Agenda

1. Second Exam: Stage II
2. Network Science
3. Centrality: Six Degrees of Kevin Bacon

Second Exam: Stage II Here is the link for the final testing data set:

`http://dl.dropboxusercontent.com/s/jfahyqc3s4blq48/second_exam_test.csv`

Social Networks Network Science is an emerging interdisciplinary field that studies the properties of large and complex networks. Network scientists are interested in both theoretical properties of networks (e.g. mathematical models for degree distribution) and data-based discoveries in real networks.

The roots of network science are in the mathematical discipline of graph theory. There are a few basic definitions that we'll need before we can proceed.

- A *graph* $G = (V, E)$ is simply a set of *vertices* (or nodes) V, and a set of *edges* (or links, or even ties) E between those nodes. It may be more convenient to think about a graph as being a *network*. For example, on Facebook, each user is a vertex, and each *friend* relation is an edge connected two users. Thus, one can think of Facebook as a *social network*, but the underlying mathematical structures is just a graph. Discrete mathematicians have been studying graphs since Leonhard Euler posed the Seven Bridges of Königsberg problem in 1736.
- Edges in graphs can be *directed* or *undirected*. The difference is whether the relationship is mutual or one-sided. For example, edges in the Facebook social network are undirected, because friendship is a mutual relationship. Conversely, edges in Twitter are directed, since you may follow someone who is not necessarily following you.
- Edges (or less commonly, vertices) may be *weighted*. The value of the weight represents some quantitative measure. For example, an airline may envision its flight network as a graph, in which each airport is a node, and edges are weighted according to the distance (in miles) from one airport to another. [If edges are unweighted, this is equivalent to setting all weights to 1.]
- A *path* is a sequence of edges that connect two vertices. There may be many paths, or no paths, between two vertices in a graph, but if there are any paths, then there is at least one *shortest path*. The notion of a shortest path is dependent upon a distance measure in the graph (usually, just the number of edges, or the sum of the edge weights). [Djikstra's algorithm is the most common way to find shortest paths, which are not necessarily unique.]
- The *diameter* of a graph is the length of the longest shortest path between any two pairs of vertices. The *eccentricity* of a vertex v in a graph, is the greatest distance between that vertex and any other vertex. Thus, in some sense a vertex with a low eccentricity is more central to the graph.
- In general, graphs do not have coordinates. Thus, there is no right way to draw a graph. Visualizing a graph is more art than science, but several graph layout algorithms are popular.
- Centrality: since graphs don't have coordinates, there is no obvious measure of *centrality*. That is, it is frequently of interest to determine which nodes are most "central" to the network. There are many notions of centrality in a graph, but we will discuss three:
  - Degree centrality: The *degree* of a vertex within a graph is the number of edges to which it is adjacent. Thus, the number of degrees is a simple measure of centrality in which more highly connected nodes rank higher.

– Betweenness centrality: If a vertex v is more central to a graph, then you would suspect that more shortest paths between vertices would pass through v. This is the notion of *betweenness centrality*. Specifically, let $\sigma(s,t)$ be the number of shortest paths between vertices s and t in a graph. Let $\sigma_v(s,t)$ be the number of shortest paths between s and t that pass through v. Then the betweenness centrality for v is the sum of the fractions $\sigma_v(s,t)/\sigma(s,t)$ over all possible pairs (s,t). This figure is often normalized by dividing by the number of pairs of vertices (excepting v).

$$C_B(v) = \frac{2}{(n-1)(n-2)} \sum_{s,t \in V \setminus \{v\}} \frac{\sigma_v(s,t)}{\sigma(s,t)}$$

– Eigenvector centrality: This is the essence of Google's PageRank algorithm, which we will discuss next Tuesday.

Note that there are also notions of *edge centrality* that we have not discussed here.

- In a social network, it is usually believed that if person Alice and Bob are friends, and Alice and Carol are friends, then it is more likely than it otherwise be that Bob and Carol are friends. This the notion of *triadic closure* and it leads to measurements of *clusters* in real-world networks.

