Project Report: On Representation Power of Character Network Feature Extraction and Inferences

Github Repo: https://github.com/annazhu1996719/CS224W-project.git
Zhining Zhu ANNAZHU@STANFORD.EDU
Kuangcong Liu CECILIA4@STANFORD.EDU
Zhen Qin ZHENQIN@STANFORD.EDU

1. Introduction

The complex structures of social networks inherently embed rich information, and thus social graphs have always been serving as a starting point for feature extractions. With representative features extracted from social networks, Machine Learning will act as a powerful tool in tasks ranging from regression and classification, to clustering and others. Our project will focus on combining feature extractions on social networks with Machine Learning. Our goal is to analyze methods of extracting representative features for social networks, and to understand the usefulness of the extracted features in making further inferences on the networks.

In order to accomplish this goal, we will use Character Movie Network, which is a social graph on relationships between characters in movies, as a representative of small scale social networks. And we have defined two specific tasks to evaluate the usefulness of network features in realistic settings, which is to predict IMDB movie ratings and genre from character movie networks. Furthermore, we target to gain insights on the relative significance of features extracted by looking into the results and weights of predictions made from Machine Learning methods.

2. Relevant Prior Work

There are several relevant papers analyzing feature extraction from different perspectives. One of them is *Mining and Modeling Character Networks* (3), which explores the usefulness of hand-picked basic graph features such as clustering coefficient, modularity, pagerank, motifs and cliques. Similar to their experiments which use these features to predict if a character network is real or fake, our experiments also utilizes the same features they are suggesting for prediction. On the other hand, *Representation Learning on Graphs: Methods and Applications* (7) emphasizes more on recent research progress on automating the process of feature generation by an encoder-decoder framework that tightly connects with Machine Learning. Inspired by their idea of node to vector representation, we developed our algorithm of complete graph to vector representation. In addition, *Exploiting character networks for movie summarization* (5) provides us with one important feature of the network by analyzing the score of each character, each of which contains information on several properties of the network such as degrees and distances. Their technique of identifying the main character provides us approaches to extract features related to main character nodes.

3. Dataset

In this project, we use two datasets:

1. *Moviegalaxies (http://moviegalaxies.com/)*: A collection of around 800 character networks extracted from movie scripts. Each character network is a weighted undirected graph with weights representing the interactions and relationships between pairs of characters. Each movie is also associated with its IMDB Id, through which we can join the character networks dataset with the IMDB movie dataset

2. *IMDB Movie Dataset:* The IMDB Movies Dataset contains information about 14,762 movies. It contains useful movie features including IMDB-rating, religion, duration, director, language, genres and so on. The IMDB-rating and genre are what we will target to predict in our experiments.

3.1. Data Preprocessing

1. *Moviegalaxies*: Each Character Movie Network comes in as an xml file with information on nodes, edges, and edge weights. To make use of them, we first scrap all the useful information into csv files, and then load each network as two directed weighted networks(PNEANet object in SNAP) into python. A subtlety here is that ideally we would like to use undi-

rected weighted networks to best represent the data. However, SNAP only supports either undirected unweighted graphs, or directed weighted networks. As a compromise, for each movie m, we created a G_{dir} and a G_{undir} PNEANet in the following way:

G_{dir}: For each connection between character u and v in movie m, create one weighted edge of weight w from u to v, and one weighted edge of weight w from v to u, where w is the weight of the connection between character u and v.

G_{undir}: For each connection between character u and v in movie m, create a weighted edge of weight w from u to v, where w is the weight of the connection between character u and v.

When computing different network properties and statistics, we are able to use whatever more convenient, G_{dir} or G_{undir}. For example, G_{undir} is more suitable in computing degree of the network, while G_{dir} is more appropriate in computing the diameters of the network.

2. *IMDB Movie Dataset*: Based on the IMDB ID of each character network's movie meta-data, we query out all 659 movies that have character graph information. Then factorize non-numerical features into integers. Finally, join the selected IMDB Dataset and graph properties with IMDB ID as index.

For regression experiment, our prediction target is IMDB-rating. IMDB-rating is a decimal ranging from 0 to 10 with step size 0.1. From the total 659 movies, the maximum rating is 9.3 and the minimum rating is 4.3.

For classification experiment, out prediction target is movie genre. A movie can have arbitrary number of genres. In IMDB dataset there are 27 total genres, out of which we choose 12 that have more than 3% of the 659 movies under the genre. Then encode the genre in to a vector of 0 and 1's, with 1 representing the movie is in this genre and 0 otherwise. In *Table 1* we represent the genre count distribution of 659 movies.

