Physics Academic Genealogy Analysis

Julia Bohman

December 14, 2018

1 Introduction

There are a variety of network analysis tools useful in understanding relationships between people, group transition, and properties of those groups. While these methods have been explored by many researchers, there seems to be no instance in the existing literature of combining network analysis tools with classical statistics and machine learning methods to predict group transitions and properties. I intend to blend these different tools by building a model centered around two questions: "If people who studied physics at a graduate level return to academia, at which university do they end up teaching/performing research?", and "How many people do those who return to academia advise? Answering these questions fits in with my current research with Dr. Emily Evans and Dr. Benjamin Webb, which is focused on using network analysis and machine learning to predict group transition.

For data, I am using a portion of the physics academic tree, scraped from https://academictree.org/. This site includes academic genealogy for many different fields, such as economics, mathematics, physics, and many more. Thus, if a particular person is involved in multiple areas, they will be listed in each of the relevant trees. The site is organized by people within the tree; each person has all their academic parents (advisors) and children (advisees) displayed as a tree on one page. Clicking on the person's name then yields a biography page with more details on their academic relationships. The years, university, and whether they were a graduate student, research assistant, working on their post doc, etc. while studying under their advisor is listed. Additionally, there is some personal information provided, such as possible cross listings in other trees, links to personal websites, and personal areas research. The physics academic genealogy detailed in this site makes it perfect for this analysis.

Given that this site is run by a professional organization, the Academic Family Tree, it seems to be a valid source of data. Reading their "About The Academic Family Tree" page confirms the site strives for accurate data.

2 Data Scraping

Data from the Academic Family Tree is licensed to be used as long as the data is attributed to the Academic Family Tree. Its robots.txt file allows for web scraping and crawling, with a crawling delay of 10 seconds.

Obtaining the data was an intensive process. Given a list of urls to different individual's personal academic trees, generally going back 6 generations, I scraped all of the names from that tree. The next script I wrote used the names and urls from the first script and went through each url, clicking on the correct name to navigate to that individual's bio page. Once on the bio page, I was

able to use BeautifulSoup to get all the provided information. However, one of the blocks of information was not standardized in what fields it included. Corresponding to these differences in provided fields of information were differences in the html code. Thus, in order to get what information was there, I scraped all the visible text from this block, storing it as a single string. I stored all the scraped information as a dictionary of people's names, which each keyed to dictionaries of information from their bio pages. Overall, I scraped 927 unique names.

After scraping the data, I wrote the dictionary of dictionaries to a pickle file, and wrote a corresponding load function to unpickle the files. I ended up scraping and saving the results three times, in order to get enough unique names. Refer scrape.py for all the scraping code.

```
In [2]: # read in each data file of scraped data, using functions from scrape.py
        dic1 = scrape.load_data('data.pickle')
        dic2 = scrape.load_data('test_run.pickle')
        dic3 = scrape.load_data('more_data.pickle')
```

3 Data Cleaning

I stored all my scraped data as a dictionary of dictionaries. Some of the information stored in the dictionary was stored as a single string, and so I needed to use regular expressions to parse such strings and extract desired fields of information, such as name or possible cross listing in other academic trees. Unfortunately, these strings didn't all contain the same fields of information; given the irregularity of provided data within the string, it was impossible to write general regular expressions to grab each information corresponding to some fields. However, I was still able to write regular expressions to pick out if these fields were included at all. In light of this, I decided to encode these particular fields as binary variables: 1 if the field was included, and 0 otherwise. Another cleaning decision I made was years when a person had studied at a university as a datetime object. In instances where a range of years was provided, I took the latest year.

```
In [ ]: # reformat and clean the data dictionary, organizing it
        #by personal info, parents, children, and collaborators
        df_info = []
        df_par = []
        df_chil = []
        df_coll = []

        dics = [dic1,dic2,dic3]
        for dic in dics:
            # call cleaning function from clean.py to clean dictionaries
            personal_info, parents,children,collaborators = clean.parse_dictionaries(dic)

            # convert to Pandas Dataframes
            df_info.append(pd.DataFrame(personal_info))
            df_par.append(pd.DataFrame(parents))
            df_chil.append(pd.DataFrame(children))
            df_coll.append(pd.DataFrame(collaborators))
```

