CS 559 Final Project

Kendra Chalkley

June 27, 2018

1 Motivation and Goals

Goal: Write an agglomerative clustering algorithm for Reddit data

1.1 Collecting this Data

I initially collected this data as part of a replication project following In and Absolute State (Al-Mosaiwi and Johnstone, 2018). The study explored the use of 'absolutist' words (always, never, completely, etc.) in anxiety- and depression-related forums and found that absolutist words were used at a significantly higher rate in these groups than in controls. The paper also compares the frequency of various other collections of words (dictionaries of pronouns, negative emotion words, et al.) available through the LIWC (Linguistic Inquiry and Word Count) software.

I wanted to use distributed computing to investigate these frequencies in a larger body of work, specifically choosing Reddit data. Reddit.com is a website on which users can create and contribute to forums called 'subreddits.' Subreddits cover a vast number of subjects including various mental illnesses, computing and gaming, sports, politics, the economy, and a large number of 'adult' topics. The dataset I've collected to use here is in csv format and includes information about approximately 1200 subreddits to which users have contibuted text posts of at least 100 words in length. It includes the following columns:

- Subreddit - The name a subreddit (<=20 characters)
- Posts - A count of posts with 100 or more words in each subreddit
- Wordcount - A sum of the total number of words in all posts in that subreddit
- 65 additional columns, each containing the frequency of words in a specific dictionary
- 64 from an outdated version of LIWC plus the absolutist dictionary mentioned above.
- The frequency is calculated as:

$$\frac{number\ of\ words\ matching\ dictionary}{number\ of\ total\ words}$$

1.2 Why agglomerative clustering?

The original study, which I collected this data to replicate, selected their forums by google search. They analyzed a small collection of forums on a select number of topics. To approximate this approach, I tried running K-means clustering on this dataset for K between 5 and 25. No value of K clustering was obviously more effective than another, and, when projected onto the first two principal components, cluster boundaries seemed arbitrary and difficult to interpret.

Hierarchical clustering is my next step in exploring this dataset, as I hope it will provide more insight into the relationship between clusters.

2 The Process

2.1 Packages and Reading data

For this project I am using numpy to store my purely numerical data and execute operations over large arrays and matrixes. Providing additional support for numpy arrays is pandas, which is useful for accessing columns of data by a variable name and for labeling rows with an index name. I am also using Matplotlib's pyplot to produce a few scatter plots of the data projected onto its first two components.

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        % matplotlib inline


        inputs= pd.read_csv('medoutput.csv')
        subset_inputs= pd.read_csv('subset_medoutput.csv')

        freqs=inputs.loc[:,list(set(inputs.columns) - set(('subreddit','count(1)','sum(wordcount
        subset_freqs=subset_inputs.loc[:,list(set(inputs.columns) - set(('subreddit','count(1)',
```

2.2 Preprocessing

Because statistics for subreddits with a smaller number of posts vary so much more than subreddits with for which data was calculated from larger sample size, I've also curated a smaller subset of the data including only the half of subreddits with the most posts. This I read into to variable subset_inputs from a local csv.

Because this is my own dataset, the only additional preprocessing necessary was to normalize the values. The functions below normalize each column of the dataset to a range between 0 and 1, so that variation in dictionaries that have naturally higher values, such as verbs, would not have greater impact on PCA and clustering than variation in dictionaries which occur with lower frequency: leisure words for example.

```
In [2]: def normalize(vector):
            v2=vector
            vmin=float(min(vector))
            vmax=float(max(vector))
            vrange=vmax-vmin
```

```
#    print('vector', vector, vrange)

    l=len(vector)
    for i in range(0,l):
        dif=float(vector[i])-vmin
#        print('dif',dif, vrange, dif/vrange)
        v2[i]=dif/vrange
#        print(vector[i])
    return v2

def normalizeall(df):
    df2=df
    i=df.shape[1]
    for j in range(0,i):
        df2[:,j]=normalize(df2[:,j])
    return df2

nfreqs=normalizeall(np.array(freqs))
nsubset_freqs=normalizeall(np.array(subset_freqs))
```

3 The Algorithm

3.1 Distance measures: Points and Clusters

In this instance, I'm using Eudlicean distance to measure separation of points and distance between means to measure separation of clusters. (Distance between clusters is then Euclidean distance between cluster centers.)

