Sean Morton
ME 395
Mechanistic Data Science
10/27/21

K-Means Clustering for Mechanical, Thermal, and Electrical Properties of Steels

Abstract

A dataset of 34 steels with 7 mechanical, 1 electrical, and 3 thermal properties will be analyzed using K-means clustering. K will be chosen for each dataset using goodness of variance fit $GVF = \frac{SDAM - SDCM}{SDAM}$. K-Means clustering will be carried out for each set of properties, and clusters will be analyzed both visually (clusters for 3D thermal properties data) and analytically (using tables of variables and clusters).

Introduction

Clustering is a data science technique with numerous applications. One application is for classification: if existing data of a certain label share similar values of variables, then a new data point can be labeled according to which cluster its variables are closest in value to. Another application of clustering is in dimensional reduction: if 900 samples are taken and they all lie within 10 distinct groups, data analysis can be carried out on 10 groups of data instead of 900 pieces of data.

One algorithm of clustering is called “K-Means Clustering”, in which the number of clusters k is defined by the user and the algorithm finds that number of k clusters that best groups the data. K-means clustering is iterative: (1) random cluster means are assigned, (2) data points are assigned to each cluster that they are closest to, (3) the new mean of the cluster is calculated. This process repeats until the mean of the cluster converges on an unchanging set of values for the variables in the n-dimensional dataset.

Two questions to be answered: What is the right value of k to choose for clustering? And what do the clusters look like (both in table form and in graph form) after the data has passed through K-Means clustering? In this procedure, values of k will be chosen to analyze the mechanical, thermal, and electrical properties of steel, using which K-Means clustering will be carried out.

Methods

Data was normalized in order to prepare the data of different scales for analysis. Each category of mechanical, thermal, and electrical properties was rescaled with a mean of 0 and a standard deviation of 1 for the set of all metal samples.

I. Selection of K

The normalized data was converted into a Numpy array and fed into sklearn.KMeans() to create an instance of the sklearn K-means clustering class. Using the class means and sample classifications that the KMeans() function produced, a SDAM and SDCM function were created to calculate the sum of deviations from array means and class means of each labeled datapoint. This was used to calculate the goodness of variance fit, $GVF = \frac{SDAM-SDCM}{SDAM}$, of the model.

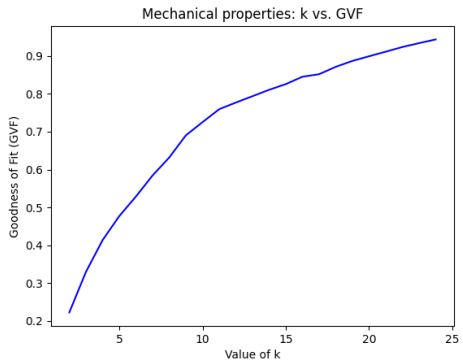

Fig. 1a. The relationship between k and GVF for the mechanical properties dataset.

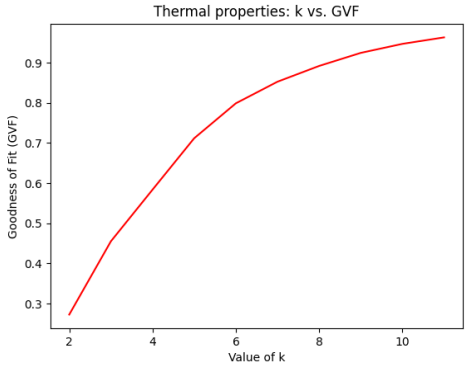

Fig. 1b. The relationship between k and GVF for the thermal properties dataset.

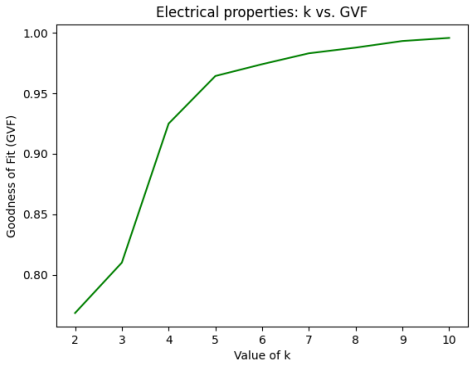

Fig. 1c. The relationship between k and GVF for the electrical properties dataset.