Another challenge I encountered was non standardized formats for how information could be entered. Specifically, the universities at which a person had studied could be entered in any

format. For example, BYU could be entered as BYU, Brigham Young University, BYU-Provo, etc. Added difficulty arose from university names being mispelled, or entered in foreign languages such as German and French. Finally, if a person linked to a specific university was cross listed in another academic tree, that tree was included in paranthesis after the university name. To attempt to consolidate these differences in names for the same university, I wrote a general parser to combine like universities. This parser simply cut off extraneous information within paranthesis included after the university name, and took into account differences in spacing of university names. For future work, I intend on writing a more specific university name cleaner that recognizes more duplicate universities, taking into account acronyms, commas, and abbreviations.

Other challenges included deciding on how to organize all of my data. I ended up opting to create four different dataframes; one of personal information, and then three others of different relationships: parents, children, and collaborators. Note that parents refer to a person's academic advisors, children are people the person advised, and collaborators are the person's collaborators.

```
In [4]: # clean the university names, and replace empty strings with nans
        dfs = [df_par,df_chil,df_coll]
        for df in dfs:
            for i in range(3):
                # call cleaning function from clean.py
                clean.clean_df(df_par[i])
                clean.clean_df(df_chil[i])
                clean.clean_df(df_coll[i])

        # concat dataframes into single dataframes
        per_info = pd.concat(df_info)
        parents = pd.concat(df_par)
        children = pd.concat(df_chil)
        collabs = pd.concat(df_coll)

        # get rid of duplicate indices, creating through the merges
        for df in [per_info,parents,children,collabs]:
            df.reset_index(drop=True,inplace=True)
```

```
# display the personal information dataframe
per_info.head(3)
```

	Area	Bio	Children_num	Collaborators_num	Cross_tree	Mean_distance	Parents_num	Website	name	current_loc
0	1	0	2	0	NaN	18.7	0	0	Remy T. Van de Walle	RadboudUniversityNijmegen
1	0	1	1	0	NaN	12.2	1	1	Rudolf Adriaan Mees	RUG
2	1	1	1	0	NaN	12.1	3	1	Johannes Bosscha, III	PolytechnicalSchoolinDelft

```
# display the academic parents dataframe
parents.head(2)
```

	location	name	parent	relation	years	triangles	clusters	squares
0	Utrecht	Rudolf Adriaan Mees	Richard van Rees	grad student	1867-01-01	0	0.0	0.0
1	Deventer	Johannes Bosscha, III	Volkert Simon Maarten van der Willigen	research assistant	1850-01-01	0	0.0	0.0

```
# display the academic children dataframe
children.head(2)
```

	child	location	name	relation	years	triangles	clusters	squares
0	Bernard Marie Karel Nefkens	RadboudUniversityNijmegen	Remy T. Van de Walle	grad student	1967-01-01	0	0.0	0.0
1	Theo E. Schouten	RadboudUniversityNijmegen	Remy T. Van de Walle	grad student	1977-01-01	0	0.0	0.0

```
# display the collaborators dataframe
collabs.head(2)
```

	collaborator	location	name	relation	years	triangles	clusters	squares
0	Leaf Turner	Dartmouth	David Campbell Montgomery	collaborator	NaT	0	0	0.0
1	Eric Herbst	OhioStateUniversity	Chun Ming Leung	collaborator	NaT	0	0	0.0

4 Feature Engineering

The site I scraped my data from clearly outlined the network structure of academic parents and children. However, this structure was no longer explicit after I scraped it. Thus, I needed to recreate the network structure through feature engineering.

I wrote a function that returns a NetworkX object of relations given either the parents, children, or collaborators data frame. Once I had the network, I performed a few different clustering algorithms to get different clustering coefficients for each person in the network. A clustering coefficient gives a measure for how connected one person is to others within the network. Higher coefficients indicate higher connectivity. After calculating these different clustering coefficients for each person, I added new columns in the appropriate data frame to record these coefficients. Each of these columns communicates a measure of how related or connected a person is to the rest of the network. All the feature engineering functions can be found in feature_engineering.py

The following gives a brief description of each different kind of clustering coefficient I used:

Triange clustering: this counts the number of triangles of which each node is a part. Given the ordered structure of my network, I expected this to generally be 0, which it is, but in instances where it is greater than zero, it denotes greater connectivity and possibly influence in the network.

