LVDT

August 24, 2020

1 K-Means clustering

K-Means clustering is a method of vector quantization that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

K-Means clustering minimizes within-cluster variances (squared Euclidean distances in which the corresponding loss function is the squared error loss and places progressively greater weight on larger errors), but not regular Euclidean distances, which would be the more difficult Weber problem.

The problem is computationally difficult (NP-hard).

1.1 Description of the situation

With a LVDT (an electromechanical device used to convert mechanical motion or vibrations, specifically rectilinear motion, into a variable electrical current, voltage or electric signals, and the reverse) and a Micrometer (a device incorporating a calibrated screw) I have obtained 20 measurements for every 0.002 millimeters between -0.500 and 0.500. At -0.500 the LVDT is expanded and the values are higher. At 0.500 the it's compressed and the values are lower.

```
In [1]: import pandas as pd

        original_df = pd.read_csv('LVDT noise at fixed measure - Sheet1.csv')
        original_df[list(original_df.columns[:3])+list(original_df.columns[-3:])]
```

Out[1]:

	0.500	0.498	0.496	-0.496	-0.498	-0.500
0	1307.0	1311.0	1307.0	1694.0	1698.0	1691.0
1	1308.0	1312.0	1311.0	1695.0	1699.0	1697.0
2	1309.0	1313.0	1311.0	1696.0	1699.0	1697.0
3	1313.0	1313.0	1311.0	1697.0	1699.0	1697.0
4	1314.0	1314.0	1312.0	1697.0	1699.0	1697.0
5	1314.0	1314.0	1314.0	1697.0	1699.0	1699.0
6	1315.0	1314.0	1315.0	1698.0	1699.0	1699.0
7	1316.0	1314.0	1316.0	1698.0	1700.0	1700.0
8	1316.0	1315.0	1316.0	1698.0	1700.0	1700.0
9	1316.0	1315.0	1316.0	1698.0	1700.0	1701.0
10	1317.0	1316.0	1316.0	1699.0	1700.0	1702.0
11	1317.0	1317.0	1317.0	1699.0	1701.0	1702.0

```
12  1318.0  1318.0  1317.0  1699.0  1701.0  1702.0
13  1319.0  1318.0  1318.0  1699.0  1701.0  1702.0
14  1319.0  1319.0  1320.0  1700.0  1701.0  1703.0
15  1319.0  1319.0  1321.0  1700.0  1702.0  1704.0
16  1319.0  1321.0  1321.0  1700.0  1702.0  1704.0
17  1321.0  1323.0  1322.0  1701.0  1702.0  1705.0
18  1323.0  1325.0  1323.0  1701.0  1702.0  1706.0
19  1324.0  1332.0  1325.0  1703.0  1705.0  1708.0
```

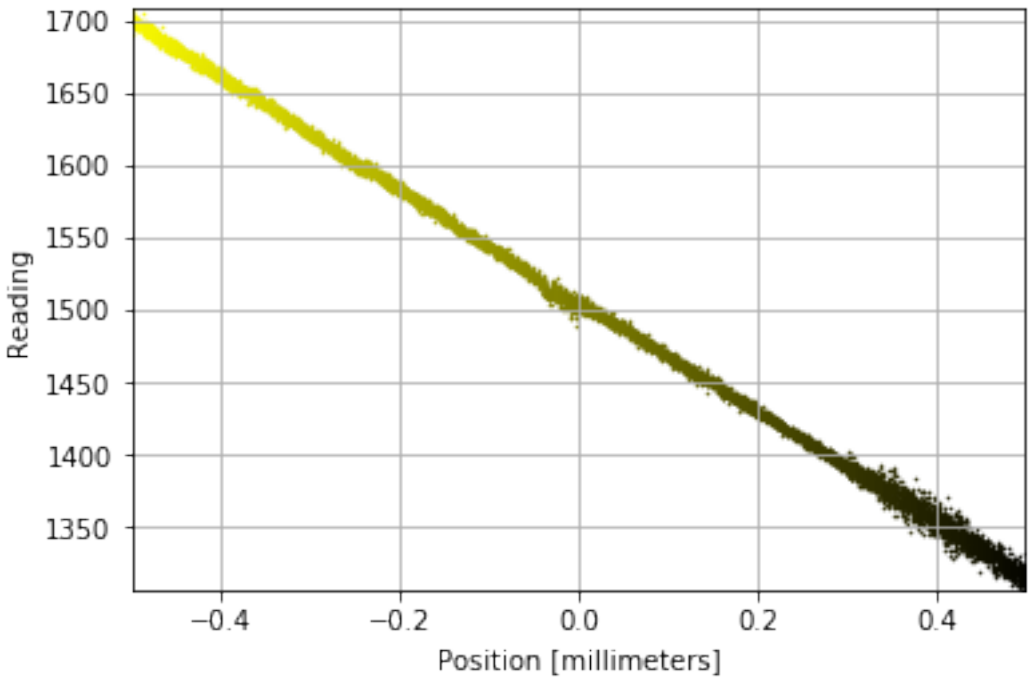
We can see that:

1. One measurement value can appear multiple times in the same column and also across multiple columns.
2. The measurements don't represent a straight linear function with the position of the LVDT. On the column 0.498 we have higher values than it's neighbouring positions.

So the problem that I have is, when receiving a value how do I know at which position the LVDT is? For example, 1315.0 is present in the columns 0.500, 0.498 and 0.496. It's obvious that I cannot differentiate every position every 0.002 millimeters. So I have to create new partitions.

```
In [2]: modMinPosition = -0.500
        modMaxPosition = 0.500
        num_centroids = 5
        random_centroids = True

In [3]: %matplotlib inline

        import time
        import matplotlib.pyplot as plt
        import matplotlib.colors as mc

        import colorsys

        import numpy as np

        from random import uniform

        dfPositions = [header for header in original_df.columns
                       if ( modMinPosition <= float(header) and float(header) <= modMaxPositio
        df = original_df[dfPositions].copy()

        minPosition = float(df.columns[len(df.columns)-1])
        maxPosition = float(df.columns[0])
        minLVDTReading = df.min().min()
        maxLVDTReading = df.max().max()

In [4]: def adjust_colour(color, amount):
            try:
                c = mc.cnames[color]
```

```
    except:
        c = color
    finally:
        c = colorsys.rgb_to_hls(*mc.to_rgb(c))
        return [colorsys.hls_to_rgb(c[0], max(0, min(1, amount * c[1])), c[2])]

figure, axes = plt.subplots()

for position in df:
    colourIndex = adjust_colour('yellow', ((df[position].mean() - minLVDTReading)/(maxL
    axes.scatter([float(position)]*len(df[position]), list(df[position]), c = colourInd

def set_axis_labelling(axis):
    global minPosition, maxPosition, minLVDTReading, maxLVDTReading
    axis.set_xlim([minPosition, maxPosition])
    axis.set_ylim([minLVDTReading, maxLVDTReading])
    axis.set_xlabel('Position [millimeters]')
    axis.set_ylabel('Reading')
    axis.grid(b = True, which = 'major', axis = 'both')
    return axis

axes = set_axis_labelling(axes)
```



