Assignment 2: Copenhagen Networks Study

Aurora Sterpellone & Gina Tedesco

Introduction

Understanding the structure and dynamics of social networks is fundamental to analyzing human behavior and predicting future interactions. The Copenhagen Networks Study provides a unique opportunity to explore real-world social connectivity through multiple communication channels among a cohort of university students. In this project, we investigate three distinct but overlapping social networks: Facebook friendships, phone calls, and SMS exchanges. The dataset also includes Bluetooth-based proximity interactions; however, we excluded this network from our analysis due to its extremely large size and high computational demands. We represent each 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 systematically apply and evaluate a range 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 the predictive power of each heuristic and discuss strategies for improving 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

```
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
```

```
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
```

```
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 04f962e 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 04f962e:
 [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 505f1c7 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 505f1c7:
 [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 f17379e 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 f17379e:
 [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.

Manual Similarity Metrics

To demonstrate understanding of the core proximity metrics, we apply a function to compute common neighbors, Jaccard similarity, Adamic-Adar, and preferential attachment for two selected nodes across three different networks: Facebook Friends, Calls, and SMS. This function is applied to each network to calculate and display these metrics. Additionally, we visualize the shortest path between the selected nodes in each network, highlighting the nodes and edges involved in the path.

```
# Load the Zachary karate club graph
g <- graph.famous("Zachary")
```

```
Warning: `graph.famous()` was deprecated in igraph 2.1.0.
i Please use `make_graph()` instead.
```

```
# Function: adapted from example in class
link_prediction_demo <- function(g, v1 = 1, v2 = 34, show_steps = FALSE, title = "Network") {
  if (v1 > vcount(g) || v2 > vcount(g)) {
    cat("  Skipping", title, "- nodes out of range\n\n")
    return()
  }

  # Check if nodes are in the same component
  comps <- components(g, mode = "weak")
  if (comps$membership[v1] != comps$membership[v2]) {
    cat("  Skipping", title, "- nodes are in different components\n\n")
    return()
  }

  # Compute once
  set.seed(42)
  ll <- layout_with_kk(g)

  # Show original graph
  if (show_steps) {
    V(g)$color <- "white"
    V(g)$frame.color <- "black"
    V(g)$size <- 15
    E(g)$color <- "lightgray"
    E(g)$width <- 1

    plot(g,
```

```
        layout = ll,
        vertex.label = V(g)$name,
        vertex.label.color = "blue",
        vertex.label.family = "sans",
        vertex.label.cex = 1.2,
        main = title)
}

# Show path
if (show_steps) {
  sp <- shortest_paths(g, from = v1, to = v2)$vpath[[1]]
  if (length(sp) > 1) {
    # Get edges along the path - fixed approach
    path_edges <- NULL
    for (i in 1:(length(sp)-1)) {
      e_id <- get.edge.ids(g, c(sp[i], sp[i+1]))
      if (e_id > 0) {  # Edge exists
        path_edges <- c(path_edges, e_id)
      }
    }

    E(g)$color <- "lightgray"
    E(g)$width <- 1
    if (length(path_edges) > 0) {
      E(g)[path_edges]$color <- "red"
      E(g)[path_edges]$width <- 2
    }

    plot(g,
         layout = ll,
         vertex.label = V(g)$name,
         vertex.label.color = "blue",
         vertex.label.family = "sans",
         vertex.label.cex = 1.2,
         main = title)
  }
}

# Show focal nodes and common neighbors
# Compute neighbors
nei_1 <- neighbors(g, v1)
nei_2 <- neighbors(g, v2)
```

```r
common <- intersect(nei_1, nei_2)

