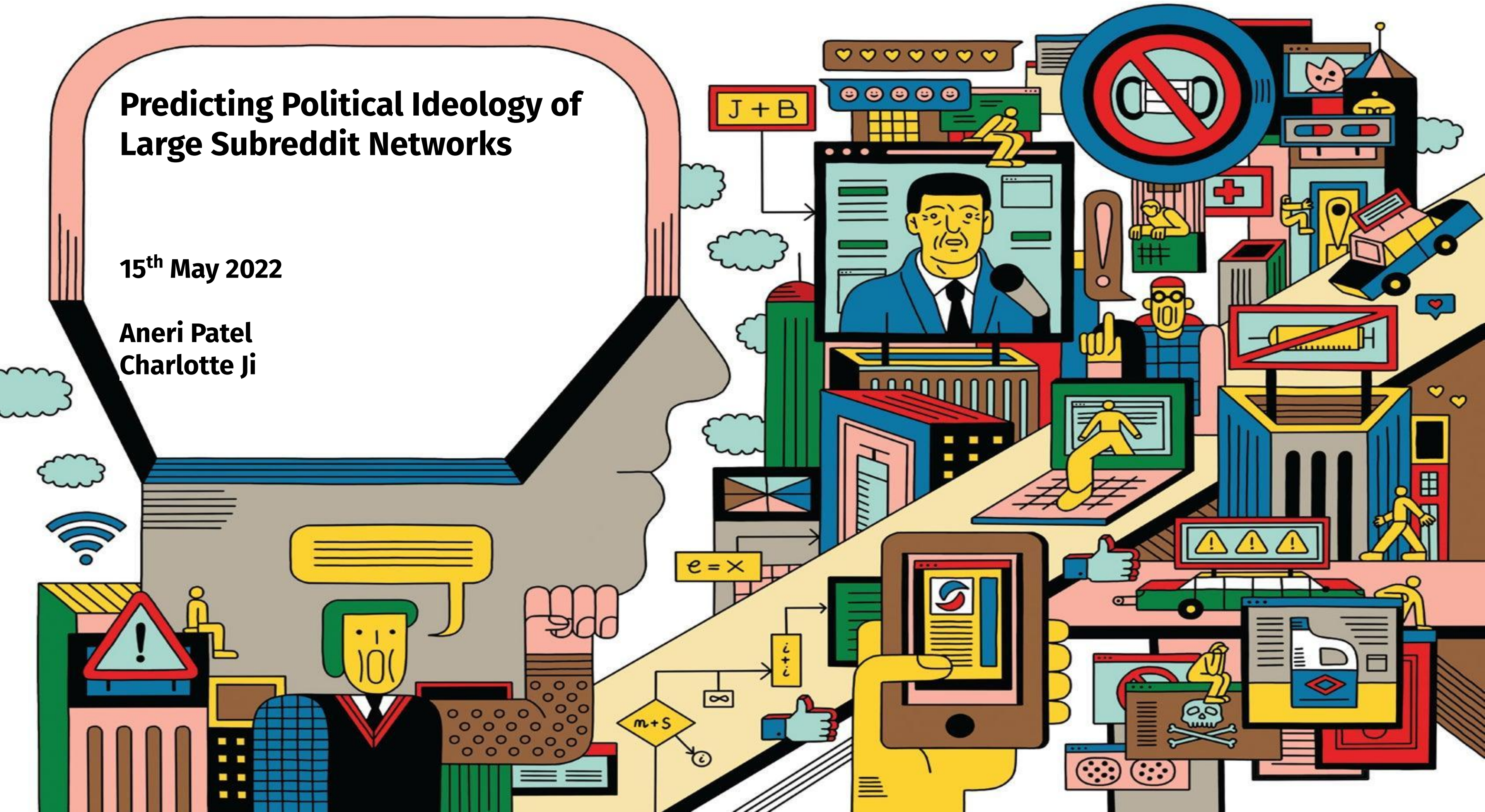
Predicting Political Ideology of Large Subreddit Networks

15th May 2022

Aneri Patel
Charlotte Ji



Problem Motivation

- Politicians and citizens increasingly engage in political conversations on social media outlets. It is important to assess the political ideology of a website (domain) mentioned in a post.
- Social media networks continue to grow in size, accessibility and complexity. Thus, making it increasingly difficult to analyze them.
- In this project, we predict the political ideology (democrat/liberal or republican/conservative) that a domain in a social media network supports.
- We first embed the network [1] for manageable representation and later input this representation to machine learning models to predict political ideology of domains in the network [2].

