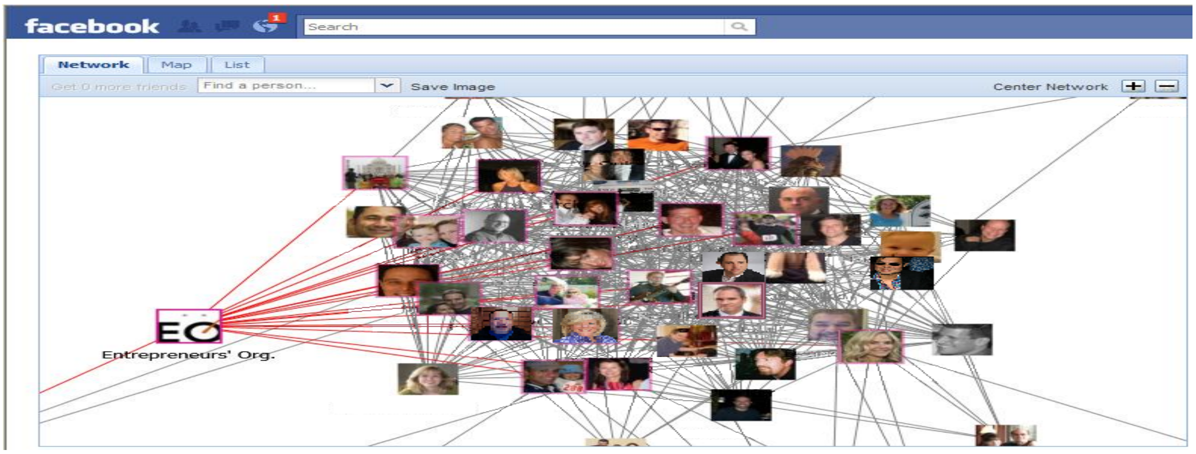
Problem

Networking & connecting with others is one of the most important features among many social media platforms. Facebook and other social media platforms need people to be able to connect users based on their common interests, degrees of separation, etc.



http://www.fmsasg.com/socialnetworkanalysis/facebook/

Data

The dataset I used is from https://snap.stanford.edu/data/ego-Facebook.html with [Number] corresponding to file names: '0','107','348','414','686','698','1684','1912','3437','3980'. It contains the following files:

1. facebook_combined.txt -> Gives all of the connected node pairs

	Node 1	Node 2
0	0	1
1	0	2
2	0	3
3	0	4
4	0	5
...	...	...
88229	4026	4030
88230	4027	4031
88231	4027	4032
88232	4027	4038
88233	4031	4038

88234 rows × 2 columns

2. [Number].featnames -> Gives the feature type, semicolon, followed by specific feature value characterized by a number

	Feature Names
0	0 birthday;anonymized feature 0
1	1 birthday;anonymized feature 1
2	2 birthday;anonymized feature 2
3	3 birthday;anonymized feature 3
4	4 birthday;anonymized feature 4
...	...
219	219 work;start_date;anonymized feature 170
220	220 work;start_date;anonymized feature 171
221	221 work;start_date;anonymized feature 203
222	222 work;start_date;anonymized feature 204
223	223 work;with;id;anonymized feature 205

224 rows × 1 columns

3. [Number].feat -> Gives the row of '1's or '0's for each node indicating whether the node has a feature or not in the corresponding [Number].featnames file rows

	Node Features
0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
1	2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
2	3 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 ...
3	4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
4	5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
...	...
342	343 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
343	344 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
344	345 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
345	346 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 ...
346	347 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 ...

347 rows × 1 columns

Data Wrangling

I discovered there were 21 unique features types by parsing through [no.].featnames files.