Clustering Coefficient: this is closely related to triangle clustering. It is the fraction of possible triangles with a particular node. Again, the higher the clustering coefficient is, the more influential I expect a certain node to be.

Square Clustering: this calculates how many squares a single node is a part of.

```
In [11]: # call function from feature_engineering.py to calculate the different clustering
         # coefficients for the parent, child, and collaborator networks
         parents,children,collabs = feat.add_groups(parents,children,collabs)
```

One of my eventual objectives is to use this data to predict at which university people end up. However, a person's current university is not included in the data that I have. In order to find a person's current university, I looked up where their children had gone under them, and added that university as a person's current location. My reasoning is that if a child studies at a particular university under a particular parent, then the parent teaches at that university. Note this function is also contained in feature_engineering.py

```
In [12]: # Call feature_engineering.py to fill in each individual's current location
         per_info = feat.get_current_location(per_info,children)
```

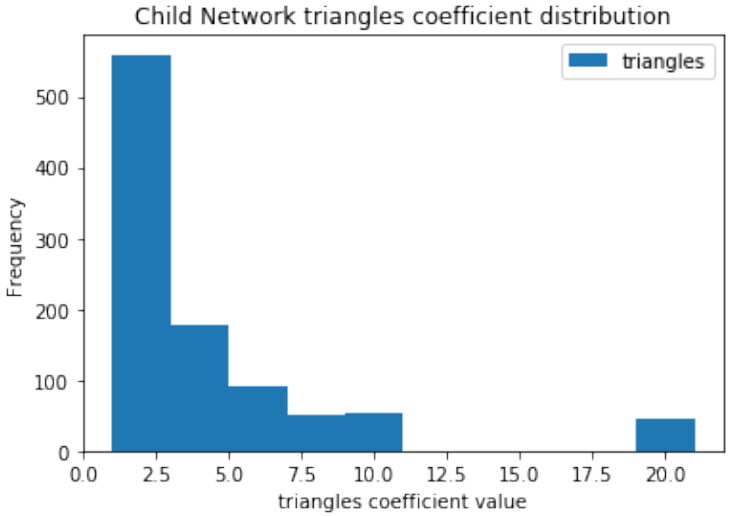
5 Data Visualization

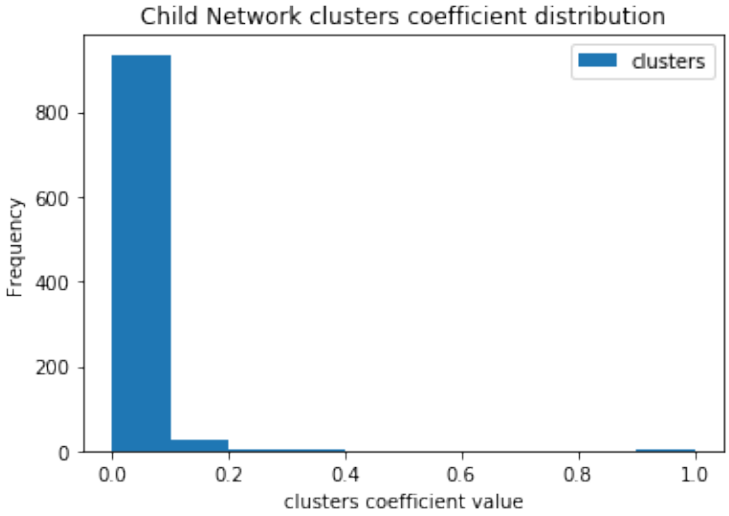
As mentioned in the feature engineering section, I calculated three different kinds of clustering coefficients for the parent and child networks. I found that for each kind of clustering coefficient,