[1] Nino Arsov and Georgina Mirceva. “Network embedding: An overview”. In:arXiv preprint arXiv:1911.11726 (2019)

[2] Megan A Brown et al. “Network Embedding Methods for Large Networks in Political Science”. In: Available at SSRN 3962536 (2021).

Executive Summary

- Reddit posts from **451 politically-oriented subreddits** (e.g. `r/politics, r/conservative, r/liberal,` subreddits that discuss politics and the news) are collected.
- Only posts that have a positive score (have **more upvotes than downvotes**) are retained. If a domain is shared `w` times in a subreddit then the domain-subreddit node pair is connected with an edge of weight `w` in the network.
- Reddit posts are collected using the `Pushshift API` [3].
- We evaluate `Node2Vec` and 3 predictor models - `ridge regression`, `random forest` and `gradient boosting`. Network embedded features corresponding to labelled nodes are entered into the predictor. We select the best combination of network embedder and predictor for our purpose. Using this process, we surpass the test performance mentioned in our reference paper.

[3] Jason Baumgartner et al. "The pushshift reddit dataset". In: Proceedings of the international AAAI conference on web and social media. Vol. 14. 2020, pp. 830–839.

Background Work

Network embeddings learns continuous feature representations for nodes in a network, which are used for downstream tasks such as node and link prediction.

Node2vec [2] is a random-walk based method that learns a mapping of nodes to a low-dimensional space of features by maximizing the likelihood of preserving network neighborhoods of nodes.

- The random walks involve an exploitation-exploration trade-off:

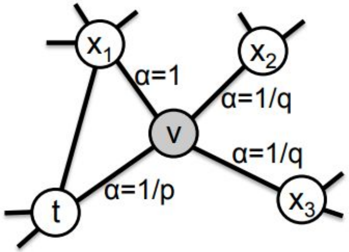


Figure 2: Illustration of the random walk procedure in *node2vec*. The walk just transitioned from t to v and is now evaluating its next step out of node v. Edge labels indicate search biases α.

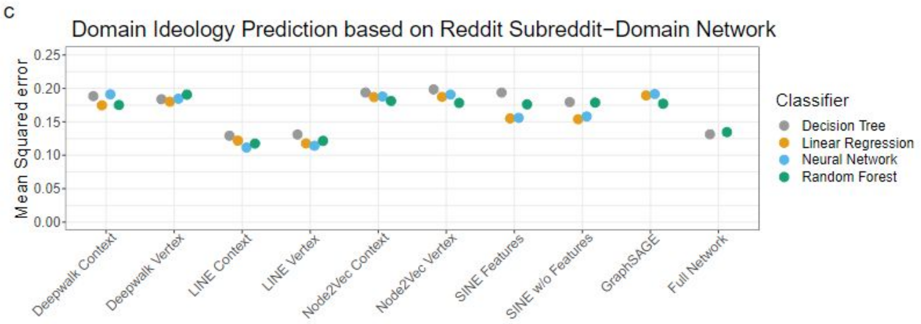
- **Objective:** $$\max_{f} \sum_{u \in V} \log Pr(N_S(u) | f(u)).$$

[1] Nino Arsov and Georgina Mirceva. “Network embedding: An overview”. In:arXiv preprint arXiv:1911.11726 (2019)

[2] Aditya Grover, Jure Leskovec. "node2vec: Scalable Feature Learning for Networks". KDD : Proceedings. International Conference on Knowledge Discovery & Data Mining. 2016: 855–864. arXiv:1607.00653 (2016)

Background Work

Brown et al. [3] compared performance of 5 different network embeddings (DeepWalk, node2vec, LINE, SINE, GraphSAGE) in **ideology scoring** tasks on reddit and twitter networks.



However, they used **default hyperparameters** for each network embedding, while embeddings like node2vec require careful tuning to maximize their performance.

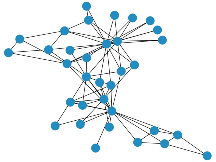
- We want to explore node2vec's [2] potential in achieving better performance.

[3] Megan A Brown et al. "Network Embedding Methods for Large Networks in Political Science". In: Available at SSRN 3962536 (2021)

[4] Ronald E. Robertson et al. "Auditing Partisan Audience Bias within Google Search". In: Proc. ACM Hum.-Comput. Interact.2.CSCW (Nov. 2018). doi: 10.1145/3274417. url: https://doi.org/10.1145/3274417.