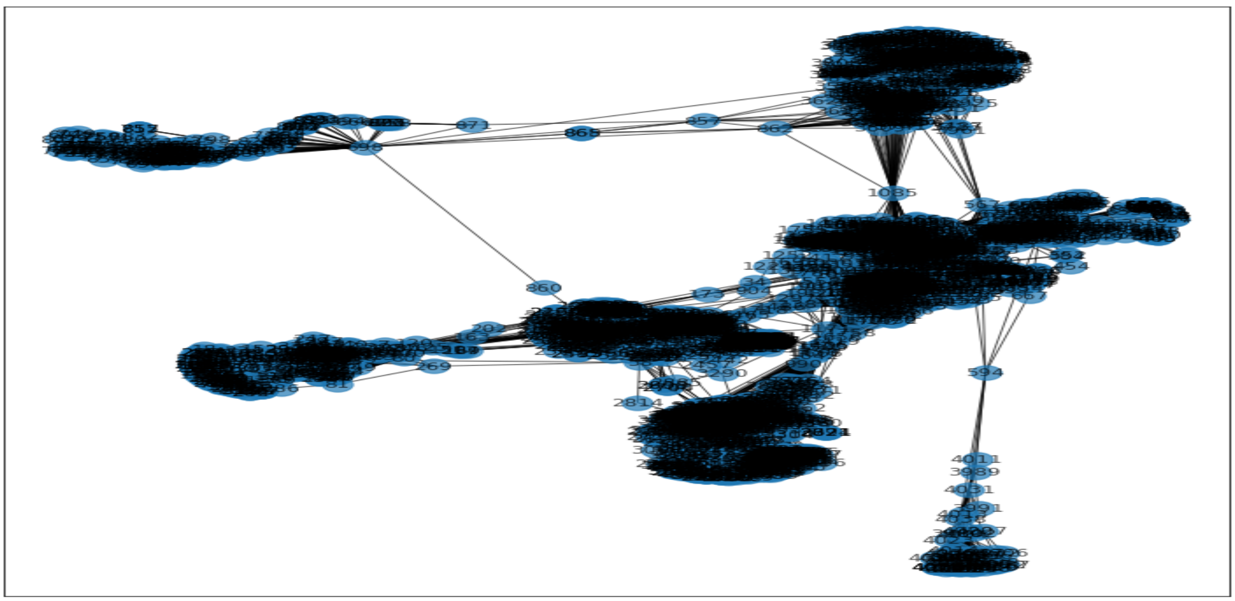
```
21 Unique Features:
{'location;id', 'work;end_date', 'hometown;id', 'work;position;id', 'first_name', 'educatio
n;year;id', 'education;with;id', 'education;type', 'locale', 'work;location;id', 'languages
;id', 'gender', 'education;school;id', 'work;employer;id', 'work;start_date', 'last_name',
'education;degree;id', 'birthday', 'work;with;id', 'education;concentration;id', 'education
;classes;id'}
```

I iterated through different node files and their attributes & values and linked them together in a dictionary.

```
{1: {'gender': [77], 'locale': [127]}, 2: {'education;school;id': [35], 'education;type': [
53, 55], 'education;year;id': [57], 'gender': [78], 'languages;id': [92, 98], 'last_name':
[114], 'locale': [126], 'location;id': [135]}, 3: {'birthday': [7], 'education;concentratio
n;id': [14], 'education;school;id': [34, 50], 'education;type': [53, 55], 'education;year;i
d': [59, 65], 'gender': [78], 'languages;id': [92], 'locale': [127], 'location;id': [137],
'work;end_date': [168, 170], 'work;location;id': [137], 'work;start_date': [164, 202]}, 4:
{'education;school;id': [50], 'education;type': [53, 55], 'education;with;id': [56], 'gende
r': [78], 'locale': [127]}, 5: {'education;school;id': [49, 50], 'education;type': [53, 54]
, 'education;year;id': [65], 'gender': [78], 'locale': [127]}, 6: {'birthday': [1], 'educat
ion;type': [53, 55], 'education;year;id': [62], 'gender': [78], 'last_name': [111], 'locale
': [127], 'work;end_date': [157], 'work;start_date': [157]}, 7: {'education;concentration;i
d': [13], 'education;school;id': [25, 43, 50], 'education;type': [53, 54, 55], 'education;y
ear;id': [59], 'gender': [78], 'last_name': [107], 'locale': [127], 'location;id': [137], '
work;employer;id': [141, 144], 'work;start_date': [196]}, 8: {'gender': [78], 'locale': [12
```


Exploratory Data Analysis

I plotted an image of a spring layout of the network of people:

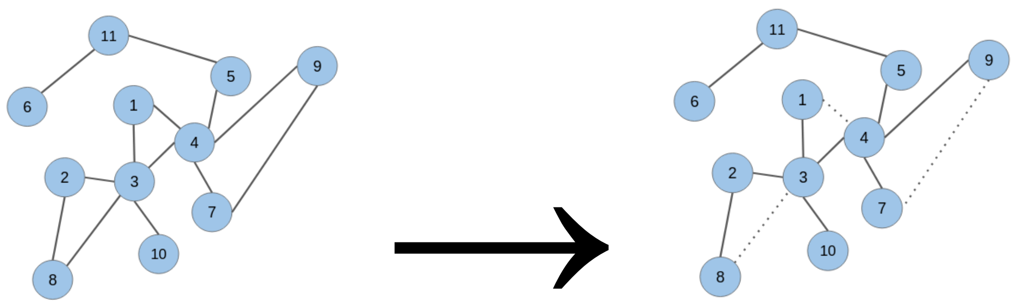


Pre-Processing

Node2Vec Pre-Processing

The goal was to generate features for the model to be able to use with a supervised learning model to predict if a pair of nodes are connected or not. The Node2Vec algorithm generates features for a node but first needs to learn how to generate those features, and it does so by understanding the graph structure and the nodes. So, a Node2Vec model will be created with an input of the graph & other hyperparameters to base its feature generation on. Instead of creating a Node2Vec Model on the entire graph (1), I created the Node2Vec model based on a new graph (2) with some deleted edges / node pairs that wouldn't change the fundamental structure of the graph since it wouldn't eliminate any nodes nor split the graph. The point of this was to have Node2Vec learn the fundamental graph structure & nodes to know how to generate features and test its accuracy by generating features on the removed & unconnected edges using a supervised learning ML model. This would give a good indication of the accuracy of the Node2Vec Model for predicting links as well as emulate the idea of social networks past & future graph structures as people constantly connect (form edges). Although in reality a link prediction model would constantly be retrained on the most current graph.

