Linguistic Analysis, Data Mining, and Clustering to Predict Document Age

Student: Keegan Freeman

Research Professor: Dr. Shishir Shah

Goal of Research

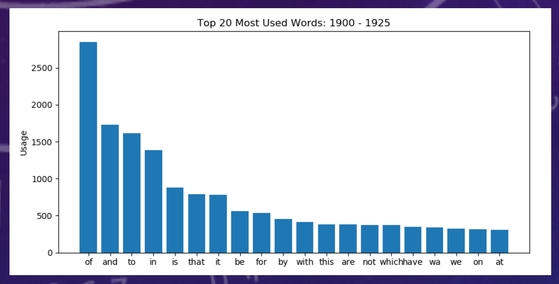
This research aims to create a program capable of iterating through an entire corpora of plain text documents and extract enough linguistic structure data so that, through data clustering, time prediction can be performed on the various documents both in and outside the given corpora. The corpora used is limited to American communication in the form of articles and periodicals less than five pages long so that a diverse corpora representing generalized American communication could be formed.

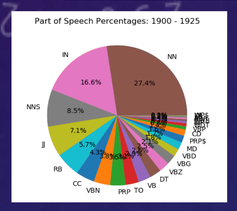
Metrics and Collected Data

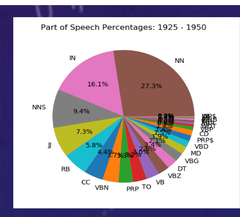
A variety of data was extracted from each text source and recorded in the written research program. This data includes for each source as well as for each time-span: an array of the stemmed words used, an array of the parts of speech for each stemmed word, word tally, part of speech tally, bigram and trigram collocations for words used together, bigram and trigram collocations for parts of speech used together, word tally percentage, part of speech percentage, and average sentence length.

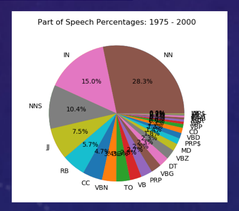
The metrics taken from this collected data to be used in clustering and therefore the data used for time prediction are as follows: average sentence length, top 25 part of speech percentages, top 20 part of speech bigram collocations, and top 20 part of speech trigram collocations.

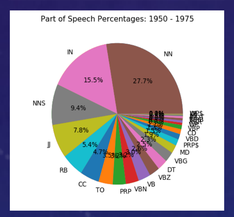
After storing each text file's metrics into a data point, filtering was performed to deem which metrics are most beneficial to be used for clustering.

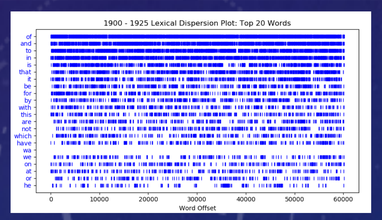






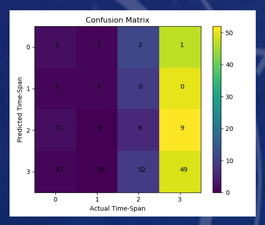


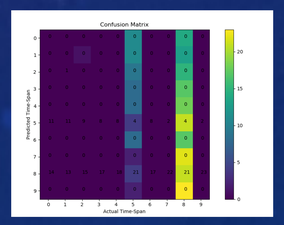




Confusion Chart

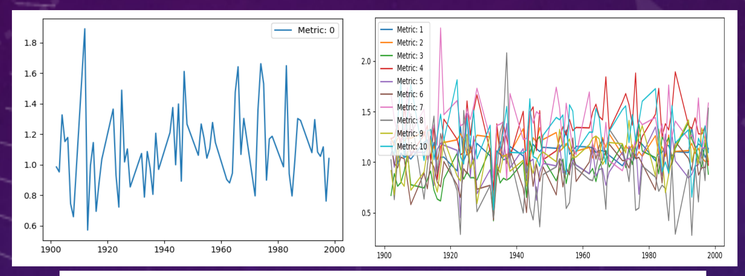
To visualize how many source clusters were correctly associated to their proper time-span clusters, a confusion chart was made. This operates by having an x axis of actual time-span values and a y axis of predicted time-span values. Tallies are placed in each cross section showing the number of actual time spans placed into the predicted time span cluster. Ideally, high tallies should be along the diagonal of the confusion chart indicating the correct predicted time-spans were placed in the proper actual time-span slots