Solution Design/Architecture

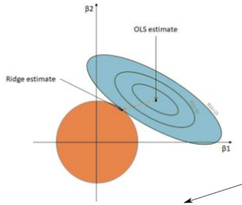
Subreddit	Domain	Edges
1350	catbox.moe	1
	ngur.com	1
	ddit.com	22

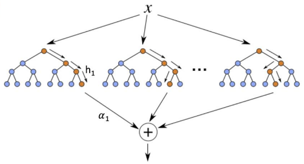


Subreddit-Domain Network Data

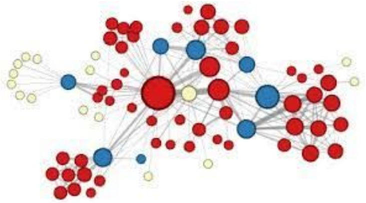
$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$

Node2Vec





Ridge Regression
Random Forest
Gradient Boosting



Political Ideology
Prediction

Approach

Basic Training Framework:

- We train node2vec unsupervised on the **Reddit subreddit to domain dataset**.
  - The dataset contains **1,000,161 edges** (subreddit-domain pairs) and **300,353 nodes**.
- The downstream task for the embeddings is to **predict the ideology of a domain node**. We have ideology scores for **9,804 domains** in the Reddit dataset for evaluation.
  - Following [3], we use domain ideology scores from Robertson et al.'s [4] paper.
- To monitor the embedding performance on downstream task, we split the evaluation domains into **train (64%), validation (16%) and test (20%)**.
- In each epoch of training, we train a predictor on train and report the **mean squared error** on train and validation.

[3] Megan A Brown et al. "Network Embedding Methods for Large Networks in Political Science". In: Available at SSRN 3962536 (2021)

[4] Ronald E. Robertson et al. "Auditing Partisan Audience Bias within Google Search". In: Proc. ACM Hum.-Comput. Interact.2.CSCW (Nov. 2018). doi: 10.1145/3274417. url: https://doi.org/10.1145/3274417.

Approach

Experiment Framework:

- First, we tune for the best node2vec parameters. We early stop at epoch 30 to trade computation resources per experiment (B) for more experiments (n).
  - The model parameters include **p**, **q**, and **random walk length**.
  - We also tune **learning rate** and **batch size**.
- Then we obtain the best performing model by training for a longer epoch (100). We also parallelized the model and evaluated the effectiveness of data parallelism by measuring training time with **1 & 2 RTX8000 & V100 GPUs and 4 V100 GPUs**.
- With the best performing node2vec model, we tune the respective parameters for the 3 types of predictors and identify the best predictor.
  - We tune **alpha** for Ridge, **n_estimators** for RF, and **n_estimator** & **learning rate** for GB.

Implementation Details

```python
'''
Predict the political ideology of each node in the subreddit-domain network using:
1. node2vec for creating(embedding) feature representations of the nodes of the graph
2. ridge regression for predicting the political inclination of each node in the graph
'''
def predict(p=1, q=1, walk_length=10, classifier="Ridge", start_epochs=0, num_epochs=30,
            batch_size=128, learning_rate=0.01):

    model = Node2Vec(data.edge_index, embedding_dim=128,
                     walk_length=walk_length, context_size=10,
                     walks_per_node=10, num_negative_samples=1,
                     p=p, q=q, sparse=True).to(device)

    optimizer = torch.optim.SparseAdam(list(model.parameters()), lr=learning_rate)
    loader = model.loader(batch_size=batch_size, shuffle=True, num_workers=8)
```