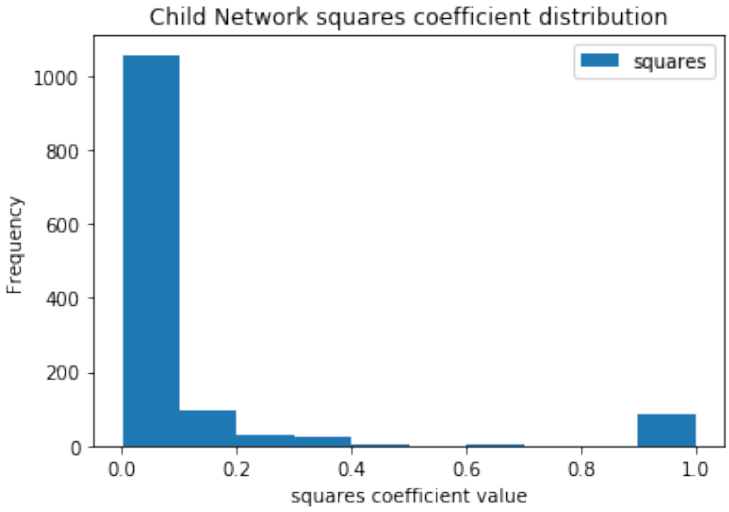
most people had a coefficient of zero. This is unsurprising given the hierarchical nature of my networks. Thus, in order to get a good idea of the distribution of clustering coefficients, I decided to exclude where the coefficient was zero.

Of particular interest to me was the density of different clustering coefficients within the child network as it gives information about connectivity between the people and their advisees.

```
In [38]: for clust in ['triangles','clusters','squares']:
             ax = children[children[clust]!=0].plot(kind='hist',y=clust,bins=10,title=f'Child Ne
             ax.set_xlabel(f'{clust} coefficient value')
             plt.show()
```



