Application 3: Closest Pairs and Clustering
Algorithmic Thinking (Part2)

Shamsuddin Rehmani

July 15, 2019

Application 3 Description

In Project 3, you implemented two methods for clustering sets of data. In this Application, we will analyze the performance of these two methods on various subsets of our county-level cancer risk data set. In particular, we will compare these two clustering methods in three areas:

- Efficiency - Which method computes clusterings more efficiently?
- Automation - Which method requires less human supervision to generate reasonable clusterings?
- Quality - Which method generates clusterings with less error?

Answer to Question 1

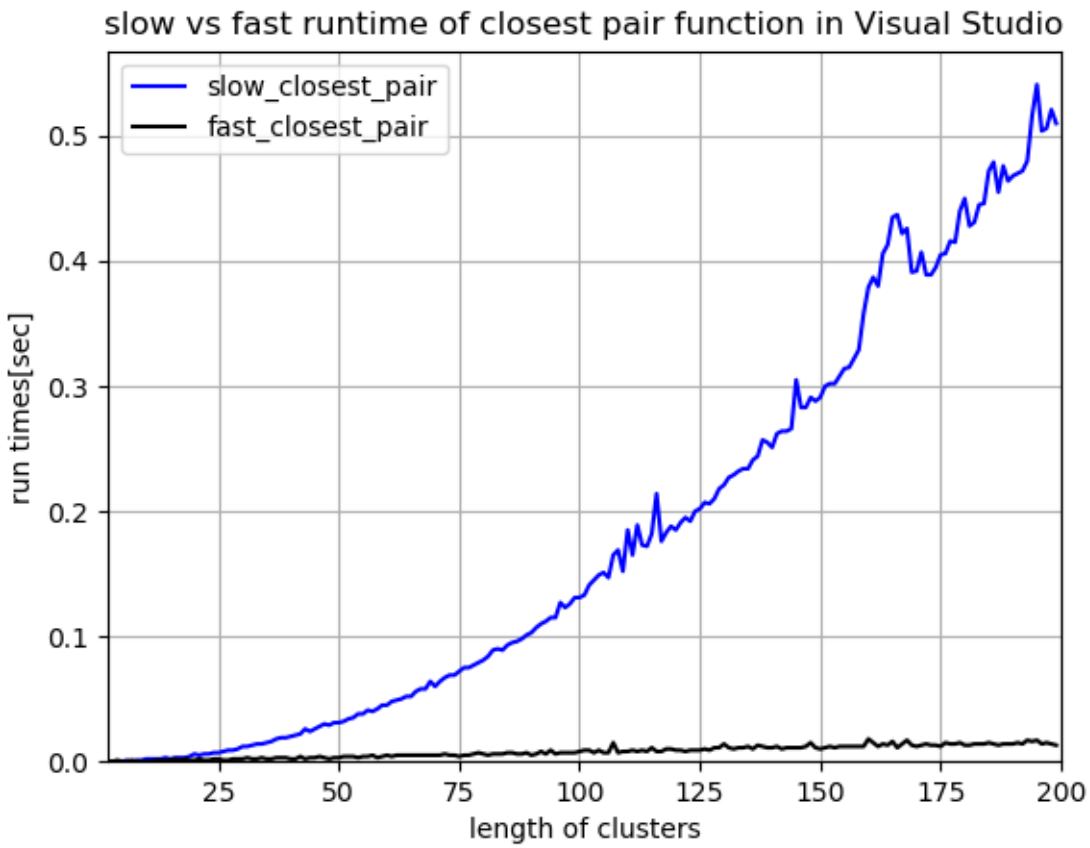

Figure 1: Fast vs slow closest pair function runtime comparison

Answer to Question 2

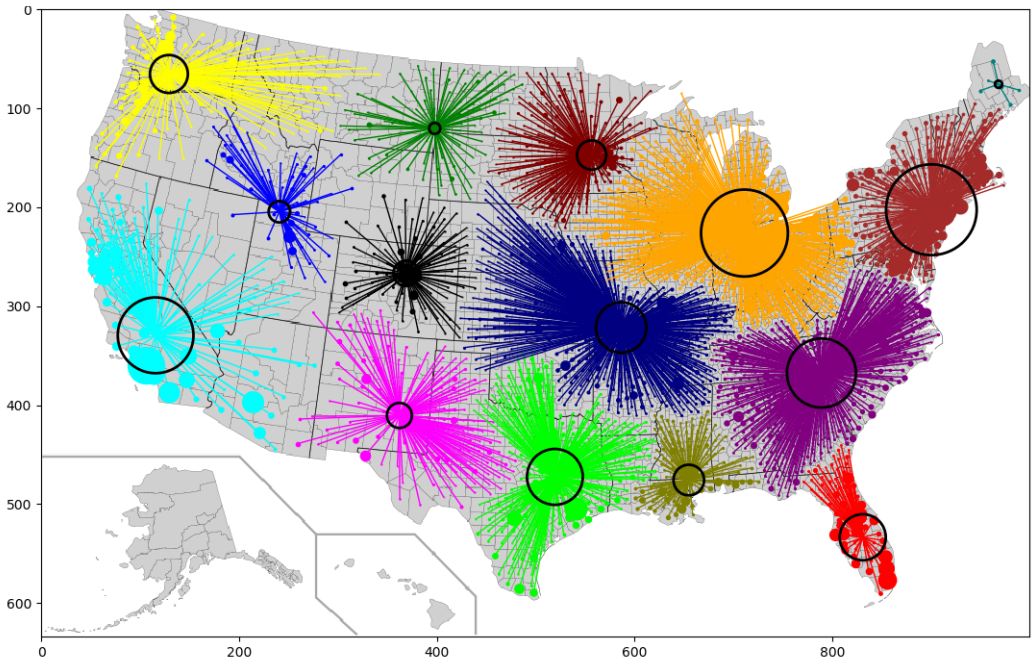

Figure 2: 15 clusters generated by hierarchical clustering for 3108 counties

Answer to Question 3

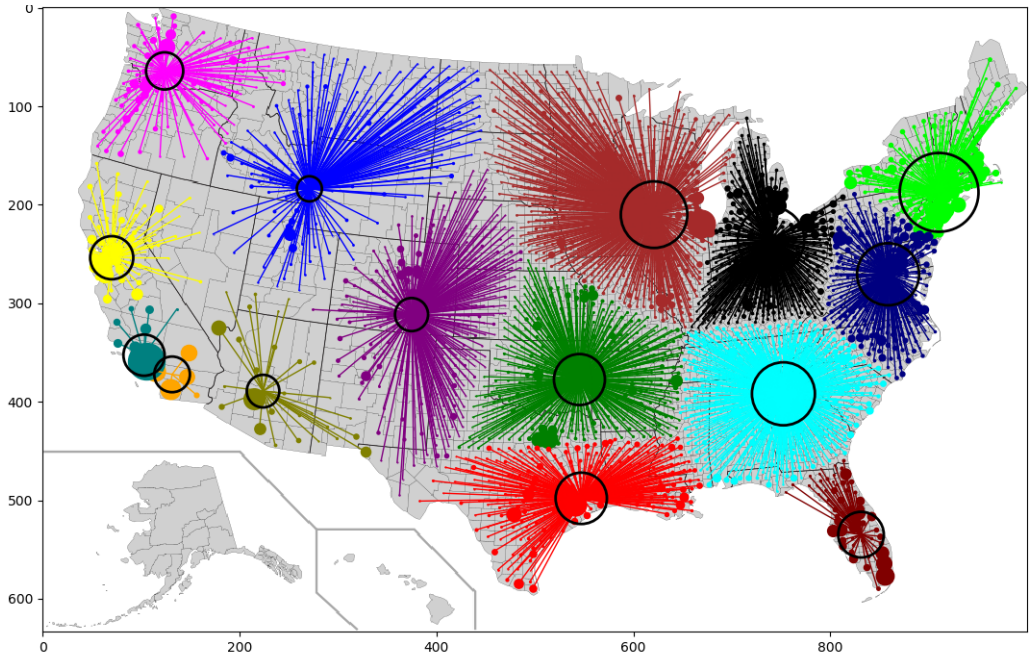

Figure 3: 15 clusters with 5 iterations generated by k-mean clustering for 3108 counties

Answer to Question 4

let k = number of clusters

n = size of the input cluster list

q = number of iterations (for k-mean clustering)

Then for hierarchical clustering, the time complexity is $O((n-k)(nlog(n)+nlog(n)^2))$ which is $O(n^2(logn)^2)$ if k is small compared to n as stated in the above question

On the other hand, the time complexity of k-mean clustering is $O(qnk)$. Since k is small compared to n and q is also a small fixed number, time complexity is $O(n)$ which is much more efficient than heirarchiacl clustering