Values of k were selected by demonstrating the relationship between the number of clusters, k, and the GVF of the resulting k-means clustering model produced for the multi-dimensional data. The larger the GVF of a model, the smaller the differences between the elements in a cluster and the mean of the cluster: GVF = 0.9 means a better fit, and GVF = 0.1 means a worse fit. Higher k is not necessarily better: for example, given there were 34 steel samples a set of 34 clusters could be made such that GVF = 1, but these clusters do not give new information. The optimal value of k was chosen as the “elbow” of the graph of k vs. GVF: the approximate location where $\frac{d(GVF)}{dk}$ sharply decreases. For each set of properties: ideal values of k were

$k_{mech} = 11$
$k_{therm} = 6$
$k_{elec} = 5$

II. K-Means Clustering

Once the ideal values of k were determined from the graphs of k vs. GVF, three rounds of K-means clustering were performed on the steels according to their mechanical, thermal, and electrical properties. This was performed on the normalized data so as to prevent the different properties from disproportionately affecting the calculations of distances from cluster means.

Once the clusters were found for the normalized data, the data was then “de-normalized” to return the data to their original scales. The de-normalized data was then exported along with the class labels to CSV files, ‘mech’ / ‘therm’ / ‘elec ‘ + ‘_array_labeled.csv’ to show which labels correspond to which groups of steels.

III. Plotting Thermal Clusters

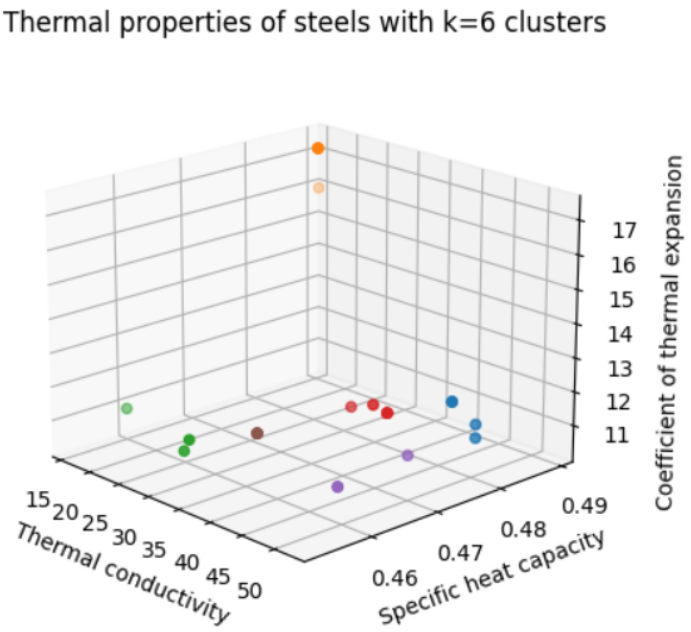

Fig. 2. A graph of three thermal properties of different steels in 3D. Different colors represent different clusters of data; darker hues indicate overlapping data points.

The K-means clustering model was tested with the 3 thermal properties of the steel dataset. The mean and standard deviation of each thermal property were calculated and recorded, then the data was normalized in Excel and exported to a CSV file. K-means clustering was carried out on the normalized data using the optimal k = 6 found in the previous step. Data was then de-normalized by calculating $d_{de-norm} = d_{norm} * \sigma + \mu$, then separated by cluster. Each cluster was then plotted in 3D using different colors, so as to visualize which data belongs to which cluster.

Discussion

I. Selection of K

The goodness of variance fit plots (GVF) for the thermal properties and electrical properties datasets converged upon a GVF of 1.0 at k = 12 and 11, respectively. As such, when GVF and k were plotted from k = 0 to ~11, clear “elbows” of the graphs stood out at k = 6 and 5, respectively. However, for the mechanical properties dataset, convergence of GVF was much slower, likely due to the mechanical properties dataset having 7 variables. The position of the elbow in the mechanical properties dataset was less clear: values of k between 9 and 17 clusters all showed sharp decreases in $\frac{d(GVF)}{dk}$.

With only 34 data points, having 17 clusters would make little sense, as many data points would have their own clusters, and the dimension of the data would not be reduced much. k = 11 was chosen as a model with a high GVF but a relatively low number of clusters for the data.

