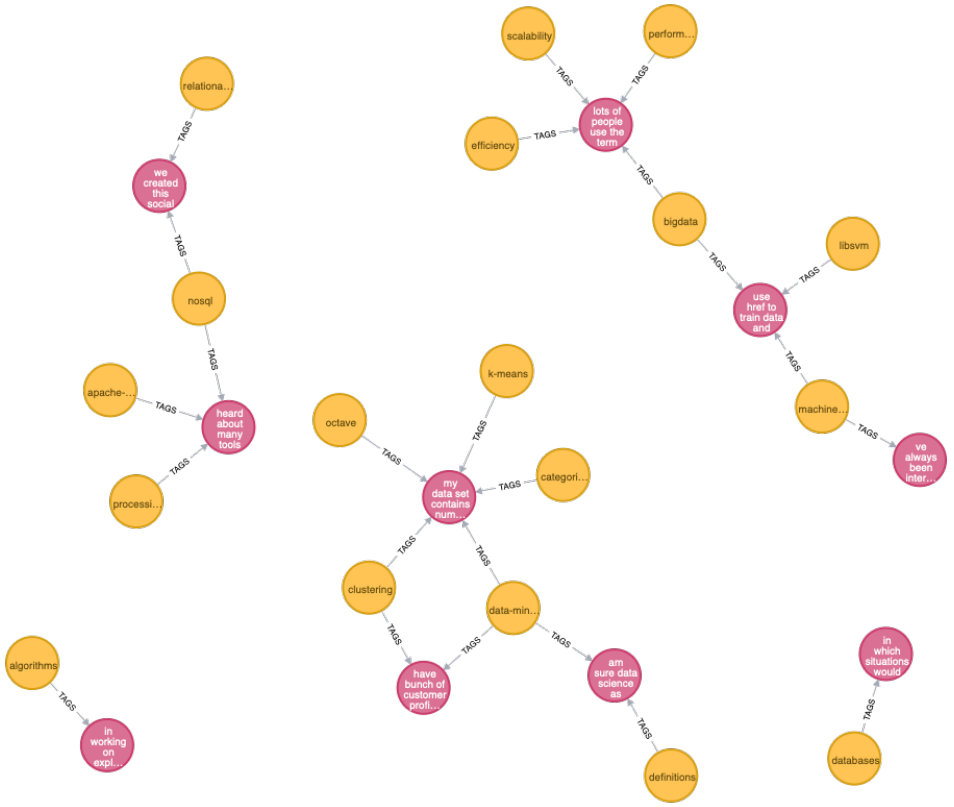

Figure 3: Visual representation of the simplified graph that was made for running node2vec. The only nodes are for tags and content. A limit of 25 was used on the query for visualization purposes.

III. Node2vec

The node2vec code that was used in this report is written in c++ and was released through the Snap project from Stanford University. Node2vec itself is an algorithmic framework for representational learning on graphs. Given any graph, it can learn continuous feature representations for the nodes, which can then be used for various downstream machine learning tasks. Learning representations from structured objects like graphs can help optimize predictive power. By optimizing a neighborhood preserving objective, low-dimensional representations for nodes are learned. Node2vec is flexible in that it accommodates various definitions of network neighborhoods by simulating biased random walks. [1]

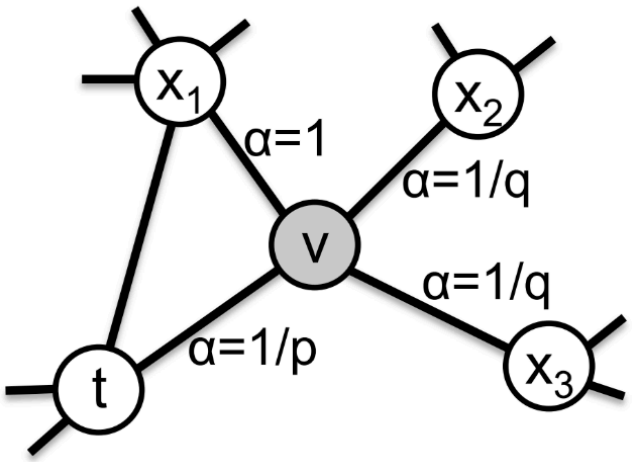

Figure 4: After transitioning to node v from t, the return hyperparameter, p and the inout hyperparameter, q control the probability of a walk staying inward revisiting nodes (t), staying close to the preceding nodes (x1), or moving outward farther away (x2, x3). From [1]