Answer to Question 5

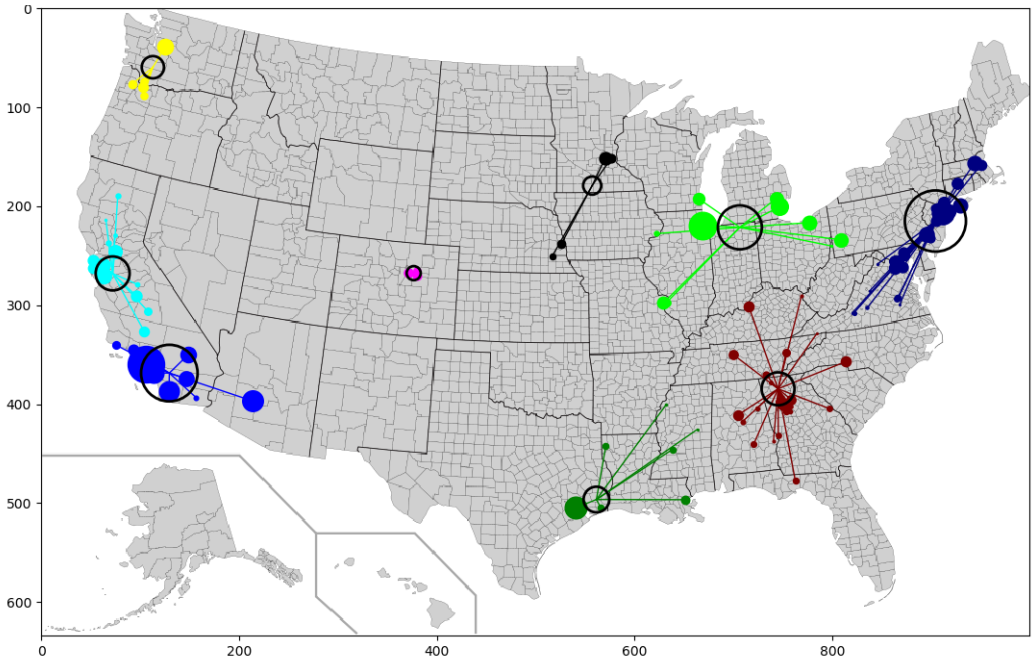

Figure 4: 9 clusters generated by hierarchical clustering for 111 counties

Answer to Question 6

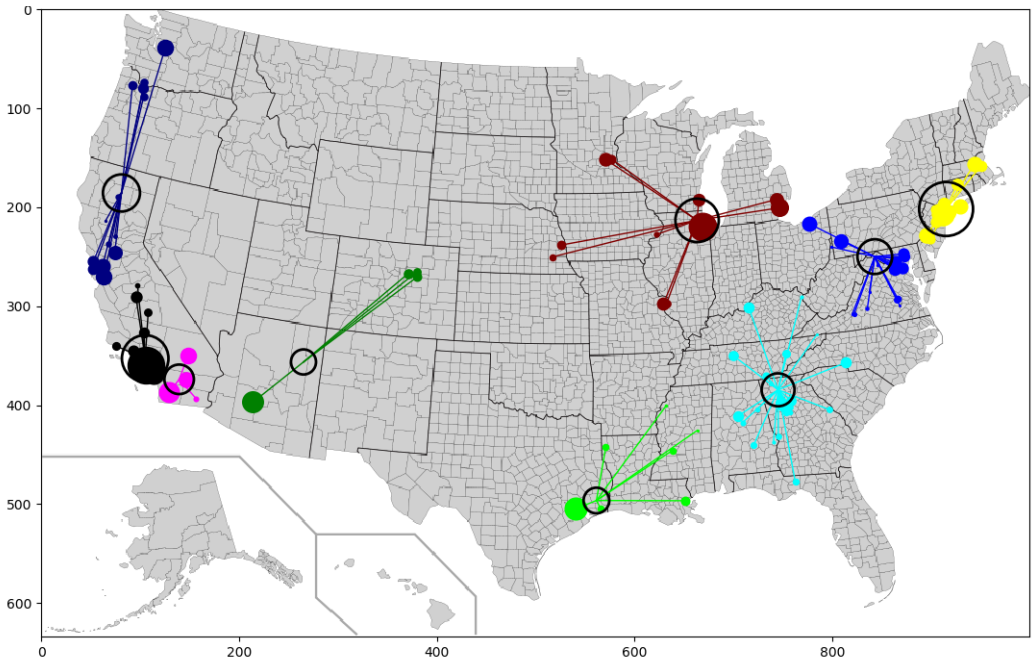

Figure 5: 9 clusters with 5 iterations generated by k-mean clustering for 111 counties

Answer to Question 7

Distortion of hierarchical clustering for 111 counties: 1.75×10^{11}

Distortion of k-mean clustering for 111 counties: 2.71×10^{11}

Answer to Question 8

For the k-mean clustering(Figure 5) we can see that the 3 cluster centers are located in the california region (one in northern while two in southern region) while of the 3 centers for the hierarchical clustering one is located in Washington state and two in the california region. The high distortion for the k-mean is due to the fact that the counties in the cluster with center in the northern California is distrbuted in Washington and southern California i.e the counties are much further than the center. The difference between the distortion values is becauses the intial clustering method for k-mean involves clustering around counties with highest population. Due to this all 3 counties(3 largest circles are black and pink) in the southern California were included in the intial clustering while none from Washington, Oregon or Northern California was selected. Thus this resulted in relatively higher distortion.

Answer to Question 9

Based on Question 8, we say that hierarchical clustering requires less human supervision as it requires only choosing the number of ouput clusters. While on the other hand, for k-mean we need a good choice of the intial cluster centers.

Answer to Question 10

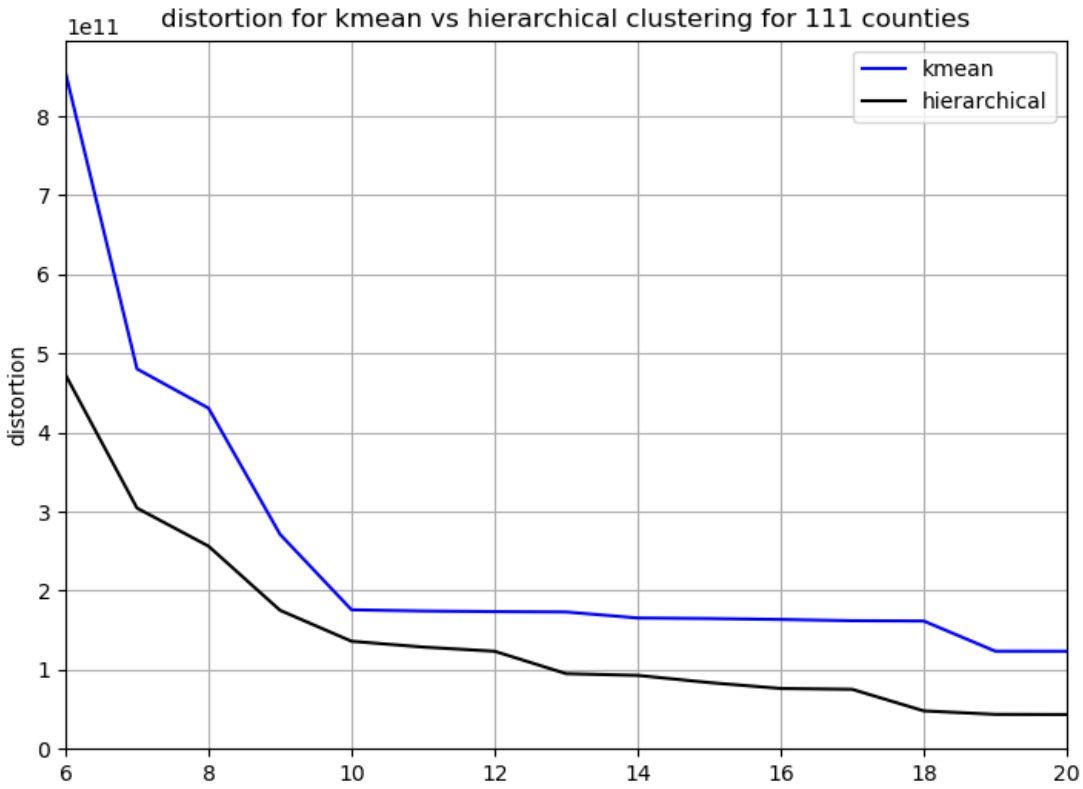

Figure 6: 9 clusters with 5 iterations generated by k-mean clustering for 111 counties