II. K-Means Clustering

1. Mechanical Clustering (Appendix A)

Clustering by mechanical properties with k = 11 clusters produced a significant number of “single-point” clusters: there were 3 clusters of only one element. These clusters (Appendix A) were each very distinct, so their separation is justified:

- Stainless Steel 316 had a significantly higher density than other comparable steels;
- Stainless Steel 302 (25%H) had unique values of Y, UTS, and Modulus of Elasticity relative to comparable steels;
- Stainless Steel 302 (annealed) had a much higher %elongation at break than its nearest neighbor, Stainless Steel 302 (25%H).

To a degree, the clusters of steel types align fairly well with the official classification numbers of the steels. For dimensional reduction, the 11 mechanical clusters could be described as:

1. AISI 1006 - 1020 Steel, cold drawn/rolled
2. AISI 1006 - 1020 Steel, other
3. AISI 1040 Steel
4. AISI 1090 and 1340 Steel
5. AISI 1095 - 8630 Steel with high yield strength and UTS
6. AISI 1095 - 8630 Steel with low Yield Strength and UTS
7. Stainless Steel 17-7 PH
8. Stainless Steel 302, annealed
9. Stainless Steel 302, 25% hardened
10. Stainless Steel 316
11. Stainless Steel 4xx

2. Thermal Clustering (Appendix B)

Clustering by thermal properties with k = 6 clusters produced clusters of size 3 or greater, which are grouped largely by the original AISI classifications. For dimensional reduction, the 6 clusters by thermal properties (Appendix B) can be described as:

1. AISI 1006 - 1040 steel
2. AISI 109x and 1340 Steel
3. AISI 5140 steel
4. AISI 4xxx and 8xxx steel
5. Stainless Steel 3xx
6. Stainless Steel, 17-7 PH and 4xx

3. Electrical Clustering (Appendix C)

Clustering by electrical properties with k = 5 clusters produced clusters of size 2 or greater, which are grouped largely by the original AISI classifications. For dimensional reduction, the 5 clusters by electrical properties (Appendix C) can be described as:

1. AISI 1xxx Steel
2. AISI 4140 - 8630 Steel
3. Stainless Steel 3xx
4. Stainless Steel 4xx
5. Stainless Steel 17-7 PH

III. Plotting of Thermal Clusters

The 3-dimensional scatter plot of thermal properties of steels shows that some clusters are distinct, while some clusters are very close to other clusters. The one cluster that stands out the most as being close to other clusters is the AISI 109x and 1340 Steels group: this group has a wide spread between data points (fig. 2, purple-colored data), is similar to the AISI 1006-1040 Steels group in thermal conductivity, and is similar to the AISI 4140-8630 Steels group in specific heat capacity. Within the AISI 109x and 1340 Steels group, there is a distinct difference between the AISI 1095 Steels and the other steel types. Based on the output of this plot, it may be appropriate to increase the number of clusters from k = 6 to k = 7 to split up the 109x and 1340 Steels group.

