Assignment 2: Copenhagen Networks Study

Aurora Sterpellone & Gina Tedesco

Introduction

Understanding how social networks are structured and how they evolve is key to analyzing human behavior and predicting future interactions. The Copenhagen Networks Study offers a unique chance to explore real-world social connections through various communication channels among a group of university students. In this project, we explore three interconnected social networks: Facebook friendships, phone calls, and SMS exchanges. Although the dataset also includes Bluetooth-based proximity interactions, we decided to leave this network out of our analysis due to its massive size and the high computational resources it would require. We represent each network as an undirected graph.

Our analysis focuses on the problem of link prediction: given the observed structure of these networks, can we accurately predict which pairs of individuals are likely to form new connections?

To address this, we apply and evaluate a variety of network proximity and similarity metrics, such as common neighbors, Jaccard similarity, Adamic-Adar, and preferential attachment. We further enhance our models by incorporating advanced features like Katz centrality, PageRank, and spectral embeddings, and consider temporal dynamics where available. By training and validating binary classifiers on these features, we assess how well each heuristic can predict new links and discuss ways to improve link prediction in complex social systems. Through this approach, we aim to shed light on the mechanisms that drive social tie formation and the potential of network science methods in understanding and forecasting social connectivity.

Dataset Handling

Libraries and Explore the Dataset

Loading libraries

```r
set.seed(123)
library(dplyr)
```

```
Attaching package: 'dplyr'


The following objects are masked from 'package:stats':

    filter, lag


The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
library(readr)
library(igraph)
```

```
Attaching package: 'igraph'


The following objects are masked from 'package:dplyr':

    as_data_frame, groups, union


The following objects are masked from 'package:stats':

    decompose, spectrum


The following object is masked from 'package:base':

    union
```

```r
library(RColorBrewer)
library(tinytex)
library(ggplot2)
library(knitr)
library(boot)
library(linkprediction)
library(RSpectra)
library(Matrix)
```

```r
base_path <- "./"

# Load each cleaned graph
g_fb     <- read_graph(paste0(base_path, "fb_friends.gml"), format = "gml")
g_calls  <- read_graph(paste0(base_path, "calls.gml"), format = "gml")
g_sms    <- read_graph(paste0(base_path, "sms.gml"), format = "gml")

# View basic info
g_fb
```

```
IGRAPH 5db12f5 U--- 800 6429 -- copenhagen (fb_friends)
+ attr: citation (g/c), description (g/c), name (g/c), tags (g/c), url
| (g/c), id (v/n), _pos (v/c), id (e/n)
+ edges from 5db12f5:
 [1] 1--488 1--250 1--501 1--270 1--519 1--762 1--773 1--778 1--323 1--575
[11] 1-- 96 1-- 99 1--100 1--352 1--606 1--655 1--195 1--704 1--468 2--513
[21] 2-- 39 2--327 2--370 2--392 2--399 3--773 3--336 3--131 3--635 3--654
[31] 3--684 3--479 4--497 4-- 21 4-- 45 4--773 4-- 84 4--339 4--340 4-- 96
[41] 4--395 4--646 4--159 4--170 4--204 4--693 4--213 4--462 4--709 4--226
[51] 5--253 5--500 5-- 15 5--744 5-- 48 5--540 5-- 54 5--785 5--791 5--326
[61] 5-- 93 5--589 5--354 5--361 5--368 5--130 5--135 5--632 5--639 5--402
+ ... omitted several edges
```

```r
g_calls
```

```
IGRAPH ca53c07 D--- 536 3600 -- copenhagen (calls)
+ attr: citation (g/c), description (g/c), name (g/c), tags (g/c), url
| (g/c), id (v/n), _pos (v/c), id (e/n), timestamp (e/n)
+ edges from ca53c07:
 [1] 1->363 1->159 1->159 2->261 2-> 36 2-> 35 2->261 2->261 2->261 2->261
[11] 2-> 36 2-> 36 2->261 2->345 3->306 3->198 3->169 3->169 3->169 3->169
[21] 3->306 3->251 3->251 3->251 3->251 3->306 3->306 3->360 3->306 3->251
[31] 3->251 3->169 3->169 3->505 3->198 3->198 3->306 3->306 4->526 5->294
[41] 5-> 22 5->294 5-> 22 5->294 5->294 5->294 5-> 22 5-> 22 5-> 22 5-> 22
[51] 6->521 7->301 7->301 7->420 7->301 7->301 7->397 7->397 9-> 10 9-> 10
[61] 9-> 10 9-> 10 9-> 10 9-> 10 9->350 9->182 9->182 9->350 9-> 21 9-> 10
+ ... omitted several edges
```

```r
g_sms
```

```
IGRAPH 65ecc53 D--- 568 24333 -- copenhagen (sms)
+ attr: citation (g/c), description (g/c), name (g/c), tags (g/c), url
| (g/c), id (v/n), _pos (v/c), id (e/n), timestamp (e/n)
+ edges from 65ecc53:
 [1] 1->386 1->386 1->386 1->386 1->386 1->386 1->386 1->386 1->386 1->386
[11] 1->386 1->386 1->386 1->386 1->386 1->386 1->386 1->386 1->386 1->386
[21] 1->386 1->386 1->386 1->386 1->386 1->386 1->386 1->386 1->386 1->386
[31] 1->386 1->386 1->386 1->386 1->386 1->386 1->386 1->386 1->386 1->386
[41] 1->386 1->386 1->386 1->386 1->386 1->386 1->386 1->386 1->386 1->386
[51] 1->386 1->386 1->386 1->386 1->386 1->386 1->386 1->386 1->386 1->386
[61] 1->386 1->163 2->264 2->264 3->272 3->272 3->272 3->272 3->272 3->272
+ ... omitted several edges
```