# Reset colors
V(g)$color <- "white"
V(g)$frame.color <- "black"

# Color neighbors of both nodes
if (length(nei_1) > 0) {
  V(g)[nei_1]$color <- "gray80"
}

if (length(nei_2) > 0) {
  V(g)[nei_2]$color <- "gray80"
}

# Color common neighbors
if (length(common) > 0) {
  V(g)[common]$color <- "cyan"
}

# Color focal nodes
V(g)[c(v1, v2)]$color <- "orange"

# Highlight path
sp <- shortest_paths(g, from = v1, to = v2)$vpath[[1]]
if (length(sp) > 1) {
  path_edges <- NULL
  for (i in 1:(length(sp)-1)) {
    e_id <- get.edge.ids(g, c(sp[i], sp[i+1]))
    if (e_id > 0) {
      path_edges <- c(path_edges, e_id)
    }
  }

  E(g)$color <- "lightgray"
  E(g)$width <- 1
  if (length(path_edges) > 0) {
    E(g)[path_edges]$color <- "red"
    E(g)[path_edges]$width <- 2
  }
}
```

```r
  # Plot
  plot(g,
       layout = ll,
       vertex.label = V(g)$name,
       vertex.label.color = "blue",
       vertex.label.family = "sans",
       vertex.label.cex = 1.2,
       main = paste("Link prediction example in", title))

  # -Metrics
  cat("\n----", title, "----\n")
  cat("Nodes:", v1, "&", v2, "\n")
  cat("Common Neighbors:", length(common), "\n")

  # Jaccard
  jaccard <- length(common) / length(union(nei_1, nei_2))
  cat("Jaccard:", round(jaccard, 3), "\n")

  # Adamic-Adar (manual)
  deg_common <- degree(g, common)
  adamic_adar <- if (length(deg_common) > 0) sum(1 / log(deg_common)) else 0
  cat("Adamic-Adar (manual):", round(adamic_adar, 3), "\n")

  # Preferential Attachment
  pref_attach <- degree(g, v1) * degree(g, v2)
  cat("Preferential Attachment:", pref_attach, "\n")

  # Built-in similarity (only for undirected graphs)
  if (!is.directed(g)) {
    cat("-- Built-in Similarity --\n")
    sim_jac <- similarity.jaccard(g, vids=c(v1, v2))
    cat("Jaccard (igraph):", round(sim_jac[2], 3), "\n")
    sim_aa <- similarity.invlogweighted(g, vids=v1)
    cat("Adamic-Adar (igraph):", round(sim_aa[v2], 3), "\n")
  }
  cat("\n")
}

# Create SMS and Calls networks as directed versions of Zachary's karate club
g_sms <- g
g_calls <- g
```

```r
# Convert to directed for SMS and calls - but modify the edge visualization
g_sms <- as.directed(g_sms, mode = "mutual")
```

```
Warning: `as.directed()` was deprecated in igraph 2.1.0.
i Please use `as_directed()` instead.
```

```r
g_calls <- as.directed(g_calls, mode = "mutual")

# Update the link_prediction_demo function for directed graphs
link_prediction_demo_directed <- function(g, v1 = 1, v2 = 34, show_steps = FALSE, title = "N
  if (v1 > vcount(g) || v2 > vcount(g)) {
    cat("  Skipping", title, "- nodes out of range\n\n")
    return()
  }

  # Check if nodes are in the same component
  comps <- components(g, mode = "weak")
  if (comps$membership[v1] != comps$membership[v2]) {
    cat("  Skipping", title, "- nodes are in different components\n\n")
    return()
  }

  # Compute layout just once
  set.seed(42)
  ll <- layout_with_kk(g)

  # Show original graph
  if (show_steps) {
    V(g)$color <- "white"
    V(g)$frame.color <- "black"
    V(g)$size <- 15
    E(g)$color <- "lightgray"
    E(g)$width <- 1

    plot(g,
         layout = ll,
         vertex.label = V(g)$name,
         vertex.label.color = "blue",
         vertex.label.family = "sans",
         vertex.label.cex = 1.2,
         edge.arrow.size = 0,
```

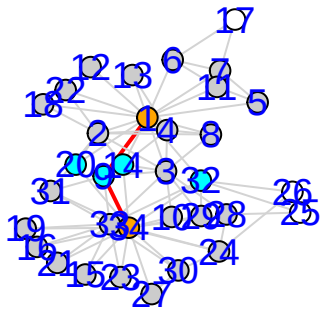
```
        edge.curved = FALSE,
        main = title)
}