Six Degrees of Separation: Kevin Bacon flavor The idea of *Six Degrees of Separation* was conjectured by a Hungarian network theorist in 1929, and later popularized by a play (and movie staring Will Smith). Stanley Milgram's famous letter-mailing *small-world* experiment supposedly lent credence to the idea that all people are connected by relatively "social hops." That is, we are all part of a social network with a relatively small diameter (perhaps as small as 6).

A popular incarnation of this idea is the Kevin Bacon game. [Compare to the notion of an Erdös number in mathematics.] The idea is that every actor in Hollywood can be connected to Kevin Bacon in at most six movie hops. We'll explore this idea using the IMDB.

We'll create a Hollywood network using actors and actresses in the IMDB. In this network, each actor or actress is a node, and two actors share an edge if they have ever appeared in a movie together. Then we'll determine the centrality of Kevin Bacon.

First, we want to determine the edges, since we can then look up the node information based on the edges that are present. One caveat is that these networks can grow very rapidly! Thus, we'll be modest by starting with only popular (at least 100,000 ratings) feature films (`kind_id = 1`) from 2012, and we'll only consider the top 20 credited roles in each film.

```
require(RMySQL)
con = dbConnect(MySQL(), user="mth292"
    , host="rucker.smith.edu"
    , password="RememberPi", dbname="imdb")
E = dbGetQuery(con, "SELECT a.person_id as src, b.person_id as dest, a.movie_id
  , a.nr_order * b.nr_order as weight, t.title, idx.info as ratings
FROM imdb.cast_info a CROSS JOIN imdb.cast_info b USING (movie_id)
LEFT JOIN imdb.title t ON a.movie_id = t.id
LEFT JOIN imdb.movie_info_idx idx ON idx.movie_id = a.movie_id
WHERE t.production_year = 2012 AND t.kind_id = 1
AND info_type_id = 100 AND idx.info > 100000
AND a.nr_order <= 20 AND b.nr_order <= 20
AND a.role_id IN (1,2) AND b.role_id IN (1,2)
AND a.person_id < b.person_id
GROUP BY src, dest, movie_id")
head(E)
```

```
##    src   dest movie_id weight           title ratings
## 1 4106 271944  2608040     52 Zero Dark Thirty  128674
## 2 4106 298636  2608040     13 Zero Dark Thirty  128674
## 3 4106 463078  2608040    143 Zero Dark Thirty  128674
## 4 4106 485497  2608040    234 Zero Dark Thirty  128674
## 5 4106 559037  2608040    260 Zero Dark Thirty  128674
## 6 4106 564481  2608040    208 Zero Dark Thirty  128674
```

This gives us

```
nrow(E)

## [1] 8427
```

connections between only 45 films:

```
head(unique(E$title))

## [1] "Zero Dark Thirty"      "The Dark Knight Rises" "Argo"
## [4] "Taken 2"               "The Expendables 2"     "The Dictator"
```

There are two popular R packages for network analysis: `igraph` and `sna`. We'll use `igraph`. To build a graph, we just have to specify the edges, and whether we want them to be directed.

```
require(igraph)
g = graph.data.frame(E, directed = FALSE)
summary(g)

## IGRAPH UNW- 831 8427 --
## attr: name (v/c), movie_id (e/n), weight (e/n), title (e/c),
##   ratings (e/c)
```

Note that we have associated metadata with each edge. Namely, information about the movie that gave rise to the edge, and a *weight* metric (that I made up) based on the order in the credits where each actor appeared. [The idea is that top-billed stars are more likely to appear on screen longer, and thus have more meaningful interactions with more of the cast.]

Now let's get information about the vertices in this graph. We could have done with another JOIN in the original query, but this is more efficient. [Why?]