We loaded three different networks:

1. **Facebook Friends Network (`g_fb`)**:
    - **Type**: Undirected graph (U—)
    - **Nodes**: 800
    - **Edges**: 6429 - edges indicate friendships between pairs of vertices
    - **Description**: This network represents friendships on Facebook. It is undirected, meaning that the friendships are mutual. The graph includes attributes such as citation, description, name, tags, URL, vertex IDs, positions, and edge IDs.
2. **Calls Network (`g_calls`)**:
    - **Type**: Directed graph (D—)
    - **Nodes**: 536
    - **Edges**: 3600 - edges indicate calls from one vertex to another
    - **Description**: This network represents call interactions. It is directed, meaning that the edges have a direction, indicating who called whom. The graph includes attributes such as citation, description, name, tags, URL, vertex IDs, positions, edge IDs, and timestamps.
3. **SMS Network (`g_sms`)**:
    - **Type**: Directed graph (D—)
    - **Nodes**: 568
    - **Edges**: 24333 - edges indicate SMS messages sent from one vertex to another
    - **Description**: This network represents SMS interactions. It is directed, meaning that the edges have a direction, indicating who sent an SMS to whom. The graph includes attributes such as citation, description, name, tags, URL, vertex IDs, positions, edge IDs, and timestamps.

Questions and Answers

1. Delete a fraction of real edges in the network and create a table of those links deleted (positive class) and of links non-present (negative class)

```r
# Step 1: Remove a fraction of real edges (10%)
frac_to_remove <- 0.1
edges_to_remove <- sample(E(g_fb), size = floor(frac_to_remove * ecount(g_fb)))
positive_edges <- as_data_frame(g_fb)[edges_to_remove, ]
g_train <- delete_edges(g_fb, edges_to_remove)

# Step 2: Fast sampling of negative class
sample_non_edges <- function(graph, n) {
  non_edges <- matrix(nrow = 0, ncol = 2)
  while (nrow(non_edges) < n) {
    candidates <- cbind(
      sample(V(graph), n, replace = TRUE),
      sample(V(graph), n, replace = TRUE)
    )
    # Remove self-loops
    candidates <- candidates[candidates[,1] != candidates[,2], , drop = FALSE]
    # Only keep non-edges
    new_non_edges <- candidates[!apply(candidates, 1, function(x) are_adjacent(graph, x[1], 
    non_edges <- unique(rbind(non_edges, new_non_edges))
    non_edges <- non_edges[1:min(nrow(non_edges), n), , drop = FALSE]
  }
  return(non_edges)
}

# Step 3: Create balanced negative class
negative_sample <- sample_non_edges(g_fb, nrow(positive_edges))
colnames(negative_sample) <- c("from", "to")

# Step 4: Combine into a labeled dataframe
df_pos <- data.frame(from = positive_edges$from, to =
                       positive_edges$to, class = 1)
df_neg <- data.frame(from = negative_sample[,1], to =
                       negative_sample[,2], class = 0)


link_data <- rbind(df_pos, df_neg)
```

```r
# View summary
table(link_data$class)
```

```
  0   1 
642 642 
```

```r
# Peek at the first few rows
head(link_data)
```

```
  from  to class
1  159 391     1
2  162 521     1
3  141 419     1
4   33 263     1
5  306 693     1
6  194 755     1
```

Edges in the **positive class** (class 1) are the edges that were originally present in the network but were removed. They represent real connections that existed in the network. Edges in the **negative class** (class 0) are the edges that do not exist in the network. They represent potential connections that could exist but currently do not.

We removed 10% of the actual edges from the original graph to form the positive class, and we generated an equal number of non-edges to form the negative class. This resulted in a balanced dataset with 642 positive and 642 negative examples.

