| Sarbjot Mann | FASS 210: Assignment 2<br>A phonological network, paths and connectedness | May 27th 2021 |
|---|---|---|

After loading the adjacency list *vitevitch.adjlist*. Using the networkx and matplotlib.pyplot python libraries and other code written by ourselves we further analyzed the properties of the network. Appendix 1 shows all of the code that was written to help analyze our network.

The network itself is connected, as there is a path between any vertex a and any vertex b. Additionally, using the is_connected function in Appendix 1 the value returned is a boolean value of True. The average shortest path in our network is 3.124338624338624. The degree of assortativity of our Vitevitch network is -0.19929528588970138, since our degree of assortativity is negative our network can be classified as diassortative (Menczer, 38).

Currently, our density has a value 0.17195767195767195. Certain nodes will make the graph disconnected if removed. which is shown by Table 1. Table 1 also shows the corresponding new density values and a list of components, which are colour coded for easier viewing.

| Node | New Density | List of Components |
|---|---|---|
| bog | 0.170940170 | [{'log', 'dig', 'dawn', 'dog', 'hog', 'fog', 'dug'}, {'cut', 'catch', 'gnat', 'coat', 'that', 'cad', 'bag', 'fat', 'calf', 'kit', 'can', 'chat', 'hat', 'kite', 'rat', 'cab', 'cot', 'cat', 'cattle', 'bat'}] |
| dog | 0.165242165 | [{'cut', 'catch', 'bog', 'gnat', 'coat', 'that', 'cad', 'fog', 'bag', 'fat', 'calf', 'kit', 'log', 'can', 'chat', 'hog', 'hat', 'kite', 'rat', 'cab', 'cot', 'bat', 'cattle', 'cat'}, {'dig', 'dug'}, {'dawn'}] |
| bag | 0.179487179 | [{'log', 'bog', 'dig', 'dawn', 'dog', 'hog', 'fog', 'dug'}, {'cut', 'catch', 'gnat', 'coat', 'that', 'cad', 'fat', 'calf', 'kit', 'can', 'chat', 'hat', 'kite', 'rat', 'cab', 'cot', 'cat', 'cattle', 'bat'}] |
| bat | 0.179487179 | [{'bag', 'log', 'bog', 'dig', 'dawn', 'dog', 'hog', 'fog', 'dug'}, {'cut', 'rat', 'gnat', 'coat', 'cab', 'cot', 'calf', 'catch', 'that', 'fat', 'kit', 'can', 'chat', 'cad', 'cattle', 'hat', 'cat', 'kite'}] |
| cat | 0.133903134 | [{'bag', 'log', 'bog', 'dig', 'dawn', 'dog', 'hog', 'fog', 'dug', 'bat'}, {'cattle'}, {'cut', 'coat', 'cot', 'kit', 'kite'}, {'calf', 'cad', 'cab', 'can'}, {'catch'}, {'rat', 'gnat', 'fat', 'that', 'chat', 'hat'}] |

Table 1: Removed Nodes That Make a Disconnected Network

As mentioned above, Table 1 shows nodes that if removed will make the graph disconnected, as well as thew new corresponding densitys and a list of the components. These nodes are "bog", "dog", "bag", "bat", "cat". Appendix 2,3,4,5,6 shows our corresponding disconnected network in graph form. The density decreases the most when "cat" is removed, followed by "dog" and "bog". However our density **increases** if "bag" or "bat" are removed. Therefore, our density increases the most if "bag" or" bat" are removed, our density decreases the least if "bog" is removed, our density decreases the most if "cat" is removed, and "dog" is in the middle of the removal of "cat" and "bog". The corresponding list of components can also be found for each graph once a specific node is removed.