Below shows a before and after images of deleting removable edges shown by the dotted lines:

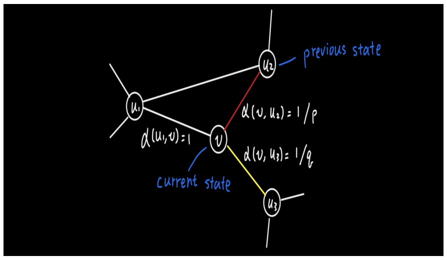


- Node2vec algorithm can be summarized as a two-part process (2):

1. First, it uses second-order biased random walks to generate sequences of nodes or "sentences" for each node according to the hyper-parameters: walk length, # of walks, and p & q values which tune for more exploratory or local walks thus producing different similarities between nodes. (1)
2. Second, once the sequences of nodes or 'sentences' are generated, they are used as an input to a word2vec skip-gram with negative sampling model. The main idea is that it maximizes the probability of predicting the correct context node given the center node. The skip-gram model first generates pairs of input and context nodes given the context window size and then feeds them into a shallow two-layer neural network. Once the neural network is trained, you can retrieve the hidden layer weights as your node embeddings. The number of neurons in the hidden layer will determine the size of the embedding, or you can alter the size based on hyperparameter tuning.

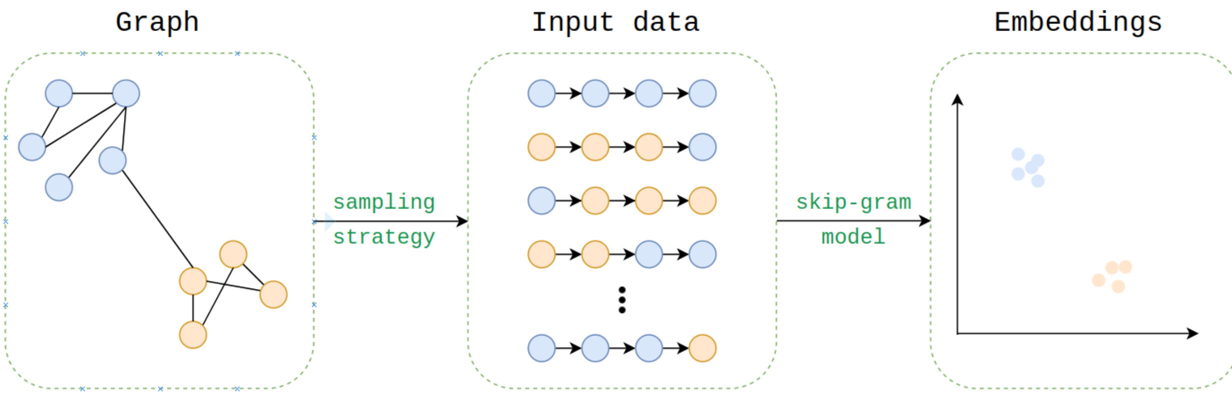
- These embeddings are a list of numbers that encapsulate the homophily or structural equivalence topological similarity in the graph, so that people closely related have similar representations & vice versa. Homophily similarity is finding nodes belonging to the same network community and structural equivalence is finding nodes that have the same structural roles. High p, low q hyperparameter values -> more depth first search / structural equivalence similarity & vice versa. (3). The embeddings are then used as features for a ML model.

1



https://towardsdatascience.com/node2vec-explained-graphically-749e49b7eb6b

2



https://towardsdatascience.com/node2vec-embeddings-for-graph-data-32a866340fef

3

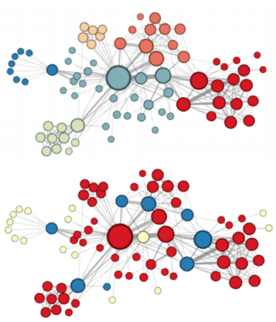


Figure 3: Complementary visualizations of Les Misérables coappearance network generated by *node2vec* with label colors reflecting homophily (top) and structural equivalence (bottom).

I followed the following steps in order to complete the Node2Vec pre-processing steps to generate features:

1. Found the negative samples (unconnected edges / node pairs) with nodes at max path length of 2 from one another to get samples that are possibly more likely to form a connection due to more common neighbors.