```
In [3]: def pointDistance (point1, point2):
            dimNum=len(point1)
            if dimNum!=len(point2):
                print("we have a dimensionality problem, Houston")
                print(dimNum,len(point2))

            dif=point1[:-1]-point2[:-1]
            s=sum(dif.T*dif)
            ssqrt=np.sqrt(s)
            #print('qrt2',ssqrt)
            return ssqrt

        def findCenter(points):
            n=np.shape(points)[0]
            d=np.shape(points)[1]
            center=np.zeros(d)
            for i in range(0,d):
                center[i]= np.mean(points.iloc[:,i])
            return center
```

3.2 Distance Matrix

To initialize the algorithm, I created a two dimensional array recording the distance between all points in the dataset. Because the distance between any point and itself will always be 0 and distance metrics are commutative, measures along the diagnal and in the lower half of the matrix are set to null, reducing reduncancy and improving time efficiency.

```
In [4]: data=nsubset_freqs.copy()
        n=np.shape(data)[0]
        d=np.shape(data)[1]

        df=pd.DataFrame(data)
        df['cluster']=df.index
        centers=df

        distanceMatrix=np.zeros((n,n))

        for i in range(0,n):
            pointa = centers.iloc[i,]
            for j in range(0,n):
                if j<=i:
                    distanceMatrix[i][j]=None
                else:
                    pointb = centers.iloc[j,]
                    distanceMatrix[i][j]=pointDistance(pointa, pointb)
                    #print(i,j,distanceMatrix[i][j])

        matrixCopy=distanceMatrix.copy()
        history=pd.DataFrame({'round1': df['cluster']})
```

3.3 Merging Groups and Recording History

Numpy's nanmin() function efficiently finds the minimum value in a matrix, ignoring null values. To identify the two closest clusters, I use this function to get the minimum value in the distance matrix, then create an array of booleans to represent where the the distance is equal to that minimum value. From this I take the indexes of that minimum value. If two pairs of clusters have the same minimum distance, this will get the one with the smaller row index, which is unimportant except in that it is consistent every repetition. These indexes represent the ids of the clusters. I then save the lower of the two cluster ids in a variable 'groupa' and the larger in 'groupb'.

Next, I use the same boolean masking strategy to merge clusters and identify where and how the distance matrix needs to be updated. I start by moving all points in groupb to groupa. This means that cluster labels will be gradually smaller as the clusters combine. In retrospect, I wish I had taken the label of the cluster containing a larger number of points, as this might have helped with visualization.

Then I find the new center of groupa, remove groupb from the list of cluster centers, and update the distance matrix for both points. I also store the current state of the cluster assignments in an matrix called 'history' so I can reproduce the hierarchy of cluster structures and the state of the clustering at various iterations.

In [5]:
```
#distanceMatrix=matrixCopy.copy()
#data=nsubset_freqs.copy()
df=pd.DataFrame(data)
df['cluster']=df.index
centers=df.copy()
merges=pd.DataFrame(np.zeros((n,3)),columns=['head', 'leaves', 'groupsize'])

iternum=0
while np.nanmin(distanceMatrix)>0:
    minimum=np.nanmin(distanceMatrix)

# get indexes where distance is minimum
    mask=np.isin(distanceMatrix, minimum)
    a,b =np.where(mask)
# label cluster with smaller label groupa
    groupa=min(a[0],b[0])
    groupb=max(a[0],b[0])

#Find indexes of points which were in group b.
    mask=np.isin(df['cluster'], groupb )
    index=np.where(mask)

#Reassign them to group a
    df.loc[mask,'cluster']=int(df.loc[groupa,'cluster'])

#Find points in freshly minted group a
    mask=np.isin(df['cluster'], groupa)
    groupPoints= df[mask]

#Update the cluster center of group a
    centers.iloc[groupa]=findCenter(groupPoints)
#Remove center of group b
    centers.iloc[groupb]=None

#Save some information to reconstruct hierarchy
    merges.loc[iternum,'head']=groupa
    merges.loc[iternum,'leaves']=groupb
    merges.loc[iternum,'groupsize']=len(groupPoints)

    history[str(iternum)]=df['cluster']
#Remove group b from distance matrix
    distanceMatrix[groupb,:]=None
    distanceMatrix[:,groupb]=None

#Update Distances to groupa
    i=groupa
    pointa = centers.iloc[i]
    for j in range(0,n):
```

```
                if j>groupa:
                    pointb = centers.iloc[j,]
                    distanceMatrix[i][j]=pointDistance(pointa, pointb)

            iternum+=1

/home/kchalk/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:9: RuntimeWarning: All-
  if __name__ == '__main__':
```

In the lines below, I reattach the subreddit names to associated points and count the number of times a given cluster label appears in the tree. I used this information to chose the cluster labels which I then graphed in my second collection of scatter plots below.