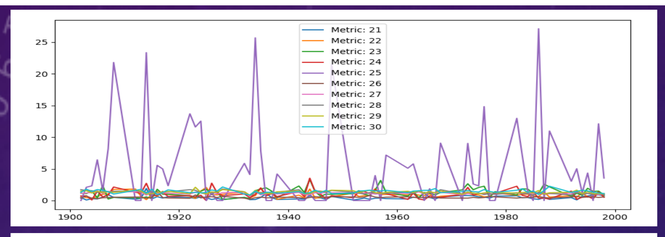


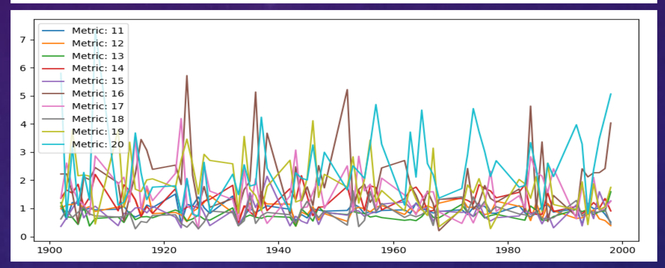


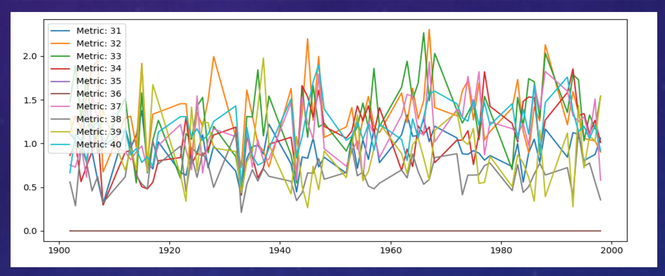
Broader Impact

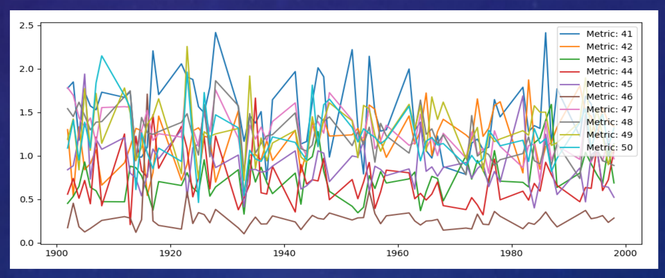
If successful, this research can provide insight as to how American communication has changed over time and model how it may continue to change over time as well.