PyTorch geometric

scikit learn

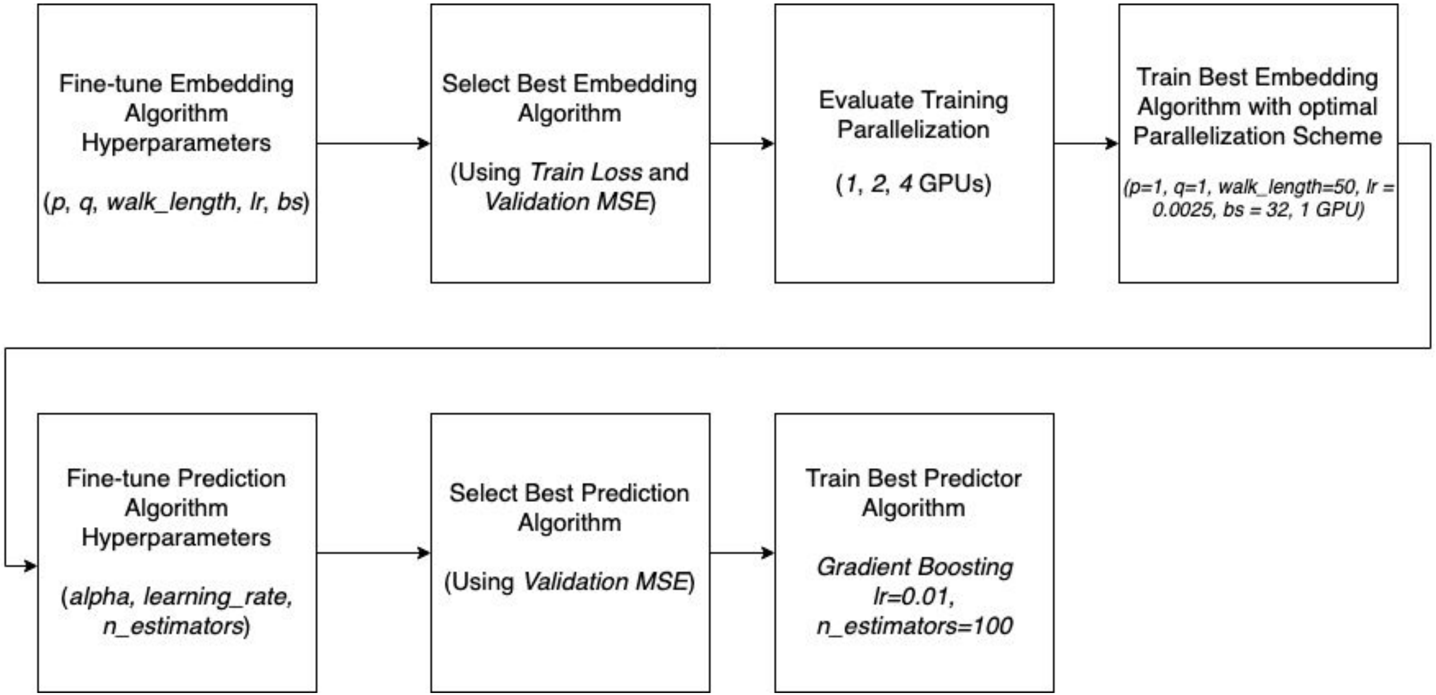
```python
def train():
    model.train()
    total_loss = 0
    for pos_rw, neg_rw in loader:
        optimizer.zero_grad()
        loss = model.loss(pos_rw.to(device), neg_rw.to(device))
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
#       scheduler.step(loss)
    return total_loss / len(loader)
```

```python
@torch.no_grad()
def test(classifier="Ridge"):
    model.eval()
    z = model()

    if classifier=="Ridge":
        clf = Ridge(alpha=0.01).fit(z[train_x].detach().cpu().numpy(), train_y)
    elif classifier=="RF":
        clf = RandomForestRegressor(max_depth=10, random_state=0).fit(z[train_x].detach().cpu().numpy(), train_y)
    elif classifier=="XGB":
        clf = xg.XGBRegressor(objective ='reg:squarederror',
                              n_estimators = 10,
                              seed = 0).fit(z[train_x].detach().cpu().numpy(), train_y)

    val_preds = clf.predict(z[val_x].detach().cpu().numpy())
    train_preds = clf.predict(z[train_x].detach().cpu().numpy())
    return mean_squared_error(train_y, train_preds), mean_squared_error(val_y, val_preds)
```

Experimental Design Flow



Experimental Design Flow

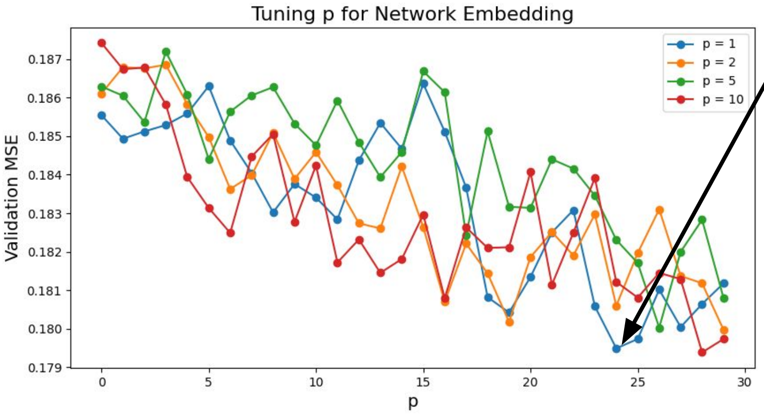
We perform experiments to fine-tune each of these hyperparameters. While fine-tuning a specific parameter, all other parameters are set to defaults as mentioned in the tables below.