By default, `igraph` has assigned the `name` attribute to each vertex to be the IMDB ID for that actor that we pulled from the database. So we'll get that list (in order!) and ask the database for more information about those people.

```
vIds = paste(V(g)$name, collapse = ",")
V = dbGetQuery(con, paste("SELECT id as imdbId, name FROM imdb.name WHERE id IN (",
    vIds, ")"))
head(V)

##   imdbId              name
## 1   4106   Abkarian, Simon
## 2   4421   Aboutboul, Alon
## 3   5482      Abtahi, Omid
## 4  10875 Adig\xfczel, Naci
## 5  11119     Adkins, Scott
## 6  12302      Affleck, Ben
```

Now we'll update the graph with this new information.

```
Vg = get.data.frame(g, what = "vertices")
Vg$vId = 1:nrow(Vg)
V = merge(x = V, y = Vg, by.x = "imdbId", by.y = "name")
V(g)[V$vId]$imdbId <- V$imdbId
V(g)[V$vId]$Name <- V$name
```

Let's visualize this network. There are *many* graphical parameters that you may wish to set, and the default choices are not always good. In this case we have over 800 vertices, so we'll make them small, and omit labels.

```
plot(g, edge.color = "lightgray", vertex.size = 2, vertex.label = NA)
```

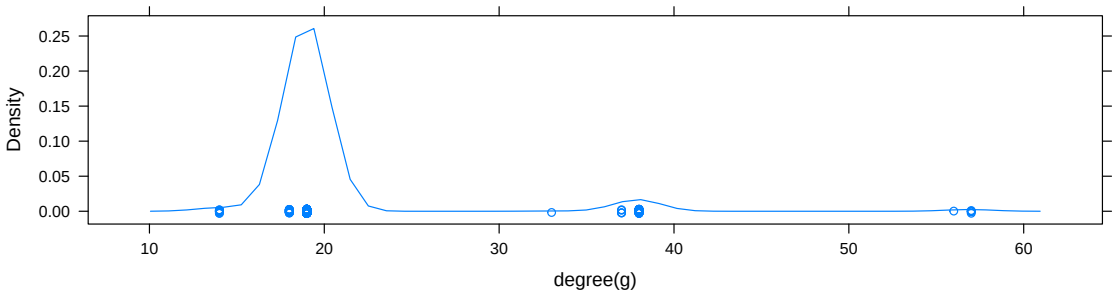
It's easy to see the clusters based on movies, but you can also see a few actors who have appeared in multiple movies, and how they tend to be more "central" to the network. If an actor has appeared in multiple movies, then it stands to reason that he will have more connections to other actors. This is captured by degree centrality.

```
V(g)$degree = degree(g)
head(get.data.frame(g, what = "vertices")[order(V(g)$degree, decreasing = TRUE),
    ])
```

```
##            name  imdbId                  Name degree
## 336923   336923  336923       Cranston, Bryan     57
## 599720   599720  599720 Gordon-Levitt, Joseph     57
## 681641   681641  681641      Hemsworth, Chris     57
## 1153136 1153136 1153136          Neeson, Liam     57
## 1555114 1555114 1555114    Stuhlbarg, Michael     57
## 1740353 1740353 1740353         Willis, Bruce     57
```

Unfortunately, the distribution of degrees is not terribly smooth. [Why?]

```
require(mosaic)
densityplot(~degree(g))
```



Why does Bryan Cranston have so many connections? The following quick function will retrieve the list of movies for a particular actor.

```
getMovies = function(imdbId, E) {
    tally(~title, data = subset(E, src == imdbId | dest == imdbId))
}
getMovies(336923, E)
```

```
##
##          Argo  John Carter Total Recall         Total
##            19            19            19            57
```

He was in three of these movies! Let's color the nodes based on their normalized degree centrality. We'll use a color palette from the ColorBrewer.

```
V(g)$deg = degree(g)/max(degree(g))
require(RColorBrewer)
palette = brewer.pal(5, "BuPu")
getColor = function(vals, palette) {
    # colorRamp returns a function that maps values in [0,1] to colors
    ramp = colorRamp(palette)
    color.mat = t(sapply(vals, FUN = ramp))
    return(rgb(color.mat, maxColorValue = 255))
}
V(g)$color = getColor(V(g)$deg, palette)
```