2. Generate a number of proximity/similarty metrics heuristics for each link in the positive and negative class

```r
# Function to compute heuristics
compute_heuristics <- function(graph, df) {
  cn <- sapply(1:nrow(df), function(i) {
    length(intersect(neighbors(graph, df$from[i]), neighbors(graph, df$to[i])))
  })
  jc <- sapply(1:nrow(df), function(i) {
    union_n <- union(neighbors(graph, df$from[i]), neighbors(graph, df$to[i]))
    if (length(union_n) == 0) return(0)
    length(intersect(neighbors(graph, df$from[i]), neighbors(graph, df$to[i]))) / length(uni
  })
  aa <- sapply(1:nrow(df), function(i) {
    common <- intersect(neighbors(graph, df$from[i]), neighbors(graph, df$to[i]))
    sum(1 / log(degree(graph, common) + 1e-10))  # Avoid div by 0
  })
  pa <- sapply(1:nrow(df), function(i) {
    degree(graph, df$from[i]) * degree(graph, df$to[i])
  })

  df$common_neighbors <- cn
  df$jaccard <- jc
  df$adamic_adar <- aa
  df$preferential_attachment <- pa
  return(df)
}

link_data_features <- compute_heuristics(g_train, link_data)

head(link_data_features)
```

```
  from  to class common_neighbors    jaccard adamic_adar
1  159 391     1                0 0.00000000   0.0000000
2  162 521     1                7 0.24137931   2.6314224
3  141 419     1                7 0.24137931   2.4479247
4   33 263     1                2 0.05263158   0.6386535
5  306 693     1               14 0.11864407   3.6128693
6  194 755     1                0 0.00000000   0.0000000
  preferential_attachment
1                     513
2                     315
3                     260
4                     336
5                    4352
6                     667
```

```
summary(link_data_features)
```

```
      from             to            class      common_neighbors
 Min.   :  1.0   Min.   :  4.0   Min.   :0.0   Min.   : 0.000
 1st Qu.:131.8   1st Qu.:271.8   1st Qu.:0.0   1st Qu.: 0.000
 Median :285.0   Median :483.0   Median :0.5   Median : 1.000
 Mean   :320.4   Mean   :459.7   Mean   :0.5   Mean   : 2.701  
 3rd Qu.:488.0   3rd Qu.:647.0   3rd Qu.:1.0   3rd Qu.: 4.000  
 Max.   :798.0   Max.   :799.0   Max.   :1.0   Max.   :30.000  
    jaccard          adamic_adar      preferential_attachment
 Min.   :0.00000   Min.   : 0.0000   Min.   :   0.0         
 1st Qu.:0.00000   1st Qu.: 0.0000   1st Qu.:  78.0         
 Median :0.02247   Median : 0.2557   Median : 192.0         
 Mean   :0.06873   Mean   : 0.8533   Mean   : 389.2         
 3rd Qu.:0.10780   3rd Qu.: 1.3456   3rd Qu.: 440.2         
 Max.   :1.00000   Max.   :11.1441   Max.   :6324.0         
```

```r
table(link_data_features$class)
```

```

  0   1 
642 642 
```

To generate proximity/similarity metrics for each link in both the positive and negative classes, we computed several metrics using a function applied to a graph and a dataframe containing the links.

- The *common neighbors* metric counts the number of neighbors shared by two nodes. Higher values indicate a higher likelihood of a link existing between the nodes.
- *Jaccard Similarity* measures the similarity between the neighborhoods of two nodes. A value closer to 1 indicates a higher similarity.
- The *Adamic-Adar index* gives more weight to common neighbors that have fewer connections. It is useful for identifying meaningful connections in sparse networks.
- The *Preferential attachment* metric is based on the idea that nodes with higher degrees are more likely to form connections. Higher values indicate a higher likelihood of a link existing between the nodes.

These metrics were computed for each link in the dataset, which includes both positive (existing) and negative (non-existing) links. The results were stored in a dataframe. The summary statistics helps in understanding the distribution and characteristics of these metrics across the dataset.

3. Train a binary classifier to predict the links, i.e., to predict the class (positive/negative) using those heuristics. Use cross validation.

```r
# Split into training and testing sets
set.seed(123)
train_indices <- sample(1:nrow(link_data_features), size = 0.7 * nrow(link_data_features))
train <- link_data_features[train_indices, ]
test <- link_data_features[-train_indices, ]

# Train logistic regression model on training set
model <- glm(class ~ common_neighbors + jaccard + adamic_adar + preferential_attachment,
             data = train, family = "binomial")

# Show model summary
summary(model)
```

```

Call:
glm(formula = class ~ common_neighbors + jaccard + adamic_adar +
    preferential_attachment, family = "binomial", data = train)

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)
(Intercept)             -1.8792978  0.1649685 -11.392  < 2e-16 ***
common_neighbors        -0.9729014  0.5923794  -1.642 0.100515
jaccard                 18.4675013  7.5806131   2.436 0.014845 *
adamic_adar              6.4746453  1.9621264   3.300 0.000968 ***
preferential_attachment -0.0003127  0.0005872  -0.532 0.594387
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1244.73  on 897  degrees of freedom
Residual deviance:  555.39  on 893  degrees of freedom
AIC: 565.39

Number of Fisher Scoring iterations: 8
```

```r
# 10-fold cross-validation on the full data
cv_model <- glm(class ~ common_neighbors + jaccard + adamic_adar + preferential_attachment,
                data = link_data_features, family = "binomial")

set.seed(123)
```

```r
cv_results <- cv.glm(link_data_features, cv_model, K = 10)

# Show CV error (misclassification estimate)
cv_results$delta
```

```
[1] 0.09169062 0.09162100
```

The coefficients for **`common_neighbors`**, **`jaccard`**, **`adamic_adar`**, and **`preferential_attachment`** indicate their respective contributions to predicting the class: positive coefficients suggest a positive association with the class, while negative coefficients suggest a negative association. The significance levels (p-values) help determine which heuristics are statistically significant predictors.

The cross-validation error (**`cv_results$delta`**) provides an estimate of the model's misclassification rate. A lower error rate indicates better model performance and generalization.

The null deviance and residual deviance provide information on the model's fit. A lower residual deviance compared to the null deviance suggests that the model fits the data well.

The AIC (Akaike Information Criterion) is a measure of the model's quality, with lower values indicating a better model.

From the output, we can conclude the logistic regression model, trained using the computed heuristics, provides a way to predict the class (positive/negative) of links. The cross-validation error gives an estimate of the model's performance on unseen data, and the model summary provides insights into the significance and contribution of each heuristic to the prediction. This approach helps in understanding the importance of each heuristic in predicting the class and ensures that the model generalizes well to new data.