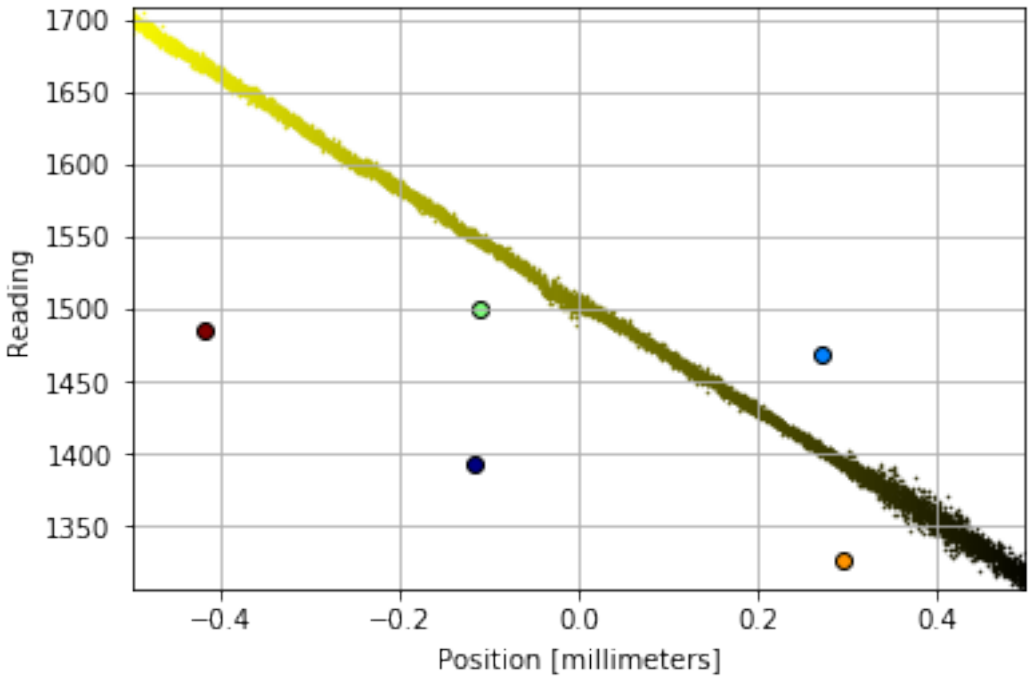
2 Initialisation

In [5]:
```
# centroids[i] = [position, reading]
if random_centroids:
    centroids = {
        i+1: [uniform(minPosition, maxPosition), uniform(minLVDTReading, maxLVDTReadin
        for i in range(num_centroids)
    }
else:
    # Centroids equally spaced along the diagonal
    x = np.linspace(minPosition, maxPosition, num_centroids+2)
    y = np.linspace(maxLVDTReading, minLVDTReading, num_centroids+2)
    centroids = {
        i+1: [x[i+1],y[i+1]]
        for i in range(num_centroids)
    }

cmap = plt.get_cmap('jet')
colmap = cmap(np.linspace(0, 1.0, num_centroids))
```

In [6]:
```
def plot_centroids(axes, centroidsDF):
    global centroids, colmap
    axesReferences = []
    for i in centroids.keys():
        ref = axes.scatter(*centroids[i], color = [colmap[i-1]], edgecolors = 'black')
        axesReferences.append(ref)
    centroidsDF['Current_position'] = axesReferences
    return axes, centroidsDF

def delete_centroids(centroidsDF):
    global centroids
    for i in centroids.keys():
        centroidsDF['Current_position'][i].remove()
    return centroidsDF
```

In [7]:
```
centroids_data = {'Current_position': [None]*num_centroids}
centroidsDF = pd.DataFrame(centroids_data, columns =
                           ['Current_position'], index=list(range(1,num_centroids+1)))

axes, centroidsDF = plot_centroids(axes, centroidsDF)
figure
```

Out[7]:



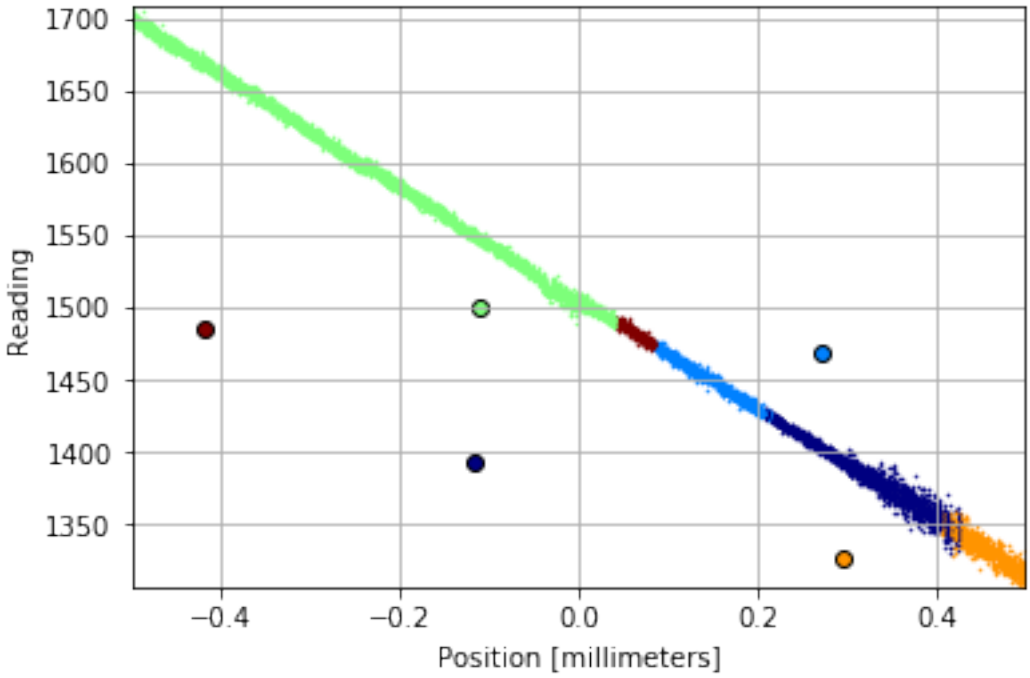
3 Assignment

```
In [8]: def assignment(df, centroids):
            global dfPositions, dfClosest

            for position in dfPositions:
                for reading in df[position]:
                    dfDistances = []
                    for i in centroids.keys():
                        dfDistances.append(position + '_distance_from_' + str(i))
                        df[position + '_distance_from_' + str(i)] = \
                            np.sqrt((float(position) - centroids[i][0])** 2 + (reading - centr
                df[position + '_closest'] = df.loc[:, dfDistances].idxmin(axis='columns')
                df[position + '_closest'] = df[position + '_closest'].map(lambda x: int(x.spli
                df[position + '_color'] = df[position + '_closest'].map(lambda x: colmap[x-1])
            return df

In [9]: dfClosest = list(map(lambda x: x + '_closest', dfPositions))
        df = assignment(df, centroids)

In [10]: def make_plot(axes):
             global dfPositions, centroids, colmap
             axes.cla()
             for position in dfPositions:
```

```
            axesReferences = []
            for _, info in df[[position, position + '_color']].iterrows():
                ref = axes.scatter(float(position), float(info[0]), color = info[1], s = (
                axesReferences.append(ref)
            df[position + '_axes'] = axesReferences
        return axes

    def new_make_plot(axes, closest_centroids, df):
        global dfPositions, centroids, colmap
        for position in dfPositions:
            for row, info in df[[position, position + '_color']].iterrows():
                if df[position + '_closest'][row] != closest_centroids[position + '_closes
                    df[position + '_axes'][row].set_color(info[1])
        return axes, df
```

In [11]:
```
axes = make_plot(axes)
axes, centroidsDF = plot_centroids(axes, centroidsDF)
axes = set_axis_labelling(axes)
figure
```

Out[11]:



4 Update Centroids Positions

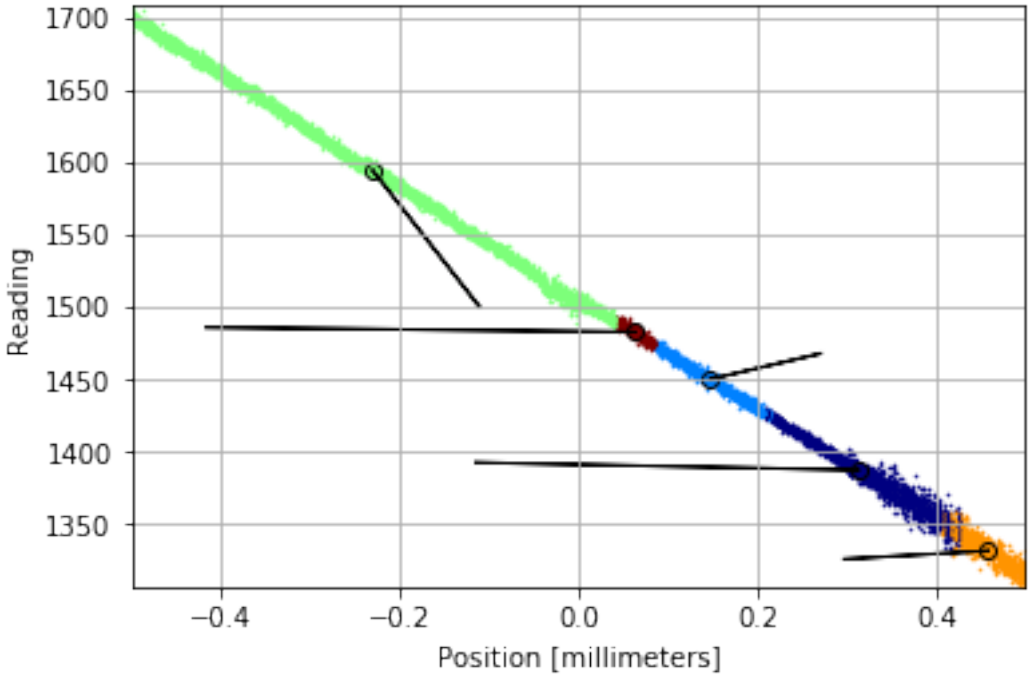
In [12]:
```
import copy
```

```
        old_centroids = copy.deepcopy(centroids)

        def update_centroids_position(centroids):
            for i in centroids.keys():
                readings = pd.Series()
                positions = []
                for position in dfPositions:
                    readings = readings.append(df.loc[df[position + '_closest'] == i, position
                    positions += [float(position)]*(df.loc[df[position + '_closest'] == i, pos
                if len(positions) > 0:
                    centroids[i][0] = np.mean(positions)
                    centroids[i][1] = np.mean(readings)
            return centroids

        centroids = update_centroids_position(centroids)
```