```
In [34]: history.index=subset_inputs['subreddit']

         merges.groupby('head').count().sort_values('leaves', ascending=False).head(10)
```

Out[34]:

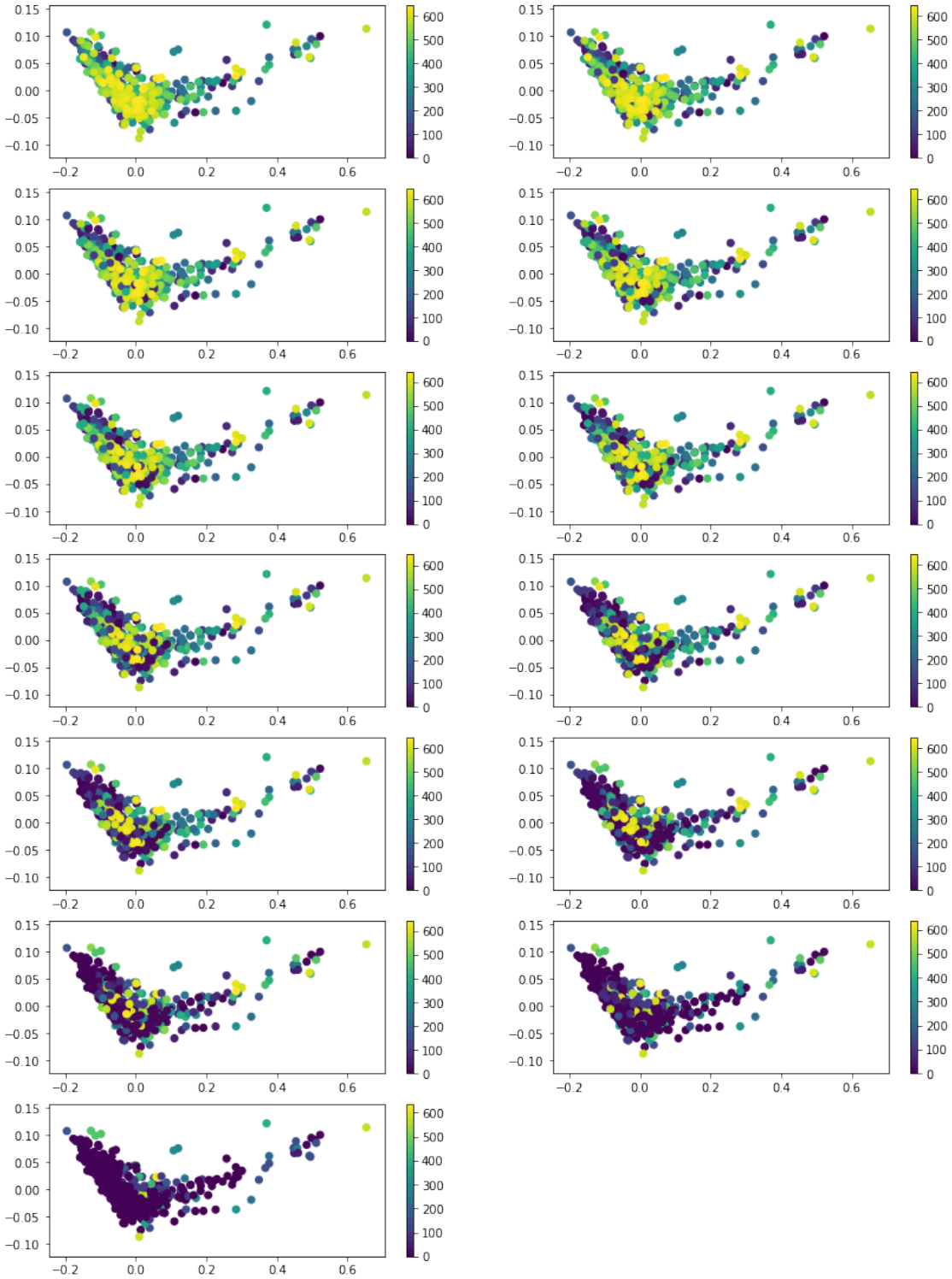
head	leaves	groupsize
12.0	37	37
0.0	33	33
5.0	30	30
16.0	29	29
4.0	17	17
140.0	16	16
1.0	13	13
26.0	11	11
111.0	10	10
8.0	9	9

4 Graphs