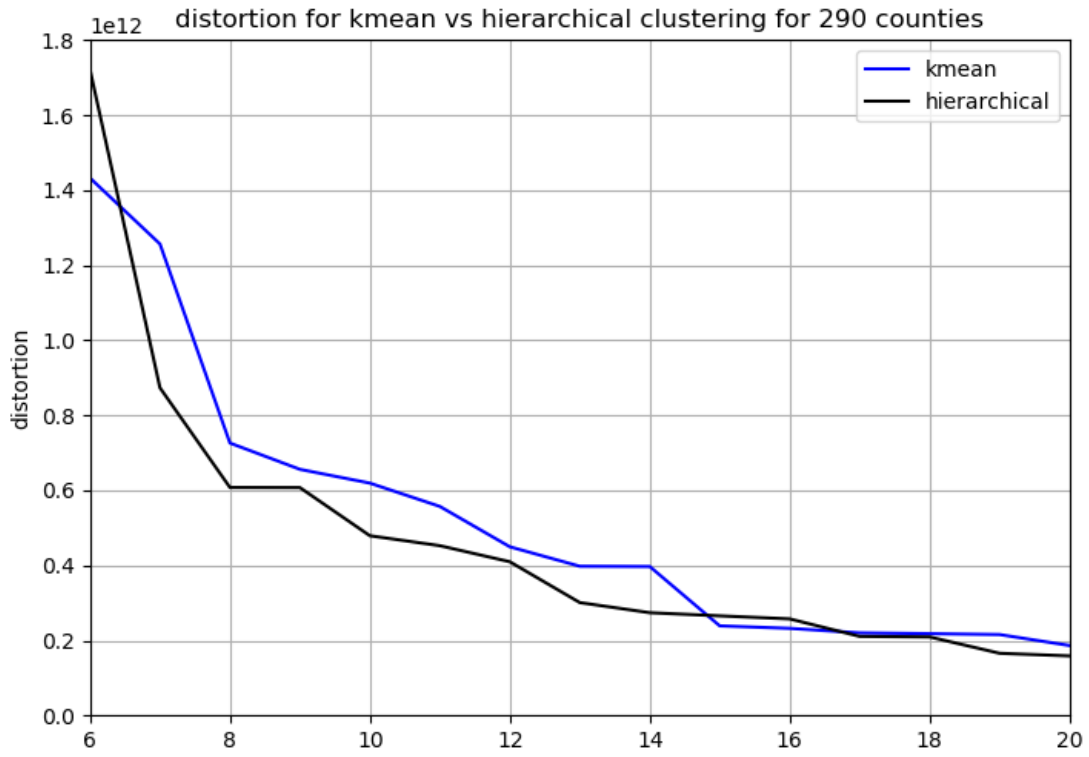

Figure 7: 9 clusters with 5 iterations generated by k-mean clustering for 111 counties

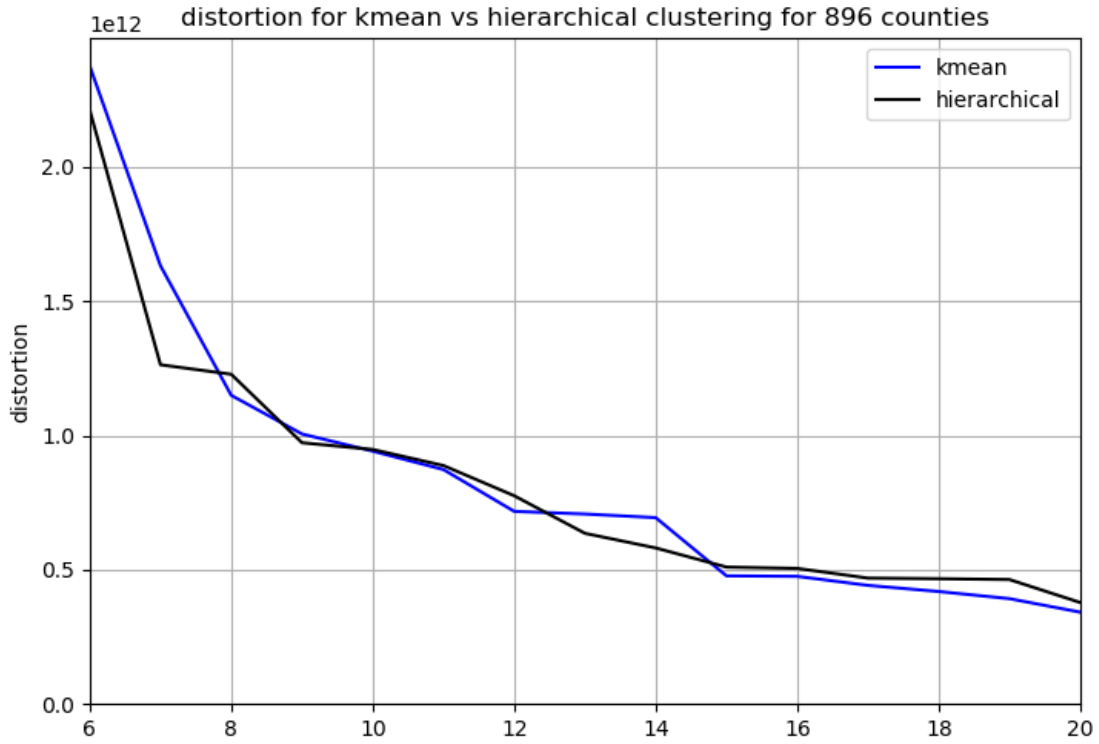

Figure 8: 9 clusters with 5 iterations generated by k-mean clustering for 111 counties