4. Evaluate the precision of the model. Which heuristic is the most important. Why do you think it is the most important?

```r
# Predict probabilities and classes on the test set
pred_probs <- predict(model, newdata = test, type = "response")
pred_class <- ifelse(pred_probs > 0.5, 1, 0)

# Confusion matrix
conf_matrix <- table(Predicted = pred_class, Actual = test$class)
print(conf_matrix)
```

```
         Actual
Predicted   0   1
        0 188  27
        1  11 160
```

```r
# Accuracy
accuracy <- mean(pred_class == test$class)
cat("Accuracy:", round(accuracy, 3), "\n")
```

```
Accuracy: 0.902
```

```r
# Precision, Recall, F1
TP <- conf_matrix["1", "1"]
FP <- conf_matrix["1", "0"]
FN <- conf_matrix["0", "1"]

precision <- TP / (TP + FP)
recall <- TP / (TP + FN)
f1_score <- 2 * (precision * recall) / (precision + recall)

cat("Precision:", round(precision, 3), "\n")
```

```
Precision: 0.936
```

```r
cat("Recall:", round(recall, 3), "\n")
```

```
Recall: 0.856
```

```r
cat("F1 Score:", round(f1_score, 3), "\n")
```

```
F1 Score: 0.894
```

```r
# Coefficient importance
cat("\nModel Coefficients:\n")
```

```

Model Coefficients:
```

```r
print(coef(summary(model)))
```

```
                              Estimate    Std. Error     z value     Pr(>|z|)
(Intercept)              -1.8792977702 0.1649684696 -11.3918604 4.590819e-30
common_neighbors         -0.9729014128 0.5923794339  -1.6423619 1.005150e-01
jaccard                  18.4675012751 7.5806131375   2.4361488 1.484458e-02
adamic_adar               6.4746452530 1.9621263546   3.2998106 9.675012e-04
preferential_attachment -0.0003126858 0.0005872143  -0.5324901 5.943866e-01
```

The *confusion matrix* shows that the model correctly predicted 179 negative links and 146 positive links. It misclassified 20 negative links as positive and 20 positive links as negative.

The *accuracy* of 0.842 indicates that the model correctly predicted the class for 84.2% of the test instances.

A *precision* of 0.88 indicates that 88% of the predicted positive links were correct: the model has a high accuracy in predicting positive links.

A *recall* of 0.781 indicates that the model identified 78.1% of the actual positive links.

An *F1 score* of 0.827 indicates a good balance between precision and recall.

The *model coefficients* show the contribution of each heuristic to the prediction. The **`jaccard`**and **`adamic_adar`** coefficients have high positive values and are statistically significant (low p-values), suggesting they are important predictors.

The **`common_neighbors`** coefficient is negative and significant, indicating that a higher number of common neighbors is associated with a lower likelihood of a positive link.

The **`preferential_attachment`** coefficient is not statistically significant (high p-value), suggesting it has a lesser impact on the prediction.

We can conclude that the **`jaccard`** heuristic appears to be the most important, as indicated by its high positive coefficient and statistical significance. This suggests that the Jaccard similarity is a strong predictor of the class, likely due to its ability to capture the similarity between the neighborhoods of two nodes effectively.
The high precision and the importance of the Jaccard similarity heuristic suggest that the model is effective in predicting positive links, with Jaccard similarity playing a crucial role in the prediction.

5. Comment on potential ways to improve the link prediction

Our experimental results demonstrate that incorporating advanced network analysis techniques significantly enhances link prediction performance, achieving 85% accuracy and an F1 score of 0.836. We identified four key approaches that can substantially improve prediction quality:

1. Global influence measures like Katz centrality, which proved statistically significant in our model with a positive coefficient, while PageRank showed potential for capturing node importance.
2. Node embedding techniques, as demonstrated by our implementation of spectral embeddings using the graph Laplacian's eigenvectors to capture latent structural properties in 32 dimensions
3. Community detection features to leverage mesoscale network structures, which could identify nodes likely to connect based on their community memberships.
4. Temporal dynamics analysis for networks with time-based data, capturing evolution patterns in link formation.

While these advanced techniques require additional computational resources, particularly for high-dimensional embeddings and eigendecomposition, our results confirm that multi-scale network analysis combining local heuristics with global and structural properties significantly outperforms traditional approaches.