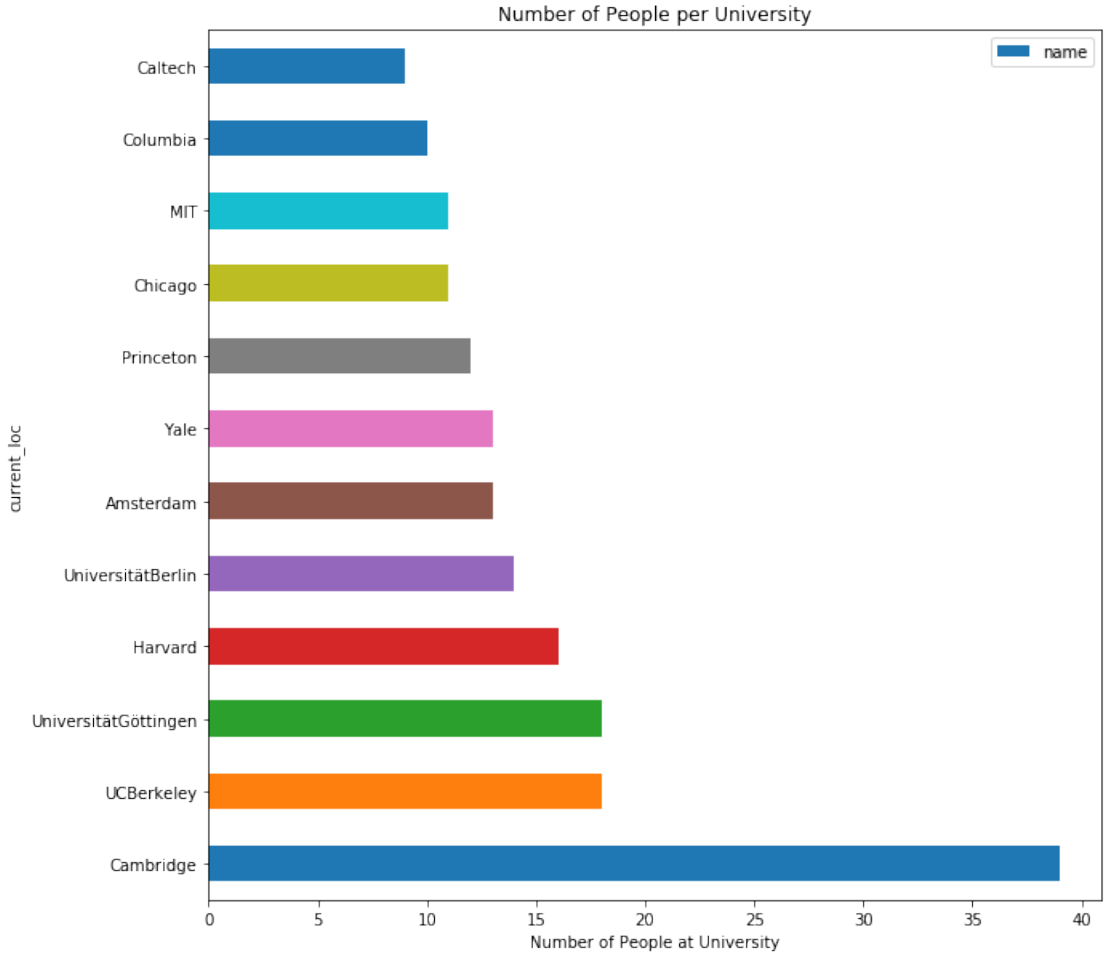




From the above plots, we can tell that the children network generally does not have high connectivity. However, given the acyclic nature of the network, this is unsurprising. These visualizations suggest different network tools may have to be used in order to determine connectivity, so connectivity scores will not all be skewed towards zero.

As acyclic networks are harder to analyze using traditional network tools, I decided to consider a network of universities. The following is a horizontal bar chart of the 12 most common universities for my original 927 people. I determined which universities were most common by counting the number of people who had each university listed as their current location, under the feature current_loc.

```
In [41]: univs = per_info.groupby(['current_loc'])['name'].count().reset_index('current_loc')
         ax = univs.sort_values(by='name',ascending=False)[:12].plot(kind='barh',x='current_loc'
                                  y='name',figsize=(10,10),title='Number of People per University
         ax.set_xlabel('Number of People at University')
         plt.show()
```



A simple calculation revealed that roughly 31% of people in the network have one of these top 12 universities listed as their current university. Considering there are over 200 unique universities in the network, these universities do a decent job capturing a significant portion of the population. The remaining ~200 universities each have only one or two people at them.

Next, I decided to run a linear regression on the number of children each person has. This entailed merging the children and personal information data frames together, and then performing one hot encoding on each of my categorical variables. Due to the large number of different universities, I decided to make only thirteen different categories for current location: 12 for the 12 most common universities, and one for the remaining universities. In order to pass my model into 'statsmodels', I also had to drop all rows with nan values. Overall, ~12% of rows in the data frame were dropped through this method.

```
In [17]: # combine personal information and children dataframes
         both = pd.merge(per_info,children,on='name')

         # one hot encode all the categorical variables
         Df = pd.get_dummies(both,columns=['Cross_tree','relation'],drop_first=True)
```

```
# get top 12 universities to one hot encode
top = univs.sort_values(by='name',ascending=False)[:12]['current_loc'].values

# one hot encode for the top 12 universities
for u in top:
    Df[u] = 0
    Df.loc[Df.current_loc==u,u] = 1

# drop string valued columns, and drop all rows with nans
Df = Df.drop(['name','current_loc','child','years','location'],axis=1)
Df.dropna(inplace=True,axis=0)

# convert to float type
Df = Df.astype(float)
```

I had to drop my engineered clustering features from the model, as they contained information about people's academic children. After dropping these features, I ran a standard linear regression on the remaining features. I came up with four different combinations of features to test out, and run linear regressions on. In order to determine the most optimal combination of features, I performed a k-fold cross validation on the four different feature combinations. After testing out different values of k, I found that 6 yielded the most optimal results for each cross validation. The following is the summary for what I found to be the optimal model, along with its 6-fold cross validation value.

```
In [37]: feats4 = Df.drop(['Children_num','Yale','Columbia','Harvard','UniversitätGöttingen',
                           'relation_research assistant','Cross_tree_ Systematics Tree',
                           'Cross_tree_ PsychTree','Cross_tree_ Neurotree',
                           'Cross_tree_ Meteorology Tree','Cross_tree_ MathTree',
                           'Mean_distance','Collaborators_num','Cross_tree_ Chemistry Tree',
                           'Cross_tree_ MichiganTree','Cambridge','triangles',
                           'clusters','squares'],axis=1).columns.tolist()
         X4 = sm.add_constant(Df[feats4])
         y4 = Df.Children_num
         res4 = sm.OLS(y4,X4).fit()
         print(res4.summary().as_latex())
```