Unconneted Node Pairs: 1395922

	Node 1	Node 2	Connected
0	0	348	0
1	0	351	0
2	0	353	0
3	0	363	0
4	0	364	0
...	...	...	...
1395917	4035	4037	0
1395918	4035	4038	0
1395919	4036	4037	0
1395920	4036	4038	0
1395921	4037	4038	0

1395922 rows × 3 columns

2. Found the positive samples (removable edges / node pairs) - edges that can be removed while preserving the base graph structure (not eliminating nodes & not splitting the graph).

Removable Node Pairs: 84196

	Node 1	Node 2	Connected
0	0	1	1
1	0	2	1
2	0	3	1
3	0	4	1
4	0	5	1
...	...	...	...
88219	4020	4030	1
88220	4020	4031	1
88223	4021	4026	1
88226	4023	4031	1
88230	4027	4031	1

84196 rows × 3 columns

3. Found the remaining edges by removing the positive edges from the list of connected edges so we can train the Node2Vec Model on those remaining edges (base structure of the graph).

Node2Vec Edges: 4038

	Node 1	Node 2
0	0	11
1	0	12
2	0	15
3	0	18
4	0	37
...	...	...
4033	4023	4038
4034	4026	4030
4035	4027	4032
4036	4027	4038
4037	4031	4038

4038 rows × 2 columns

4. Combined negative & positive samples into final dataframe after reducing the # of negative

samples to balance the final dataset to about equal positive & negative samples to be used for more accurate modeling.

```
Unconnected & Removable Edges: 169196
0    85000
1    84196
Name: Connected, dtype: int64
```

	Node 1	Node 2	Connected
0	0	364	0
1	0	906	0
2	0	961	0
3	0	970	0
4	0	978	0
...	...	...	...
169191	4020	4030	1
169192	4020	4031	1
169193	4021	4026	1
169194	4023	4031	1
169195	4027	4031	1

169196 rows × 3 columns

5. Used the trained Node2Vec model to generate features for each node in a pair & sum them for a final set of features for that pair. Then, I test train split data for modeling.

```
Features (X): (169196, 100)
[[ 0.05423658  0.38881892 -0.7825276  ... -0.3608516   0.13439018
  -0.14539672]
 [-0.06878684  0.9884709  -0.85741556 ... -0.45005986 -0.20884675
  -0.87278837]
 [-0.4779037   0.9238887  -1.0211918  ... -0.7530286   0.5291232
  -0.5641365 ]
 ...
 [-0.29917532  0.92449725 -1.1229932  ... -0.36626023 -0.29428327
  -0.489528  ]
 [ 0.3246755   1.0931772  -0.41049516 ... -0.01053643 -0.36503953
   0.14392054]
 [ 0.48538733  0.91636264 -0.595041   ...  0.15409005 -0.65976167
   0.33824128]]
```

NetworkX Link Prediction Pre-Processing

I used the NetworkX library link prediction algorithms to calculate the link metric for each pair in the list of unconnected (negative samples) & removable edges (positive samples) to generate features that will be applied to a supervised learning approach to predict probabilities of each node pairs in forming a connection. I used the following link prediction algorithms: common_neighbors, jaccard_coefficient, resource_allocation index, adamic_adar_index, & preferential_attachment. Created & scaled the training set and testing set of data to create the final data below to be used for modeling:

	Node 1	Node 2	Connected	Common Neighbors	Jaccard Coefficient	Resource Allocation Index	Adamic Adar Index	Preferential Attachment
0	0	364	0	-0.688733	-0.940612	-0.639678	-0.704201	-0.371761
1	0	906	0	-0.688733	-0.942633	-0.871550	-0.733825	1.547445
2	0	961	0	-0.688733	-0.940750	-0.871550	-0.733825	-0.260503
3	0	970	0	-0.688733	-0.940818	-0.871550	-0.733825	-0.204874
4	0	978	0	-0.688733	-0.943233	-0.871550	-0.733825	2.270625
...	...	...	...	...	...	...	...	...
169191	4020	4030	1	-0.665866	-0.600397	-0.545057	-0.672454	-0.609909
169192	4020	4031	1	-0.597264	0.621218	0.571919	-0.502564	-0.615039
169193	4021	4026	1	-0.574396	0.936068	0.460927	-0.483467	-0.614157
169194	4023	4031	1	-0.597264	-0.034718	0.638937	-0.497499	-0.606222
169195	4027	4031	1	-0.620131	0.306369	0.240177	-0.556099	-0.615921

The link prediction algorithms employ the following equations to create the feature values and revolve around using the set of common neighbors between two nodes in a pair:

1. Common_neighbors

   `common_neighbors` (G, u, v) [source]

   Return the common neighbors of two nodes in a graph.

2. Jaccard_coefficient

   Jaccard coefficient of nodes **u** and **v** is defined as

   $$\frac{|\Gamma(u) \cap \Gamma(v)|}{|\Gamma(u) \cup \Gamma(v)|}$$

   where $\Gamma(u)$ denotes the set of neighbors of u.