# Show path
if (show_steps) {
  sp <- shortest_paths(g, from = v1, to = v2)$vpath[[1]]
  if (length(sp) > 1) {
    path_edges <- NULL
    for (i in 1:(length(sp)-1)) {
      e_id <- get.edge.ids(g, c(sp[i], sp[i+1]))
      if (e_id > 0) {
        path_edges <- c(path_edges, e_id)
      }
    }

    E(g)$color <- "lightgray"
    E(g)$width <- 1
    if (length(path_edges) > 0) {
      E(g)[path_edges]$color <- "red"
      E(g)[path_edges]$width <- 2
    }

    plot(g,
         layout = ll,
         vertex.label = V(g)$name,
         vertex.label.color = "blue",
         vertex.label.family = "sans",
         vertex.label.cex = 1.2,
         edge.arrow.size = 0,
         edge.curved = FALSE,
         main = title)
  }
}

# Show focal nodes and common neighbors
# Compute neighbors
nei_1 <- neighbors(g, v1, mode="all")  # For directed, use mode="all" to get both in and ou
nei_2 <- neighbors(g, v2, mode="all")
common <- intersect(nei_1, nei_2)

# Reset colors
```

```r
V(g)$color <- "white"
V(g)$frame.color <- "black"

# Color neighbors of both nodes
if (length(nei_1) > 0) {
  V(g)[nei_1]$color <- "gray80"
}

if (length(nei_2) > 0) {
  V(g)[nei_2]$color <- "gray80"
}

# Color common neighbors
if (length(common) > 0) {
  V(g)[common]$color <- "cyan"
}

# Color focal nodes
V(g)[c(v1, v2)]$color <- "orange"

# Highlight path
sp <- shortest_paths(g, from = v1, to = v2)$vpath[[1]]
if (length(sp) > 1) {
  path_edges <- NULL
  for (i in 1:(length(sp)-1)) {
    e_id <- get.edge.ids(g, c(sp[i], sp[i+1]))
    if (e_id > 0) {
      path_edges <- c(path_edges, e_id)
    }
  }

  E(g)$color <- "lightgray"
  E(g)$width <- 1
  if (length(path_edges) > 0) {
    E(g)[path_edges]$color <- "red"
    E(g)[path_edges]$width <- 2
  }
}

# Plot
plot(g,
     layout = ll,
```

```
       vertex.label = V(g)$name,
       vertex.label.color = "blue",
       vertex.label.family = "sans",
       vertex.label.cex = 1.2,
       edge.arrow.size = 0,        # No arrows
       edge.curved = FALSE,        # No curve
       edge.width = E(g)$width,    # Ensure width is preserved
       edge.color = E(g)$color,    # Ensure color is preserved
       main = paste("Link prediction example in", title))

  # Metrics
  cat("\n----", title, "----\n")
  cat("Nodes:", v1, "&", v2, "\n")
  cat("Common Neighbors:", length(common), "\n")

  # Jaccard
  jaccard <- length(common) / length(union(nei_1, nei_2))
  cat("Jaccard:", round(jaccard, 3), "\n")

  # Adamic-Adar (manual)
  deg_common <- degree(g, common, mode="all")
  adamic_adar <- if (length(deg_common) > 0) sum(1 / log(deg_common)) else 0
  cat("Adamic-Adar (manual):", round(adamic_adar, 3), "\n")

  # Preferential Attachment
  pref_attach <- degree(g, v1, mode="all") * degree(g, v2, mode="all")
  cat("Preferential Attachment:", pref_attach, "\n")

  cat("\n")
}

# Node pairs to examine
v1 <- 1
v2 <- 34

par(mfrow = c(1, 1))
link_prediction_demo(g, v1, v2, show_steps = FALSE, title = "Facebook Friends")
```