In [13]:
```
        def plot_updated_centroids(axes, centroidsDF):
            global old_centroids, centroids, colmap
            axesReferences = []
            for i in old_centroids.keys():
                old_x = old_centroids[i][0]
                old_y = old_centroids[i][1]
                dx = (centroids[i][0] - old_centroids[i][0])
                dy = (centroids[i][1] - old_centroids[i][1])
                ref = axes.arrow(old_x, old_y, dx, dy, fc = colmap[i-1], ec = 'black')
                axesReferences.append(ref)
            centroidsDF['Updated_position'] = axesReferences
            return axes, centroidsDF

        def delete_updated_centroids(centroidsDF):
            global centroids
            for i in centroids.keys():
                centroidsDF['Updated_position'][i].remove()
            return centroidsDF
```

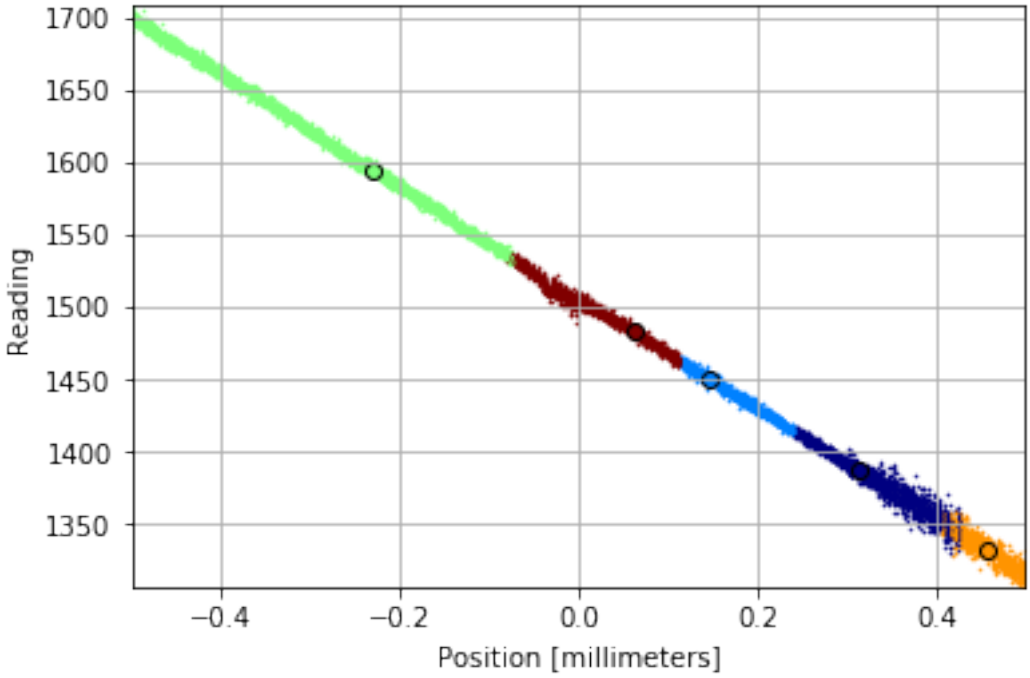
In [14]:
```
        centroidsDF = delete_centroids(centroidsDF)
        # Plots the new position
        axes, centroidsDF = plot_centroids(axes, centroidsDF)
        # Plots the lines between points
        axes, centroidsDF = plot_updated_centroids(axes, centroidsDF)
        figure
```