Genre Count	0	1	2	3
Movie Count	2	84	229	344

Table 1. Histogram of genre count in our dataset

4. Approaches and Preliminary Findings

In high level, our approach is to first find an abundant set of features to represent the networks, and then pass them as inputs to machine learning frameworks.

Network Property	Network Used	Formula
num_characters	G_{undir}	$\|V\|$
num_edges_unweighted	G_{undir}	$\|E\|$
weighted_degree_sum	G_{undir}	$\sum_{i=1}^{\|V\|} d_i$
weighted_degree_max	G_{undir}	$\max_{i=1}^{\|V\|} d_i$
weighted_degre_avg	G_{undir}	$\frac{1}{\|V\|}\sum_{i=1}^{\|V\|} d_i$
clustering_coefficients	G_{undir}	$\frac{1}{\|V\|}\sum_{i=1}^{\|V\|} \frac{2e_i}{d_i(d_i-1)}$
density	G_{undir}	$\frac{\sum_{i=1}^{\|E\|} w_i}{\|V\|}$
max_shortest_path	G_{dir}	$max_{i\neq j\in\{1\ldots\|V\|\}^2} p_{ij}$
avg_shortest_path	G_{dir}	$\frac{\sum_{i\neq j\in\{1\ldots\|V\|\}^2} p_{ij}}{\|V\|^2}$

Table 2. Basic Network Properties
d_i := out degree of the i-th node
w_i := weight of i-th edge
e_i := num of edges between neighbors of node i
p_{ij} := length of shortest path between node i & j

4.1. Graph Representations

We mainly experiment with five types of network features in representing the Movie Character Networks. We will describe below how to extract each types of the features in details, and summarize the initial findings and statistics.

4.1.1. Basic Network Properties

The basic properties we studied on include number of characters, number of edges, total weighted degrees, max weighted degree, average weighted degrees, clustering coefficient, density, diameter (max length of shortest paths), and average length of shortest paths. The formulas are specified in *Table 2*.

See the distribution of basic network properties over the 773 Character Movie Networks in *Figure 1*.

We have also attempted to calculate the number of connected components and the size of the largest connected component for each Movie Character Network. However, it turns out that there is only one connected component for each network, and thus the size of the largest connected components is just the total number of nodes. As a result, we discard these two features to avoid highly correlated features.

4.1.2. Motif counts

Motifs are potentially very useful in interpreting Character Movie Networks since they embed the relationship between characters. Therefore, we also look at the counts of size 3 and 4 motifs in each network. Because Character Movie Networks are undirected, we limit our study to undirected version of motifs (*Figure*

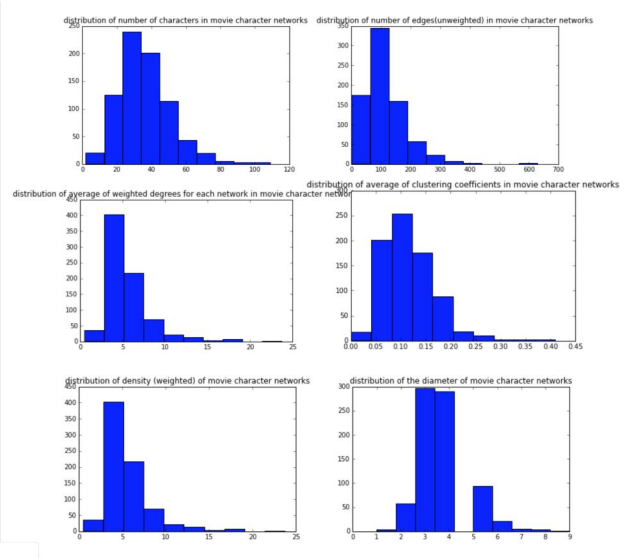

Figure 1. distribution of the basic network properties over the 773 Character Movie Networks

2).

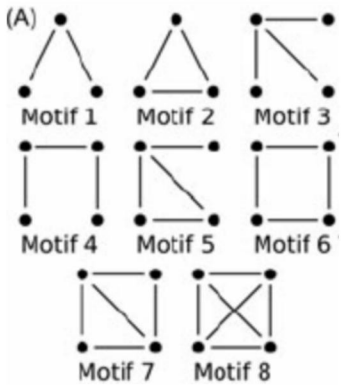

Figure 2. Undirected motifs of size 3 and size 4

The distribution of proportion of each motif over the networks is shown in the boxplot (*Figure 3*). it is noticeable that Motif 3 occurs most frequently, which implies that having central characters that connect with a lot of other characters is a universal pattern in movies.