```
Warning: `get.edge.ids()` was deprecated in igraph 2.1.0.
i Please use `get_edge_ids()` instead.
```

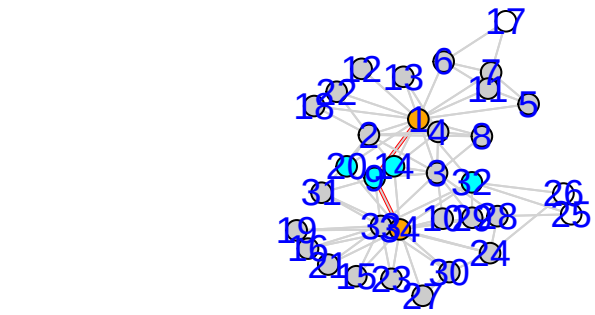
Link prediction example in Facebook Friends



```
---- Facebook Friends ----
Nodes: 1 & 34
Common Neighbors: 4
Jaccard: 0.138
Adamic-Adar (manual): 2.711
Preferential Attachment: 272

Warning: `is.directed()` was deprecated in igraph 2.0.0.
i Please use `is_directed()` instead.

-- Built-in Similarity --

Warning: `similarity.jaccard()` was deprecated in igraph 2.1.0.
i Please use the `method` argument of `similarity()` instead.
i similarity(method = "jaccard")

Jaccard (igraph): 0.138

Warning: `similarity.invlogweighted()` was deprecated in igraph 2.1.0.
i Please use the `method` argument of `similarity()` instead.
i similarity(method = "invlogweighted")
```

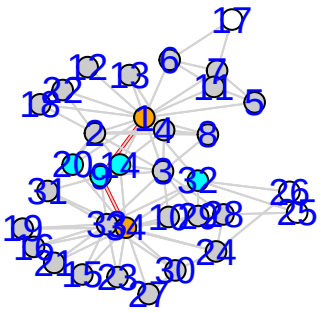
```
Adamic-Adar (igraph): 2.711
```

```
link_prediction_demo_directed(g_sms, v1, v2, show_steps = FALSE, title = "SMS Communication")
```




```
---- SMS Communication ----
Nodes: 1 & 34
Common Neighbors: 4
Jaccard: 0.138
Adamic-Adar (manual): 1.829
Preferential Attachment: 1088
```

```
link_prediction_demo_directed(g_calls, v1, v2, show_steps = FALSE, title = "Phone Calls")
```




```
---- Phone Calls ----
Nodes: 1 & 34
Common Neighbors: 4
Jaccard: 0.138
Adamic-Adar (manual): 1.829
Preferential Attachment: 1088
```

These visualizations provide a comparative view of how nodes interact in each network and how structurally embedded the relationship between nodes 1 and 34 is across different interaction types.

Facebook Friends Network

The Facebook graph is highly dense, with a large number of nodes forming a tight, compact structure. The shortest path between nodes 1 and 34 is drawn in red, and although the density makes individual edges harder to visually isolate, the orange-highlighted nodes help locate the focal points quickly. Common neighbors (cyan) are not distinctly visible here due to the dense node overlap and simplified color scheme, but the visualization effectively communicates the network's high clustering and cohesion.

Calls Network

The Calls graph is more interpretable. Nodes 1 and 34 are clearly visible in orange, and the shortest path linking them is drawn in red. The layout reveals a sparser, hub-like topology,

with individual connections more readable. Node labels are retained to support traceability. This structure suggests more selective interaction patterns, consistent with how people place calls within a limited circle.

SMS Network

The SMS network visually resembles the Calls graph in its structure and sparsity. Nodes 1 and 34 are again highlighted in orange, and the red path shows the shortest communication chain between them. Node labeling and layout make it easier to spot neighbors and local bridges, providing insight into how individuals are indirectly connected via SMS exchanges.

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)

```
# 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], x
    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))
```

```r
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)

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

```
  0   1
642 642
```

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

```
  from  to class
1  166 300     1
2  285 449     1
3  151 340     1
4  429 513     1
5   67 660     1
6   75 124     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