Appendix A: Mechanical Clustering

		Class Labels	Density	Hardness	Tensile Strength Ultimate	Yield Strength	Modulus of Elasticity	Poisson's Ratio	%Elongation at Break
AISI 1006 Steel	cold drawn	0	7.87	95	330	286	205	0.29	20
AISI 1020 Steel	cold rolled	0	7.87	121	421	350	205	0.29	15
AISI 1095 Steel	normalized	1	7.85	293	1015	505	205	0.29	10
AISI 4140 Steel	normalized	1	7.85	302	1018	656	205	0.29	18
AISI 4140 Steel	oil quenched	1	7.85	311	1075	985	205	0.29	15
AISI 4340 Steel	normalized	1	7.85	363	1281	863	205	0.29	12
AISI 4340 Steel	oil quenched	1	7.85	352	1208	1146	205	0.29	14
AISI 5140 Steel	oil quenched	1	7.85	293	971	840	205	0.29	18
AISI 8630 Steel	water quenched	1	7.85	293	1015	911	205	0.29	16
Stainless Steel 405	annealed	2	7.80	150	447	276	200	0.27	30
Stainless Steel 434	annealed	2	7.80	164	516	344	200	0.27	25
Stainless Steel 316	annealed	3	8.00	143	579	289	193	0.27	50
AISI 1040 Steel	cold drawn	4	7.85	170	586	515	200	0.29	12
AISI 1040 Steel	annealed	4	7.85	149	516	350	200	0.29	30
AISI 1040 Steel	hot rolled	4	7.85	149	526	289	200	0.29	18
AISI 1040 Steel	normalized	4	7.85	170	595	370	200	0.29	25
Stainless Steel 302	annealed	5	7.86	147	620	276	193	0.25	55
Stainless Steel 17-7 PH	cold rolled	6	7.80	378	1379	1211	204	0.27	1
Stainless Steel 17-7 PH	precipitation hardened	6	7.80	433	1650	1591	204	0.27	1
Stainless Steel 302	25% hardened	7	7.86	261	861	515	193	0.25	12
AISI 1095 Steel	annealed	8	7.85	192	665	379	205	0.29	13
AISI 4140 Steel	annealed	8	7.85	197	655	415	205	0.29	26
AISI 4340 Steel	annealed	8	7.85	217	744	469	205	0.29	22
AISI 5140 Steel	annealed	8	7.85	167	570	295	205	0.29	29
AISI 5140 Steel	normalized	8	7.85	229	794	469	205	0.29	23
AISI 8630 Steel	annealed	8	7.85	156	564	370	205	0.29	29
AISI 8630 Steel	normalized	8	7.85	187	649	424	205	0.29	24
AISI 1006 Steel	hot rolled	9	7.87	86	295	166	200	0.29	30
AISI 1020 Steel	normalized	9	7.87	131	440	344	200	0.29	36
AISI 1020 Steel	annealed	9	7.87	111	396	295	200	0.29	36
AISI 1090 Steel	hot rolled	10	7.87	248	842	460	200	0.29	10
AISI 1340 Steel	normalized	10	7.87	248	835	556	200	0.29	22
AISI 1340 Steel	oil quenched	10	7.87	285	952	834	200	0.29	19
AISI 1340 Steel	annealed	10	7.87	207	703	434	200	0.29	26

Table 1. A list of different steel types and 7 mechanical properties for each steel type. Each steel type is sorted by its cluster after a K-means clustering with k = 11 clusters.

Appendix B: Thermal Clustering

		Class Labels	Thermal Conductivity	Specific Heat Capacity	Coefficient of Thermal Expansion
AISI 1006 Steel	cold drawn	0	51.9	0.481	12.6
AISI 1006 Steel	hot rolled	0	51.9	0.481	12.6
AISI 1020 Steel	cold rolled	0	51.9	0.486	11.7
AISI 1020 Steel	normalized	0	51.9	0.486	11.7
AISI 1020 Steel	annealed	0	51.9	0.486	11.7
AISI 1040 Steel	cold drawn	0	51.9	0.486	11.3
AISI 1040 Steel	annealed	0	51.9	0.486	11.3
AISI 1040 Steel	hot rolled	0	51.9	0.486	11.3
AISI 1040 Steel	normalized	0	51.9	0.486	11.3
Stainless Steel 302	25% hardened	1	16.2	0.500	17.2
Stainless Steel 302	annealed	1	16.2	0.500	17.2
Stainless Steel 316	annealed	1	16.3	0.500	16.0
Stainless Steel 17-7 PH	cold rolled	2	16.4	0.460	11.0
Stainless Steel 17-7 PH	precipitation hardened	2	16.4	0.460	11.0
Stainless Steel 405	annealed	2	27.0	0.460	10.8
Stainless Steel 434	annealed	2	26.2	0.460	10.4
AISI 4140 Steel	normalized	3	42.6	0.473	12.2
AISI 4140 Steel	oil quenched	3	42.6	0.473	12.2
AISI 4140 Steel	annealed	3	42.6	0.473	12.2
AISI 4340 Steel	normalized	3	44.5	0.475	12.3
AISI 4340 Steel	annealed	3	44.5	0.475	12.3
AISI 4340 Steel	oil quenched	3	44.5	0.475	12.3
AISI 8630 Steel	annealed	3	46.6	0.475	12.2
AISI 8630 Steel	normalized	3	46.6	0.475	12.2
AISI 8630 Steel	water quenched	3	46.6	0.475	12.2
AISI 1090 Steel	hot rolled	4	51.9	0.472	11.5
AISI 1095 Steel	annealed	4	49.8	0.461	11.0
AISI 1095 Steel	normalized	4	49.8	0.461	11.0
AISI 1340 Steel	normalized	4	51.9	0.472	11.5
AISI 1340 Steel	oil quenched	4	51.9	0.472	11.5
AISI 1340 Steel	annealed	4	51.9	0.472	11.5
AISI 5140 Steel	annealed	5	44.6	0.452	12.6
AISI 5140 Steel	normalized	5	44.6	0.452	12.6
AISI 5140 Steel	oil quenched	5	44.6	0.452	12.6

Table 2. A list of different steel types and 3 thermal properties for each steel type. Each steel type is sorted by its cluster after a K-means clustering with k = 6 clusters.