Node2vec was introduced in the paper, “node2vec: Scalable feature learning for networks” and has been cited 2348 times. Grover et al have designed a random walk procedure to efficiently explore diverse neighborhoods. They do this by defining a flexible notion of a node’s network neighborhood. Their algorithm generalizes prior work which is based on rigid notions of network neighborhoods. They argue that adding flexibility in exploring networks is the key to learning richer representations. [2]

Embeddings in node2vec are learned by using a skip-gram model in the same way that they are used in word2vec. Like word2vec, node2vec generates a corpus. To generate a corpus from a graph, node2vec uses a sampling strategy. Word2vec is essentially already embedding a graph albeit a very simple type and this is a directed acyclic graph with a maximum out-degree of 1 where each node is a word in a sentence. By using a sampling strategy that can be tweaked by hyperparameters, node2vec can extend the logic of word2vec to more complex graphs. Random walks are generated from each node of a graph. [3] On top of general skip-gram parameters, node2vec accepts 4 hyperparameters:

1. Number of walks: the number of random walks from each node in the graph.
2. Walk length: The number of nodes in each random walk.
3. P: Return hyperparameter.
4. Q: Inout hyperparameter.

Figure 4 which comes from the original paper gives more of an explanation on how the parameters P and Q work. P controls the probability to go back to <t> after visiting <v>. Q controls the probability of going to explore unvisited parts of the graph. The final travel probability is a function of 3 things [3]:

1. The previous node in the walk
2. P and Q
3. Edge weight

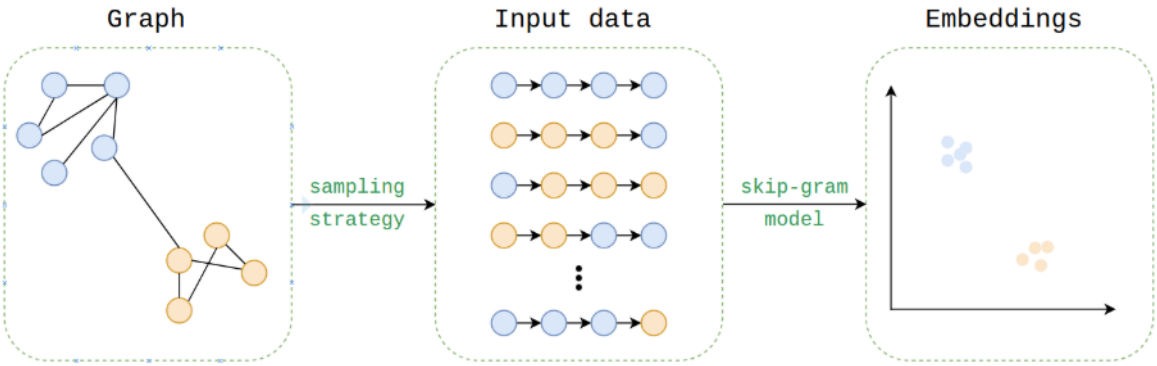

Figure 5: Node2vec embedding process. From [3].

IV. Modelling: Neo4j -> node2vec

1. Convert to xml: I used code that I found online to convert xml to csv.
2. Clean data: I cleaned the body column in the posts.csv file. I tokenized the content, handled negations, converted all text to lower case and then later took out stop words. I reused code that I had from assignment 4 to do all of this.
3. Assemble dataframe: In this step I merged the posts with their tags to create a dataframe. This was done to create a simplified structure that mirrors the steps I took to create a graph in Neo4j in the same way that I did for assignment 3.
4. Import data into Neo4j: I created a very simple graph in Neo4j. The reason for having such a basic structure is to learn how to use node2vec.

```
USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM "file:////tag_and_post.csv" as row
MERGE (s:TAG{tag:row.tag})
MERGE (t:CONTENT{text:row.text})
CREATE (s)-[:TAGS]->(t)
```