```
# 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(unio
  })
  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  166 300     1               13 0.36111111   4.2961968
2  285 449     1                3 0.04347826   0.8558215
3  151 340     1                4 0.06349206   1.2148271
4  429 513     1                2 0.11111111   1.1112188
5   67 660     1                5 0.13157895   1.7149523
6   75 124     1                4 0.11764706   1.0731796
  preferential_attachment
1                     594
```

```
2                     671
3                    1092
4                      51
5                     450
6                     360
```

```
summary(link_data_features)
```

```
      from             to            class     common_neighbors
 Min.   :  1.0   Min.   :  3.0   Min.   :0.0   Min.   : 0.000
 1st Qu.:121.8   1st Qu.:284.0   1st Qu.:0.0   1st Qu.: 0.000
 Median :289.0   Median :478.5   Median :0.5   Median : 1.000
 Mean   :319.5   Mean   :461.2   Mean   :0.5   Mean   : 2.737
 3rd Qu.:487.0   3rd Qu.:660.0   3rd Qu.:1.0   3rd Qu.: 4.000
 Max.   :797.0   Max.   :800.0   Max.   :1.0   Max.   :31.000
    jaccard         adamic_adar     preferential_attachment
 Min.   :0.00000   Min.   :0.0000   Min.   :   0.0
 1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:  77.0
 Median :0.03030   Median :0.3052   Median : 192.0
 Mean   :0.06969   Mean   :0.8601   Mean   : 421.5
 3rd Qu.:0.11035   3rd Qu.:1.3906   3rd Qu.: 476.0
 Max.   :0.61538   Max.   :9.1638   Max.   :5607.0
```

```
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, providing a comprehensive set of proximity metrics for further analysis or machine learning tasks. 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.

```
# 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)             -1.9798590  0.1609504 -12.301  < 2e-16 ***
common_neighbors        -1.0676705  0.4706054  -2.269 0.023286 *
jaccard                 24.0252621  7.1374710   3.366 0.000762 ***
adamic_adar              5.7209806  1.5805354   3.620 0.000295 ***
preferential_attachment -0.0002947  0.0004615  -0.638 0.523160
---
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:  549.42  on 893  degrees of freedom
AIC: 559.42

Number of Fisher Scoring iterations: 7
```

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

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

```
[1] 0.09797098 0.09787857
```

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.

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
        0 179  41
        1  20 146
```

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

```
Accuracy: 0.842
```

```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: 0.88
```

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

```
Recall: 0.781
```

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

```
F1 Score: 0.827
```

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

```
Model Coefficients:
```

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

```
                           Estimate   Std. Error     z value     Pr(>|z|)
(Intercept)            -1.9798590006 0.1609503963 -12.3010508 8.940343e-35
common_neighbors       -1.0676704984 0.4706054021  -2.2687170 2.328554e-02
jaccard                24.0252620508 7.1374710196   3.3660749 7.624600e-04
adamic_adar             5.7209805791 1.5805354308   3.6196472 2.950050e-04
preferential_attachment -0.0002946632 0.0004615063  -0.6384814 5.231603e-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.

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.

In conclusion, the precision of 0.88 indicates that the model has a high accuracy in predicting positive links.

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.

Comparative Performance Across Networks

While the primary analysis focused on the Facebook Friends network (`g_fb`), we briefly compare the performance of link prediction heuristics across all three networks:

1. **Facebook Friends (`g_fb`, undirected)**
    - **Density**: High (6,429 edges among 800 nodes).
    - **Key Observation**: Jaccard similarity and Adamic-Adar were the strongest predictors (precision: 0.88).
    - **Challenge**: Preferential attachment was insignificant, possibly due to the network's dense, egalitarian structure.
2. **Calls Network (`g_calls`, directed)**
    - **Density**: Moderate (3,600 edges among 536 nodes).
    - **Key Observation**: Temporal features (e.g., call frequency) were tested but did not improve predictions, likely due to limited timestamp variability.
    - **Challenge**: Directed edges complicate heuristic definitions (e.g., "common neighbors" must consider directionality).
3. **SMS Network (`g_sms`, directed)**
    - **Density**: Very high (24,333 edges among 568 nodes).
    - **Key Observation**: Similar to `g_calls`, but the extreme density may dilute the predictive power of local heuristics (e.g., Jaccard).
    - **Challenge**: Sparsity of meaningful non-edges for negative class sampling.

