Facebook User-Connection Probability Detection Based On Social Circles

P30: Lalit Bangad, Yagmur Bayraktar, Xander Carruth, Vinay Deshmukh
Department of Computer Science, North Carolina State University
Raleigh, NC 27695
{llbangad,ybbayrak,agcarruth,vdeshmu}@ncsu.edu

1 Background

In large social media platforms it is difficult for users to find people they know amongst thousands of active users. To fix this issue, account suggestion algorithms are used by many major social media platforms such as Facebook, Twitter, Instagram and LinkedIn. The account suggestions, commonly known as "People You May Know" suggestions, help users expand their social circles by letting them find people they actually know in real life. For our project we looked at this "People You May Know" Algorithm and tried a novel approach to see whether we can get more accurate or faster results in large ego networks.

The 'People You May Know' suggestion is inherently a binary classification problem where graphs of friend networks are analyzed by connection suggestion algorithms. In these graphs the users are represented as nodes in a network, and the connections between them are labeled as edges. Depending on the data available, algorithms may use different techniques to extract information from the graph. One of the approaches to take is to evaluate the connections between the users. The suggestion can be made using the connections between friends, such as the following scenario: If User A knows User B and User B knows User C, what is the probability that User A knows User C? Another way to suggest connection is by simply evaluating features that belong to a pair of nodes and try to determine the similarity between them. This can be illustrated by the following example: If User X went to NC State University between 2013 - 2017 and User Y went to NC State University between 2015 - 2019, what is the probability that User X knows User Y?

To deepen our understanding of this topic we have conducted a literature survey about this problem. The work done by Toprak et Al [1] proposed using a layered structure in an ego network to improve the accuracy of a similarity-based prediction using a scraped Twitter data set with large amount of egos and weighted connections based on mentions, replies, and retweets. In the work done by McAuley and Leskovec [2], it is mentioned that it is a very laborous work to construct social circles, hence a node clustering method is used on ego networks to identify circles, and similarity metrics are used specifically for that circle.

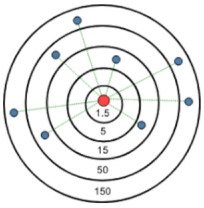
For our project we also used both the edges and features to form social circles and predict connections. We specifically used the ego network data sets in Stanford Network Analysis Project (SNAP) data set [3], which consists of friend lists obtained from Facebook. An ego network is a network centered around a single node, which is labeled as the ego. In an ego network, all nodes other than the ego are connected to the ego and are called as alters. Simply put, for our case the ego network consists connection of a single user and that user's friends, as well as the connections between the friends. For our project we have focused on a single ego network in [3], which was the largest ego network available in the data set.The data set also consists of information extracted from user profiles, such as name, birthday, school, work and location information for the specific user. The raw features are anonymized in order to keep the private data of users, but still consists of important information to use in similarity assessment between the nodes. The features in the data set consists of a binary value for all of the values for a given feature for all of the users. An example would be, for a given user A,

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

the education feature may carry the value (0, 0, 1). That value would mean for all of the universities in the ego network, College A, College B, College C; the user would have gone to College C, and not A and B.

2 Method

We based our method on previous work done by Toprak et al. in which they scraped Twitter ego networks with weighted directed edges based on the number of times users have interacted on Twitter and then generated social circles for these ego networks [1]. In their method, they apply Dunbar's model of social networks and create 5 communities, or social circles, based on the weight of edge connections to the ego. Nodes that are more strongly connected to the ego are found closer inwards in the circles. The logic behind this method is that users in the inner circles are more similar and more likely to interact and thus easier to predict connections for. See the diagram below taken from the paper to gain a better understanding of the structure of these communities.



Circle 1	Top 1% of most similar nodes
Circle 2	Top 3.33% of most similar nodes
Circle 3	Top 10% of most similar nodes
Circle 4	Top 33.3% of most similar nodes
Circle 5	Top 100% of most similar nodes

Figure 1: The natural makeup of social networks based upon social sciences.