4.1.3. Representing Clustering and Communities

Since we are experimenting with social networks, the community and clustering structures are worth extensive study of. We mainly use the following two methods to represent the microstructure:

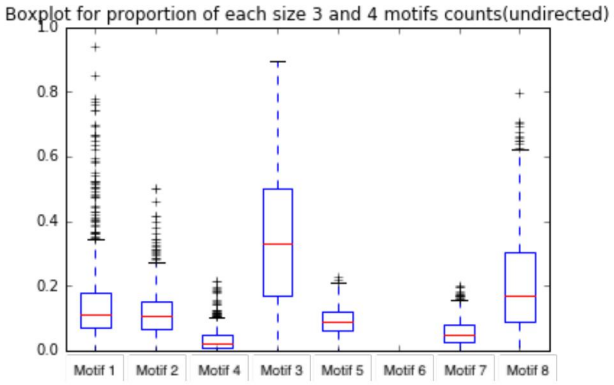

Figure 3. Distribution of proportion of each motif over the 773 networks

(a) K-core Features

A k-core of a graph G is a maximal subgraph of G in which all vertices have degree at least k, and it represents the clustering structure of the graph. In our case, the number of k-core of a movie character network means the number of closely-connected small communities of characters with appropriate grouping criteria. In other words, number of k-core can be interpreted as the number of ways we can group the characters into sub-communities in which everyone has interactions with at least k other people.

For prediction purposes, we extract the number of k-cores for $k \in \{1, 2, 3, 4, 5\}$ for each Character Movie Network as features.

(b) Modularity

Modularity is a measure of how well a network is partitioned into communities. Formally,

$$Q(G, S) = \frac{1}{2m} \sum_{s \in S} \sum_{i \in s} \sum_{j \in s} (A_{ij} - \frac{k_i k_j}{2m})$$

.

We first use two community detection algorithms (Girvan-Newman algorithm(4) and Clauset-Newman-Moore algorithm(1)) to partition graphs into communities, and then compute the modularities of the resulting communities from two community detection techniques respectively.

In order to get an intuitive understanding of how community detection algorithms are performing, as well as what graphs have high modularities, we choose two typical Character Movie Networks to make some visualizations. We choose network 3 and 5, of which one has high modularity and the

other has low modularity. *Table 3* displays the modularities of network 3 and network 5 after performing the two community detection algorithms, and *Figure 4* visualizes the original network structures of network 3 and network 5. We also use colors of nodes to label the communities detected by the two algorithms respectively. Obviously, although the two algorithms divide different nodes to communities, there is a clear pattering of clustering in network 3 with both algorithms, while it is hard to find any clearly-divided communities from network 5. As a result, detecting communities in network 5 results in a low and even negative modularity.

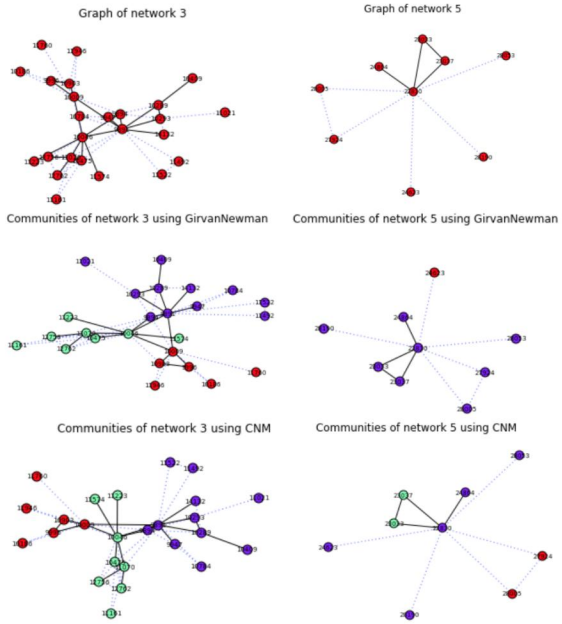

Figure 4. Community Detection

Network	Girvan-Newman	CNM
3	0.47261204	0.47261204
5	-0.005	0.16

Table 3. Modularities of network 3 and 5 using different community detection algorithms

4.1.4. Egonet

For each character network, we extract the egonet features of the main character. The specific algorithm involve two steps:

(a) Identify the Main Character

To find the main character, we combine several centrality measures.