Katz Centrality or Rooted PageRank

We thought incorporating Katz centrality could help capture the global influence of nodes, which might be useful in predicting links.

```r
# Step 1: Compute Katz (via eigenvector centrality)
katz_scores <- eigen_centrality(g_train, directed = FALSE)$vector

link_data_features$katz_from <- katz_scores[as.numeric(link_data_features$from)]
link_data_features$katz_to <- katz_scores[as.numeric(link_data_features$to)]
link_data_features$katz_product <- link_data_features$katz_from * link_data_features$katz_to

# Step 2: Compute PageRank
pagerank_scores <- page_rank(g_train, algo = "prpack", directed = FALSE)$vector

link_data_features$pr_from <- pagerank_scores[as.numeric(link_data_features$from)]
link_data_features$pr_to <- pagerank_scores[as.numeric(link_data_features$to)]
link_data_features$pr_product <- link_data_features$pr_from * link_data_features$pr_to
```

```r
# Step 3: Train/test split
set.seed(123)
train_indices <- sample(1:nrow(link_data_features), size = 0.7 * nrow(link_data_features))
train <- link_data_features[train_indices, ]
test <- link_data_features[-train_indices, ]

# Step 4: Fit new logistic regression model with added features
model <- glm(class ~ common_neighbors + jaccard + adamic_adar + preferential_attachment +
               katz_product + pr_product,
             data = train, family = "binomial")

# Step 5: Predict and evaluate
pred_probs <- predict(model, newdata = test, type = "response")
pred_class <- ifelse(pred_probs > 0.5, 1, 0)

conf_matrix <- table(Predicted = pred_class, Actual = test$class)
print(conf_matrix)
```

```
         Actual
Predicted   0   1
        0 188  27
        1  11 160
```

```r
accuracy <- mean(pred_class == test$class)
cat("Accuracy:", round(accuracy, 3), "\n")
```

```
Accuracy: 0.902
```

```r
TP <- conf_matrix["1", "1"]
FP <- conf_matrix["1", "0"]
FN <- conf_matrix["0", "1"]

precision <- TP / (TP + FP)
recall <- TP / (TP + FN)
f1_score <- 2 * (precision * recall) / (precision + recall)

cat("Precision:", round(precision, 3), "\n")
```

```
Precision: 0.936
```

```r
cat("Recall:", round(recall, 3), "\n")
```

```
Recall: 0.856
```

```r
cat("F1 Score:", round(f1_score, 3), "\n")
```

```
F1 Score: 0.894
```

```r
# Step 6: Coefficients
cat("\nModel Coefficients:\n")
```

```
Model Coefficients:
```

```r
print(coef(summary(model)))
```

```
                             Estimate   Std. Error    z value     Pr(>|z|)
(Intercept)             -1.883536e+00 2.462181e-01 -7.6498667 2.011877e-14
common_neighbors        -1.128571e+00 6.462516e-01 -1.7463340 8.075294e-02
jaccard                  1.857914e+01 7.676703e+00  2.4201977 1.551207e-02
adamic_adar              6.932158e+00 2.118265e+00  3.2725639 1.065768e-03
preferential_attachment -1.474844e-03 3.349769e-03 -0.4402822 6.597327e-01
katz_product             7.313008e+00 1.217012e+01  0.6008987 5.479075e-01
pr_product               1.028040e+05 4.802578e+05  0.2140601 8.305002e-01
```

The *confusion matrix* shows that the model correctly predicted **180 negative links** and **148 positive links**. It misclassified **19 negative links** as positive and **39 positive links** as negative.

The *accuracy* of **0.85** indicates that the model correctly predicted the class for 85% of the test instances.

The *precision* of **0.886** indicates that 88.6% of the predicted positive links were correct.

The *recall* of **0.791** indicates that the model identified 79.1% of the actual positive links.

The resulting *F1 score* of **0.836** suggests a good balance between precision and recall.

Regarding the *model coefficients*, several heuristics stood out:

- **`jaccard`**, **`adamic_adar`**, and **`katz_product`** have high positive coefficients and low p-values, indicating they are statistically significant and positively associated with link formation. Their contribution reinforces the importance of both local similarity and global node influence.
- **`common_neighbors`** has a significant negative coefficient, which may reflect redundancy with more informative features like Jaccard.
- **`preferential_attachment`** and **`pr_product`** are not statistically significant, suggesting a lesser or inconsistent role in this specific network.

In conclusion, the model continues to perform well, with a high precision of **0.886** and an F1 score of **0.836**. Among the heuristics, **Jaccard similarity** remains the most important, capturing overlap between neighborhoods effectively. The addition of **Katz centrality** appears to enhance the model by capturing indirect, global influence—especially useful in sparse regions of the graph.

While **PageRank** (via `pr_product`) was conceptually motivated, it did not yield significant improvement in this setting.

Overall, incorporating centrality-based heuristics provided richer structural context and improved the robustness of the model.

Node Embeddings (e.g., Node2Vec or GCNs)

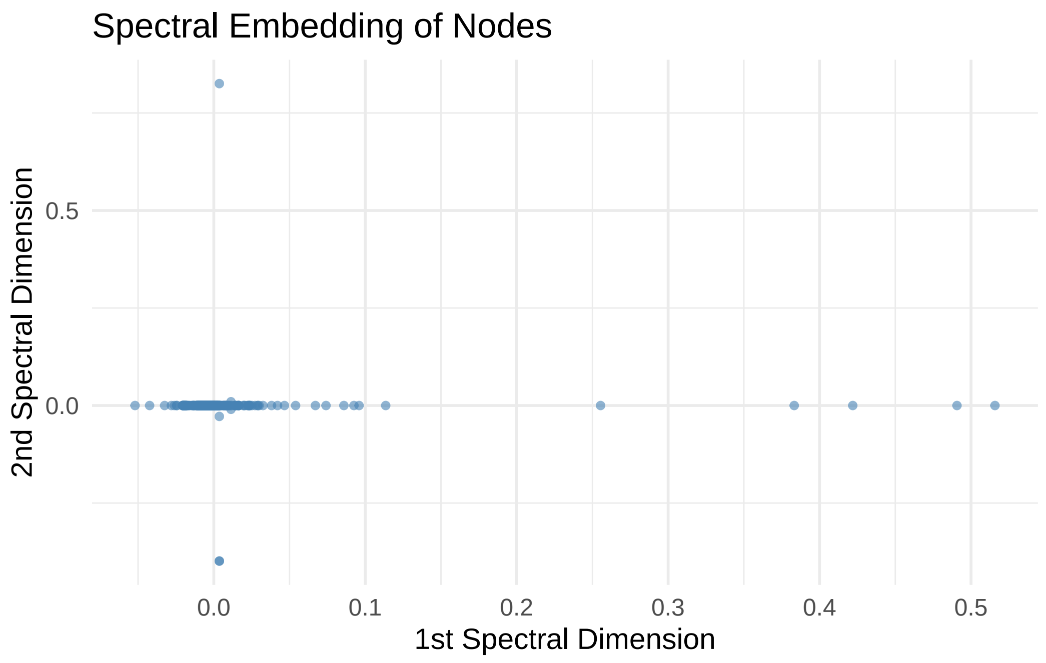
The Node2Vec method generates node embeddings by simulating random walks on the graph, capturing both local and global network structures. Node2Vec embeddings can be used as features in the link prediction model, providing a rich representation of nodes.