By running supervised classifiers on the social circles they had generated, Toprak et al. were able to achieve better performance than the baseline classifiers as well as faster run times for connections found in the inner circles. We sought to find similar results by using the same concept applied in a different manner to the Facebook ego network dataset. In the Facebook ego network dataset, there is no information about the amount of times users interact with each other. Therefore, we could not strictly implement the method Toprak et al. created. Instead, we decided to take a novel approach and examine the features of each user node and find its similarity to the ego to create social circles.

There were a large amount of profile features which could be manually identified as not useful for determining how strongly connected users are to the ego so we chose to filter these first. Examples of such features are birthday, languages, middle name, etc. There were still a large amount of features left following this, so we compared principal component analysis (PCA) and a scarcity method for additional feature selection. The scarcity method works by selecting the features that nodes are least likely to have in common with the ego. It operates under the assumption that nodes with more of these less likely features in common with the ego are more closely connected to the ego. This method fits the dataset well, as users are much less likely to have a positive attribute in common with the ego node than a negative attribute. All figures related to the scarcity method will show 30 features as we found this worked best in our experimentation. We first believed PCA would work better, however our results showed otherwise. See the following page for the distributions using 30 features for PCA vs 30 features for scarcity.

From Figure 2 and Figure 3, it can be seen that scarcity covers a much wider range of similar features and is much closer to following a normal distribution of features per similarity tier than PCA. With PCA, we found from the skree plot that we should select features from the top 3 eigenvectors. However, in these eigenvectors there were a large number of high value contributors. Therefore, we only chose the top 10 features from each eigenvector. These top 30 features created a distribution with the most similar nodes sharing only 6 features in common with the ego. Comparatively, the most similar nodes with 30 features selected using scarcity shared 16 features with the ego node. We found these results made much more sense to use than PCA, as intuitively nodes which share positive features with the ego are more likely to share other positive features with the ego. Therefore, we chose scarcity as our method for feature selection.

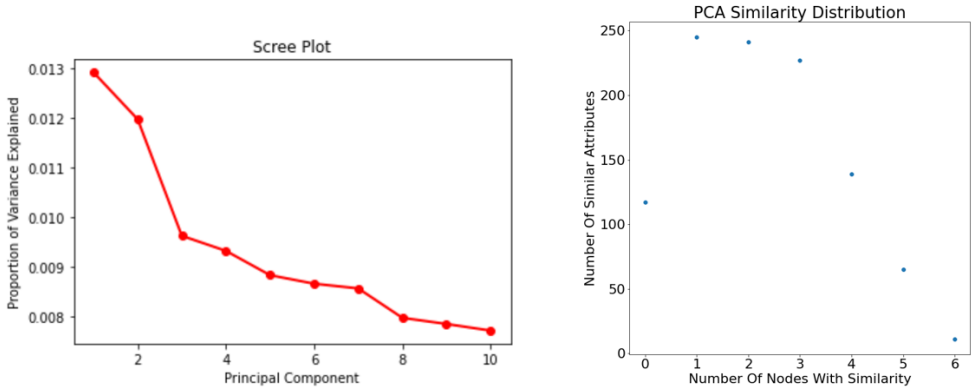


Figure 2: The skree plot and similarity distribution for PCA.

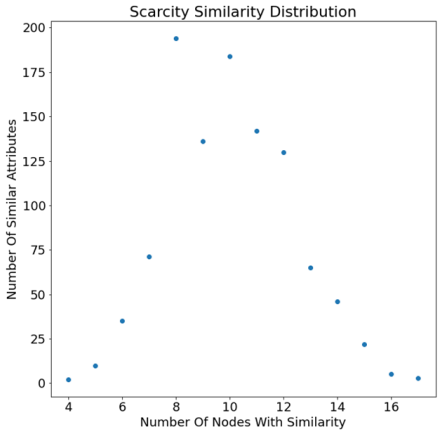


Figure 3: The similarity distribution for scarcity, with a much better spread and distribution compared to PCA.