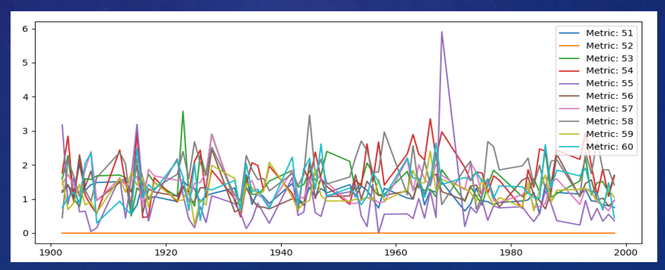


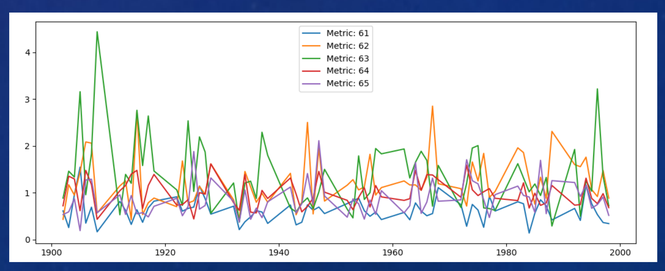












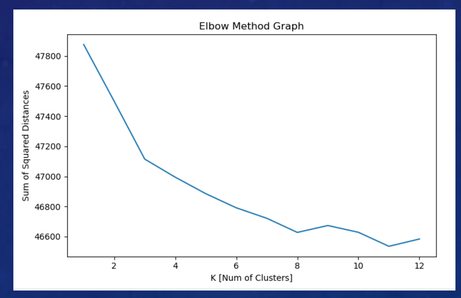
Clustering

To accomplish prediction, data clustering is used. Clustering uses several data points, each representing a different text source. In the instance of this research the clusters are 66 dimensional points and grouped in close proximity with an association to a single cluster group. This process is highly iterative and continuously changes the location of the mother cluster points until the distance to all of its associated points is nominal. Essentially, clustering groups points with similar metrics and is used to represent a different time-span once the iterative process is completed.

```
def cluster(cluster_data, k, max_iter):
    distance = 0
    difference = 0
    lowest = 99999
    clusters = []
    various_return_variables = []
    average_points = []

    #print("K " + str(k) + " clusters")


    for i in range(k):                                  # Make a list containing k arrays to act as clusters
        clusters.append([])
        average_points.append([])  # Add array for each

    for i in range(k):
        #print("First cluster iteration")
        for j in range(len(cluster_data[1])):
            clusters[i].append(random.randint(0,max_iter))  # Assign random values between the limits for every cluster-point
            average_points[i].append(-1)                    # Assign array of -1's of len(cluster_data) to know which average points are reassigned later on
        #print(clusters[i])                                  # Used to debug
    close_enough = 0

    while(not close_enough):                            # re-iterate through the while loop until the clusters don't change distance by more than 0.05
        # average_points = []                           # take out?

        for i in range(len(cluster_data)):
            lowest = 99999
            for j in range(len(clusters)):
                #print(len(cluster_data))
                distance = dis(cluster_data[i][0],clusters[j])          # define the distance between every point and every cluster
                #  print(distance)

                if distance < lowest:                   # Find which distance is the lowest and assign that cluster number to a variable in cluster data
                    lowest = distance
                    #print(lowest)
                    cluster_data[i][2] = j

        if (not -1 in average_points[cluster_data[i][2]]):
            average_points[cluster_data[i][2]] = point_ave(average_points[cluster_data[i][2]], cluster_data[i][0])  # Find the average points between all the points assigned to a specific cluster
        else:
            average_points[cluster_data[i][2]] = cluster_data[i][0]                                                 # If average points is still full of -1's assign given point to average points variable

        difference = 0
        for i in range(len(clusters)):
            if len(clusters[i]) == len(average_points[i]) and average_points[i][0] > 0:
                #print("this works")
                #print(clusters[i])
                difference += dis(clusters[i], average_points[i])       # Add distance between each point and it's assigned cluster,, add that to distance variable
                clusters[i] = average_points[i]                         # Assign each average to be the new cluster center,,, if lengths of arrays are the same and average p
            #else:
                #print("Pls' dog, are points are still aren't changing, It's preferring only a few clusters")  # Used for debugging, alert the user if a cluster is not used
                #print(average_points[i])
                #print(clusters[i])
        difference = difference / len(clusters)                         # Determine the amount of distance changed for each cluster

        if (difference < 0.05):                                         # If that distance is low enough, break out of the loop
            close_enough = 1

    various_return_variables.append(cluster_data)                       # Add cluster data to return variable
    various_return_variables.append(clusters)                           # Add clusters to return variable

    #print("Cluster Method Completed...")
    return various_return_variables
```

Elbow Method

To determine how many clusters, and therefore time-spans, the elbow method was used. The elbow method performs data clustering for a number of clusters and determines the sum of squared distances for each cluster and its respective points. When these distances are graphed opposed to the number of clusters used, a trend will arise where the distance decreases from a near vertical slope and levels out after a number of clusters, forming what looks to be an elbow.



Results

Due to the limited corpora used, prediction beyond precision of an indiscriminate projection has not yet been achieved in this research project.

Special Acknowledgments

Dr. Claude Willan

Director of the UH Digital Research Commons

UNIVERSITY of HOUSTON