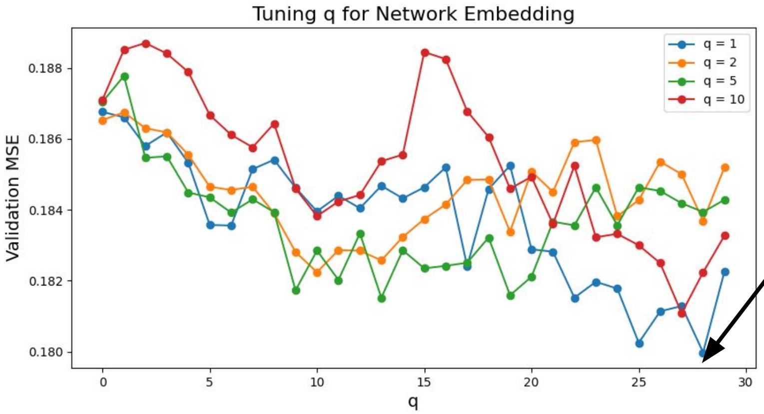
Embedders		
Hyperparameter	**Default**	**Tuning Range**
p	1	[1, 2, 5, 10]
q	1	[1, 2, 5, 10]
random walk length	10	[10, 20, 50]
epochs	30	
learning rate	0.01	[0.001, 0.01, 0.1, 1]
batch size	128	[32, 64, 128, 256]

Predictors		
Hyperparameter	**Default**	**Tuning Range**
Ridge - alpha	0.01	[0.001, 0.01, 0.1, 1]
Gradient Boosting - n_estimators	100	[50, 100, 200, 500]
Gradient Boosting - learning_rate	0.1	[0.001, 0.01, 0.1, 1]