5. Create edge list: The next step was to create an edge list that could be then imported into node2vec. The edge list is just a list of all edges consisting of the source and target nodes.

```
result = session.run("""\
MATCH (c:CONTENT)--(other)
RETURN id(c) AS source, id(other) AS target
""")

writer = csv.writer(edges_file, delimiter=" ")
for row in result:
    writer.writerow([row["source"], row["target"]]
```

6. Node2vec: Node2vec is used to create the embeddings. To run node2vec I had to install Snap which is a project out of Stanford University. It is c++ code that takes the edge list from step 5 as the parameter. It exports an emb file. The output from this file can be seen in Figure N.
7. Store embeddings in Neo4j: The embedding information for each node is imported into Neo4j. A TAG node now looks like this:
   "tag": "machine-learning"
   "embedding": [0.677453, -0.604607, 0.257911 …]

	source	embedding	tags
0	0	[0.677453, -0.604607, 0.257911, 0.467539, 0.04...	[0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, ...
1	2	[0.0387761, 0.0941891, 0.0417972, 0.0645476, -...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
2	4	[0.0524003, 0.0732689, 0.0213013, 0.0458899, 0...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
3	5	[-0.00247167, 0.129757, 0.106142, 0.116246, 0....	[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4	7	[0.0485555, 0.0876447, 0.0168011, 0.0788839, -...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...

8. Import genism KeyedVectors: Using a function to find the most similar movies based on cosine similarity (it uses the .most_similar method in genism and then searches neo4j for the right node):

```
print(neo4j_most_similar_tag(model, 'keras'))
```

```
r 0.9959577918052673
classification 0.9937408566474915
nlp 0.9897832870483398
scikit-learn 0.987450361251831
time-series 0.9830454587936401
clustering 0.9818791151046753
tensorflow 0.9793992638587952
deep-learning 0.9774578809738159
data-mining 0.9734617471694946
neural-network 0.9691236615180969
```

If we look at the similarity between 'keras' and 'machine-learning' with the command: model.similarity('1833', '0'), we get a very low score of 0.2914106. This is perhaps because keras is a neural network library.

V. Hypotheses

1. Recommendations via node2vec: The first hypothesis involved with this submission is in recommending tags or posts to Users that are related to their questions, provided by embeddings. I realized too late that my node2vec implementation was not utilizing the actual text information stored in the post nodes. To fully implement my idea, I would maybe have to tokenize post content and then put each token into a graph as a separate node. This is something that I wish I had more time to try. Some of the results are shown above in modelling step 8. These are cosine similarity results based on posts:

```
print(neo4j_most_similar(model, 'as researcher and instructor looking for open source books or similar materials
```

```
we are developing classification system where the categories are fixed but many of them are 0.9876089096069336
was going through the official documentation of scikit learn learn after going through book 0.9870911240577698
is it possible to train neural network to solve polynomial equation what about any non linea 0.986779510974884
was looking at kernel implementation for text classification and the following piece of cod 0.9847907423973083
ve built an rl agent using the following ol li href 0.984783411026001
have problem where need to weight models for ensemble learning however some models aren 0.9847713112831116
am building neural network to solve regression problem the output is single numerical va 0.9839391112327576
would like to clip the reward in keras saw it is possible to clip the norm and clip the valu 0.9837677478790283
have variety of nfl datasets that think might make good side project but have not done 0.9831158518791199
would like to understand regularization shrinkage in the light of mle gradient descent know 0.9823760986328125
```

Conclusion: Node2vec is interesting but I am still not completely sure how I could use it for full effect. It seems to work well for predicting tags but for predicting posts that may be useful for a question it is a little more difficult. Figure 7 below is an attempt to visit the results of finding the most similar tags to keras.