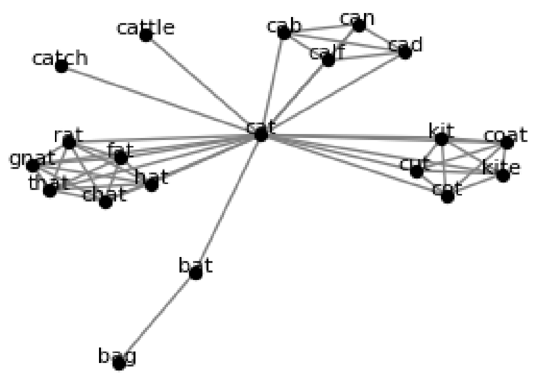
## References

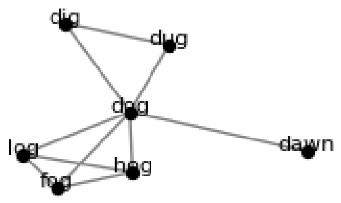
“2. Small Worlds.” *A First Course in Network Science*, by Filippo Menczer et al., Cambridge University Press, 2020, pp. 37–38.

## Appendix 1 - Code Used

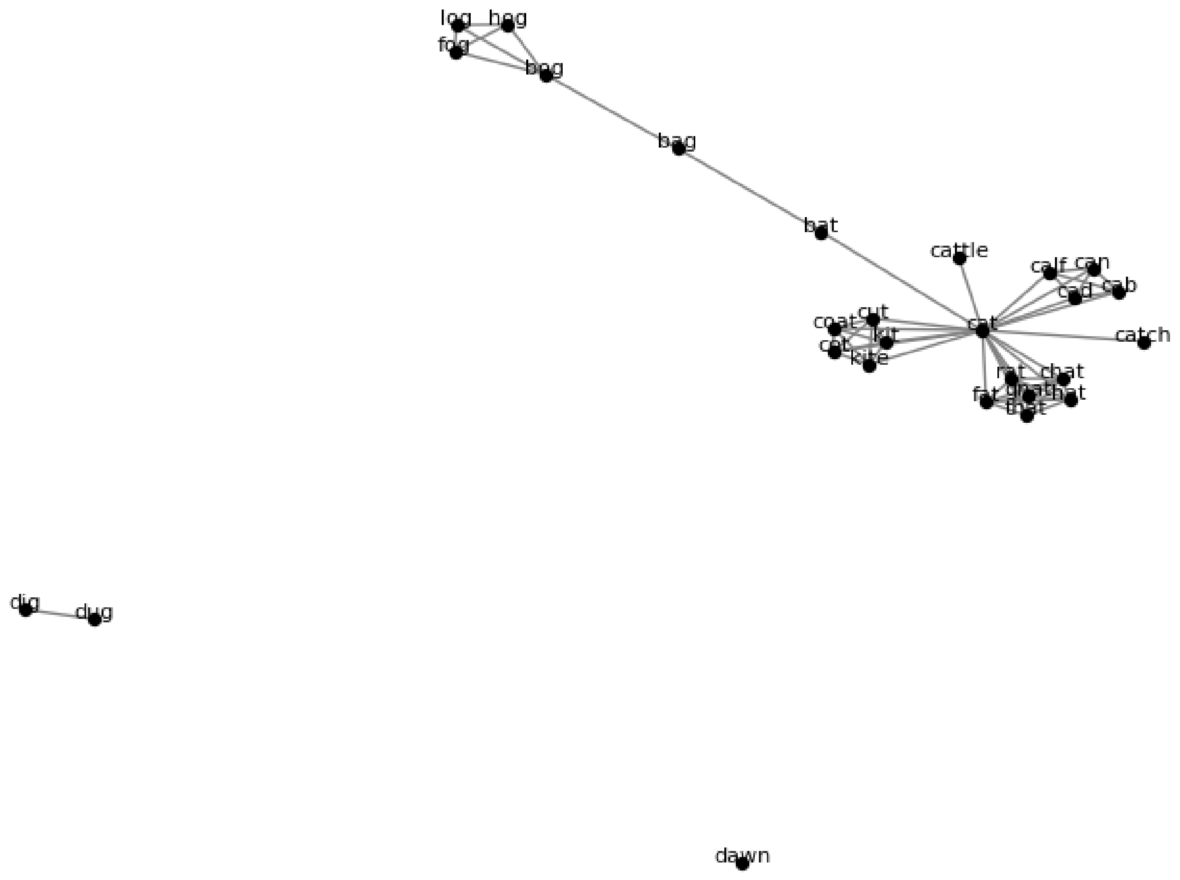
```python
import networkx as nx
import matplotlib.pyplot as plt  # Since we are not using a notebook we will import like this


def main():
    graph = nx.read_adjlist('../Datasets/vitevitch.adjlist')
    connected = is_connected(graph)
    avg_short_path = shortest_path(graph)
    deg_asrt = assortativity(graph)
    removed_nodes = disconnected(graph)
    density = network_density(graph.number_of_nodes(), graph.number_of_edges())
    reduce_density = reduce_least_density(removed_nodes)
    removed_components = components(removed_nodes)
    print(f"The truth value of connectedness is {connected} \n"
          f"The average shortest path in our network is {avg_short_path} \n"
          f"The degree of assortativity is {deg_asrt} \n"
          f"If we remove the nodes {removed_nodes} our graph will become disconnected \n"
          f"The Current density of our network is {density} \n"
          f"Removing the nodes {removed_nodes} resuls in densitys of {reduce_density} respectively
              \n"
          f"The list of components are as followed:  \n "
          f"\t - {removed_components[0]} \n"
          f"\t - {removed_components[1]} \n"
          f"\t - {removed_components[2]} \n"
          f"\t - {removed_components[3]} \n"
          f"\t - {removed_components[4]} \n")
    print(len(removed_components))