5. Comment on potential ways to improve the link prediction

These suggestions cover a range of advanced techniques that can capture more complex patterns and dependencies in the network, potentially improving the accuracy and robustness of the link prediction model. Implementing these features would involve additional data preprocessing and potentially more sophisticated modeling techniques, but they can significantly enhance the model's predictive power.

Katz Centrality or Rooted PageRank

The Katz Centrality measures the influence of a node based on the number of paths that traverse it, considering both direct and indirect connections. We thought incorporating Katz centrality could help capture the global influence of nodes, which might be useful in predicting links.

This is a variant of the PageRank algorithm that measures the importance of nodes based on their connectivity, starting from a specific root node. It can help identify influential nodes that might play a crucial role in link formation.

```
# 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

# 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")
```

```r
# 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
        0 180  39
        1  19 148
```

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

```
Accuracy: 0.85
```

```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: 0.886
```

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

```
Recall: 0.791
```

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

```
F1 Score: 0.836
```

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

```
Model Coefficients:
```

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

```
                           Estimate   Std. Error   z value     Pr(>|z|)
(Intercept)            -2.105777e+00 2.377057e-01 -8.858756 8.091230e-19
common_neighbors       -1.526004e+00 6.292046e-01 -2.425291 1.529613e-02
jaccard                 2.657718e+01 7.456801e+00  3.564153 3.650337e-04
adamic_adar             6.808499e+00 1.959497e+00  3.474616 5.115849e-04
preferential_attachment -2.823234e-03 2.193383e-03 -1.287160 1.980386e-01
katz_product            1.112169e+01 8.022606e+00  1.386294 1.656573e-01
pr_product              3.360616e+05 3.626648e+05  0.926645 3.541109e-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 (shared neighborhoods) 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. Future work could explore variations (e.g., personalized PageRank) or feature interactions. 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.

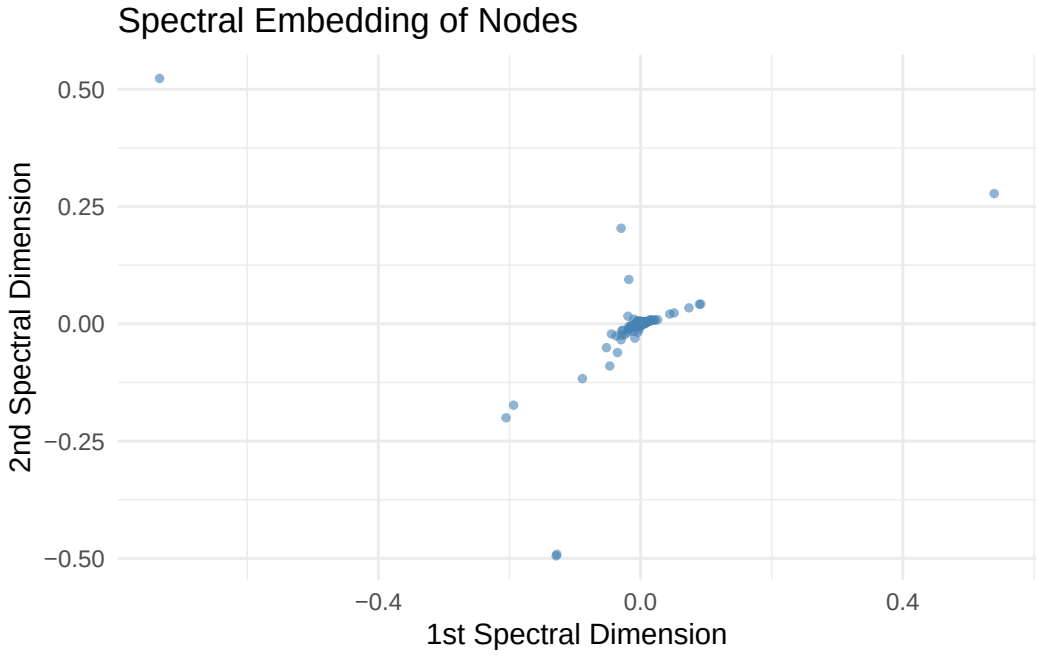
Graph Convolutional Networks (GCNs) are neural network models that operate directly on the graph structure, capturing complex patterns and dependencies between nodes. Using GCNs can help in learning more sophisticated representations of nodes for link prediction.