In details, for each Movie Character Network, we first compute the Closeness Centrality, Betweenness Centrality, and Page Rank score for each node, whose formal definitions are elaborated in *Table 4*. Then, we select the top 2 central nodes based on each measure. After that, with the output of the above three measures, we define the main character of the network to be the node that is identified as "central" most often, and break ties randomly.

Centrality	Formula
Closeness	$c_{clos}(x) = \frac{1}{\sum_y d(y,x)}$
Betweenness	$c_{bet}(x) = \sum_{y,z \neq 1x, \sigma_{yz} \neq 0} \frac{\sigma_{yz}(x)}{\sigma_{yz}}$ σ_{yz}: num of shortest path from y to z $\sigma_{yz}(x)$: num of such paths pass through x
Page Rank	$c_{pr}(x) = \alpha \sum_{y \to x} \frac{c_{pr}(y)}{d_{out}(y)} + \frac{1-\alpha}{n}$

Table 4. Formal Definition of Centrality Measures

Again, *Figure 5* visualizes the central nodes in network 3 and network 5 with the three centrality measures. In network 3, all three centrality measures find the same two nodes as the central nodes, so each central node is selected for exactly three times. Therefore, we break tie randomly and define node 8691 as the "main character". In network 5, however, all three measures regard node 22830 along with a different node as the top 2 central nodes, so we pick node 22830 as the "main character" because it is the only overlap among three groups of central nodes.

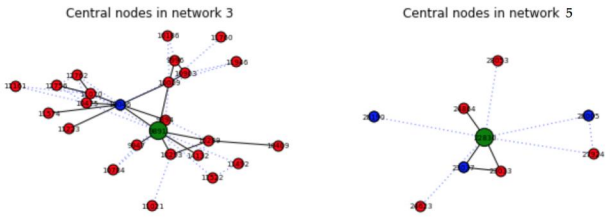

Figure 5. Central Characters: blue nodes are central nodes found by the three centrality measures (may have overlappings); green node is the "main character" following our definition

(b) Compute the Egonet features

When calculating the Egonet features, we combine basic features with recursive features of the main character to obtain more comprehensive structural information.

For each node i, basic features are the degree of node i, the number of edges in the egonet of node i and the number of edges between node i's egonet and the rest of the graph, as $V_i^{(0)}$ shown below.

Recursive features concatenate each node's basic features with the mean and sum of their neighbors features, and we denote it as $V_i^{(1)}$. We repeat this process for twice and get $V_i^{(2)} \in R^{27}$, to generate more information about this network. We select the vectors of our two main characters as parts of out features of the whole graph.

$$V_i^{(0)} = [d_i, in_i, out_i]$$
$$V_i^{(1)} = [V_i^{(0)}, \frac{1}{|N(i)|} \sum_{j \in N(i)} V_j^{(0)}, \sum_{j \in N(i)} V_j^{(0)}]$$

4.1.5. Network Embeddings

(a) Node2Vec

Grover *et al* (2) proposed a method to represent each node with a low-dimensional vectors of features that maximize the likelihood of maintaining network properties. If two nodes have similar network neighborhoods, then their vector representations should also be close.

After exploring all the character networks, we find out there is only one connected component in each network and the average number of nodes is around 30, which means our networks are relatively small. Then it is more intuitive to learn more about the local features rather than the global features of our networks.

Therefore, we choose the return parameter $p = 0.1$ which gives high probability in random walk for each node to return back to the previous node. Also select the In-out parameter $q = 1$. This will be likely to the Breadth First Search method and generate more helpful information about neighborhood of each node.

For each network, after getting the vector representations of each node, we simply calculate the sum and average of all the node embeddings, and then add those result vectors to our final feature representation of each graph.

(b) Anonymous Walk Embedding Feature-Based (AWE-FB)

Anonymous Walk Embeddings (6) also proposes insightful method for graph embedding, which is able to reconstruct graph information as a whole. There are two approaches to embedding anonymous walk, the feature-based and the data-driven embeddings.

AWE-FB embedding of a graph G is a vector, with the size of all possible anonymous walks of some length we choose. i-th element in the vector is the probability of i-th anonymous walk a_i on graph G:

$$V = [p(a_1), p(a_2), ..., p(a_n)]$$

When the network is large or the length of anonymous walk is long, we couldn't find all possible anonymous walks, and therefore they use a Monte-Carlo sampling method to approximate the true distribution. Under the situation of Character Network, we choose anonymous walk length of 7 and sampling 10000 examples for each graph.