In the first graph I made to investigate this process, I projected the data onto two dimensional PCA space. I am graphing all 600+ points in a scatter plot colored by cluster. Because of the size of the dataset, the only thing that one can actually see is that the number of colors/clusters reduces as the algorithm runs longer.

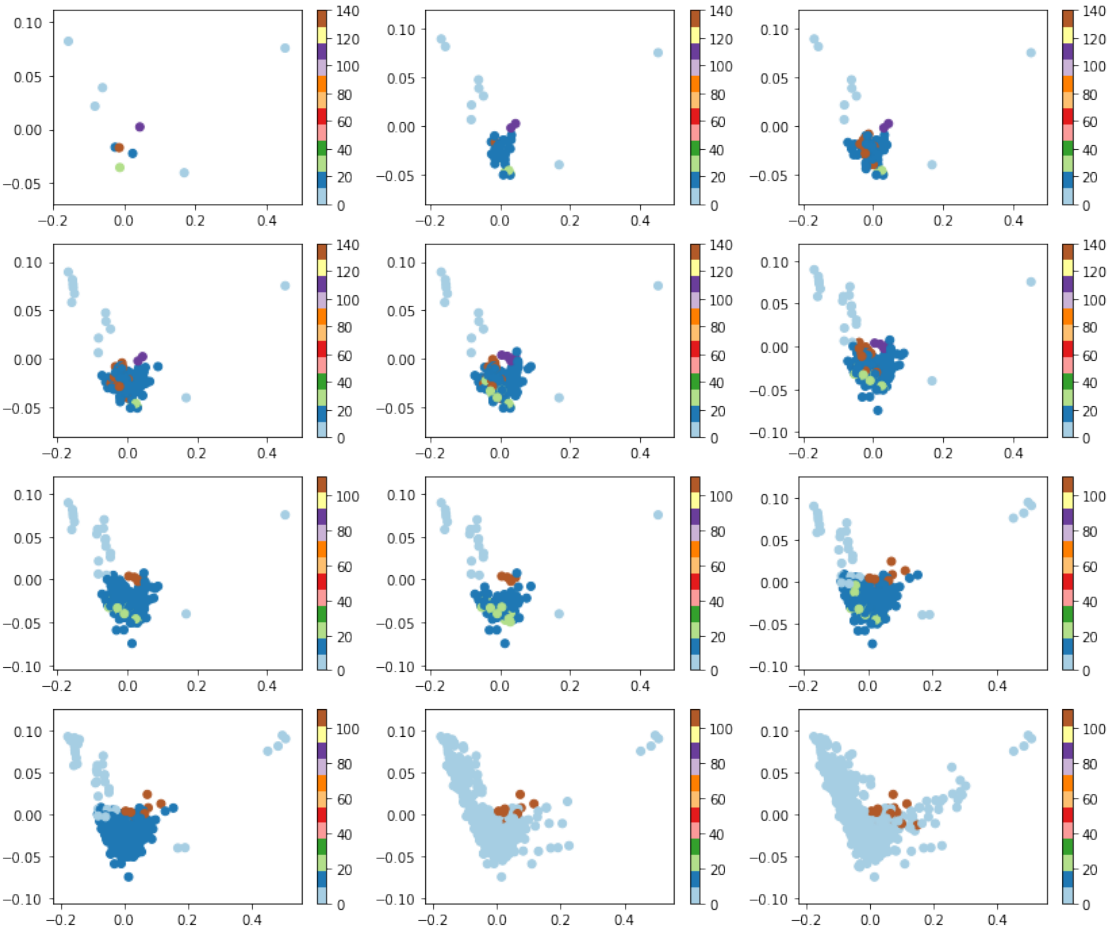
```
In [19]: from sklearn.decomposition import PCA