Appendix C: Electrical Clustering

		Class Labels	Electrical Resistivity
AISI 1006 Steel	cold drawn	0	1.74E-05
AISI 1006 Steel	hot rolled	0	1.74E-05
AISI 1020 Steel	cold rolled	0	1.59E-05
AISI 1020 Steel	normalized	0	1.59E-05
AISI 1020 Steel	annealed	0	1.59E-05
AISI 1040 Steel	cold drawn	0	1.71E-05
AISI 1040 Steel	annealed	0	1.71E-05
AISI 1040 Steel	hot rolled	0	1.71E-05
AISI 1040 Steel	normalized	0	1.71E-05
AISI 1090 Steel	hot rolled	0	1.74E-05
AISI 1095 Steel	annealed	0	1.80E-05
AISI 1095 Steel	normalized	0	1.80E-05
AISI 1340 Steel	normalized	0	1.74E-05
AISI 1340 Steel	oil quenched	0	1.74E-05
AISI 1340 Steel	annealed	0	1.74E-05
Stainless Steel 302	25% hardened	1	7.20E-05
Stainless Steel 302	annealed	1	7.20E-05
Stainless Steel 316	annealed	1	7.40E-05
AISI 4140 Steel	normalized	2	2.20E-05
AISI 4140 Steel	oil quenched	2	2.20E-05
AISI 4140 Steel	annealed	2	2.20E-05
AISI 4340 Steel	normalized	2	2.48E-05
AISI 4340 Steel	annealed	2	2.48E-05
AISI 4340 Steel	oil quenched	2	2.48E-05
AISI 5140 Steel	annealed	2	2.28E-05
AISI 5140 Steel	normalized	2	2.28E-05
AISI 5140 Steel	oil quenched	2	2.28E-05
AISI 8630 Steel	annealed	2	2.34E-05
AISI 8630 Steel	normalized	2	2.34E-05
AISI 8630 Steel	water quenched	2	2.34E-05
Stainless Steel 405	annealed	3	6.00E-05
Stainless Steel 434	annealed	3	6.00E-05
Stainless Steel 17-7 PH	cold rolled	4	8.30E-05
Stainless Steel 17-7 PH	precipitation hardened	4	8.30E-05

Table 3. A list of different steel types and their electrical resistivity. Each steel type is sorted by its cluster after a K-means clustering with k = 5 clusters.