def components(nodes_list):
    return_components = []
    for nodes in nodes_list:
        graph_r = nx.read_adjlist('../Datasets/vitevitch.adjlist')
        graph_r.remove_node(nodes)
        return_components.append(list(nx.connected_components(graph_r)))
    return return_components

def reduce_least_density(nodes_list):
    difference = 0
    min_node = []
    for nodes in nodes_list:
        graph_r = nx.read_adjlist('../Datasets/vitevitch.adjlist')
        graph_r.remove_node(nodes)
        density = nx.density(graph_r)
        min_node.append(density)
    return min_node


def network_density(N, L):
    numerator = 2 * L
    denominator = N * (N - 1)
    return numerator / denominator


def disconnected(graph):
    return_nodes = []
    for node in list(graph.nodes()):
        graph_r = nx.read_adjlist('../Datasets/vitevitch.adjlist')
        graph_r.remove_node(node)
        if not nx.is_connected(graph_r):
            return_nodes.append(node)
            nx.draw(graph_r,
                    with_labels=True,
                    node_color='black',
                    node_size=18,
                    font_size=8,
                    verticalalignment='baseline',
                    edge_color='grey')
            plt.title(f"Removed node {node}")
            plt.savefig(f'{node}.png')
            plt.show()

    return return_nodes


def shortest_path(G):
    return nx.average_shortest_path_length(G)


def is_connected(G):
    return nx.is_connected(G)


def assortativity(G):
    return nx.degree_assortativity_coefficient(G)


if __name__ == '__main__':
    main()
```