Out[14]:



5 Update Assignment

```
In [15]: closest_centroids = df[dfClosest].copy(deep = True)
         df = assignment(df, centroids)

In [16]: centroidsDF = delete_updated_centroids(centroidsDF)
         axes, df = new_make_plot(axes, closest_centroids, df)
         axes = set_axis_labelling(axes)
         figure

Out[16]:
```

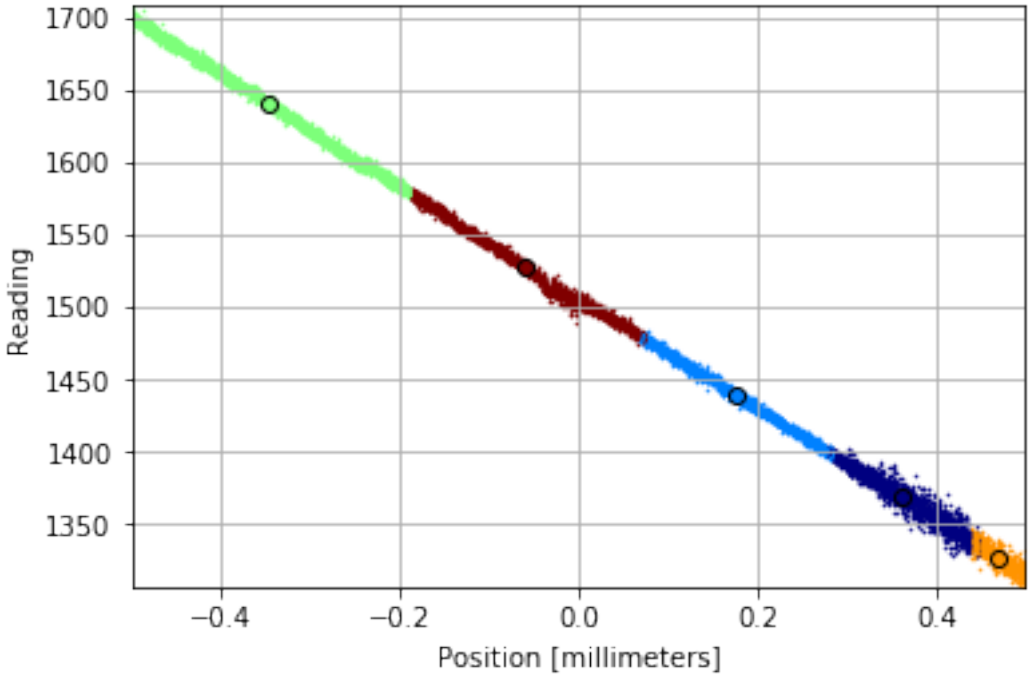


6 Continue To Update Until No Changes

```
In [17]: number_of_steps = 0
         previous_closest_centroids = df[dfClosest].copy(deep = True)
         while True:
             centroids = update_centroids_position(centroids)
             closest_centroids = df[dfClosest].copy(deep = True)
             df = assignment(df, centroids)
             number_of_steps += 1
             if closest_centroids.equals(df[dfClosest]):
                 break
```

```
In [18]: centroidsDF = delete_centroids(centroidsDF)
         axes, df = new_make_plot(axes, previous_closest_centroids, df)
         axes, centroidsDF = plot_centroids(axes, centroidsDF)
         axes = set_axis_labelling(axes)
         figure
```

Out[18]:



modMinPosition = -0.500
modMaxPosition = 0.500
num_centroids = 5
random_centroids = True
Execution time: 4:43