A Python code used to answer the Application Questions

```
"""
Contains the answers to all the questions for
Application #3 - Comparision of Clustering Algorithm
"""

import random
import copy
import matplotlib.pyplot as plt
import time
import alg_cluster
import alg_clusters_matplotlib as clust_plt
import alg_project3_solution as pj3_sol
import alg_project3_viz as pj3_viz

##############################################################
##### Code for Q1 Solution #####
# Q1: Write a function gen_random_clusters(num_clusters) that creates
# a list of clusters where each cluster in this list corresponds to one
# randomly generated point in the square with corners (+/-1,+/-1). Use
# this function and your favorite Python timing code to compute the
# running times of the functions slow_closest_pair and fast_closest_pair
# for lists of clusters of size 2 to 200. Once you have computed the running
# times for both functions, plot the result as two curves combined in a single
# plot. (Use a line plot for each curve.) The horizontal axis for your plot
# should be the the number of initial clusters while the vertical axis should
# be the running time of the function in seconds. Please include a legend in
# your plot that distinguishes the two curves.

def gen_random_clusters(num_clusters):
    """[summary]

    Arguments:
        num_clusters {[type]} -- [description]

    Returns:
        [type] -- [description]
    """

    # initialize the cluster list to be returned
    cluster_lst = num_clusters * [0]

    for idx in range(num_clusters):
        # generate a random point between -1 and 1
        horz_center = random.uniform(-1.0, 1.0)
        vert_center = random.uniform(-1.0, 1.0)
        # create the cluster and add it to the list
        cluster_lst[idx] = alg_cluster.Cluster(set(), horz_center, vert_center, 1, 0)

    return cluster_lst

# intialize the range of cluster size to be used
clust_lens = range(2, 200)
# intialize the fast and slow pair function times
time_slow_closest = []
time_fast_closest = []

# clsts = gen_random_clusters(4)
# print len(clsts)
# print pj3_sol.slow_closest_pair(clsts)

# calcualte the time to run the slow and fast functions for the closest pair
for clust_len in clust_lens:

    # generate the cluster list of size clust_lsn
    cluster_list = gen_random_clusters(clust_len)

    # calculate the closest pair in the cluster using slow algorithm and
    # store the time it takes to run the function for given cluster size
    start = time.time()
    pj3_sol.slow_closest_pair(cluster_list)
    end = time.time()
    time_slow_closest.append((end - start))
    # calculate the closest pair in the cluster using fast algorithm and
    # store the time it takes to run the function for given cluster size
    start = time.time()
    pj3_sol.fast_closest_pair(cluster_list)
    end = time.time()
    time_fast_closest.append((end - start))

# plot the graphs of resilience vs number of nodes removed for each of the 3 graphs
#plt.figure(1)
plt.plot(clust_lens, time_slow_closest, '-b', label = 'slow_closest_pair')
plt.plot(clust_lens, time_fast_closest, '-k', label = 'fast_closest_pair')
plt.title('slow vs fast runtime of closest pair function in Visual Studio' )
plt.xlabel('length of clusters')
plt.ylabel('run times[sec]')
plt.legend(loc = 'upper left')
plt.xlim(2, 200)
plt.ylim(0, None)
plt.grid()
plt.show()
# uncommet to save the plot
#plt.savefig("Q1_closest_pair_comparision.png")