**Appendix 2 - Bog Removal**

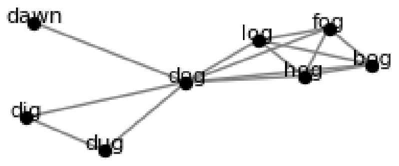


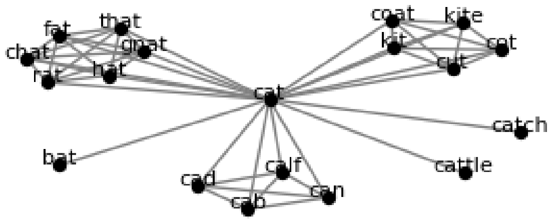


**Appendix 3 - Dog Removal**

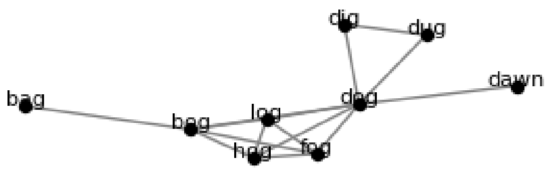


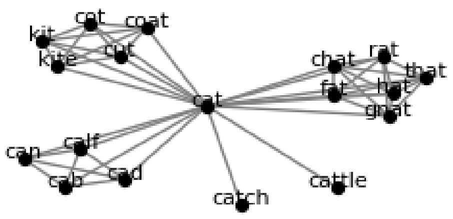
## Appendix 4 - Bag Removal



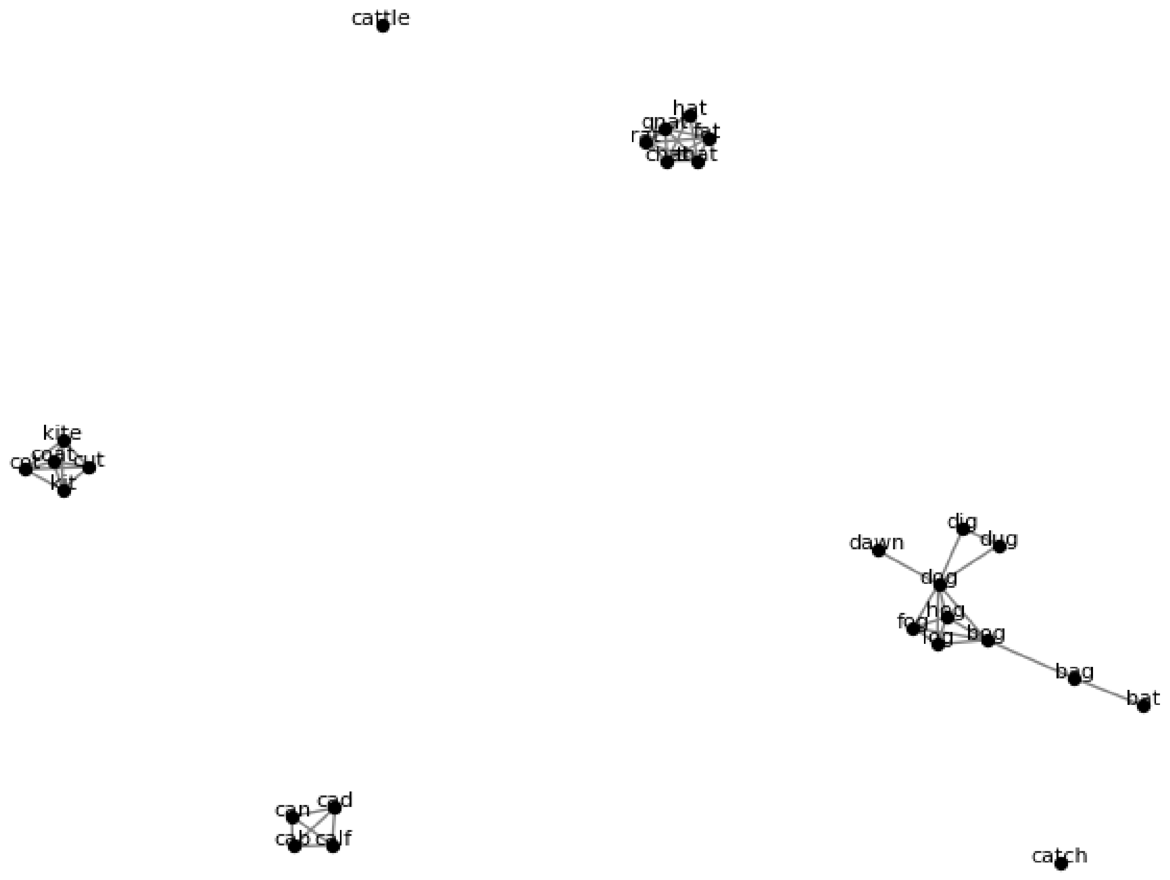


**Appendix 5 - Bat Removal**





**Appendix 6 - Cat Removal**


cattle
hat
gnat
rat
fat
chat
kite
coat
cot
cut
kit
dawn
dig
dug
dog
hog
fog
log
bog
bag
bat
can
cad
cab
calf
catch