An Examination of Gender Imbalances in Hollywood Movies

by Nyslai Bolaños, Sophie Hwang, and Ayesha Tariq

With assistance from:
Instructors: Takis, Stella, Smaranda
TA's: Angel, Becky, Moji, Hana, Jiarui, Maria

Introduction

Inequalities based on identity can be seen within nearly every facet of society, and gender and race are arguably the two greatest defining characteristics of identity. Historical oppression and the marginalization of individuals considered minorities have led to the imbalances of such identities in practically every societal sphere. From politics and academia to job opportunities and income disparities, imbalances have existed throughout time. We aim to examine imbalances of gender identity within the context of art and creative spaces. Within this realm, we can dive deeper into films and productions, as there have been great historical gender imbalances, with the rolies of producers, directors, actors, and crew often largely male and not having diverse gender identities. One 2017 study examining gender imbalances within director roles found that within the top 1,100 movies, only 4.3% of directors were female ("Hollywood's Hiring Freeze", 2018). This extreme statistic is representative of imbalances within one role within film productions, but we want to learn more about the same issue regarding the roles of actors. To understand more about this phenomenon, we will explore occurrences of gender imbalance of actors within the Hollywood film industry and will do so by examining data regarding Hollywood movies. The dataset we will be using contains variables such as "movie," "actor," and "gender," and looks at the top-grossing movies of 2015 ("The Next Bechdel Test").

We will examine the data by going through a series of tests, including the Bechdel test, which consists of the following questions:

- Does it have at least two named female characters?
- Do those characters have at least one conversation that is not about a man?

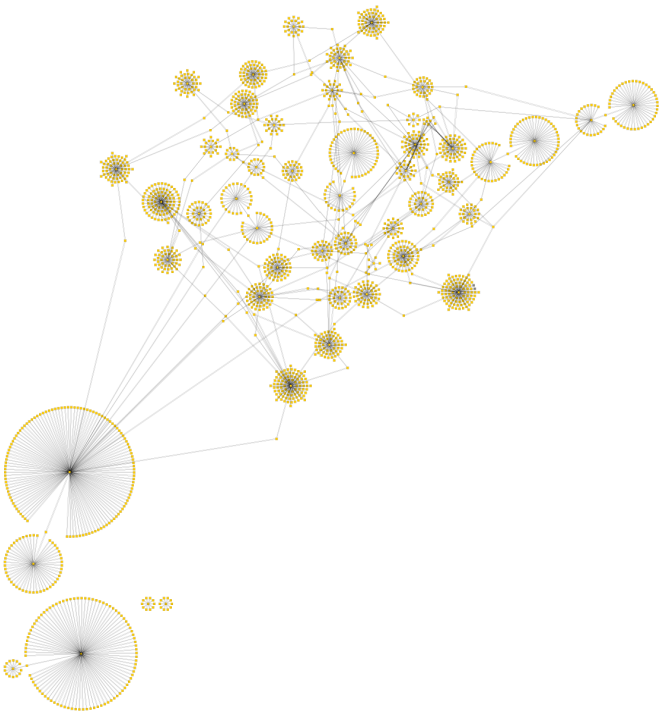
We will be modifying these questions to fit our dataset by asking the following question instead: is the movie's cast over 48% female? We will also be looking at the connections between actors and the movies they have had roles in, and vice versa.

Methods

Task 1.1 and 1.2:

In the first task, we implemented the HollywoodApp class. The HollywoodApp class parses through a file of movie information. Within the constructor, we create a hashtable called "castGender," a graph called "g," and vectors of strings called "actors" and "movies." Then,

within the AdjListsGraphFromFile() method we take a parameter input of the file, and then scan the file. In doing this, we populate the graph with vertices being each movie and actor within the file, and with the edges being vectors that have actors within a movie and the movies an actor has starred in. Within readCastandMovies(), we read through the file and filled the hashtable with the gender of the cast, and filled actors and movies as vectors. Our graph is undirected, and below is a yED visualization of it, alongside our response within the original Task 1.2 submission.



- **Task 1.2 Answer:**
- It was stated in the instructions that this is a bipartite graph. Because of this, we figured the graph would be in two rows. However, after inputting our data and creating a graph it looked different. In visualizing the relationships between actors and movies in the dataset through this graph, it can be seen that some of the movies included in the dataset, as shown in the bottom left, have a very large quantity of actors in their casts. When continuing onto the next task, which aims to discover whether a particular movie passes the Bechdel test, we predict that perhaps such movies may be more likely to do so, since they contain such large casts and are likely to have more female characters. On the contrary, movies with smaller casts, such as the ones depicted towards the top and right of the graph, may be less likely to do so. We wonder whether this prediction will actually hold true, and by completing task 1.3, we will be able to see exactly that.