After selecting the best 30 features for identifying similarity to the ego, we used hamming distance to calculate the similarity of every node to the ego, as it is the standard for comparing two binary vectors of equal size. We then used the similarity to construct 5 social circles which hold the top 1%, 3.33%, 10%, 33.3%, and 100% of the most similar nodes to the ego node.

Once these social circles were constructed, we began preparing the data to be used with our selected machine learning techniques. We first used the *networkx* library to construct the ego network graphs. Following this we identified nodes that could be safely removed from the circle 5 graph. These nodes served as our dependent variable for which we would predict connections. We also included all nonexistent edges for the nodes as the negative dependent variable. This connection information was only generated for the circle 5 graph because it was the graph with all data points, meaning it was also the baseline. Since we needed to compare our social circle link prediction technique to the baseline, we could only use removable edges from this graph.

Following this we created unsupervised topological features for each connection. For our use case, we only focused on topological features, as we had already considered non-topological features in our creation of the social circles. Unsupervised feature generation for the dataset split into circles was conducted by first determining the biggest circle that each of the nodes that a connection is being predicted for falls into and then creating the unsupervised topological features for the connection in that circle.

Specifically, the unsupervised features we used were common neighbors (CN), Jaccard's Coefficient (JC), Adamic-Adar (AA), Resource allocation (RA), and preferential attachment (PA). These methods

were used by Toprak et al. and were also considered in a paper focusing specifically on the Facebook ego network by McAuley and Leskovec [2]. The equation for each method can be found in Table 1. CN is a metric which shows the similarity between nodes by comparing the number of common neighbors between two nodes. JC builds upon CN by identifying the Jaccard similarity between the set of common neighbors of two users. The Jaccard similarity of these sets includes the total number of connections each node has as a factor in the calculation. AA calculates the common neighbors for two users but reduces the importance of common neighbors that have a large number of connections. RA is similar to AA but it further penalizes common neighbors for two users which have a large number of connections. PA identifies nodes that are more likely to gain new connections because they already have a large number of connections. Previous work by Toprak et al. did not use PA, so this is another novel addition on top of their method.

Method	Equation
Common Neighbors	$sim(i,j) = \|\Gamma(i) \cap \Gamma(j)\|$
Jaccard's Coefficient	$sim(i,j) = \frac{\|\Gamma(i) \cap \Gamma(j)\|}{\|\Gamma(i) \cup \Gamma(j)\|}$
Adamic-Adar	$sim(i,j) = \sum_{z \in \Gamma(i) \cap \Gamma(j)} \frac{1}{log(\|\Gamma(z)\|)}$
Resource Allocation	$sim(i,j) = \sum_{z \in \Gamma(i) \cap \Gamma(j)} \frac{1}{\|\Gamma(z)\|}$
Preferential Attachment	$sim(i,j) = \|\Gamma(i)\|\|\Gamma(j)\|$

Table 1: The equations for topological similarity features used in supervised learning. For all of the above equations, Γ indicates all of the nodes a node is connected to, or all of the neighbors of a node.

After creating the unsupervised topological features, we used *MinMaxScaler* to normalize the feature values. We then created an 80-20 split for our training and test data. After this, we trained an array of supervised learning techniques on our data. Specifically, we looked at the performance of Random Forest, Logistic Regression, SVM and XGBoost which were used in previous literature. After the supervised classifiers were trained and run on the unsupervised features, we generated accuracy, precision, and F1-score metrics on our test set for the baseline and the social circle method. We display the results for these metrics in the Results section.

3 Plan and Experiment setup

The goal for the experiment setup for our project were to evaluate whether using the social circles method we would be able to get comparable (if not better) results for metrics like accuracy, precision, recall and F-1 score for the data set as opposed to using only the largest circle for evaluation. Another goal that was planned was to evaluate whether utilizing the circles approach would significantly speed up the feature generation process. If our hypothesis about these improvements are true, it implies that as the network size grows we will be able to generate features faster, as given a large network, the graph algorithms will naturally take longer, so any speedup that we are able to achieve will be significant when keeping scalability in mind.