```

```
95
96  ######################################################
97  # ##### Code for Q2 Solution #####
98  # Use alg_project3_viz to create an image of the 15 clusters generated by applying
99  # hierarchical clustering to the 3108 county cancer risk data set. You may submit
100 # an image with the 3108 counties colored by clusters or an enhanced visualization
101 # with the original counties colored by cluster and linked to the center of their
102 # corresponding clusters by lines. You can generate such an enhanced plot using our
103 # alg_clusters_matplotlib code by modifying the last parameter of plot_clusters to be
104 # True. Note that plotting only the resulting cluster centers is not acceptable
105
106 # load the data
107 data_table = pj3_viz.load_data_table(pj3_viz.DATA_3108_URL)
108
109 # generate cluster from the data
110 singleton_list = []
111 for line in data_table:
112     singleton_list.append(alg_cluster.Cluster(set([line[0]]), line[1], line[2], line[3], line[4]))
113
114 # create a deep copy since hierarchical_clustering modifies the references to the list
115 singleton_list_cpy = copy.deepcopy(singleton_list)
116
117 # form clusters based on hierarchical clustering algorithm
118 cluster_list = pj3_sol.hierarchical_clustering(singleton_list_cpy, 15)
119 print "Displaying", len(cluster_list), "hierarchical clusters"
120
121 # generate the image for the 15 clusters formed using hierarchical clustering algorithm
122 clust_plt.plot_clusters(data_table, cluster_list, True)
123
124 ######################################################
125 ##### Code for Q3 Solution #####
126 # Use alg_project3_viz to create an image of the 15 clusters generated by applying 5
127 # iterations of k-means clustering to the 3108 county cancer risk data set. You may
128 # submit an image with the 3108 counties colored by clusters or an enhanced visualization
129 # with the original counties colored by cluster and linked to the center of their corresponding
130 # clusters by lines. As in Project 3, the initial clusters should correspond to the 15 counties
131 # with the largest populations.
132
133 # form clusters based on k-mean clustering algorithm
134 cluster_list = pj3_sol.kmeans_clustering(singleton_list, 15, 5)
135 print "Displaying", len(cluster_list), "k-mean clusters"
136
137 clust_plt.plot_clusters(data_table, cluster_list, True)
138
139 ######################################################
140 ##### Q4 Solution #####
141 # Which clustering method is faster when the number of output clusters is either a small
142 # fixed number or a small fraction of the number of input clusters? Provide a short
143 # explanation in terms of the asymptotic running times of both methods. You should assume
144 # that hierarchical_clustering uses fast_closest_pair and that k-means clustering always
145 # uses a small fixed number of iterations.
146 #
147 # Ans: let k = number of clusters
148 #          n = size of the input cluster_list
149 #          q = number of iterations (for k-mean clustering)
150
151 # Then for hierarchical clustering, the time complexity is O((n - k) * (n * logn + n * (logn)^2)
152 # which is ~ O(n^2 * (logn)^2) if k is small compared to n as stated in the above question
153
154 # On the other hand, the time complexity of k-mean clustering is ~ O(q * n * k). Since k is small
155 # compared to n and q is also a small fixed number, time complexity is O(n) which is much more
156 # efficient than heirarchiacl clustering
157
158 ######################################################
159 ##### Q5 Solution #####
160 # Use alg_project3_viz to create an image of the 9 clusters generated by applying hierarchical
161 # clustering to the 111 county cancer risk data set. You may submit an image with the 111
162 # counties colored by clusters or an enhanced visualization with the original counties colored
163 # by cluster and linked to the center of their corresponding clusters by lines.
164
165 # load the data
166 data_table = pj3_viz.load_data_table(pj3_viz.DATA_111_URL)
167
168 # generate cluster from the data
169 singleton_list = []
170 for line in data_table:
171     singleton_list.append(alg_cluster.Cluster(set([line[0]]), line[1], line[2], line[3], line[4]))
172
173 # create a deep copy since hierarchical_clustering modifies the references to the list
174 singleton_list_cpy = copy.deepcopy(singleton_list)
175 # form clusters based on hierarchical clustering algorithm
176 cluster_list_hierarc = pj3_sol.hierarchical_clustering(singleton_list_cpy, 9)
177
178 # generate the image for the 9 clusters formed using hierarchical clustering algorithm
179 print "Displaying", len(cluster_list_hierarc), "hierarchical clusters"
180 clust_plt.plot_clusters(data_table, cluster_list_hierarc, True)
181
182 ######################################################
183 ##### Q6 Solution #####
184 # Use alg_project3_viz to create an image of the 9 clusters generated by applying 5
185 # iterations of k-means clustering to the 111 county cancer risk data set. You may
186 # submit an image with the 111 counties colored by clusters or an enhanced visualization
187 # with the original counties colored by cluster and linked to the center of their
188 # corresponding clusters by lines. As in Project 3, the initial clusters should correspond
189 # to the 9 counties with the largest populations.
190
```

```
# form clusters based on k-mean clustering algorithm
cluster_list_kmean = pj3_sol.kmeans_clustering(singleton_list, 9, 5)

# generate the image for the 9 clusters formed using k-means clustering algorithm
print "Displaying", len(cluster_list_hierarc), "k-mean clusters"
clust_plt.plot_clusters(data_table, cluster_list_kmean, True)

######################################################
##### Q7 Solution #####
# Write a function compute_distortion(cluster_list) that takes a list of clusters and
# uses cluster_error to compute its distortion. Now, use compute_distortion to compute
# the distortions of the two clusterings in questions 5 and 6

def compute_distortion(cluster_list, data_table):

    distortion = sum([cluster.cluster_error(data_table) for cluster in cluster_list])

    return distortion

print compute_distortion(cluster_list_hierarc, data_table)
print compute_distortion(cluster_list_kmean, data_table)

######################################################