Experimental Evaluation

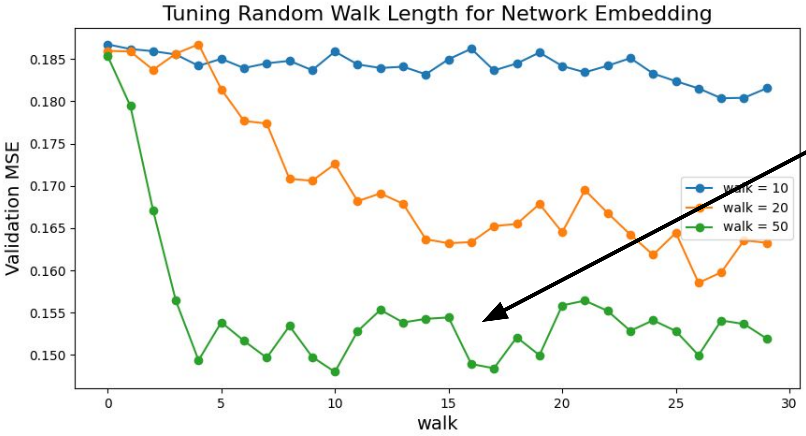
p=1





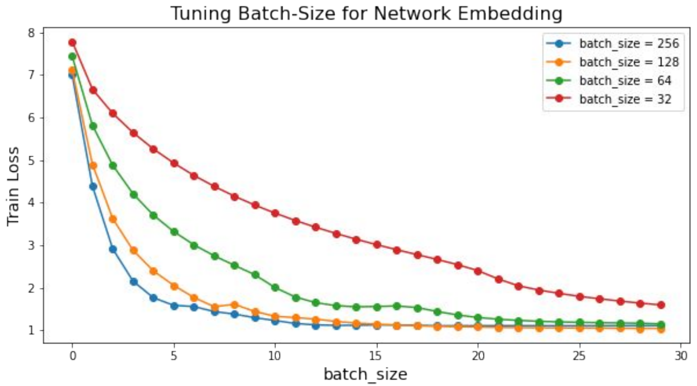
q=1

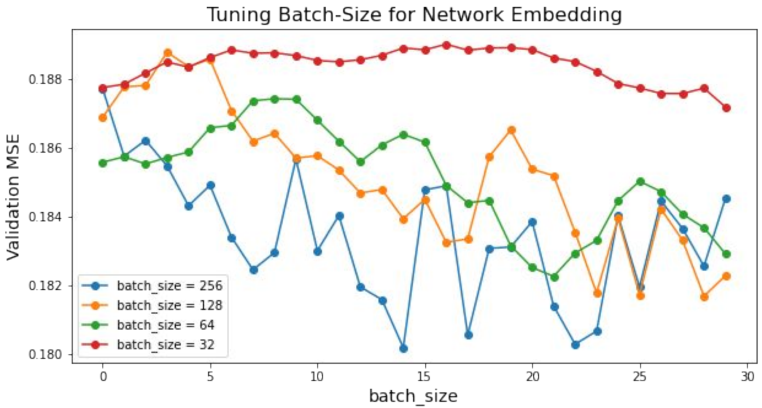
- p=1 and q=1 imply the model is using unbiased random walk while p≠1 and q≠1 indicate biased random walks.
- Thus, we find that unbiased random walks perform better than biased random walks and increasing walk length - i.e. increasing the neighborhood of influence of a node significantly impacts the embedding capability.

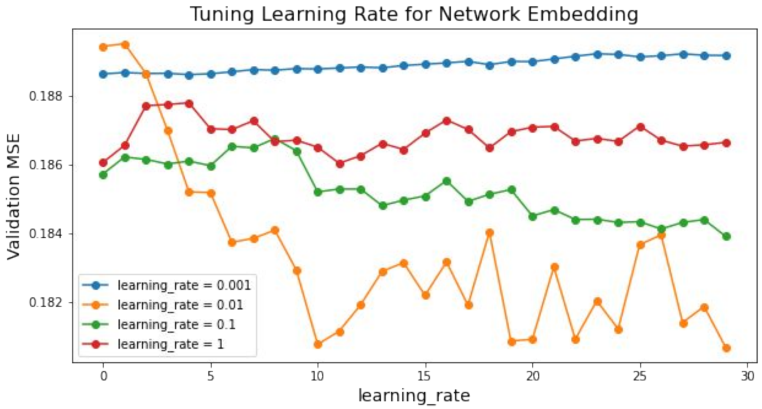


walk length: 50

Experimental Evaluation



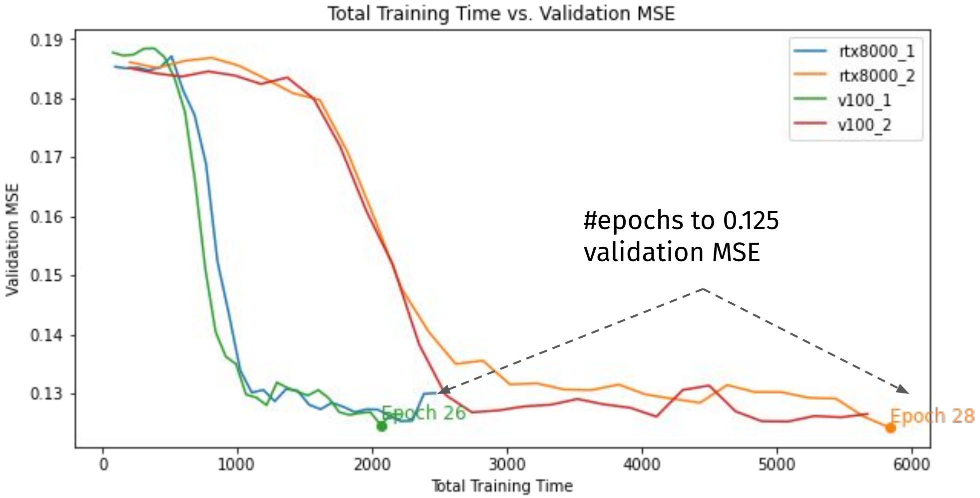


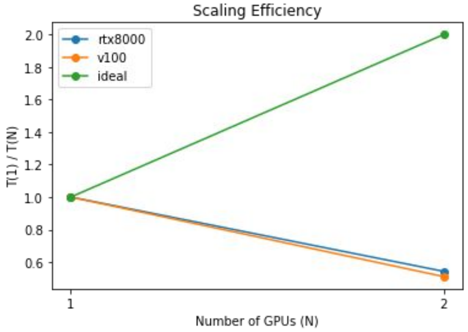


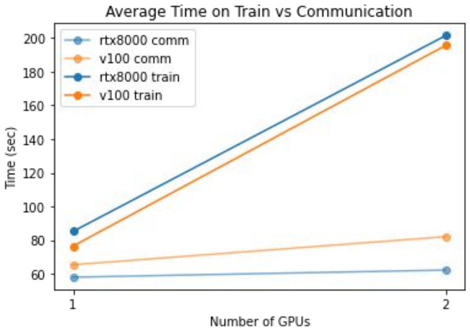
- Optimal learning rate for batch size 128 was found to be 0.01
- To find optimal batch size, we scale learning rate with (bs/128)x0.01 for each batch size experiment
- With higher batch sizes, we find that the model gets stuck at multiple steep local minima. We find that batch size 32 works better as it converges fastest to a flatter minima for MSE.