Appendix D: Python Script

```
import numpy as np
import matplotlib.pyplot as plt
import math
import time

from sklearn.cluster import KMeans


def read_data():

    #open up our training data
    f = open('normalized_csv.csv', 'r')
    counter = -1

    #create two arrays: one for names, one for data for each metal
    name_array = []
    data_array = []

    for line in f.readlines():

        #skip the first line in the spreadsheet
        counter += 1
        if counter == 0:
            continue

        #split line by csv
        line = line.replace('\n','')
        line_list = line.split(',')

        #once we get to an empty line, stop reading lines
        if line_list[0] == '':
            continue

        #let each index of each list correspond to the data of one metal
        name = line_list[0:2]
        data = line_list[2:]
        data = [float(x) for x in data]

        name_array.append(name)
        data_array.append(data)

        #time.sleep(1)

    means_and_stdevs = data_array[-2:]
    data_points_array = data_array[:-2]
```

```
    return name_array, data_points_array, means_and_stdevs

def fsdam(array):
    '''calculates the sum of deviations from array mean

    Args:
     - array: an (m*n) array of m samples (rows), and n attributes
        (columns)

    Calculated within:
     - array_mean: a (1*n) array of n attributes, where each index
        in the array gives the mean of each attribute

    Returns:
     - value of sdam
    '''

    #value of deviations from array mean for one row of array
    #for normalized data, means are 0 for all data
    array_means = [0]*len(array[0])

    #this should give the number of rows
    num_samples = len(array)
    distances_array_means = [np.linalg.norm(array[i]-array_means) for i in
range(num_samples)]
    sum_distances = sum(distances_array_means)

    return sum_distances

def fsdcm(array, class_means, class_ids):
    '''calculates the sum of deviations from class mean

    Args:
     - array: an (m*n) array of m samples (rows), and n attributes
        (columns)
     - class_means: a (k*n) array of k clusters (rows) and n attributes
        of the data (columns). each row corresponds to the sample mean
        of a different cluster
     - classifications: a (1*m) array that shows which  samples correspond
        to which labels

    Returns:
     - value of sdcm
     '''
    num_samples = len(array)
```

```
    dcms = []

    #iterate through all the samples; use indexing to find which class
    #they belong to and find deviation from class mean
    for i in range(num_samples):
        class_id = class_ids[i]
        sample_array = array[i]
        cluster_mean_array = class_means[class_id]
        distance_cluster_mean = np.linalg.norm(sample_array - cluster_mean_array)
        dcms.append(distance_cluster_mean)

    return sum(dcms)

def fgvf(array, class_means, class_ids):
    sdcm = fsdcm(array, class_means, class_ids)
    sdam = fsdam(array)
    gvf = (sdam - sdcm) / sdam
    return gvf

def find_k_vs_gvf(array, k_max = 10):
    #try out different values of k to determine what the optimal
    #value of k is ("the elbow of the GVF plot")
    #keep in mind we only have 34 data points. don't make too many clusters
    k_array = list(range(2,k_max))
    gvf_array = []

    for k in k_array:

        print('\nValue of k: ' + str(k))
        #use SKLearn KMeans class to analyze the data
        kmeans = KMeans(n_clusters = k, random_state = 0).fit(array)
        class_means = kmeans.cluster_centers_
        class_ids = kmeans.labels_
        gvf = fgvf(array, class_means, class_ids)
        gvf_array.append(gvf)

    #plot the relation between k and GVf
    return k_array, gvf_array

def plot_k_vs_gvf(mech_array, therm_array, elec_array):
    #make plots to determine optimal k for each property
    k_array_mech, gvf_array_mech = find_k_vs_gvf(mech_array, k_max = 25)
    k_array_therm, gvf_array_therm = find_k_vs_gvf(therm_array,  k_max = 12)
    k_array_elec, gvf_array_elec = find_k_vs_gvf(elec_array,k_max = 11)

    plt.figure(1)
```

```
    plt.plot(k_array_mech, gvf_array_mech, c='blue')
    plt.title('Mechanical properties: k vs. GVF')
    plt.xlabel('Value of k')
    plt.ylabel('Goodness of Fit (GVF)')

    plt.figure(2)
    plt.plot(k_array_therm,gvf_array_therm, c='red')
    plt.title('Thermal properties: k vs. GVF')
    plt.xlabel('Value of k')
    plt.ylabel('Goodness of Fit (GVF)')

    plt.figure(3)
    plt.plot(k_array_elec, gvf_array_elec, c='green')
    plt.title('Electrical properties: k vs. GVF')
    plt.xlabel('Value of k')
    plt.ylabel('Goodness of Fit (GVF)')
    plt.show()

def plot_elec_clusters(elec_array, elec_k):
    #graph electrical properties to test this out

    elec_kmeans = KMeans(n_clusters = elec_k, random_state = 0).fit(elec_array)
    elec_class_ids = elec_kmeans.labels_
    elec_class_means = elec_kmeans.cluster_centers_

    #separate the data points into different lists based on which