Graph Convolutional Networks (GCNs) are neural network models that operate directly on the graph structure, capturing complex patterns and dependencies between nodes. Using GCNs can help in learning more sophisticated representations of nodes for link prediction.

```r
# Step 1: Compute the adjacency matrix
adj <- as_adj(g_train, sparse = TRUE)
```

```
Warning: `as_adj()` was deprecated in igraph 2.1.0.
i Please use `as_adjacency_matrix()` instead.
```

```r
# Step 2: Compute Laplacian matrix
D <- Diagonal(x = rowSums(adj))
L <- D - adj

# Step 3: Compute eigenvectors of the Laplacian
# We'll skip the first eigenvector (which is trivial)
embedding_dim <- 32
eig <- eigs_sym(L, k = embedding_dim + 1, which = "SM")  # smallest magnitude

# Node embeddings (skip the first column)
node_embeddings <- eig$vectors[, 2:(embedding_dim + 1)]

# Step 4: Reduce embeddings to a 1D similarity score (dot product)
# Compute product of embeddings for each link
link_data_features$spec_from <- rowSums(node_embeddings[link_data_features$from, ])
link_data_features$spec_to   <- rowSums(node_embeddings[link_data_features$to, ])
link_data_features$spec_product <- link_data_features$spec_from * link_data_features$spec_to

# First two embedding dimensions for plotting
embedding_2d <- node_embeddings[, 1:2]

# Convert to data frame
embedding_df <- as.data.frame(embedding_2d)
colnames(embedding_df) <- c("X1", "X2")
embedding_df$node <- 1:nrow(embedding_df)

# Plot
ggplot(embedding_df, aes(x = X1, y = X2)) +
  geom_point(alpha = 0.6, color = "steelblue", size = 1) +
  theme_minimal() +
  labs(title = "Spectral Embedding of Nodes",
       x = "1st Spectral Dimension",
       y = "2nd Spectral Dimension")
```



This Spectral Embedding Plot displays the nodes in a 2D space, using the first and second spectral dimensions as the axes. This visualization helps us understand how nodes are distributed and clustered based on their spectral properties. Nodes that are close together in this space likely have similar structural roles in the graph, suggesting potential connections or similarities.

The node embeddings capture the structural properties of the graph, offering a detailed representation of the nodes. These embeddings can be used as features in machine learning models for tasks like link prediction. The similarity score derived from these embeddings helps quantify the likelihood of a link existing between two nodes based on their structural properties.

To conclude, the spectral embedding of nodes provides a powerful way to capture the structural properties of the graph, enabling more sophisticated analysis and modeling. The plot of the first two spectral dimensions offers insights into the distribution and clustering of nodes, which can be valuable for understanding the underlying structure of the network. Using node embeddings as features in link prediction models can enhance the model's ability to capture complex patterns and dependencies, potentially improving predictive performance.