##### Q8 Solution #####
# Examine the clusterings generated in Questions 5 and 6. In particular, focus your
# attention on the number and shape of the clusters located on the west coast of the USA.
# Describe the difference between the shapes of the clusters produced by these two methods
# on the west coast of the USA. What caused one method to produce a clustering with a much
# higher distortion? To help you answer this question, you should consider how k-means
# clustering generates its initial clustering in this case.
#
# Ans: For the k-mean clustering(Figure 5) we can see that the 3 cluster centers are located
# in the california region (one in northern while two in southern region) while of the 3
# centers for the hierarchical clustering one is located in Washington state  and two in the
# california region. The high distortion for the k-mean is due to the fact that the counties
# in the cluster with center in the northern California is distrbuted in Washington and
# southern California i.e the counties are much further than the center. The difference between
# the distortion values is becauses the intial clustering method for k-mean involves clustering
# around counties with highest population. Due to this all 3 counties(3 largest circles are black
# and pink) in the southern California were included in the intial clustering while none from
# Washington, Oregon or Northern California was selected. Thus this resulted in relatively
# higher distortion.

######################################################
##### Q9 Solution #####
# Based on your answer to Question 8, which method (hierarchical clustering or k-means
# clustering) requires less human supervision to produce clusterings with relatively
# low distortion?
#
# Ans: based on Q8, we say that hierarchical clustering requires less human supervision
# as it requires only choosing the number of ouput clusters. While on the other hand
# for k-mean we need a good choice of the intial cluster centers.

######################################################
##### Q10 Solution #####

hierarc_distortion = 15 * [0]
kmean_distortion = 15 * [0]
num_clusters_range = range(20, 5, -1)

## do the calculations for 111 data set
# load the data
data_table = pj3_viz.load_data_table(pj3_viz.DATA_111_URL)

# generate cluster from the data
singleton_list = []
for line in data_table:
    singleton_list.append(alg_cluster.Cluster(set([line[0]]), line[1], line[2], line[3], line[4]))

singleton_list_cpy = copy.deepcopy(singleton_list)

for num_clusters in num_clusters_range:
    cluster_list_hierarc = pj3_sol.hierarchical_clustering(singleton_list_cpy, num_clusters)
    cluster_list_kmean = pj3_sol.kmeans_clustering(singleton_list, num_clusters, 5)
    hierarc_distortion[num_clusters - 6] = compute_distortion(cluster_list_hierarc, data_table)
    kmean_distortion[num_clusters - 6] = compute_distortion(cluster_list_kmean, data_table)

# reverse the num_cluster_range
num_clusters_range.reverse()

# plot the graphs for distortion for two clustering method for 111 counties
#plt.figure(2)
plt.plot(num_clusters_range, kmean_distortion, '-b', label = 'kmean')
plt.plot(num_clusters_range, hierarc_distortion, '-k', label = 'hierarchical')
plt.title('distortion for kmean vs hierarchical clustering for 111 counties' )
plt.xlabel('number of clusters')
plt.ylabel('distortion')
plt.legend(loc = 'upper right')
plt.xlim(6, 20)
plt.ylim(0, None)
plt.grid()
plt.show()

## do the calculations for 290 data set
# load the data
data_table = pj3_viz.load_data_table(pj3_viz.DATA_290_URL)
```

```
287
288 # generate cluster from the data
289 singleton_list = []
290 for line in data_table:
291     singleton_list.append(alg_cluster.Cluster(set([line[0]]), line[1], line[2], line[3], line[4]))
292
293 num_clusters_range = range(20, 5, -1)
294 singleton_list_cpy = copy.deepcopy(singleton_list)
295
296 for num_clusters in num_clusters_range:
297     cluster_list_hierarc = pj3_sol.hierarchical_clustering(singleton_list_cpy, num_clusters)
298     cluster_list_kmean = pj3_sol.kmeans_clustering(singleton_list, num_clusters, 5)
299     hierarc_distortion[num_clusters - 6] = compute_distortion(cluster_list_hierarc, data_table)
300     kmean_distortion[num_clusters - 6] = compute_distortion(cluster_list_kmean, data_table)
301
302 # reverse the num_cluster_range
303 num_clusters_range.reverse()
304
305 # plot the graphs for distortion for two clustering method for 290 counties
306 #plt.figure(3)
307 plt.plot(num_clusters_range, kmean_distortion, '-b', label = 'kmean')
308 plt.plot(num_clusters_range, hierarc_distortion, '-k', label = 'hierarchical')
309 plt.title('distortion for kmean vs hierarchical clustering for 290 counties' )
310 plt.xlabel('number of clusters')
311 plt.ylabel('distortion')
312 plt.legend(loc = 'upper right')
313 plt.xlim(6, 20)
314 plt.ylim(0, None)
315 plt.grid()
316 plt.show()
317
318 ## do the calculations for 896 data set
319 # load the data
320 data_table = pj3_viz.load_data_table(pj3_viz.DATA_896_URL)
321
322 # generate cluster from the data
323 singleton_list = []
324 for line in data_table:
325     singleton_list.append(alg_cluster.Cluster(set([line[0]]), line[1], line[2], line[3], line[4]))
326
327 num_clusters_range = range(20, 5, -1)
328 singleton_list_cpy = copy.deepcopy(singleton_list)
329
330 for num_clusters in num_clusters_range:
331     cluster_list_hierarc = pj3_sol.hierarchical_clustering(singleton_list_cpy, num_clusters)
332     cluster_list_kmean = pj3_sol.kmeans_clustering(singleton_list, num_clusters, 5)
333     hierarc_distortion[num_clusters - 6] = compute_distortion(cluster_list_hierarc, data_table)
334     kmean_distortion[num_clusters - 6] = compute_distortion(cluster_list_kmean, data_table)
335
336 # reverse the num_cluster_range
337 num_clusters_range.reverse()
338
339 # plot the graphs for distortion for two clustering method for 290 counties
340 #plt.figure(3)
341 plt.plot(num_clusters_range, kmean_distortion, '-b', label = 'kmean')
342 plt.plot(num_clusters_range, hierarc_distortion, '-k', label = 'hierarchical')
343 plt.title('distortion for kmean vs hierarchical clustering for 896 counties' )
344 plt.xlabel('number of clusters')
345 plt.ylabel('distortion')
346 plt.legend(loc = 'upper right')
347 plt.xlim(6, 20)
348 plt.ylim(0, None)
349 plt.grid()
350 plt.show()
351
352 ######################################################
353 ##### Q11 Solution #####
354 # For each data set (111, 290, and 896 counties), does one clustering method consistently
355 # produce lower distortion clusterings when the number of output clusters is in the range
356 # 6 to 20? Is so, indicate on which data set(s) one method is superior to the other.
357 #
358 # Ans: for 111 counties, hierarchical clustering consistently produces less clusteriing
359 # For other two data set, there is no one method that consistently produces lower distorion
```