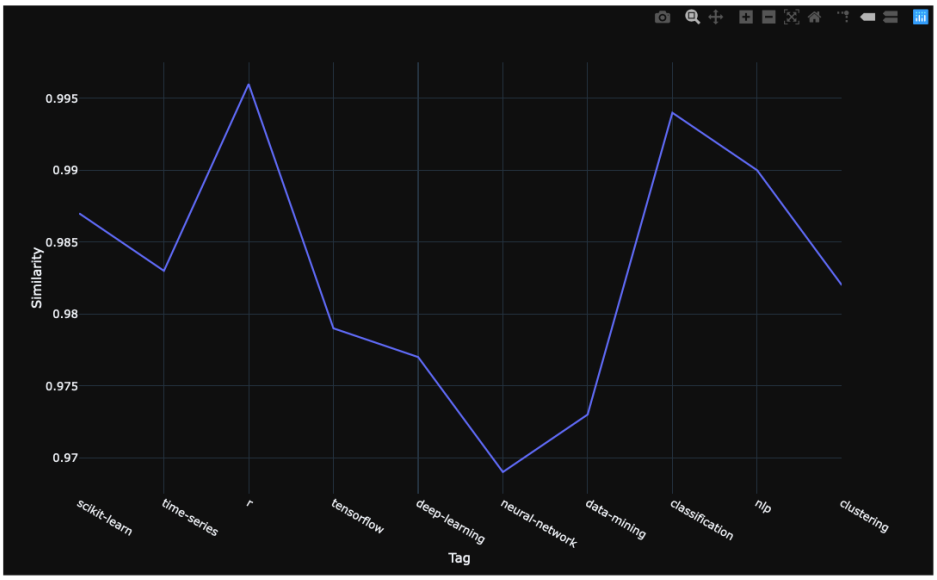

Figure 6: node2vec: most similar tags to 'keras'

2. Prediction – Users with higher reputations have a higher reputation per answer score: The logic behind this train of thought is that having a high reputation score makes a person more well-known in the data science community on stack. This

person has provided an answer that works and new answers that they may provide get more attention than other people answering.

	name	rep	u.location	answers	rep/ans
16	Martin Thoma	7157	München, Deutschland	108	66.268519
14	Dawny33	5798	Gurgaon, India	112	51.767857
0	Neil Slater	18824	Durham, United Kingdom	393	47.898219
19	ncasas	4201	Barcelona, España	103	40.786408
5	Emre	8881	Silicon Valley, CA, United States	227	39.123348
7	Jan van der Vegt	6995	Amsterdam	184	38.016304
11	oW_	3906	San Diego, CA, United States	116	33.672414
8	Esmailian	5069	None	152	33.348684
6	JahKnows	5782	Montreal, QC, Canada	203	28.482759
9	Kasra Manshaei	3900	Bonn, Germany	147	26.530612

Figure 7: top 10 Users by reputation / # of answers

Conclusion: There are definitely some heavy hitters like Neil Slater who has answered 393 times and has a rep/ans score of over 47. Martin Thoma however only has 108 answers and scores over 66 on rep/ans. Neil Slater had a total reputation of 18824 while Martin Thoma only has 7157. The chief outlier is Erwan who has answered 114 times but has a low rep/ans score close to 12. A lot more data would need to be analyzed to draw any real conclusions.