Task 1.3:
Here, we looked at the movies within the dataset and determined whether their casts were over 48% female. We implemented this by creating a method called bechdelTest(), which parsed through castGender, and using a for loop, for each movie within the string of movies, it counted how many females there were, and then divided it by the total number of actors within the movie's cast. We compared this number to our final int variable which was equal to 48, and if the resulting number was greater than our variable of 48, then it would state that the movie passed the Bechdel test. It would do the opposite if the resulting number was below 48.

Task 2.1 + 2.2:
In the first two parts of the second task, we were assigned to answer two questions:

- ➢ Given an actor, what is the list of movies that the actor has played in?
- ➢ Given a movie, what is the list of all actors who have played in that movie?

To answer these questions we implemented two methods: givenActor() and givenMovie().
In the method, givenActor takes as a parameter the name of an actor, (a String). In order to store the list of all the movies this actor has played in, we decided to create a LinkedList called "movieLists" that would store that information. In order to add information into this LinkedList we had to iterate through all the movies. In our instance method, we created a Vector of movies that contains the names of all movies in the given file. For each movie in the movies Vector we checked if there was an edge between the actor name passed as a parameter and the current movie; we did so by using the isEdge() method. If an edge between the actor and the current movie existed, we checked whether or not movieLists already contained the movie. If not, then we would add the current movie to the list. Finally, once all the movies for a given actor had been added, we would return movieLists.

Task 2.3:
In task 2.3 we were asked to find the degree of separation between two actors. We created a method called doS that takes in two String parameters, called actor1 and actor2, corresponding to the two actors we are trying to find the degree of separation for. We created a LinkedList named "first" which stores actor1. We then created a LinkedQueue named "traversalQueue" to keep track of the paths in the BFS. The first list is enqueued to traversalQueue so that the BFS algorithm can start from the first actor (actor1). We decided to store our visited nodes in a vector named "visited." We added actor1 to it since it was the first one we visited. We added another LinkedList named "finalPath" to store the final desired path that we want. We did a try/catch math before performing our BFS search to make sure that exceptions were handled correctly. We created a while loop that runs as long as the "visited" vector does not contain the name of the second actor. In this while loop, we declared two linked lists "adjacent" and "nextPath". Adjacent stores the list of adjacent nodes to the current location being visited. Next path, will be the new path that will be enqueued. We dequeue a path from the traversalQueue and store it in

the path, we get the last location from the path and store it in “location”. This is so that we can get the location where the BFS will continue from. We checked if the location was a movie or an actor. It does this by checking if the given location is in the “movies” list. If the location is in the movies list, then that means the location is a movie and it calls the givenMovie method to get the list of actors who acted in that movie. If the location is not present in the movies list, it means that the location is an actor. It then calls the givenActor method to get the list of movies in which the actor acted in.

We created a for-loop to iterate through the list of adjacent nodes. It gets the current adjacent node using the adjacent.get(i) method. It creates a clone of the current path using the path.clone() and assigns it to the nextPath variable. We did this to avoid changing the original path and to create this path that can be modified. Inside this for loop there is an if statement, which checks if the adjacent node has not been visited. If so, it adds the adjacent node to the nextPath using the nextPath.add(nextLocation) method. It then enqueues the nextPath into the traversalQueue, because we want to explore all the nodes adjacent to the current node. As we do this, we also add the adjacent node to the visited set using the visited.add(nextLocation) method. Inside this if statement there is another if statement. If the adjacent is the target node(actor2) it assigns the nextPath to the finalPath. This is done to keep track of the shortest path between the start and target nodes. Once we exit this loop, we return the finalPath.size()/2-1. This expression calculates the number of edges in the shortest path between the two actors. Dividing the finalPath.size by 2 gives us the number of edges in the path, since each pair of adjacent nodes in the finalPath corresponds to a single edge we subtract one. This result gives us the number of edges in the shortest path.

Collaboration

Unfortunately, due to our differing schedules, we weren’t able to meet all together as a full group very often. However, we did meet successfully in pairs throughout the duration of the project. Work distribution was generally equal, with some individuals taking lead on certain tasks and others taking lead on different tasks and the report.

Works Cited

Fivethirtyeight. “The Next Bechdel Test.” *GitHub*, https://github.com/fivethirtyeight/data/tree/master/next-bechdel.

Staff, Communication and Marketing. “Hollywood's Hiring Freeze: Female, Black and Asian Directors Rarely Worked in 2017 Films.” *USC Annenberg School for Communication and Journalism*, 3 Jan. 2018, https://annenberg.usc.edu/news/research/hollywoods-hiring-freeze-female-black-and-asian-directors-rarely-worked-2017-films.