B All functions for project 4 used in the application

```
1  """
2  All the algorithms for Project 3 for closest
3  pair and clustering
4  """
5
6  import alg_cluster
7
8  ######################################################
9  # Code for closest pairs of clusters
10 def slow_closest_pair(cluster_list):
11     """
12     Uses the brute-force algorithm (O(n^2)) to find the closest pair of clusters in a list
13
14     Arguments:
15         cluster_list {Cluster} -- a list of Cluster objects
16
17     Returns:
18         tuple -- returns a closest pair where the pair is represented by the tuple
```

```
                (dist, idx1, idx2) with idx1 < idx2 where dist is the distance between
                the closest pair cluster_list[idx1] and cluster_list[idx2].
    """
    # initialize the tuple that will store the closest pair of cluster distance and index
    (dist, idx1, idx2) = (float("inf"), -1, -1)

    for clusteri_idx in range(len(cluster_list)):
        for clusterj_idx in range(len(cluster_list)):
            if (clusteri_idx != clusterj_idx):
                curr_dist = cluster_list[clusteri_idx].distance(cluster_list[clusterj_idx])
                (dist, idx1, idx2) = (min(set([(dist, idx1, idx2), (curr_dist, clusteri_idx, clusterj_idx)]),
                    key = lambda tup: tup[0]))

    if (idx2 > idx1):
        return (dist, idx1, idx2)
    else:
        return (dist, idx2, idx1)

def fast_closest_pair(cluster_list):
    """
    Compute the distance between the closest pair of clusters in a list using fast algorithm
    [O(n*(logn)^2]

    Arguments:
        cluster_list {list} -- list of clusters sorted based on the horizontal distance of
                               their centers in ascending order

    Returns:
        tuple -- tuple of the form (dist, idx1, idx2) where the centers of the clusters
                 cluster_list[idx1] and cluster_list[idx2] have minimum distance dist.
    """

    # base case
    if len(cluster_list) <= 3:
        (dist, idx1, idx2) = slow_closest_pair(cluster_list)

    # inductive case
    else:
        # divide the problem in half, solve it and then merge the results from both
        idx_m = len(cluster_list) / 2
        # solve for the left half of the cluster list
        (d_left, idxi_l, idxj_l) = fast_closest_pair(cluster_list[ : idx_m])
        # solve for right half of the cluster list
        (d_right, idxi_r, idxj_r) = fast_closest_pair(cluster_list[idx_m : ])
        # find the minimum of the left and right paritiion minimum distances
        (dist, idx1, idx2) = (min(set([(d_left, idxi_l, idxj_l), (d_right, idxi_r + idx_m, idxj_r + idx_m)]),
            key = lambda tup: tup[0]))
        # find the the horizontal position of the strip's vertical center line i.e the midpoint of the
        # horizontal position of the last element of the left perition and first element of the right parition
        horiz_center = 0.5 * (cluster_list[idx_m - 1].horiz_center() + cluster_list[idx_m].horiz_center())
        # find the minimum of the minimum distance found earlier and the closest pair in the strip
        (dist, idx1, idx2) = (min(set([(dist, idx1, idx2), closest_pair_strip(cluster_list, horiz_center, dist)
            ]),
            key = lambda tup: tup[0]))

    return (dist, idx1, idx2)

def closest_pair_strip(cluster_list, horiz_center, half_width) :
    """
    Helper function to compute the closest pair of clusters in a vertical strip

    Arguments:
        cluster_list {list} -- a list of Cluster objects
        horiz_center {integer} -- the horizontal position of the strip's vertical center line
        half_width {integer} -- the half the width of the strip (i.e; the maximum horizontal distance
                                that a cluster can lie from the center line)

    Returns:
        tuple -- returns a tuple of the form (dist, idx1, idx2) where the centers of the clusters
                 cluster_list[idx1] and cluster_list[idx2] lie in the strip and have minimum distance dist.
    """

    # store all cluster whose centres are within the vertical strip specified by horiz_center and half_width
    strip_clust = [(cluster_list[clst_idx], clst_idx) for clst_idx in range(len(cluster_list))
        if abs(cluster_list[clst_idx].horiz_center() - horiz_center) < half_width ]

    # sort the cluster based on their vertical distances
    strip_clust.sort(key = lambda cluster: cluster[0].vert_center())

    # intialize the minimium ditance and their indices
    (dist, idx1, idx2) = (float("inf"), -1, -1)

    # for each cluster inspect the next 3 ones and record the pair of cluster indices that
    # corresponds to closest pair thus found
    for clsti_idx in range(len(strip_clust) - 1):
        for clstj_idx in range(clsti_idx + 1, min((clsti_idx + 4), len(strip_clust))):
            curr_dist = strip_clust[clsti_idx][0].distance(strip_clust[clstj_idx][0])
            (dist, idx1, idx2) = (min(set([(dist, idx1, idx2),
                (curr_dist, strip_clust[clsti_idx][1], strip_clust[clstj_idx][1])]), key = lambda tup: tup[0]))

    # return minimum distance and their indicies (dist, idx1, idx2) where idx1 < idx2
    if (idx2 > idx1):
        return (dist, idx1, idx2)
    else:
        return (dist, idx2, idx1)