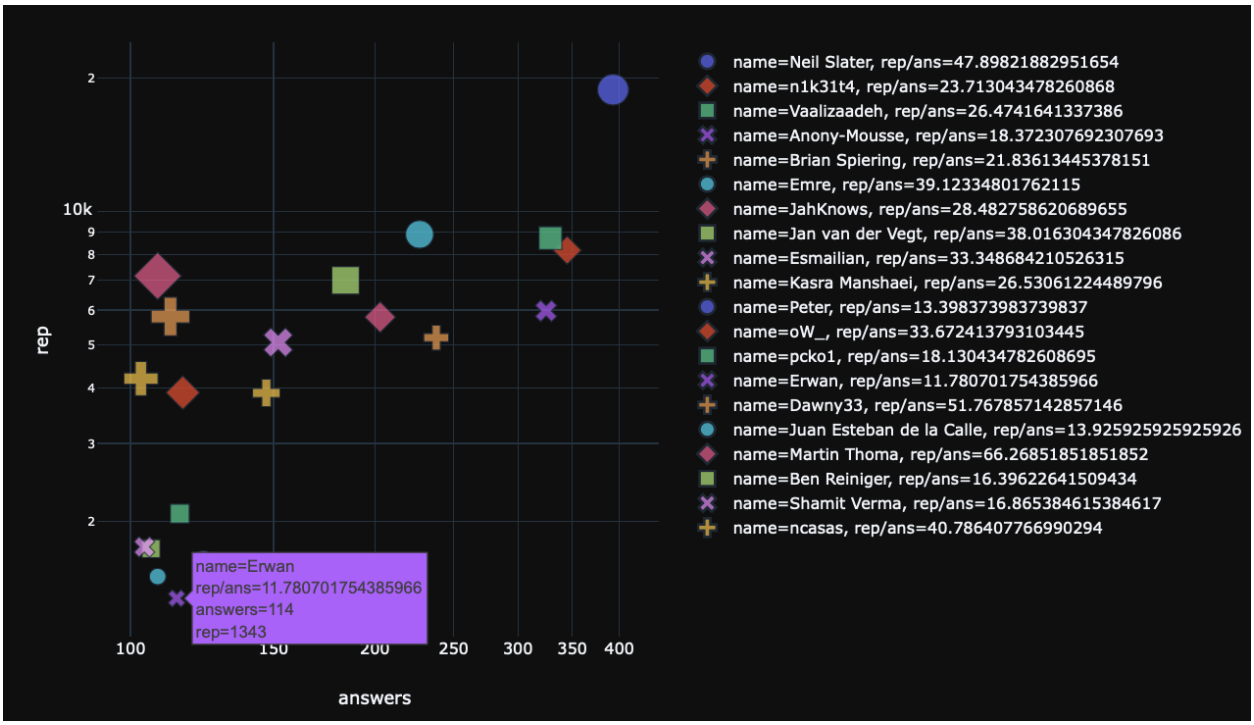

Figure 8: reputation X answers – the size of the marker is proportional to the reputations score / by the number of answers that a User has.

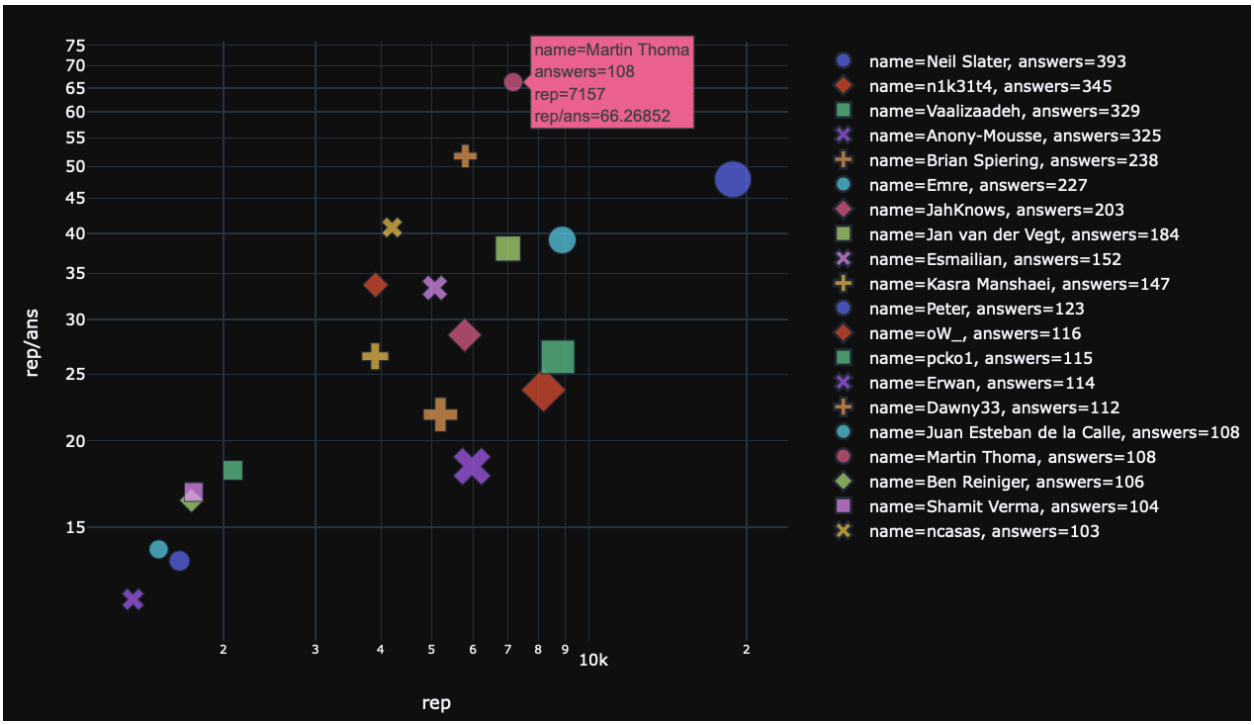

Figure 9: reputation/answers X reputation – the size of the marker is proportional to the number of answers that a User has.