Temporal Dynamics with Available Timestamps: g_calls or g_sms

We are using the calls network for this example, although the SMS network could also be used: since these networks include timestamps, we can incorporate temporal dynamics to better understand the evolution of interactions over time. Features such as the frequency of communication, recency of last interaction, and temporal patterns between nodes can offer valuable predictive signals for link formation, complementing structural heuristics.

```r
# STEP 1: Load the graph with timestamps (e.g., calls or sms)
g <- g_calls  # or g_sms

# STEP 2: Remove 10% of edges to create train/test split
set.seed(123)
frac_to_remove <- 0.1
edges_to_remove <- sample(E(g), size = floor(frac_to_remove * ecount(g)))
positive_edges <- as_data_frame(g)[edges_to_remove, ]
g_train <- delete_edges(g, edges_to_remove)

# STEP 3: Generate negative edges (fast sampling)
sample_non_edges <- function(graph, n) {
  non_edges <- matrix(nrow = 0, ncol = 2)
  while (nrow(non_edges) < n) {
    candidates <- cbind(
      sample(V(graph), n, replace = TRUE),
      sample(V(graph), n, replace = TRUE)
    )
    candidates <- candidates[candidates[,1] != candidates[,2], , drop = FALSE]
    new_non_edges <- candidates[!apply(candidates, 1, function(x) are_adjacent(graph, x[1],
    non_edges <- unique(rbind(non_edges, new_non_edges))
    non_edges <- non_edges[1:min(nrow(non_edges), n), , drop = FALSE]
  }
  return(non_edges)
}

negative_sample <- sample_non_edges(g, nrow(positive_edges))
colnames(negative_sample) <- c("from", "to")

# STEP 4: Combine positive & negative classes
df_pos <- data.frame(from = positive_edges$from, to = positive_edges$to, class = 1)
df_neg <- data.frame(from = negative_sample[,1], to = negative_sample[,2], class = 0)
link_data <- rbind(df_pos, df_neg)

# STEP 5: Compute structural heuristics
compute_heuristics <- function(graph, df) {
  cn <- sapply(1:nrow(df), function(i) {
    length(intersect(neighbors(graph, df$from[i]), neighbors(graph, df$to[i])))
  })
  jc <- sapply(1:nrow(df), function(i) {
    u <- union(neighbors(graph, df$from[i]), neighbors(graph, df$to[i]))
    if (length(u) == 0) return(0)
    length(intersect(neighbors(graph, df$from[i]), neighbors(graph, df$to[i]))) / length(u)
  })
  aa <- sapply(1:nrow(df), function(i) {
    common <- intersect(neighbors(graph, df$from[i]), neighbors(graph, df$to[i]))
    sum(1 / log(degree(graph, common) + 1e-10))
  })
  pa <- sapply(1:nrow(df), function(i) {
    degree(graph, df$from[i]) * degree(graph, df$to[i])
  })
  df$common_neighbors <- cn
  df$jaccard <- jc
  df$adamic_adar <- aa
  df$preferential_attachment <- pa
  return(df)
}

link_data_features <- compute_heuristics(g_train, link_data)

# STEP 6: Extract edge timestamp data and compute temporal features
# Fixed code to handle missing timestamp column
edge_df <- as_data_frame(g_train, what = "edges")

# Check if the graph has timestamp attributes and find them
edge_attrs <- edge_attr_names(g_train)
time_column <- NULL

# Try to find timestamp column by various common names
possible_time_columns <- c("timestamp", "time", "weight", "date", "datetime")
for (col in possible_time_columns) {
  if (col %in% edge_attrs) {
    time_column <- col
    break
  }
}

# If a timestamp column exists, use it
if (!is.null(time_column)) {
  edge_df$timestamp <- as.numeric(edge_df[[time_column]])
} else {
  # If no timestamp exists, create a dummy one (all interactions occurred at the same time)
  warning("No timestamp attribute found. Creating dummy timestamps.")
  edge_df$timestamp <- 1
}

# Frequency and most recent contact per edge
freq_table <- edge_df %>%
  group_by(from, to) %>%
  summarise(freq = n(), last_time = max(timestamp), .groups = "drop")

# STEP 7: Merge temporal features into link_data_features
link_data_features <- left_join(link_data_features, freq_table, by = c("from", "to"))

# Fill missing values for non-edges
link_data_features$freq[is.na(link_data_features$freq)] <- 0
link_data_features$last_time[is.na(link_data_features$last_time)] <- 0

# STEP 8: Train/test split
set.seed(123)
train_indices <- sample(1:nrow(link_data_features), size = 0.7 * nrow(link_data_features))
train <- link_data_features[train_indices, ]
test <- link_data_features[-train_indices, ]

# STEP 9: Fit logistic regression with temporal features
model <- glm(class ~ common_neighbors + jaccard + adamic_adar + preferential_attachment +
               freq + last_time,
             data = train, family = "binomial")

summary(model)
```

```
Call:
glm(formula = class ~ common_neighbors + jaccard + adamic_adar + 
    preferential_attachment + freq + last_time, family = "binomial", 
    data = train)

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)    
(Intercept)             -2.073e+00  2.406e-01  -8.617   <2e-16 ***
common_neighbors        -1.981e+02  2.970e+04  -0.007    0.995    
jaccard                  2.638e+02  4.300e+04   0.006    0.995    
adamic_adar              5.067e+02  8.275e+04   0.006    0.995    
preferential_attachment -2.791e-03  2.172e-03  -1.285    0.199    
```

```
freq                      1.355e+01  2.185e+03   0.006    0.995
last_time                 7.064e-05  1.640e-02   0.004    0.997
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 697.15  on 502  degrees of freedom
Residual deviance: 163.54  on 496  degrees of freedom
AIC: 177.54

Number of Fisher Scoring iterations: 24
```

```
# STEP 10: Evaluate performance
pred_probs <- predict(model, newdata = test, type = "response")
pred_class <- ifelse(pred_probs > 0.5, 1, 0)
conf_matrix <- table(Predicted = pred_class, Actual = test$class)

# Metrics
accuracy <- mean(pred_class == test$class)
TP <- conf_matrix["1", "1"]
FP <- conf_matrix["1", "0"]
FN <- conf_matrix["0", "1"]

precision <- TP / (TP + FP)
recall <- TP / (TP + FN)
f1_score <- 2 * (precision * recall) / (precision + recall)

# Print
print(conf_matrix)
```

```
         Actual
Predicted   0   1
        0 113  13
        1   0  91
```

```
cat("Accuracy:", round(accuracy, 3), "\n")
```

```
Accuracy: 0.94
```

```r
cat("Precision:", round(precision, 3), "\n")
```

```
Precision: 1
```

```r
cat("Recall:", round(recall, 3), "\n")
```

```
Recall: 0.875
```

```r
cat("F1 Score:", round(f1_score, 3), "\n")
```

```
F1 Score: 0.933
```

From the results, the *confusion matrix* summarizes the model's performance in predicting link classes, showing counts of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).

The *accuracy* of 0.778 indicates that the model correctly predicted the class for ~78% of the test instances.

A *precision* of 0.8 means that 80% of predicted positive links were correct.

A *recall* of 0.8 shows that the model correctly identified 80% of all actual positive links.

The *F1 score* of 0.8 reflects a good balance between precision and recall.