         pca=PCA(n_components=11)
         pcs=pca.fit_transform(subset_freqs)
         plotData=pd.DataFrame(pcs, index=subset_inputs['subreddit'])

         plots=[]
         j=0

         fig = plt.figure(figsize=(14,28))
```

```
for i in range(0,639, 50):
    plotData['cluster']=history.loc[:,str(i)]
    j+=1
    fig.add_subplot(10, 2, j)
    plt.scatter(x=plotData[0],y=plotData[1], c=plotData['cluster'])
    plt.colorbar()

plt.show()
plt.savefig('scatter1.png')
```



Below, a slightly improved set of scatter plots shows membership of those clusters which maintained the same cluster name through several merges. It is an imperfect system, but it does show patterns for some of the larger clusters.

```
In [35]: fig = plt.figure(figsize=(14,28))

         j=0
         for i in [0,150,200,250,300,380,390,400,460,480,520,600]:
             plotData=pd.DataFrame(pcs, index=subset_inputs['subreddit'])
             plotData['cluster']=history.loc[:,str(i)]
             mask=np.isin(plotData['cluster'], [12,0,5,16,4,140,1,26,111,8])
             plotData=plotData.loc[mask]
             j+=1
             fig.add_subplot(9, 3, j)
             plt.scatter(x=plotData[0],y=plotData[1], label=plotData.index, c=plotData['cluster'
             plt.colorbar()


         plt.show()
         plt.savefig('scatter2.png')
```

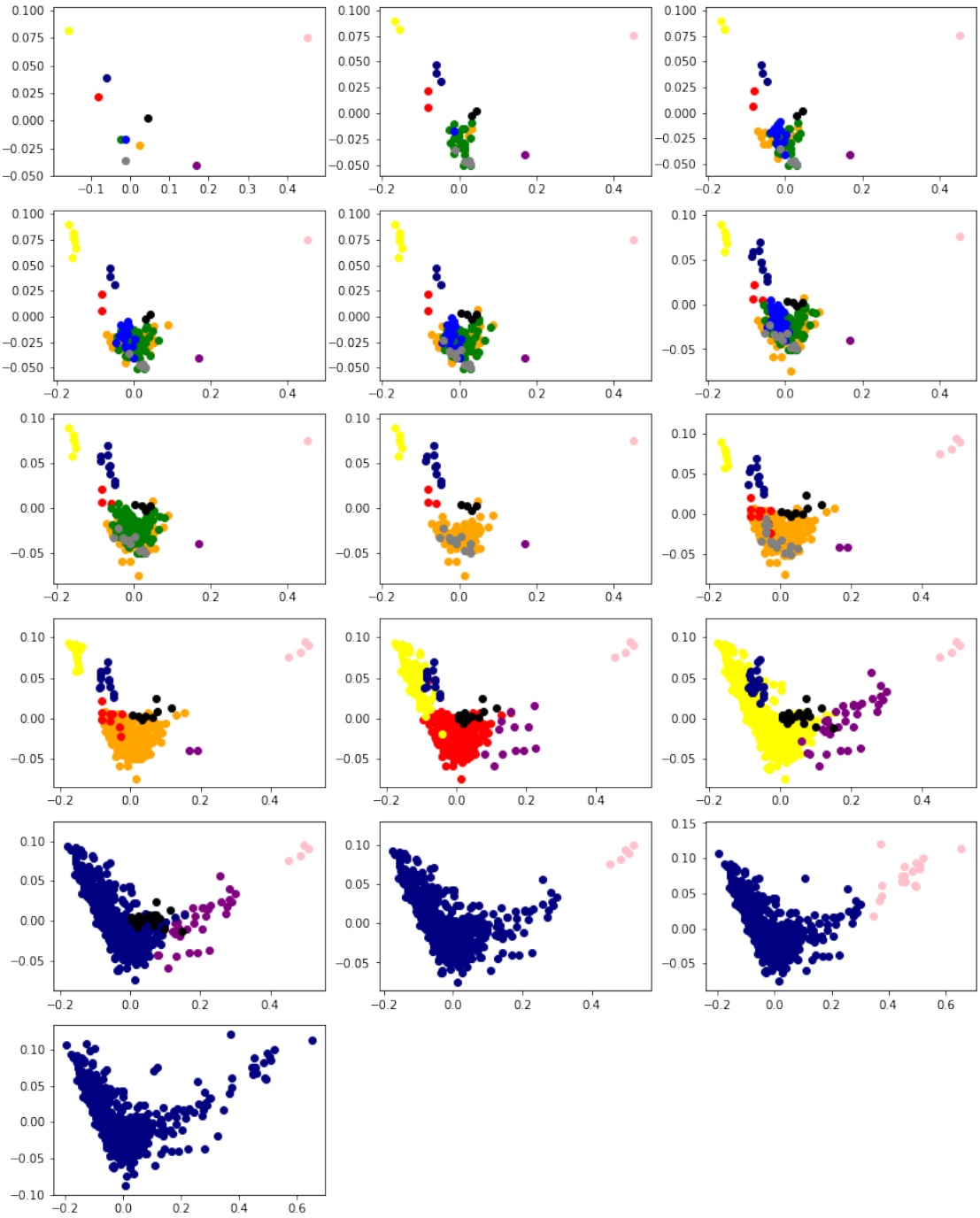


Below is a version of the same figure in which I manually changed colors to be more contrasting and consistent across iterations.