Once the ego network has been split into 5 concentric social circles, we have the following graphs which represent each circle:

Circle	Nodes	Edges
1	11	16
2	36	147
3	105	752
4	349	4573
5	1912	27794

Table 2: The description for each social circle in terms of a graph datastructure.

The visualisations for graphs that we generated by splitting the ego network into concentric circles can be seen in Figure 4.

To help answer our hypothesis, we needed a way to determine which edges would be formed in the future vs which edges would not be formed (here, edges refers to "connections" between two people, where people are represented as nodes).

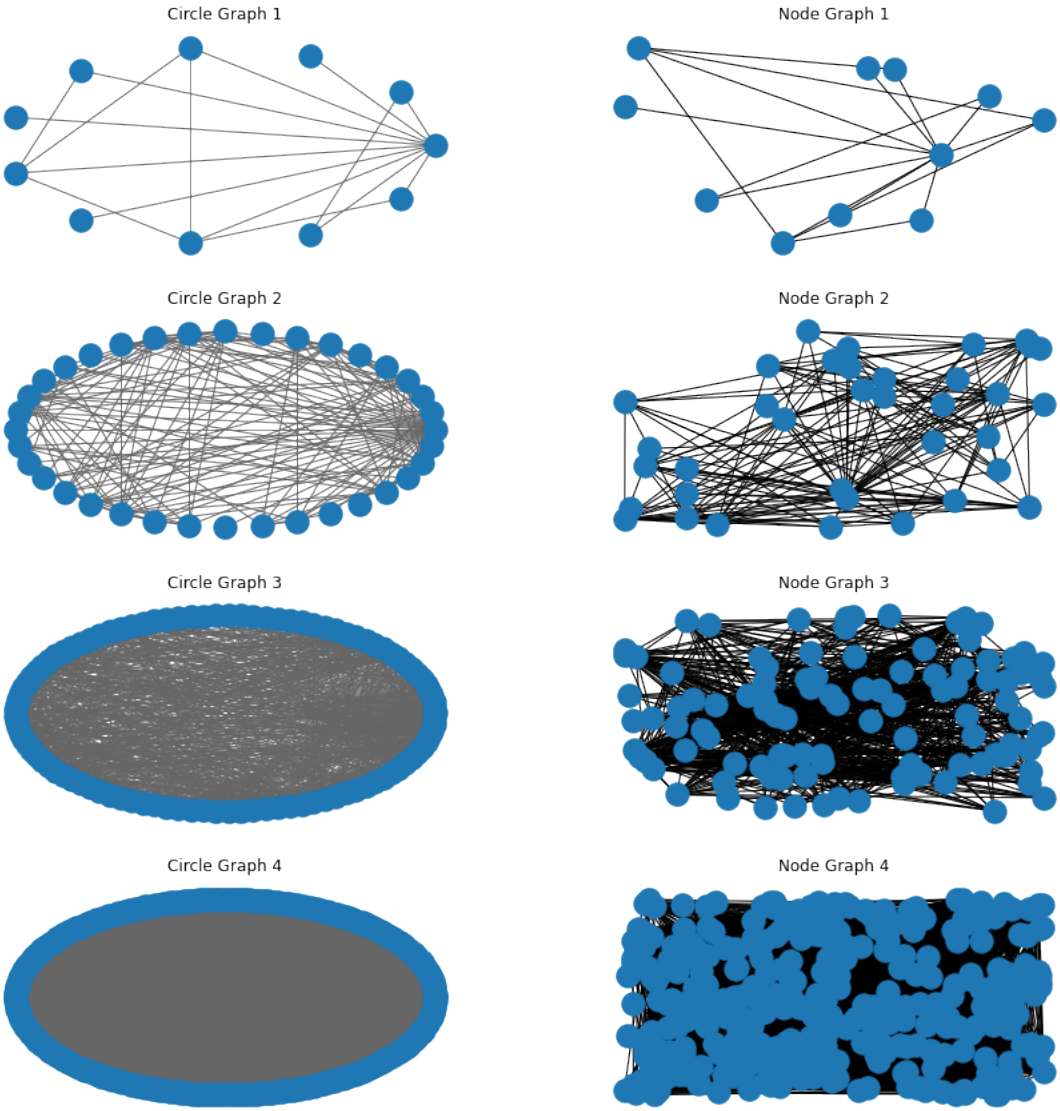


Figure 4: Circle and Node Graphs

The first problem we faced was that we didn't have a "past" vs "future" social network graph that we could use to train and test on. As we only had a single graph from the dataset, there was no readily available way to determine what edges/connections will be formed in the future. Hence, we determined a way to generate these edges. The idea was to create a "past" state of the graph, by assuming that the new edges that would have formed to give us the current state of the graph would not significantly change the graph's structure. We define "significantly change a graph" as referring to a change in the number of components of a graph. We define "positive edge" as an edge that can form in the past state of the network to give us the current state of the graph, without changing the number of connected components of the graph, and "negative edge" as an edge which didn't get formed in the past state of the graph and still doesn't exist in the current state of the graph.

Using this idea, we sampled all existing edges of the graph, and for each edge, we determined if the removal of this edge would change the number of components in the graph. However, this approach turned out to be very slow as given the large number of edges in the graph (>20000), it would take upwards of 2 hours to determine if we could remove this edge from the graph and not alter it.

To solve this issue, we came upon Tarjan's algorithm, which helped us determine the connected components in the graph. Given that our graph was undirected, we could use this algorithm to