3. Resource_allocation index

   Resource allocation index of **u** and **v** is defined as

   $$\sum_{w \in \Gamma(u) \cap \Gamma(v)} \frac{1}{|\Gamma(w)|}$$

   where $\Gamma(u)$ denotes the set of neighbors of u.

4. Adamic_adar_index

Adamic-Adar index of **u** and **v** is defined as

$$\sum_{w \in \Gamma(u) \cap \Gamma(v)} \frac{1}{\log |\Gamma(w)|}$$

where $\Gamma(u)$ denotes the set of neighbors of u. This index leads to zero-division for nodes only connected via self-loops. It is intended to be used when no self-loops are present.

5. Preferential_attachment

Preferential attachment score of **u** and **v** is defined as

$$|\Gamma(u)||\Gamma(v)|$$

where $\Gamma(u)$ denotes the set of neighbors of u.

Modeling

I trained & tested 4 ML models to predict my binary classification of 1 or 0 representing connected status of connected or not connected respectively. I used logistic regression, Random Forest Classifier, Gradient Boosting Classifier, and a Multi-Layer Perceptron Classifier. I wanted a simple classifier so I picked a logistic regression classifier, a random forest and gradient boosting classifier to test tree based classifiers, and the mlp classifier to try a neural network classifier to test my data on.

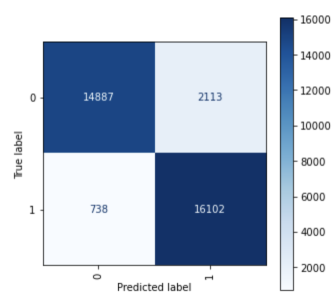
I took the following 5 steps to implement each of the 4 ML models:

1. I used GridSearchCV for hyperparameter tuning to pick the best model version.
2. I fit the model and made predictions for the test set.
3. Printed the classification report displaying the accuracy, precision, recall, and F1-scores.
4. Calculated the accuracy, precision, recall, F1-score, log loss score, and ROC-AUC (area under the curve) for the model.
5. Lastly, displayed the confusion matrix to see the distribution of predictions being made across all genres.

MLP Classification Report

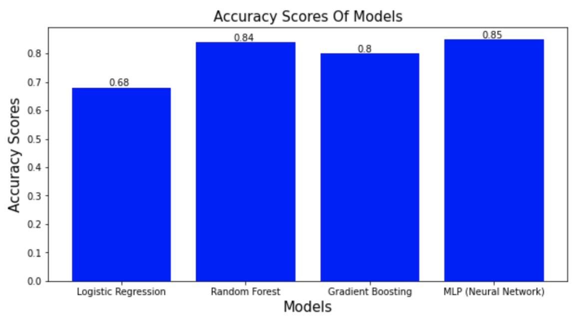
	precision	recall	f1-score	support
0	0.95	0.88	0.91	17000
1	0.88	0.96	0.92	16840
accuracy			0.92	33840
macro avg	0.92	0.92	0.92	33840
weighted avg	0.92	0.92	0.92	33840

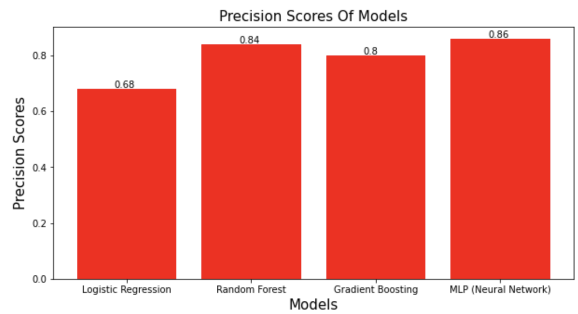
MLP Confusion Matrix

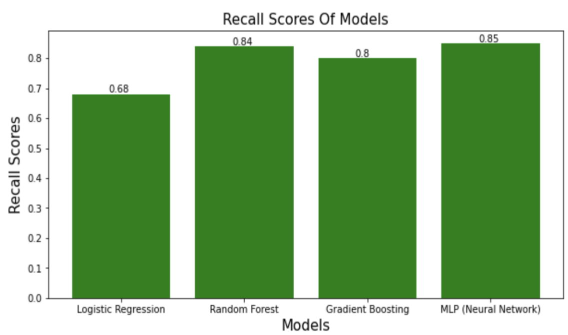


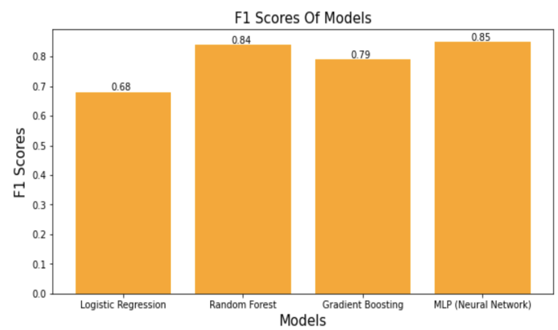
Node2Vec Model Comparison

The best model is the Multilayer Perceptron Classifier (Neural Network Classifier). The delta between the best and worst model in accuracy: 17%, precision: 18%, recall: 17%, f1: 17%. Here are the accuracies, precisions, recalls, and F1 scores for all 4 models.