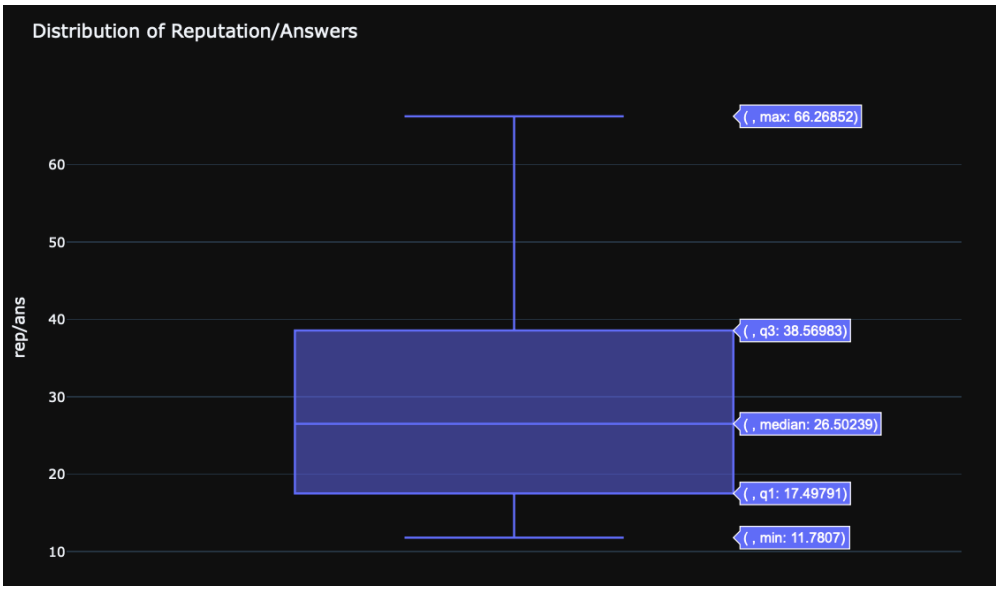

Figure 10: distribution of rep/ans scores for top 20 Users by the number of answers they have posted.

3. Relationship between the number of comments on a post and the score associated with a post. As can be seen in Figure 11, there are some posts that have a much higher score than what would be normally expected.

   Conclusion: The scores for posts with 9 answers generally fall inside of a relatively narrow range. There are a few posts with

extremely high scores. The highest scoring post with a score of 165 is entitled, 'Public Available Datasets' which is interesting. I was unable to draw any conclusions from this hypothesis.