    #classID they are. iterate through each of the class IDs to figure out
    classid = 0
    lst_by_class = []

    for classid in range(elec_k):
        data = [elec_array[i][0] for i in range(len(elec_array)) if
                elec_class_ids[i] == classid]
        lst_by_class.append(data)

    #iterate through and plot the data of different classes by color
    plt.figure(1)
    for i in range(elec_k):
        classx = lst_by_class[i]
        y_axis = [0]*len(classx)
        plt.scatter(classx, y_axis, s=32)

    plt.show()

def plot_therm_clusters(therm_array, k_therm, means_and_stdevs):
    #plot thermal data in 3D to visualize clusters
```

```
kmeans_therm = KMeans(n_clusters = k_therm, random_state = 0).fit(therm_array)
therm_class_ids = kmeans_therm.labels_
therm_class_means = kmeans_therm.cluster_centers_

#de-normalize data using d = d_normal * sigma + mu
denorm_array = []
for data in therm_array:
    denorm_array.append([data[i] * means_and_stdevs[1][i]
            + means_and_stdevs[0][i] for i in range(len(data))] )

therm_array = denorm_array[:]

#separate the data points into different lists based on which
#classID they are. iterate through each of the class IDs to figure out
classid = 0
lst_by_class = []

for classid in range(k_therm):
    data = [therm_array[i] for i in range(len(therm_array)) if
            therm_class_ids[i] == classid]
    lst_by_class.append(data)

#iterate through and plot the data of different classes by color
fig = plt.figure(1)
ax = fig.add_subplot(projection = '3d')

for i in range(k_therm):

    class_points = lst_by_class[i]
    class_x = [point[0] for point in class_points]
    class_y = [point[1] for point in class_points]
    class_z = [point[2] for point in class_points]

    ax.scatter(class_x, class_y, class_z)

ax.set_xlabel('Thermal conductivity')
ax.set_ylabel('Specific heat capacity')
ax.set_zlabel('Coefficient of thermal expansion')
ax.set_title('Thermal properties of steels with k=6 clusters')
plt.show()
```

```
def classify_and_export(mech_array, therm_array, elec_array):
    #selected values of k
    k_mech = 11
    k_therm = 6
    k_elec = 5

    #find clusters of data and show them with the normalized data
    mech_kmeans = KMeans(n_clusters = k_mech, random_state = 0).fit(mech_array)
    therm_kmeans = KMeans(n_clusters = k_therm, random_state = 0).fit(therm_array)
    elec_kmeans = KMeans(n_clusters = k_elec, random_state = 0).fit(elec_array)

    mech_class_ids = mech_kmeans.labels_
    therm_class_ids = therm_kmeans.labels_
    elec_class_ids = elec_kmeans.labels_

    #group together the normalized data and their classIDs to export to csv
    mech_array_and_classid = [list(np.concatenate(( [mech_class_ids[i]], mech_array[i]
)) )  for i in range(len(mech_array)) ]
    mech_array_and_classid = np.array(mech_array_and_classid)

    therm_array_and_classid = [list(np.concatenate(( [therm_class_ids[i]],
therm_array[i] )) ) for i in range(len(therm_array)) ]
    therm_array_and_classid = np.array(therm_array_and_classid)

    elec_array_and_classid = [list(np.concatenate(( [elec_class_ids[i]], elec_array[i]
)) ) for i in range(len(elec_array)) ]
    elec_array_and_classid = np.array(elec_array_and_classid)

    #export all these class data arrays to CSV
    f = open('mech_array_labeled.csv', 'w')
    g = open('therm_array_labeled.csv', 'w')
    h = open('elec_array_labeled.csv', 'w')

    num_points = len(mech_array_and_classid)
    for i in range(num_points):

        str_mech = ','.join([str(x) for x in mech_array_and_classid[i]])
        str_therm = ','.join([str(x) for x in therm_array_and_classid[i]])
        str_elec = ','.join([str(x) for x in elec_array_and_classid[i]])

        f.write(str_mech + '\n')
        g.write(str_therm + '\n')
        h.write(str_elec + '\n')

    f.close()
    g.close()
```

```
    h.close()

if __name__ == '__main__':
    name_list, data_list, means_and_stdevs = read_data()

    #extract mechanical, thermal, electrical data from array of lists
    mech_data = [inner_list[0:7] for inner_list in data_list]
    therm_data = [inner_list[7:10] for inner_list in data_list]
    elec_data = [inner_list[10:11] for inner_list in data_list]

    #means and stdevs for data, for use in de-normalizing data
    mech_mus_sigmas = [inner_list[0:7] for inner_list in means_and_stdevs]
    therm_mus_sigmas = [inner_list[7:10] for inner_list in means_and_stdevs]
    elec_mus_sigmas = [inner_list[10:11] for inner_list in means_and_stdevs]

    #convert to numpy format
    mech_array = np.asarray(mech_data)
    therm_array = np.asarray(therm_data)
    elec_array = np.asarray(elec_data)

    #for finding ideal values of k for each set of properties
    #plot_k_vs_gvf(mech_array, therm_array, elec_array)

    #selected values of k
    k_mech = 11
    k_therm = 6
    k_elec = 5

    #plot_elec_clusters(elec_array, k_elec)
    plot_therm_clusters(therm_array, k_therm, therm_mus_sigmas)
    #classify_and_export(mech_array, therm_array, elec_array)
```