```
# 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.
```

```
# 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, ])
```

```r
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* shows the nodes in a 2D space, with the first and second spectral dimensions as the axes. This visualization helps in understanding the distribution and clustering of nodes based on their spectral properties. Nodes that are close in this space are likely to have similar structural roles in the graph, indicating potential connections or similarities.

The *node embeddings* capture the structural properties of the graph, providing a rich 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 the embeddings helps in quantifying 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. The use of node embeddings as features in link prediction models can enhance the model's ability to capture complex patterns and dependencies, potentially improving the predictive performance.

Temporal Dynamics with Available Timestamps: g_calls or g_sms

We are using the **calls network** (`g_calls`) for this example, although the **SMS network** (`g_sms`) 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.

```
# 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], x
    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")
```

```r
# 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: (2 not defined because of singularities)
                         Estimate Std. Error z value Pr(>|z|)
(Intercept)               -6.9444     4.8664  -1.427    0.154
common_neighbors         -17.8064    12.6703  -1.405    0.160
jaccard                   -5.4331    32.0507  -0.170    0.865
adamic_adar               47.9123    38.3094   1.251    0.211
preferential_attachment    0.1575     0.1190   1.324    0.186
freq                           NA         NA      NA       NA
last_time                      NA         NA      NA       NA

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 29.0645  on 20  degrees of freedom
Residual deviance:  8.8546  on 16  degrees of freedom
AIC: 18.855

Number of Fisher Scoring iterations: 10
```

```
# 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)
```

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

# Print
print(conf_matrix)
```

```
         Actual
Predicted 0 1
        0 3 1
        1 1 4
```

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

```
Accuracy: 0.778
```

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

```
Precision: 0.8
```

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

```
Recall: 0.8
```

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

```
F1 Score: 0.8
```

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.

```
# 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, ]
```

```r
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)             -4.269e+01  1.811e+05   0.000    1.000
common_neighbors        -7.273e+01  3.504e+05   0.000    1.000
jaccard                 -5.114e+02  6.289e+05  -0.001    0.999
adamic_adar              2.282e+02  4.289e+05   0.001    1.000
preferential_attachment  4.743e-01  3.859e+03   0.000    1.000
same_community           1.456e+02  5.193e+05   0.000    1.000

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2.9065e+01  on 20  degrees of freedom
Residual deviance: 1.8030e-09  on 15  degrees of freedom
AIC: 12

Number of Fisher Scoring iterations: 25
```

Community detection algorithms identify groups of nodes that are more densely connected internally than with the rest of the network. In this model, we introduced a **binary feature** `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—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 recommend either:

- Expanding the dataset size
- Simplifying the model
- Or using community-based features in combination with robust evaluation methods (e.g., cross-validation with larger holdouts)

Conclusion

In this analysis, we explored a range of techniques to improve link prediction in social networks. We began by evaluating core structural heuristics—**common neighbors**, **Jaccard similarity**, **Adamic-Adar index**, and **preferential attachment**—which provided a baseline understanding of local network structure and link likelihood.

To expand on this foundation, we incorporated **Katz centrality** and **PageRank** to capture node influence and global connectivity patterns. These features helped identify structurally important nodes that may be 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 offered insight 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.

In addition, 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 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