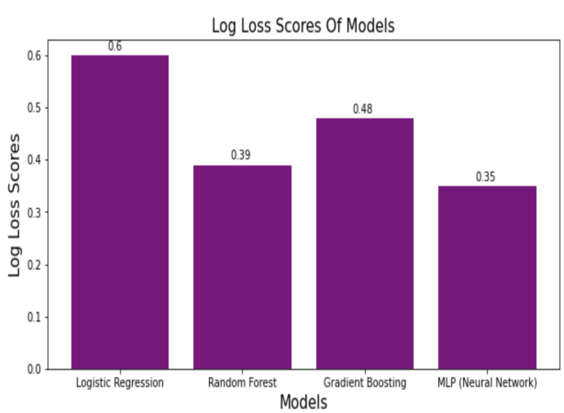


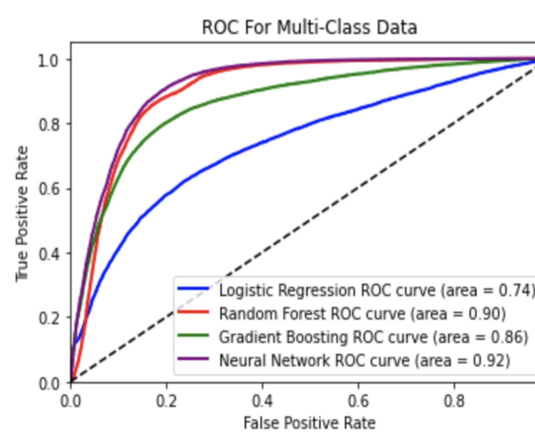






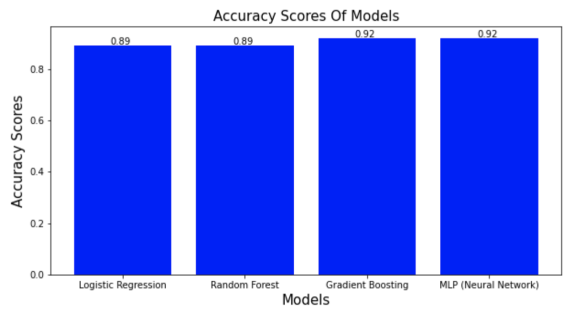
The lowest Log Loss score was 0.35 for the Multilayer Perceptron Classifier model. The ROC curves and the area under the curve (AUC) is the largest at 0.92 for the Multilayer Perceptron Classifier model.

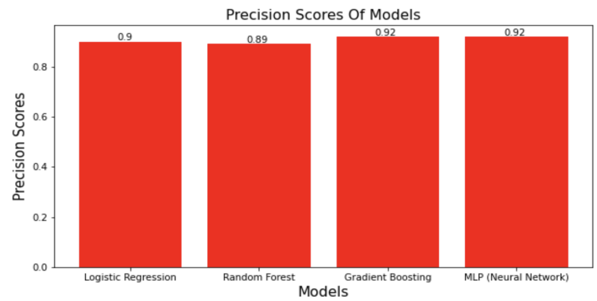


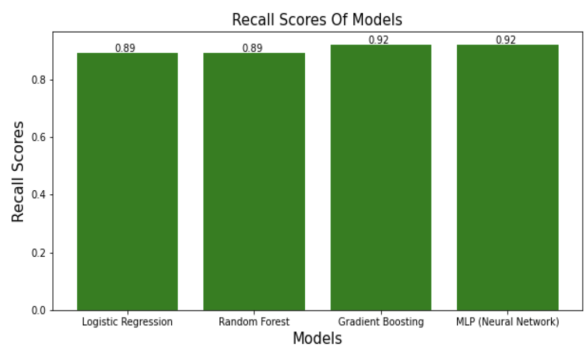


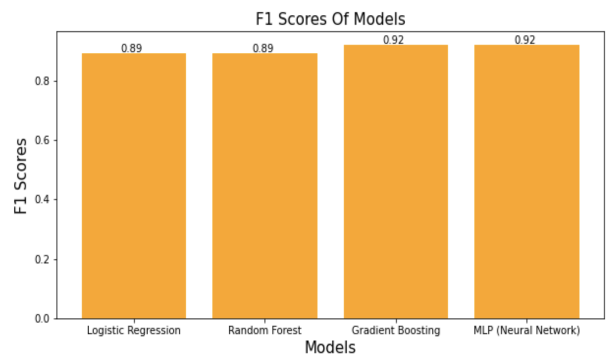
NetworkX LP Model Comparison

The best models were the Multilayer Perceptron Classifier (Neural Network Classifier) & Gradient Boosting Classifier. The delta between the best and worst model in accuracy: 3%, precision: 2%, recall: 3%, f1: 3%. Here are the accuracies, precisions, recalls, and F1 scores for all 4 models:

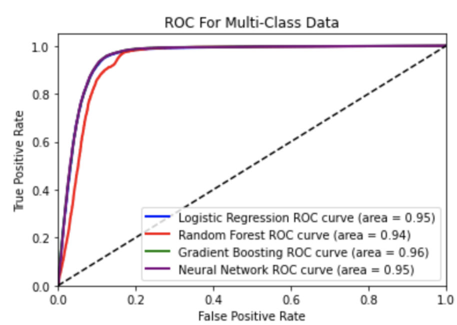








The lowest Log Loss score was 0.24 for the Multilayer Perceptron Classifier & Gradient Boosting Classifier models. The ROC curves and the area under the curve (AUC) is the largest at 0.96 for the GB model and 0.95 for the Multilayer Perceptron Classifier model & the LR model.



Node2Vec & NetworkX LP Recommender Functions

I created a Recommed_friends function which gets the probabilities for each unconnected pair that includes the user requested user # and prints the top k as requested by the user. I took the following steps to create the functions:

1. Prompts to pick the user # to recommend friends for.
2. Prompts the user for # of recommendations.
3. Uses MLP classifier & 'predict_proba' method to get the probabilities of all unconnected node pairs being connected.
4. Ranks the probabilities and prints out the requested number of recommended friends user #s.

Node2Vec Function

NetworkX LP Function

```
In [339]:  1  recommend_friends()

Please enter the user (0–4038) to get friend recommendations for:
65
Please enter the k number of friend recommendations out of 22 that you
would like to recieve:
22
Friend recommendations for user 65 are:

User 168 at 82.56% chance of a future link
User 221 at 73.65% chance of a future link
User 315 at 66.85% chance of a future link
User 56 at 62.53% chance of a future link
User 274 at 54.61% chance of a future link
User 75 at 53.54% chance of a future link
User 325 at 53.07% chance of a future link
User 3 at 48.2% chance of a future link
User 30 at 42.2% chance of a future link
User 73 at 29.01% chance of a future link
User 234 at 18.27% chance of a future link
User 229 at 17.61% chance of a future link
User 55 at 13.17% chance of a future link
User 92 at 10.66% chance of a future link
User 195 at 10.35% chance of a future link
User 177 at 4.59% chance of a future link
User 139 at 3.42% chance of a future link
User 167 at 0.55% chance of a future link
User 81 at 0.35% chance of a future link
User 93 at 0.32% chance of a future link
User 216 at 0.21% chance of a future link
User 6 at 0.03% chance of a future link
```

```
In [104]:  1  recommend_friends_nx()

Please enter the user (0–4038) to get friend recommendations for:
65
Please enter the k number of friend recommendations out of 22 that you would like to recieve:
22
Friend recommendations for user 65 are:

User: 56.0 at 81.09% chance of a future link
User: 325.0 at 74.75% chance of a future link
User: 315.0 at 63.15% chance of a future link
User: 168.0 at 51.07% chance of a future link
User: 55.0 at 35.69% chance of a future link
User: 221.0 at 5.89% chance of a future link
User: 73.0 at 5.85% chance of a future link
User: 3.0 at 5.69% chance of a future link
User: 216.0 at 1.55% chance of a future link
User: 234.0 at 1.55% chance of a future link
User: 81.0 at 1.49% chance of a future link
User: 229.0 at 1.35% chance of a future link
User: 6.0 at 1.35% chance of a future link
User: 167.0 at 1.32% chance of a future link
User: 93.0 at 1.28% chance of a future link
User: 195.0 at 1.26% chance of a future link
User: 139.0 at 1.26% chance of a future link
User: 177.0 at 1.21% chance of a future link
User: 75.0 at 1.15% chance of a future link
User: 274.0 at 1.15% chance of a future link
User: 30.0 at 1.1% chance of a future link
User: 92.0 at 1.06% chance of a future link
```