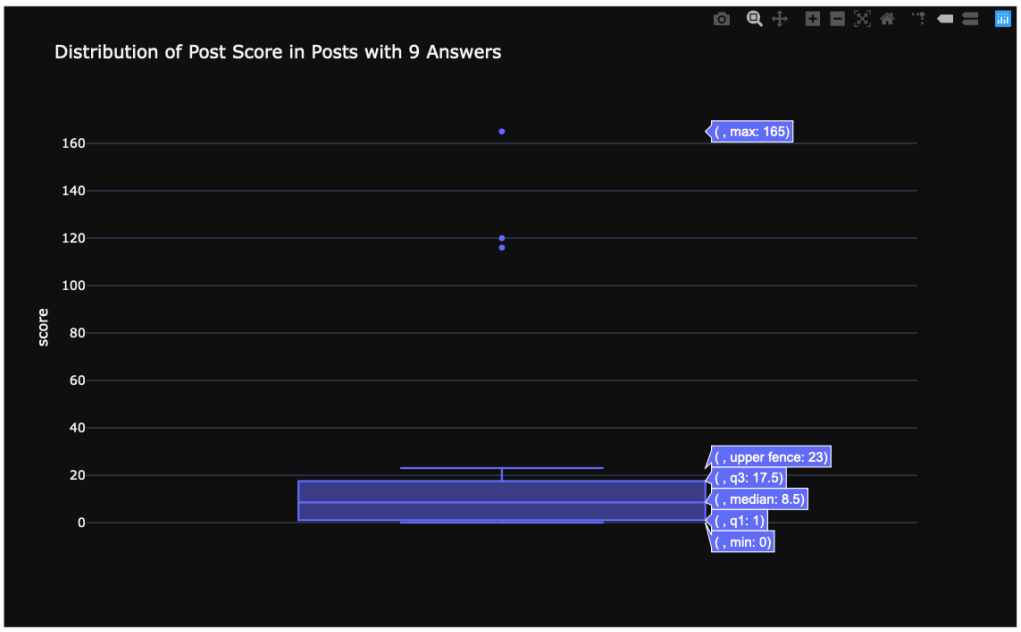

Figure 11: distribution of post scores for in posts with 9 answers.

VI. Conclusion

In this project, I combined ideas and methods from the section on graphs and the section on text processing. To do this I used Neo4j as a graph database and node2vec for text processing. Node2vec mimics word2vec but creates embeddings from a graph instead of a sentence, using random walks. I had hoped to utilize the textual information in each post to make better recommendations but this was outside the scope of this project. These ideas formed the basis for my first hypothesis.

For the second hypothesis, I looked at the relationship between a User's reputation and the number of answers they have posted. I thought that having a higher reputation would result in a higher total reputation / number of answers score (rep/ans) as people would naturally think Users with higher reputations would have better answers. The results gathered from following this idea were far from conclusive. Generally, Users with higher reputations did also have higher rep/ans scores but this was probably just because these are people who generally post better answers.

With the last hypothesis I looked at the relationship between scores and comments in Posts. I hypothesized that posts with a lot of comments should have high scores. I was unable to draw any conclusions

after following this hypothesis. I looked only at posts with 9 comments due to space and time considerations.
This was a challenging project. We were asked to combine and apply two chapters from this course to model our data. I was not comfortable having one model to simply be data tables, whether SQL or dataframes (I did end up extensively using dataframes as well). I wanted to challenge myself and apply word2vec-like embeddings in neo4j. Learning to use node2vec was far from easy. Neo4j is extremely challenging too I think, especially for a new User. Applying node2vec and building the graphs in neo4j took a considerable amount of time. Being a one-person group is something that should not be underestimated. I had trouble coming up with a good third hypothesis and it was in times like this when I would have liked to have someone to bounce ideas around with.

```
$ ./node2vec -i:graph/stack_dec12.edgelist -o:emb/stack_dec12.emb -l:80 -d:100 -p:0.3
-dr -v

An algorithmic framework for representational learning on graphs. [Dec  2 2019]
================================================================================
Input graph path (-i:)=graph/stack_dec12.edgelist
Output graph path (-o:)=emb/stack_dec12.emb
Number of dimensions. Default is 128 (-d:)=100
Length of walk per source. Default is 80 (-l:)=80
Number of walks per source. Default is 10 (-r:)=10
Context size for optimization. Default is 10 (-k:)=10
Number of epochs in SGD. Default is 1 (-e:)=1
Return hyperparameter. Default is 1 (-p:)=0.3
Inout hyperparameter. Default is 1 (-q:)=1
Verbose output. (-v)=YES
Graph is directed. (-dr)=YES
Graph is weighted. (-w)=NO
Output random walks instead of embeddings. (-ow)=NO
Read 53952 lines from graph/stack_dec12.edgelist
Preprocessing progress: 99.61%
Walking Progress: 95.88%
Learning Progress: 99.94%
```

Figure 12: node2vec output from Snap code on github

References

[1] “Scalable Feature Learning for Networks.” *node2vec*, snap.stanford.edu/node2vec/.

[2] Grover, Aditya, and Jure Leskovec. "node2vec: Scalable feature learning for networks." *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016.

[3] Cohen, Elior. “node2vec: Embeddings for Graph Data.” *Medium*, Towards Data Science, 23 Apr. 2018, https://towardsdatascience.com/node2vec-embeddings-for-graph-data-32a866340fef.

CSC 501 Final Project – Jonathan Healy