Experimental Evaluation

- We parallelized training on 1 and 2 RTX8000 and V100 GPUs using PyTorch's `DataParallel`.
- We observed that using more GPUs increased total training time.







- We separately measured **train time** (forward + backward + optimizer step) and **communication time** (data loading and CPU-GPU transfer).
- Results show that train time actually dominates the increase in total time.
- We conclude that the current pytorch geometric implementation of node2vec does not support data parallelism well.

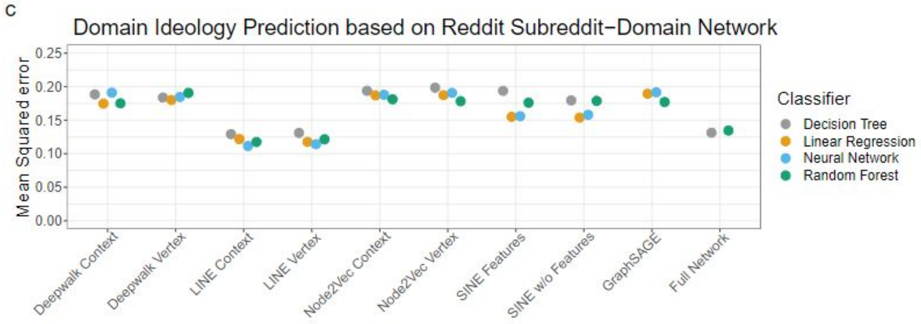
Experimental Evaluation

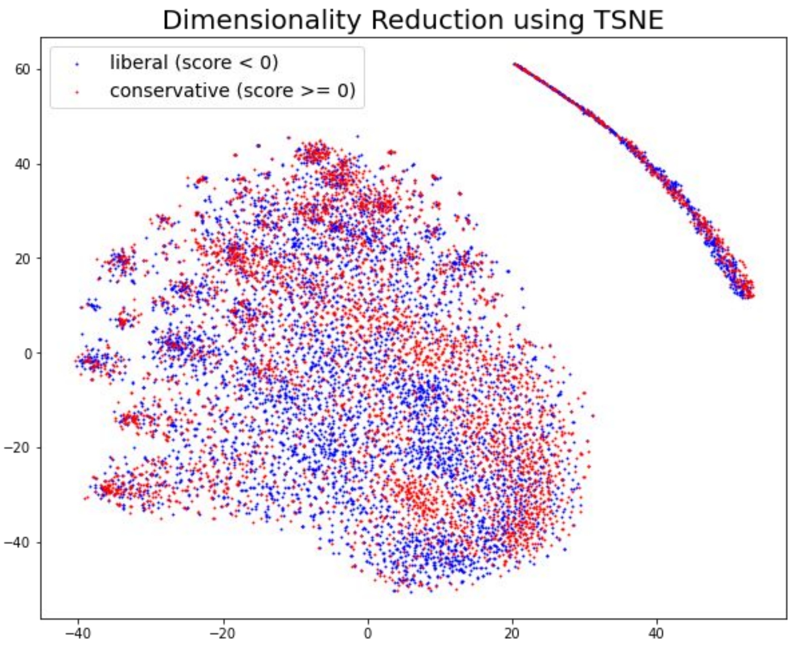
We visualized the best performing embeddings for the 9,804 labeled domains using the results of dimensionality reduction with **T-SNE**.

Political ideology score < 0

Political ideology score >= 0

We observe certain clusters that are distinctively liberal or conservative.