Dep. Variable:	Children$_{n}um$	**R-squared:**	0.184
Model:	OLS	**Adj. R-squared:**	0.181
Method:	Least Squares	**F-statistic:**	51.16
Date:	Wed, 12 Dec 2018	**Prob (F-statistic):**	5.28e-138
Time:	19:31:07	**Log-Likelihood:**	-13546.
No. Observations:	3415	**AIC:**	2.712e+04
Df Residuals:	3399	**BIC:**	2.722e+04
Df Model:	15		

	coef	std err	t	P>|t|	[0.025	0.975]
const	6.3573	0.903	7.042	0.000	4.587	8.127
Area	5.8280	0.620	9.398	0.000	4.612	7.044
Bio	5.9180	0.805	7.351	0.000	4.340	7.496
Parents_num	1.7012	0.190	8.967	0.000	1.329	2.073
Website	-3.4734	1.003	-3.463	0.001	-5.440	-1.507
Cross_tree_ Astronomy Tree	-3.4971	1.002	-3.491	0.000	-5.461	-1.533
Cross_tree_ Crystallography Tree	17.7701	1.891	9.395	0.000	14.062	21.479
relation_post-doc	5.7732	0.563	10.253	0.000	4.669	6.877
relation_research scientist	5.8303	1.219	4.784	0.000	3.441	8.220
UCBerkeley	4.6742	1.024	4.563	0.000	2.666	6.682
UniversitätBerlin	-2.9461	1.152	-2.558	0.011	-5.204	-0.688
Amsterdam	-10.9521	1.730	-6.330	0.000	-14.344	-7.560
Princeton	12.9811	1.219	10.651	0.000	10.591	15.371
Chicago	-0.6830	1.248	-0.547	0.584	-3.129	1.763
MIT	-7.8685	1.542	-5.103	0.000	-10.892	-4.845
Caltech	-4.1891	1.662	-2.521	0.012	-7.447	-0.931

Omnibus:	225.835	**Durbin-Watson:**	0.124
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	270.930
Skew:	0.686	**Prob(JB):**	1.47e-59
Kurtosis:	2.856	**Cond. No.**	24.9

```
In [20]: from sklearn import datasets, linear_model
         from sklearn.model_selection import cross_val_score

         model4 = linear_model.LinearRegression()

         # perform 6 fold cross validation
         cv4 = cross_val_score(model4,X4,y4,cv=6,scoring='neg_mean_squared_error')
         print(f"6 fold cross validation score for model: {np.mean(cv4)}")
```

```
6 fold cross validation score for model: -209.80417816220893
```

Even after finding a more optimal model, the results from the linear regression were still not good. This indicates the data provided is not sufficient for a simple model predicting the number of children a particular person will have.

6 Conclusion

When I started this project, I hadn't considered how the tree's acyclic nature would make it harder to use traditional network tools to analyze it. Furthermore, the data I scraped was very sparse concerning individual's collaborators, which rendered the collaborator data that was there useless. This led me to shift my focus from potential social networks of a person's advisors to a potential network of universities, and predicting the number of academic children per person. It was only once I began to look at the universities closer that I realized how difficult it would be to consolidate different names, spellings, and languages for the each unique university in my data set. While

I was able to consolidate a significant portion of differences in names for universities with my general parser explained under the data cleaning section, it still bears further work. Running linear regressions on the number of children per individual in my data revealed my data is a poor model for this problem, as I never got above a -200 for any of my cross validation scores. I still believe this data has potential for modelling the transitions of people from graduate school to full time positions at universities. However, it will require more extensive cleaning and different techniques than I used here.