(c) Anonymous Walk Embedding Data-Driven (AWE-DD)

Anonymous Walk Embeddings (6) proposes AWE-DD method as a solution to the case when a network has sparse feature-based vector. This method is really similar to the method of finding representation vectors for paragraph in a text document.

For each node u in a graph G, AWE-DD (6) first samples user-specified k number of random walks starting from u, such as $r_1, r_2, ..., r_k$. Find the corresponding anonymous walks representation$a_1, a_2, ..., a_k$ and select a_i as target anonymous walk. Then calculate the probability of that target anonymous walk given the rest anonymous walk and the representation vector of this graph d, which is $p(a_i|a_1, ..., a_{i-1}, a_{i+1}, ..., a_k, d)$. Try to maximize this probability for all nodes in all graphs by finding best representation of anonymous walks for all graphs and best representation vector of each graph.

(d) Compare node2vec, AWE-DD and AWE-FB

Figure 6 shows us the similarities between second network and other networks. The similarity is calculated by distance between the vector representations of networks. The black edges indicate edge with weight more than 1 and the blue dashed edges indicate weight less or equal to 1.

In node2vec algorithm, Network 832 has the highest similarity with Network 2, while Network 719

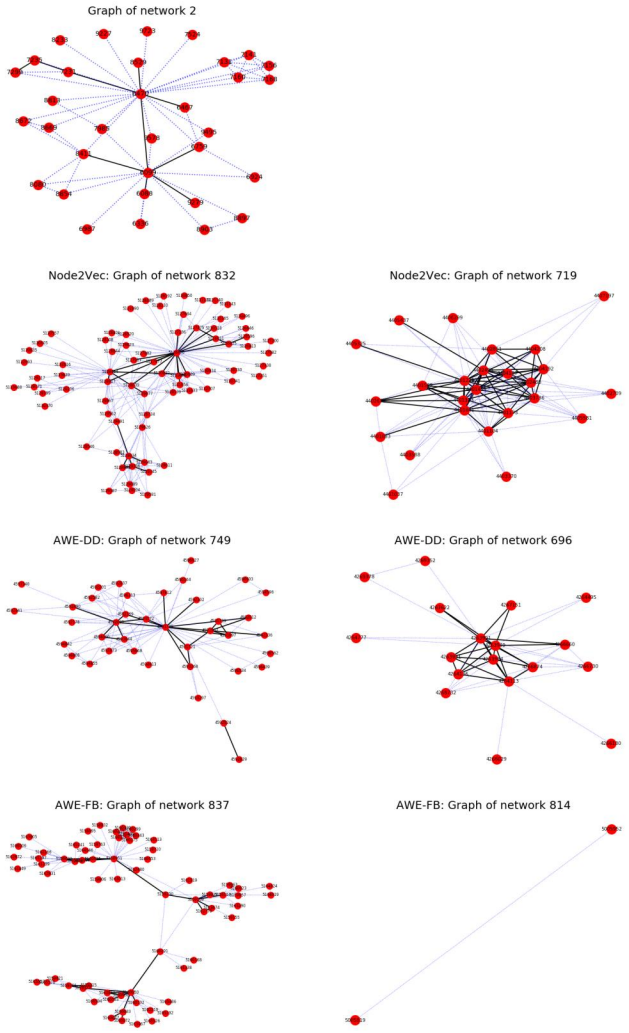

Figure 6. Graphs of Network2, and the most and the least similar networks with Network2 calculated under node2vec, AWE-DD and AWE-FB algorithms

has lowest similarity. If we use AWE-DD method, Network 749 has the highest similarity with Network 2, while Network 696 has lowest similarity. In AWE-FB method, Network 837 has the highest similarity with Network 2, while Network 814 has lowest similarity.

From this figure, we could see that node2vec and AWE-DD seem to perform better than AWE-FB because Network 2 have two relative center nodes with high weights (black edges), which is similar to the structure of network 832 and network 749, while Network 837 seems to have many center nodes with high weights (black edges).