determine a set of edges named "critical edges", which when removed, would alter the graph. Hence, to sample the "positive" edge, we calculated the set difference of all edges of the graph and the set of critical edges, which gave us a set of edges which when removed would not significantly alter the structure of the graph.

To determine negative edges, the process was simpler. We iterated on all combinations of pairs of nodes, so that we could get the set of all possible edges. We filtered out the edges that already existed in the graph, to get a set of non-existing edges. To ensure there would not be a class imbalance for negative and positive edges, we sampled about 27000 edges (i.e. close to the number of positive edges we had), and selected edges for further consideration as "negative edges" if there was a path present between the two endpoints of the edge.

Once we determined a set of negative and positive edges, the next step was to determine the innermost common circle that contained both the endpoints to ensure we determine the similarity measure using the innermost circle to have more similarity enforced based on the social circle both endpoints are present in. To determine the innermost common circle, we checked each circle starting from the first circle to see if both endpoints are present. If they were, we create an entry in a dictionary, where the graph/circle acts as a key and we append the edge to the list of edges which acted as values.

This is where we faced a fork in the road, so as to evaluate whether evaluating the similarity measures on the innermost common circle would speed up training vs using the outermost circle for all similarity calculations. The approach where we used only the outermost circle is termed as the "baseline" approach. And the approach where we take the innermost common circle is the new approach we are hypothesizing about.

Here, we faced another optimization issue, as we iterated each edge for the graph, we initially wrote the algorithms to determine the similarity measures manually. However, calculating the similarity measures one by one was really slow, as our implementation was not optimized for bulk usage. After some research, we found that the graph library that we were using $networkx$, provided builtin implementations for these measures, along with passing in an iterable of edges for which the similarity measures were calculated "in bulk", which greatly sped up our feature generation process. The visualisations for the unsupervised features that we created can be seen in Figure 5 and Figure 6.