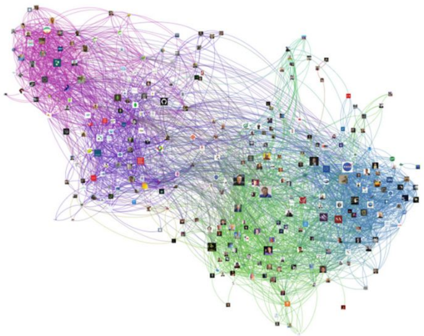


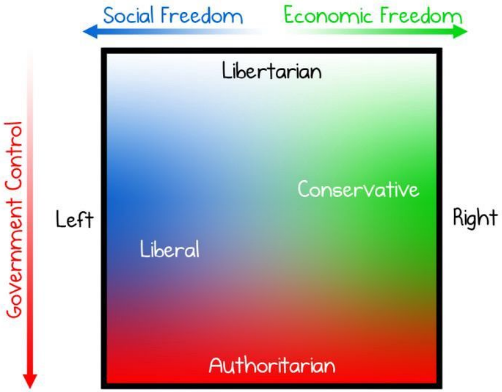
The best performing embedding model + predictor achieved a test MSE of **0.1262**, which outperformed node2vec in Brown et al. [3] and is comparable to their best test performance achieved by LINE [5].

[5] Jian Tang et al. “LINE: Large-scale Information Network Embedding”. In: Proceedings of the 24th International Conference on World Wide Web (May 2015). doi: 10.1145/2736277.2741093. url:http://dx.doi.org/10.1145/2736277.2741093

Technical Challenges

- **Dataset**
  - The dataset is a relatively large graph containing 1 million edges.
    - However, the graph still fits in memory and we are able to create in-memory PyTorch Geometric dataset out of it.
  - It also contained heterogeneous types of nodes (subreddit and domain).
    - Due to the limitation of node2vec, we do not differentiate node types and treat all nodes as homogeneous.
- **Metric**
  - Problem statement (predicting political ideology) is simplified substantially by converting it to a scalar between -1 and 1

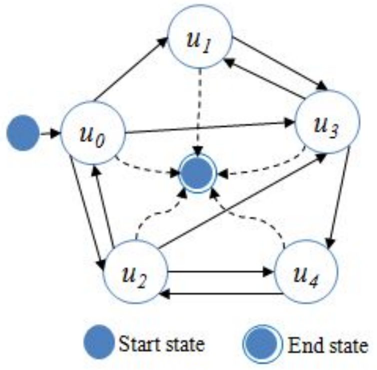




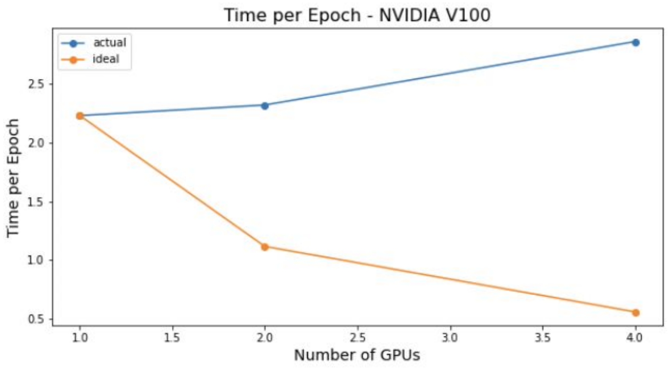
https://dzone.com/articles/process-massive-data-graphs-on-your-pc-with-this-d
https://darbymatt.medium.com/political-ideologies-2b5bf9f0ffc1

Technical Challenges

- **Method**
  - There is a tradeoff between getting better representations of node neighborhoods through longer random walks, and computation resources.
  - We limit the number of epochs to efficiently find the best model configurations.
  - Random walks are computation and memory intensive
    - The data loading takes longer, which affects GPU utilization and overall training time.
    - We are also limited to testing walk lengths up to 50 due to memory constraints.



- **Low Parallelizability in PyTorch - Geometric implementation of Node2Vec**
  - We were not able to speed up training using multiple GPUs as the time per epoch increased on scaling to 1,2, and 4 NVIDIA V100 GPUs instead of decreasing as expected.



G. Liao, X. Huang, M. Mao, C. Wan, X. Liu and D. Liu, "Group Event Recommendation in Event-Based Social Networks Considering Unexperienced Events," in *IEEE Access*, vol. 7, pp. 96650-96671, 2019, doi: 10.1109/ACCESS.2019.2929247.

Conclusion

- Node2Vec can achieve better performance at ideology scoring tasks after hyperparameter tuning.
- A longer random walk is more important for achieving better performance, compared to the exploitation-exploration ratio.
- Our best performing node2vec + predictor model outperformed node2vec from Brown et al.’s [3] paper, and reached best performance achieved by LINE on test.
- Current PyTorch Geometric implementation of node2vec doesn’t support data parallelism very well.

Code & Documentation

https://github.com/mginabluebox/idls_final_project