Algorithms	Kendall's τ	Spearman's ρ
node2vec, AWE-DD	0.01	0.016
node2vec, AWE-FB	-0.028	-0.041
AWE-DD, AWE-FB	-0.026	-0.039

Table 5. Rank Correlation of three algorithms

After calculate the similarities rank of all networks with Network 2 by using these three algorithms, we could calculate the rank correlation by 2 methods, Kendall's Tau and Spearman's Rho. The results are shown in *Table 5*. From this table, we could see that the correlation of rank vectors between algorithms are really low and there are even negative correlations, which means that each algorithm predicts similarities with Network 2 in quite different ways, and the main cause for this problem is probably lack of data.

4.2. Predictions

4.2.1. Cross validation

Due to limited available data, we use k-fold cross validation for evaluation in all prediction tasks. Randomly divide our training data into 11 equally sized folds, each with 60 graphs, expect last fold with 59. For each iteration, use one of the fold as test set and 10 other folds to train the model. Then average the evaluation result, over 11 iteration as final evaluation.

4.2.2. Genre Classification

Since there are 12 different genres, and each movie can have multiple genres, we need to predict 12 different classification tasks. We use the multi-label classification method to predict each genre one by one and output a prediction vector with 12 dimensions. Each element $y_{\text{predict}_i} \in \{0, 1\}$ represents whether this movie is in i-th genre.

We define two metrics for genre classification evaluation, precision and recall. Precision is the total number of true positives divide the sum of true positives and false positives. Recall is the total number of true positives divide the sum of true positives and false negatives. To be more specific, assume y_{predict_i} is our predicted genre label for ith graph and y_{true_i} is its true label:

$$\text{Precision} = \frac{\sum_i 1\{y_{\text{predict}_i} = 1 \text{ and } y_{\text{true}_i} = 1\}}{\sum_i 1\{y_{\text{predict}_i} = 1\}}$$

$$\text{Recall} = \frac{\sum_i 1\{y_{\text{predict}_i} = 1 \text{ and } y_{\text{true}_i} = 1\}}{\sum_i 1\{y_{\text{true}_i} = 1\}}$$

1. Support Vector Machine (SVM) use hinge loss to minimize the distance to margin. The slack variable ξ_n allows some instances to fall of margin but penalizes them. In addition, with kernels, SVM maps the inputs into a higher dimensional feature space. In our experiment we use Gaussian kernels. SVM has objective:

$$\min \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \forall i, y_i(wx_i + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0$$

2. Single layer neural network (Perceptron) is a linear classifier with weight and bias, and a non linear output activation function. We use a perceptron with L_2 regularization penalty term and apply stochastic gradient descent to update weight and bias.

4.2.3. Rating Regression

We use mean square error(MSE) to evaluate regression tasks on all iterations to predict the final test set. More specifically, we use the following regression methods:

1. Lasso Regression combines least square regression loss with L_1 norm regularization on the weights. Since L_1 norm push non-relevant features' weights to 0. Lasso regression is helpful for subset selection. It's objective is:

$$\min_w \|wX - Y\|_2^2 + \alpha\|w\|_1$$

2. Support Vector Machine Regression(SVR) uses the same principles as the SVM for classification. In the case of regression, the margin (ϵ) is set in approximation to the SVM, with slack variables ξ_n and ξ_n^* for each point as soft margin. More specifically, the optimization problem for SVR is:

$$\min \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

$$\text{s.t. } \forall i, y_i(wx_i + b) \leq \epsilon + \xi_i$$

$$(wx_i + b) - y \leq \epsilon + \xi_i^*$$

$$\xi_i, \xi_i^* \geq 0$$

In our prediction we use SVR with linear kernels, where parameter w can be completely described as a linear combination of the training observations using the equation. With linear kernel, we can easily visualize the weights for each feature after training completes.

5. Results and Analysis

5.1. Classification

We perform classification on 7 different set of features mentioned previously, and compare their recall and precision. We find that there is trade-off between precision and recall while tuning the hyper-parameters. For example, *Figure 7* represents how precision and recall change while sweeping different values of C, the penalty term for SVM.