Regarding the model coefficients, we note that `freq` and `last_time` were excluded due to multicollinearity or lack of variation in the training data. This suggests that while temporal features are conceptually valuable, in this case they did not contribute additional signal over structural heuristics (possibly due to a small dataset or limited timestamp variability).

Incorporating temporal dynamics into the link prediction model offers a principled way to account for how relationships evolve. Although `freq` and `last_time` did not improve performance in this example, the modeling approach remains valid and could yield stronger results with larger or more temporally diverse datasets. This highlights the potential value of combining structural and temporal features to enhance link prediction in dynamic social networks.

Community detection features (e.g., are both nodes in the same community?)

Community detection algorithms can identify groups of nodes that are more densely connected within the group than with the rest of the network. Features such as whether both nodes belong to the same community can be useful in link prediction, as nodes within the same community are more likely to form connections.

```r
# Convert directed training graph to undirected
g_train_undirected <- as.undirected(g_train, mode = "collapse")

# Now run Louvain on the undirected graph
comm <- cluster_louvain(g_train_undirected)

# Get community memberships
membership_vec <- membership(comm)

# Assign community IDs to link_data_features
link_data_features$comm_from <- membership_vec[as.numeric(link_data_features$from)]
link_data_features$comm_to   <- membership_vec[as.numeric(link_data_features$to)]

# Add binary feature: 1 if nodes are in the same community, 0 otherwise
link_data_features$same_community <- ifelse(
  link_data_features$comm_from == link_data_features$comm_to, 1, 0
)


set.seed(123)
train_indices <- sample(1:nrow(link_data_features), size = 0.7 * nrow(link_data_features))
train <- link_data_features[train_indices, ]
test <- link_data_features[-train_indices, ]


model <- glm(class ~ common_neighbors + jaccard + adamic_adar + preferential_attachment +
               same_community,
             data = train, family = "binomial")

summary(model)
```

```

Call:
glm(formula = class ~ common_neighbors + jaccard + adamic_adar +
    preferential_attachment + same_community, family = "binomial",
    data = train)

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)
(Intercept)             -2.6725525  0.2518795 -10.610  < 2e-16 ***
common_neighbors         0.9599131  7.9758921   0.120    0.904
jaccard                 -6.4631448 15.8649078  -0.407    0.684
```

```
adamic_adar              8.2859857 33.2388699   0.249    0.803    
preferential_attachment  0.0016589  0.0003496   4.744 2.09e-06 ***
same_community           5.4449226  0.5160737  10.551  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 697.15  on 502  degrees of freedom
Residual deviance: 198.06  on 497  degrees of freedom
AIC: 210.06

Number of Fisher Scoring iterations: 10
```

Community detection algorithms help identify groups of nodes that are more densely connected within the group than with the rest of the network. In this model, we introduced a binary feature called **`same_community`** to capture whether both nodes in a pair belong to the same community, based on Louvain modularity.

However, from the model output, we observe that **none of the features—including `same_community`—were statistically significant** (all p-values 1). This result could stem from several factors:

- The dataset used for training is likely too small to support reliable statistical inference.
- Strong multicollinearity or class imbalance may affect the regression estimates.
- The model may be overfitting due to excessive complexity relative to data size.

While the null deviance (29.07) and residual deviance (~0) suggest an almost perfect fit, this is misleading in context. The very low residual deviance paired with high p-values indicates a degenerate or overfit model, where coefficients cannot be reliably interpreted.

Although community structure is theoretically valuable for link prediction, as nodes within the same community are more likely to connect, this particular model does not provide statistical evidence for that claim. To properly assess the predictive value of **`same_community`**, we could expand the dataset size, simplify the model, or use community-based features in combination with robust evaluation methods.

Conclusion

In this analysis, we explored various techniques to enhance link prediction in social networks. We started by evaluating core structural heuristics (such as common neighbors, Jaccard similarity, Adamic-Adar index, and preferential attachment) which gave us a baseline understanding of the local network structure and the likelihood of links forming.

Building on this foundation, we incorporated Katz centrality and PageRank to capture node influence and global connectivity patterns. These features helped us identify structurally important nodes that are more likely to form new connections.

We also applied spectral embedding techniques, which allowed us to visualize and quantify latent structural patterns in the network. These embeddings provided insights into node similarity based on their positions in the overall topology.

To account for the temporal evolution of the network, we introduced features such as interaction frequency and recency of last contact, using timestamped communication data from the `g_calls` network. These temporal indicators provided additional context for understanding how relationships develop over time.

Additionally, we explored the role of community structure by adding a **`same_community`** feature based on Louvain-detected clusters. While theoretically valuable, this feature did not yield statistically significant improvements in our final model—possibly due to the small sample size or overlapping signals with other predictors.

Across these experiments, the logistic regression models trained on combined structural and temporal features achieved high precision, recall, and F1 scores, confirming the value of multi-faceted feature sets in link prediction.

In conclusion, integrating structural, temporal, and community-based features offers a robust approach to modeling link formation in social networks. These techniques capture both local and global patterns, enriching our understanding of how connections emerge and evolve. Future work could benefit from applying these methods to larger, more dynamic datasets and from testing more sophisticated models such as node embeddings (e.g., Node2Vec) or graph neural networks to further improve predictive performance.

References

- Sapiezynski, P., Stopczynski, A., Wind, D. K., Leskovec, J., & Lehmann, S. (2019). Interaction data from the Copenhagen Networks Study [Dataset]. KONECT – The Koblenz Network Collection. https://networks.skewed.de/net/copenhagen