We don't want to show labels for everyone. We only want to show them for the highly central actors.

```
V(g)$label = ifelse(V(g)$deg > 0.9, V(g)$Name, NA)
# E(g)£label = ifelse(E(g)£weight == 0.5, E(g)£title, NA)
```

Let's also draw the more heavily-weighted edges thicker.

```
E(g)$width = 2 * E(g)$weight/max(E(g)$weight)
```

```
plot(g, edge.width = E(g)$width, edge.color = "lightgray", vertex.color = V(g)$color,
    vertex.label = V(g)$label, vertex.size = 2)
```

Degree centrality does not take into account the weights on the edges. If we want to emphasize the pathways through leading actors and actresses, we could consider betweenness centrality.

```
V(g)$btw = betweenness(g, normalized = TRUE)
head(get.data.frame(g, what = "vertices")[order(V(g)$btw, decreasing = TRUE),
    ])
```

```
##            name  imdbId                 Name degree    deg   color
## 599720   599720  599720 Gordon-Levitt, Joseph     57 1.0000 #810F7C
## 2646845 2646845 2646845      Stewart, Kristen     38 0.6667 #896BB1
## 89248     89248   89248       Bale, Christian     19 0.3333 #A6BAD9
## 1740353 1740353 1740353         Willis, Bruce     57 1.0000 #810F7C
## 374263   374263  374263     Day-Lewis, Daniel     19 0.3333 #A6BAD9
## 886717   886717  886717         LaBeouf, Shia     19 0.3333 #A6BAD9
##                         label    btw
## 599720  Gordon-Levitt, Joseph 0.2206
## 2646845                  <NA> 0.1771
## 89248                    <NA> 0.1751
## 1740353         Willis, Bruce 0.1683
## 374263                   <NA> 0.1630
## 886717                   <NA> 0.1543

getMovies(599720, E)
```

```
##
##              Lincoln               Looper The Dark Knight Rises
##                   19                   19                   19
##                Total
##                   57
```

How does this plot differ from the previous one?

```
V(g)$color = getColor(V(g)$btw/max(V(g)$btw), palette)
V(g)$label = iconv(ifelse(V(g)$btw > 0.1, V(g)$Name, NA), "latin1", "UTF-8")
plot(g, edge.width = E(g)$width, edge.color = "lightgray", vertex.color = V(g)$color,
    vertex.label = V(g)$label, vertex.size = 2)
```

If Joseph Gordon-Levitt (imdbId 599720) is very central to this network, then perhaps instead of a Bacon number, we could consider a JGL number. Christian Bale's JGL number is obviously 1, since they appeared in *The Dark Knight Rises* together:

```
jgl = V(g)[Name == "Gordon-Levitt, Joseph"]
get.shortest.paths(g, from = jgl, to = V(g)[Name == "Bale, Christian"], weights = NA)

## [[1]]
## [1] 198  22
```

On the other hand, his distance from Kristen Stewart is:

```
p = get.shortest.paths(g, from = jgl, to = V(g)[Name == "Stewart, Kristen"],
    weights = NA)[[1]]
length(p)

## [1] 5

V(g)[p]$Name

## [1] "Gordon-Levitt, Joseph" "Hardy, Tom"            "Pearce, Guy"
## [4] "Theron, Charlize"      "Stewart, Kristen"

E(g, path = p)$title

## [1] "The Dark Knight Rises"       "Lawless"
## [3] "Prometheus"                  "Snow White and the Huntsman"
```

Note that while the diameter of the graph is:

```
diameter(g, weights = NA)

## [1] 9
```

the eccentricity of JGL is:

```
eccentricity(g, vids = jgl)

## 599720
##      5
```

Thus, there is no actor in the network whose JGL number is greater than 5.
What will happen if you expand this network by going back further in time?

Further For more sophisticated graph visualization software, see Gephi.