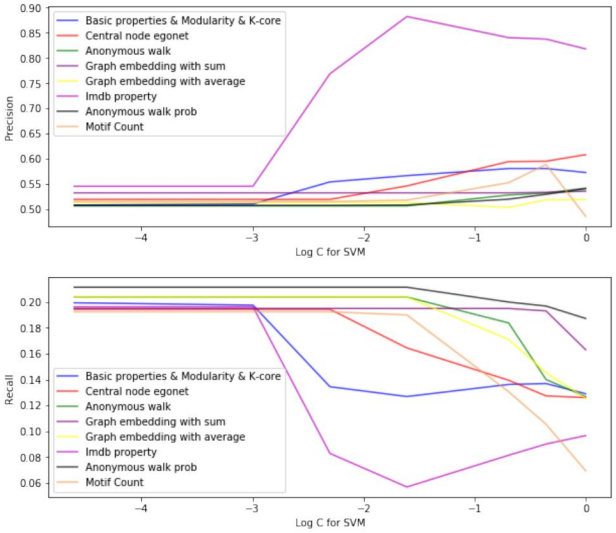

Figure 7. Precision & Recall in SVM

Upon tuning SVM for each feature set, we list out the precision and recall for the SVM with the best hyper-parameters in *Table 6*. We find that although IMDB features set achieves good precision, the model is very conservative in predicting positive values, and therefore the recall is unusually low. On the other hand, two sets of graph features achieve reasonable precision and recall rates. One is "basic network property, modularity, and number of nodes", and the other is "egonet of central nodes".

Features	Precision	Recall
Basic,Modularity,K-core	0.580669	0.136838
Egonet of central node	0.608169	0.125977
Anonymous walk embedding	0.528106	0.183756
Node2Vec sum embedding	0.532468	0.194945
Node2Vec mean embedding	0.513196	0.203783
IMDB features	0.84047	0.081382
Anonymous walk probability	0.540545	0.187217
Motif count	0.517879	0.189929

Table 6. SVM classification on different features

Similarly, we present the best Perceptron classification results in *Table 7*. Overall, Perceptron classifier performs worse than SVM classifier. This is because perceptron is a linear classifier and has a weaker representation power than SVM. What's interesting is that on motif feature set, perceptron achieves pretty high precision. This might imply that local structural features are linearly separable in genre classification problem.

Features	Precision	Recall
Basic,Modularity,K-core	0.576271	0.021465
Egonet of central node	0.636587	0.039782
Anonymous walk embedding	0.319654	0.280235
Node2Vec sum embedding	0.468644	0.052745
Node2Vec mean embedding	0.280620	0.203718
IMDB features	0.692090	0.023341
Anonymous walk probability	0.281971	0.329475
Motif count	0.788136	0.022059

Table 7. Perceptron classification on different features

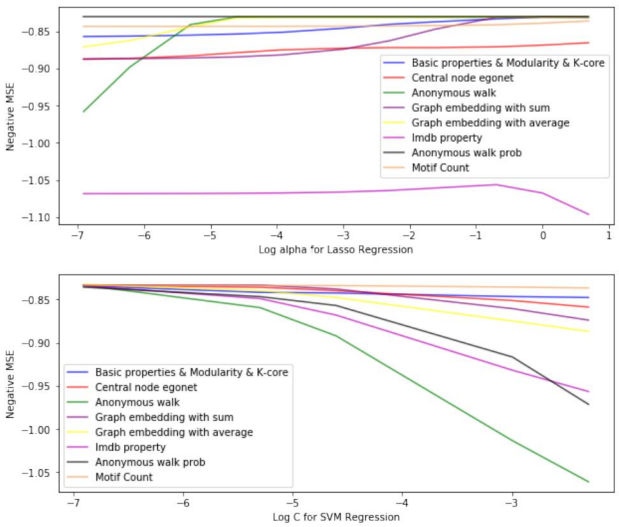

Figure 8. Regression parameter tuning

Features	Lasso	SVR
Basic,Modularity,K-core	0.830236	0.834300
Egonet of central node	0.865048	0.833672
Anonymous walk embedding	0.830521	0.833632
Node2Vec sum embedding	0.830876	0.833323
Node2Vec mean embedding	0.830404	0.832751
IMDB features	1.056176	0.834290
Anonymous walk probability	0.830521	0.835874
Motif count	0.835694	0.833596

Table 8. Mean Square error on different features

Overall, for both SVM and Perceptron, features that are mesoscale characterizations of the networks, such as motifs, and egonet features turn out to be more useful. One justification could be that in movies, which is a miniature of the real world social networks, complexity within small close communities are what really distinguish one network from another.

5.2. Regression

Similarly, we first tune the hyper-parameters, regularization coefficient, alpha, for Lasso regression and penalty term, C, for SVM Regression. *Figure 8* shows how Mean Square error(MSE) changes during tuning. In addition, *Table 8* lists the MSE for different feature sets and regression methods after tuning. Among these features, basic graph property, modularity and k-core count with Lasso regression output the lowest MSE.