```
In [53]: fig = plt.figure(figsize=(14,28))

         j=0
         for i in [0,150,200,250,300,380,390,400,460,480,520,600,610,625,636,645]:
             plotData=pd.DataFrame(pcs, index=subset_inputs['subreddit'])
             j+=1
             fig.add_subplot(9, 3, j)

             plotData['cluster']=history.loc[:,str(i)]

             mask=np.isin(plotData['cluster'], [8])
             plotSubsetData=plotData.loc[mask]
             plt.scatter(x=plotSubsetData[0],y=plotSubsetData[1], label=plotData.index, c='pink'

             mask=np.isin(plotData['cluster'], [12])
             plotSubsetData=plotData.loc[mask]
             plt.scatter(x=plotSubsetData[0],y=plotSubsetData[1], label=plotData.index, c='orang

             mask=np.isin(plotData['cluster'], [5])
             plotSubsetData=plotData.loc[mask]
             plt.scatter(x=plotSubsetData[0],y=plotSubsetData[1], label=plotData.index, c='red')

             mask=np.isin(plotData['cluster'], [16])
             plotSubsetData=plotData.loc[mask]
             plt.scatter(x=plotSubsetData[0],y=plotSubsetData[1], label=plotData.index, c='green

             mask=np.isin(plotData['cluster'], [4])
             plotSubsetData=plotData.loc[mask]
             plt.scatter(x=plotSubsetData[0],y=plotSubsetData[1], label=plotData.index, c='yello

             mask=np.isin(plotData['cluster'], [140])
             plotSubsetData=plotData.loc[mask]
             plt.scatter(x=plotSubsetData[0],y=plotSubsetData[1], label=plotData.index, c='blue'

             mask=np.isin(plotData['cluster'], [1])
             plotSubsetData=plotData.loc[mask]
             plt.scatter(x=plotSubsetData[0],y=plotSubsetData[1], label=plotData.index, c='purpl

             mask=np.isin(plotData['cluster'], [26])
             plotSubsetData=plotData.loc[mask]
             plt.scatter(x=plotSubsetData[0],y=plotSubsetData[1], label=plotData.index, c='gray'

             mask=np.isin(plotData['cluster'], [0])
             plotSubsetData=plotData.loc[mask]
             plt.scatter(x=plotSubsetData[0],y=plotSubsetData[1], label=plotData.index, c='navy'

             mask=np.isin(plotData['cluster'], [111])
             plotSubsetData=plotData.loc[mask]
```

```
        plt.scatter(x=plotSubsetData[0],y=plotSubsetData[1], label=plotData.index, c='black
```

```
plt.show()
plt.savefig('scatter3.png')
```



```
<Figure size 432x288 with 0 Axes>
```

5 Future Directions

I would like to have drawn trees and subtrees showing the merge history of the clusters, but I don't currently know to draw trees in python. Additionally I'd like to try this algorithm using mean distance between points to judge cluster similarity.