```

```
##############################################################################
# Code for hierarchical clustering
def hierarchical_clustering(cluster_list, num_clusters):
    """
    Compute a hierarchical clustering of a set of clusters
    Note: the function mutates cluster_list to have length num_clusters

    Arguments:
        cluster_list {list} -- a list of Cluster objects
        num_clusters {integer} -- integer number of clusters to be made from cluster_list

    Returns:
        list -- a list of Cluster objects whose length is num_clusters
    """

    while len(cluster_list) > num_clusters:
        # sort the cluster based on their horizontal distances
        #print cluster_list
        cluster_list.sort(key = lambda cluster: cluster.horiz_center())
        #print cluster_list
        (dummy_dist, idx1, idx2) = fast_closest_pair(cluster_list)
        cluster_list[idx1].merge_clusters(cluster_list[idx2])
        #print cluster_list
        cluster_list.pop(idx2)

    return cluster_list

##############################################################################
# Code for k-means clustering

def kmeans_clustering(cluster_list, num_clusters, num_iterations):
    """
    Compute the k-means clustering of a set of clusters
    Note: the function does not mutate cluster_list to have length num_clusters

    Arguments:
        cluster_list {list} -- a list of Cluster objects
        num_clusters {integer} -- integer number of clusters to be made from cluster_list
        num_iterations {integer} -- number of iterations

    Returns:
        list -- a list of Cluster objects whose length is num_clusters
    """
    # compute an initial list of clusters with the property that each cluster consists of
    # a single county chosen from the set of the num_cluster counties with the largest populations
    cluster_list_cpy = list(cluster_list)
    cluster_list_cpy.sort(key = lambda cluster: cluster.total_population())
    cluster_len = len(cluster_list)
    old_cluster = cluster_list_cpy[cluster_len - num_clusters : ]

    for dummy_idx in range(num_iterations):
        # initialze an empty cluster
        new_cluster = [alg_cluster.Cluster(set(), 0.0, 0.0, 0, 0) for dummy_i in range(num_clusters)]
        for idx_i in range(cluster_len):
            min_dist = float("inf")
            min_idx = -1
            # find the cluster from old_cluster and its index that is closest to current cluster
            # in the cluster_list
            for idx_j in range(num_clusters):
                curr_dist = old_cluster[idx_j].distance(cluster_list[idx_i])
                if curr_dist < min_dist:
                    min_dist = curr_dist
                    min_idx = idx_j
            # add the cluster to the new_cluster at index = min_idx
            new_cluster[min_idx].merge_clusters(cluster_list[idx_i])

        old_cluster = new_cluster

    return new_cluster
```