To further investigate the significance of impact from different network properties, we visualize the regression weight on network features and then select several features with highest average absolute weight from different feature sets. The features we select include:

- From basic graph property: *max_shortest_path*
- 2-core node count
- CNM Modularity & GN Modularity
- From anonymous walk embedding: embed_108, embed_41, embed_26, embed_1
- From central node egonet: node_0_0, node_0_11, node_0_12, node_0_3
- From IMDB dataset: year, nrOfWins, nrOfNominations.

The final MSE with theses features is 0.8640806810970486 for Lasso Regression and 0.8212124857898835 for SVM Regression. *Figure 9* displays the average weights from both classifiers for these features.

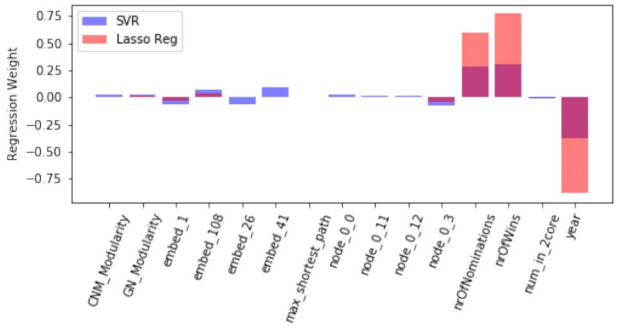

Figure 9. Lasso & SVR Regression weights on best features

One interesting observation from *Figure 9* is that although most of the network features take on nonzero weights, which means that they are at least somewhat relevant to our regression task, all of their weights are relatively small compared to weights of IMDB features. There are many possible explanations.

First, for AWE-DD embeddings, one possibility is that we didn't find a good random walk length or the embedding length for each graph. We set random walk length to be 10 in the experiments and embedding length to be 128 for each graph, however we only have 773 graphs which is pretty few compared to the number of features. More fine-tune on those parameters are needed.

Secondly, the networks we used might be too small. The number of nodes range from 2 to 100. For such small graphs, network features might not have enough variations to be informational, especially for difficult tasks such as regression.

In addition, IMDB features are more explicit than network features. We can think of IMDB features themselves as a prediction result from the network characteristics. From this perspective, they are products of preprocessing from the more lower level network features, and thus regression tasks are more likely to use them directly because they embed more useful information.

5.3. Limitations and Future Directions

5.3.1. Size of data set

The data set we use is too small for a machine learning task, both in terms of number of data points and size of the networks. Also, due to the specialty of Character Network, we couldn't do data augmentation on this dataset as people usually do on many other datasets, because each edge and node in Character Network have unique meaning from movies. Therefore, a future direction could be to find a more comprehensive data set, and redo the feature extractions and predictions on it to get more robust prediction results. More aggressively, instead of finding a bigger data set, we could even augment the original data set by generating similar networks in terms of network embedding.

5.3.2. Overfitting and Lack of Generalization

A lot of the machine learning algorithms we tried are actually overfitting to the training set. Thus, another potential direction is to study on ways of removing features from the feature vectors. While we tried standard ways of feature selection such as Lasso, it would be better to have some algorithms that incorporate domain knowledge of networks in eliminating features.

5.3.3. Automate feature extraction

In feature extraction phases, although we have applied automatic feature encoding methods such as Node2Vec and AWE, most of the features are still manually computed. And the manual computed features, especially features that incorporate community information, like modularity, turn out to work better in our prediction tasks. Therefore, another promising research topic would be to develop more general algorithms for network feature auto-encoding.

References

[1] Cristopher Moore Aaron Clauset, M. E. J. Newman. Finding community structure in very large networks. 2004.

[2] Jure Leskovec Aditya Grover. node2vec: Scalable feature learning for networks. 2016.

[3] Ethan R. Elenberg David F. Gleich Anthony Bonato, David Ryan DAngelo1 and Yangyang Hou. Mining and modeling character networks. 2016.

[4] M. E. J. Newman M. Girvan. Community structure in social and biological networks. 2002.

[5] O-Joun Lee Jai E. Jung2 Quang Dieu Tran, Dosam Hwang. Exploiting character networks for movie summarization. 2017.

[6] Evgeny Burnaev Sergey Ivanov. Anonymous walk embeddings. 2018.

[7] Jure Leskovec. William L. Hamilton, Rex Ying. Representation learning on graphs: Methods and applications. 2017.