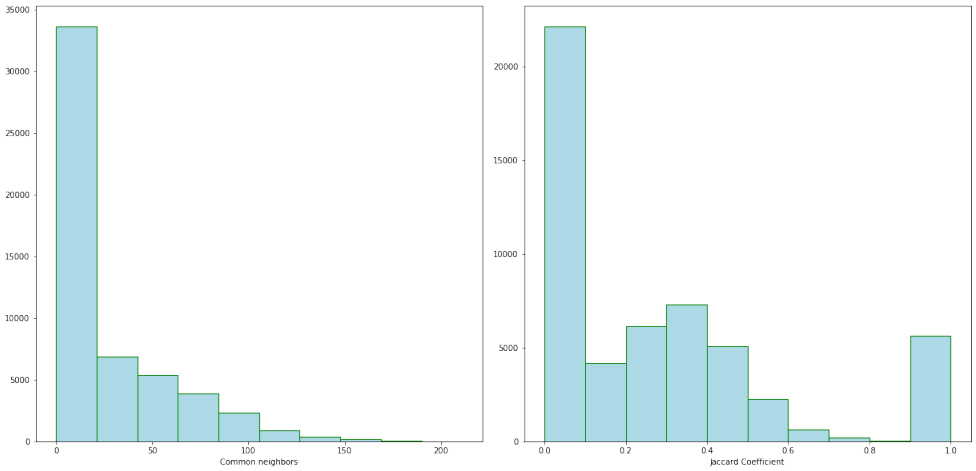


Figure 5: Common Neighbour and Jaccard Coefficient

Once we generated all the training features (i.e. similarity measures), we created a dataframe which was indexed by the endpoints of the edge, with all the similarity measures present as the columns. Along with the features, we created a label column "Connection" which would be set to 1 if the edge was a positive edge, else 0.

There was one more transformation we needed to consider before running machine learning classifiers on the data. The range for the similarity measures wasn't uniform, so there was the possibility that directly training on these features would have a feature imbalance where a feature with a large value would influence the output. Hence, to combat this, we used a $MinMaxScaler$ to adjust all values to be between 0 and 1 so all affect the output equally.

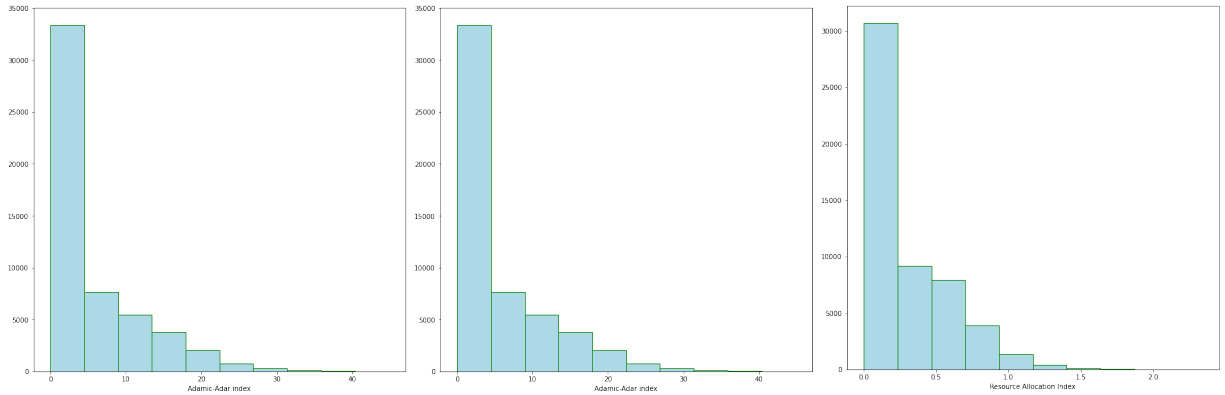


Figure 6: Adamic-Addar, Resource Allocation Index and Preferential Attachment

Once we have all the features and their respective labels, we ran the data through multiple classifiers like Random Forest Classifier, Logistic Regression, XGBoost and SVM.

Using these classifiers (apart from XGBoost), we got comparable results to the baseline approach, along with slightly less time taken to generate the similarity measures for the edges, thereby proving our hypothesis that it would be faster to generate the similarity features on smaller circles/graphs as it would take less time to run the algorithm on a smaller graph. With this, we can take up a future scope on analyzing the speedup factor that can occur in a larger graph if use our novel approach to determine the training features.