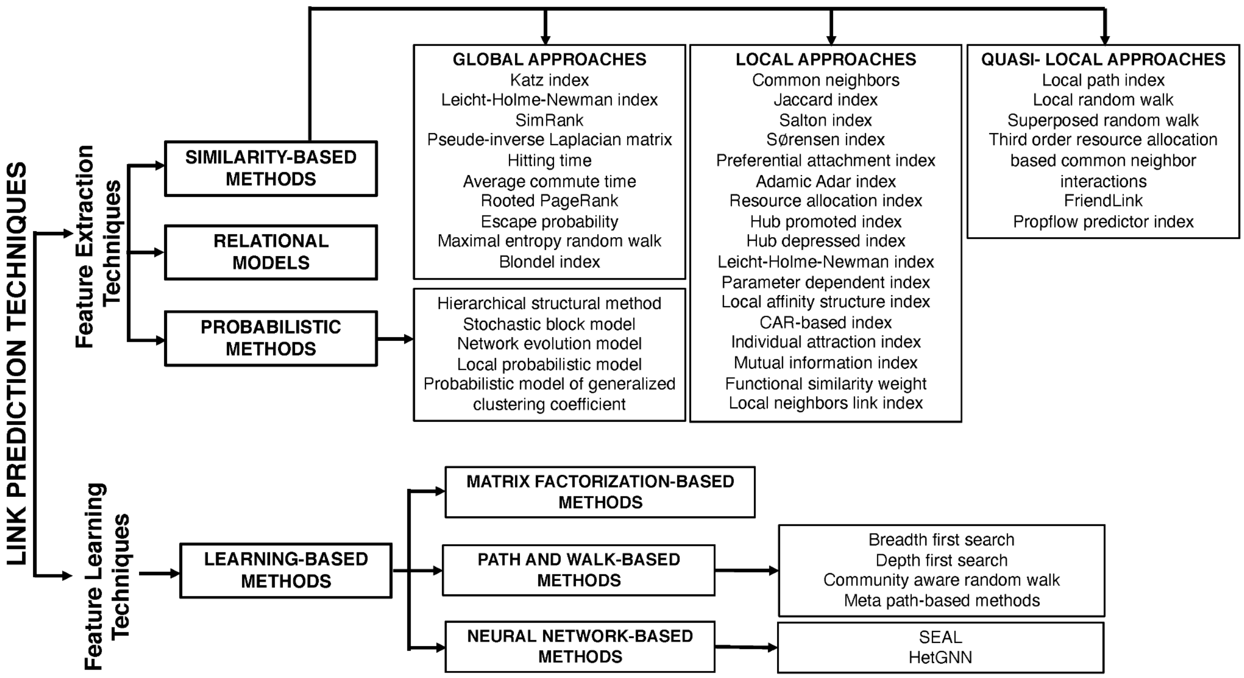
Takeaways

In the Node2Vec & NetworkX features creation pre-processing approaches, the Multi-layer perceptron classifier was the best model & the gradient boosting classifier as well on the NetworkX LP approach. Also, the features generated by node2vec don't work for predicting links as well as the features generated by the NetworkX library like prediction algorithms.

Model	Hyperperameters	Accuracy	Precision	Recall	F1-Score	Log Loss	ROC-AUC
Node2Vec: MLP Classifier (Multi-layer Perceptron Classifier)	activation='tanh', max_iter=1000, random_state=0, solver='sgd'	0.85	0.86	0.85	0.85	0.35	0.92
NetworkX: MLP Classifier (Multi-layer Perceptron Classifier)	activation='tanh', max_iter=1000, random_state=0, solver='sgd'	0.92	0.92	0.92	0.92	0.24	0.95
NetworkX: Gradient Boosting Classifier	learning_rate=0.05, random_state=0	0.92	0.92	0.92	0.92	0.24	0.96

Further Research

I would try to implement different approaches to get link predictions other than the Similarity-Based Local Approaches (NetworkX LP) & Path & Walk-Based Method (Node2Vec).


LINK PREDICTION TECHNIQUES
Feature Extraction Techniques
SIMILARITY-BASED METHODS
RELATIONAL MODELS
PROBABILISTIC METHODS
GLOBAL APPROACHES
Katz index
Leicht-Holme-Newman index
SimRank
Pseude-inverse Laplacian matrix
Hitting time
Average commute time
Rooted PageRank
Escape probability
Maximal entropy random walk
Blondel index
LOCAL APPROACHES
Common neighbors
Jaccard index
Salton index
SØrensen index
Preferential attachment index
Adamic Adar index
Resource allocation index
Hub promoted index
Hub depressed index
Leicht-Holme-Newman index
Parameter dependent index
Local affinity structure index
CAR-based index
Individual attraction index
Mutual information index
Functional similarity weight
Local neighbors link index
QUASI- LOCAL APPROACHES
Local path index
Local random walk
Superposed random walk
Third order resource allocation based common neighbor interactions
FriendLink
Propflow predictor index
Hierarchical structural method
Stochastic block model
Network evolution model
Local probabilistic model
Probabilistic model of generalized clustering coefficient
Feature Learning Techniques
LEARNING-BASED METHODS
MATRIX FACTORIZATION-BASED METHODS
PATH AND WALK-BASED METHODS
NEURAL NETWORK-BASED METHODS
Breadth first search
Depth first search
Community aware random walk
Meta path-based methods
SEAL
HetGNN