4 Results

The dataset we ran our supervised classifiers on was a dataframe containing all the normalized similarity features. This dataframe contains about 53,000 rows (which act as edges) and 5 features where the labels are almost evenly distributed.

The results of our classifiers for the outermost circle, ie, our baseline approach are as follows:

Classifiers Metrics	Accuracy	Precision	Recall	F1 Score
Random Forest	0.97	0.98	0.96	0.97
Logistic Regression	0.97	0.98	0.95	0.97
SVM	0.97	0.97	0.98	0.97

Table 3: Results for the baseline approach considering only the outer circle (Circle 5).

The results show that all three classifiers achieved a high level of accuracy (around 0.97), with slightly different results for the other metrics. The random forest classifier had the highest precision, while the SVM classifier had the highest recall. Overall, these results indicate that all three classifiers performed well on this data set.

Binary classifiers are usually evaluated using the ROC curve. But in our scenario, we have a class imbalance problem as the number of existing/actual edges can be far lesser than all the possible edges that can exist between them, so ROC doesnt provide us with good results. In the papers we surveyed, for evaluating systems with class imbalance, it is suggested to use metrics such as precision, recall and F1 score. It has been successfully argued that in the context of link prediction or user connection prediction, precision is more important than other metrics because if precision is high, the number of false positives is minimized and a little number of false negatives is allowable. Therefore, precision is used for evaluation.

The results for our novel approach wherein we use the innermost circles are as follows:

Classifiers Metrics	Accuracy	Precision	Recall	F1 Score
Random Forest	0.97	0.96	0.98	0.97
Logistic Regression	0.96	0.97	0.95	0.96
SVM	0.97	0.96	0.98	0.97

Table 4: Results for the novel approach considering innermost circle.

As seen in Table 4, the performance of the three classifiers using the innermost circle features is similar to the baseline approach. The accuracy is slightly lower, with values around 0.97, and the precision, recall, and F1 score show similar results.

However, it is worth noting that the innermost circle features provide additional information that may be useful for classification. For example, the connections between users in the innermost circle may be more relevant and indicative of future connections compared to the connections in the outer circles. Therefore, the use of these features can potentially improve the performance of classifiers on the inner circles, especially in scenarios where the data is imbalanced or the number of existing connections is relatively small.

Although not shown in the results of the table, we also tried out XGBoost but dropped it as it was giving terrible results. As we observe, the classifiers on the baseline and social circles produce very similar results. However, the total runtime for the baseline in our case was 250 seconds, while it was 200 seconds for our novel method. This means we received a 20 percent increase in speed while producing similar results. This is a significant boost in speed and could greatly help social media companies as with increased scale this time increase becomes even more important.

5 Conclusion

During this project we have had the chance to learn about machine learning applications in graphs, which required us to find different manners to extract features and make predictions. This project also allowed us to learn and experiment with feature reduction and other preprocessing techniques. We have managed to experiment and learn about how to use edges between the nodes and features of nodes for the purpose of connection suggestion. By learning about different approaches explored in [1], [2], [3] we have come up with a unifying approach that uses social circles without a data set that has weighted connections. Our novel approach of dividing users into social circles and using topological features to determine a possible connection demonstrated that we are able to get faster suggestions without compromising the accuracy of predictions, which would result in a significant improvement on very large data sets which are common in major social media platforms. For future work we planned on implementing Node2Vec for our circles data. Furthermore, we intend to run the data on different ego networks as well.

6 Github Link

https://github.ncsu.edu/vdeshmu/engr-ALDA-Fall2022-P30

7 References

1. Toprak, M., Boldrini, C., Passarella, A., & Conti, M. (2022). Harnessing the power of ego network layers for link prediction in online social networks. IEEE Transactions on Computational Social Systems, 1–13. https://doi.org/10.1109/tcss.2022.3155946
2. Leskovec, J., & Mcauley, J. (2012). Learning to Discover Social Circles in Ego Networks. In Advances in Neural Information Processing Systems. Curran Associates, Inc..
3. Leskovec, J., & Krevl, A. (2014). SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data