Implementation and Use of The K-Means Algorithm

Peter Mucsi

September 11, 2014

Contents

Abstract

The *k-means* algorithm is described and an implementation in Python is presented. The algorithm is applied to a data set obtained from the UCI Machine Learning Repository. The result of each iteration of the *k-means* algorithm is illustrated and variations in the results are studied for different values of k.

1 Introduction

The *k-means* algorithm is a type of unsupervised machine learning algorithm. It is a clustering algorithm that operates on n-dimensional data sets. In this paper, for the sake of data visualization, it will only be applied to 2-dimensional data sets. The objective of *k-means* is to find points in the data space that are near the centers of groupings of data (Bishop, 2006, pp. 424–428). One of the difficulties of implementing the algorithm is that it is usually not known how many groupings there are in the data. Plotting the J-value against the number of centers (k) can help determine the correct number.

2 Python Implementation of *k-means*

This section contains a summary of the *k-means* algorithm, an implementation of it in Python, and some demonstrations of the algorithm's behavior on two-dimensional data.

2.1 *k-means*

The *k-means* algorithm operates by choosing random centers in the data space, then fine-tuning the location of those centers by minimizing the sum of squared distances from the points of the group assigned to a particular center. It is an iterative algorithm, which alternately refines the position of the centers, then reassigns data points to the nearest center if necessary. The assignment of a data point to single center result in the formation of clusters.

Not all dimensions of the data set is useful for the clustering algorithm, so an important step in the implementation of *k-means* is feature selection. For the sake of data visualization, only 2 attributes of the data set are selected to form a basis for clustering. Dimensionality reduction can also occur in real life applications since performing calculations involving multi-dimensional data is expensive.

2.2 Implementation

In this section, the Python implementation of the *k-means* clustering algorithm is discussed. The algorithm operates on multidimensional data, and its purpose is to find centers in Euclidian space that minimize the square means of distances of the data points from these centers. It is practical to create a function that calculates J, the value that needs to be minimized. It is obviously necessary to create a function that is the implementation of k-means and return an array of centers, which are n-dimensional vectors representing a point in Euclidian space.

```
def calcJ(data,centers):
    diffsq = (centers[:,np.newaxis,:] - data)**2
    return np.sum(np.min(np.sum(diffsq,axis=2), axis=0))
```

The function *calcJ* takes 2 arguments, the data set and the centers. The data set has more dimensions than the centers, therefore a new dimension is introduced by using *np.newaxis* (which is just a synonym for Python's 'None') so that we can perform the matrix operation for calculating the sum of square distances from *each* data point in the data set.

The *kmeans* function takes three arguments: the data set, the number of centers, and the number of iterations (we will see later how to approach a good value for both).

The algorithm chooses a random center without replacement from the data set, then assigns each point to the closest center. Then refines the location of centers by finding the mean of data points in the group.

```
def kmeans(data, k = 2, n = 5):
    # Initialize centers and list J to track performance metric
    centers = data[np.random.choice(range(data.shape[0]),k,replace=False), :]
    J = []

    # Repeat n times
    for iteration in range(n):

        # Which center is each sample closest to?
        sqdistances = np.sum((centers[:,np.newaxis,:] - data)**2, axis=2)
        closest = np.argmin(sqdistances, axis=0)

        # Calculate J and append to list J
        J.append(calcJ(data,centers))

        # Update cluster centers
        for i in range(k):
            centers[i,:] = data[closest==i,:].mean(axis=0)

    # Calculate J one final time and return results
    J.append(calcJ(data,centers))
    return centers,J,closest
```

2.3 Demonstrations

In this section the *k-means* algorithm is demonstrated on a simple, artificial data set. Figure 1 is a scatter plot of the artificially created data. It simply represents the following 2 sets: x = 2, 4, 5, 9, 10, 12, y = 3, 8, 11, 15, 16, 17. By visualizing the data, it seems that the samples belong 2 clusters. Let's see what happens when we apply k-mean to the data set using the following Python code:

```
d_data = np.vstack((x, y)).T
```

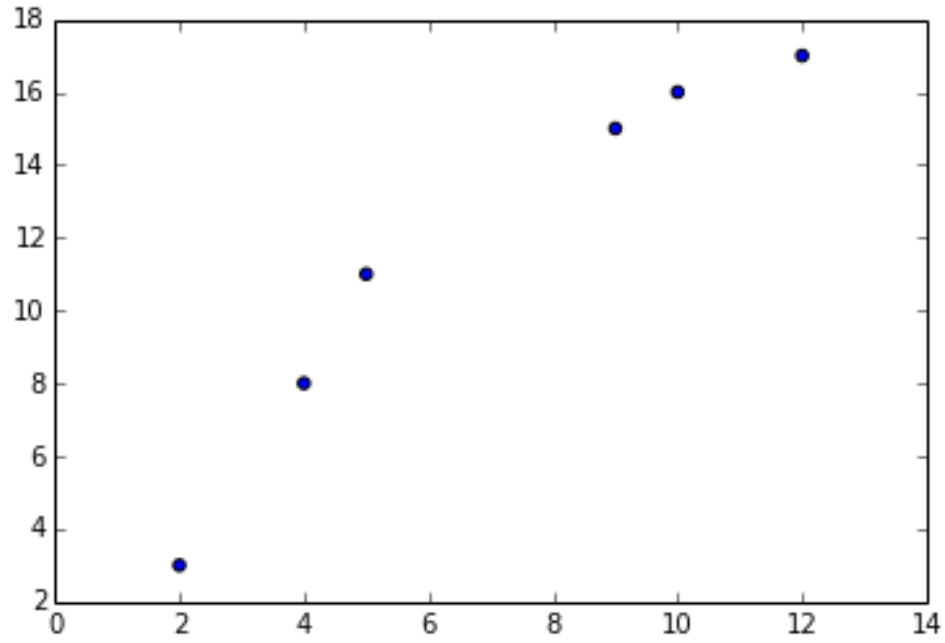

Figure 1: This is a simple, 2-dimensional data set in which we might be able to find clusters

```
d_centers = kmeans(d_data, 2, 10)
d_closest = closest(d_centers, d_data)

plt.close()
for idx, cen in enumerate(d_closest):
    m = markers[cen];
    c = colors[cen];
    plt.scatter(d_data[idx,0], d_data[idx,1], marker=m, color=c)
```

After running *k-means* with 10 iterations and targeting 2 groups, the algorithm finds 2 clusters as it can be seen in Figure 2. During the experiment there were some variations in the assignment of samples to the centers. This can be explained by the fact that the algorithm contains a random element, therefore the outcome cannot always be reproduced.

3 Experiments

This section describes the experiments performed on a dataset from the UCI Repository using the k-means clustering algorithm. The selected data contains dimensional measurements of 3 types of seed.

The data have kernels belonging to three different varieties of wheat: Kama, Rosa and Canadian, 70 elements each, randomly selected for the experiment. High quality visualization of the internal kernel structure was detected using a soft X-ray technique. Studies were conducted using combine harvested wheat grain originating from experimental fields, explored at the Institute of Agrophysics of the Polish Academy of Sciences in Lublin.

3.1 Data Attributes

To construct the data, seven geometric parameters of wheat kernels were measured:

1. area A
2. perimeter P

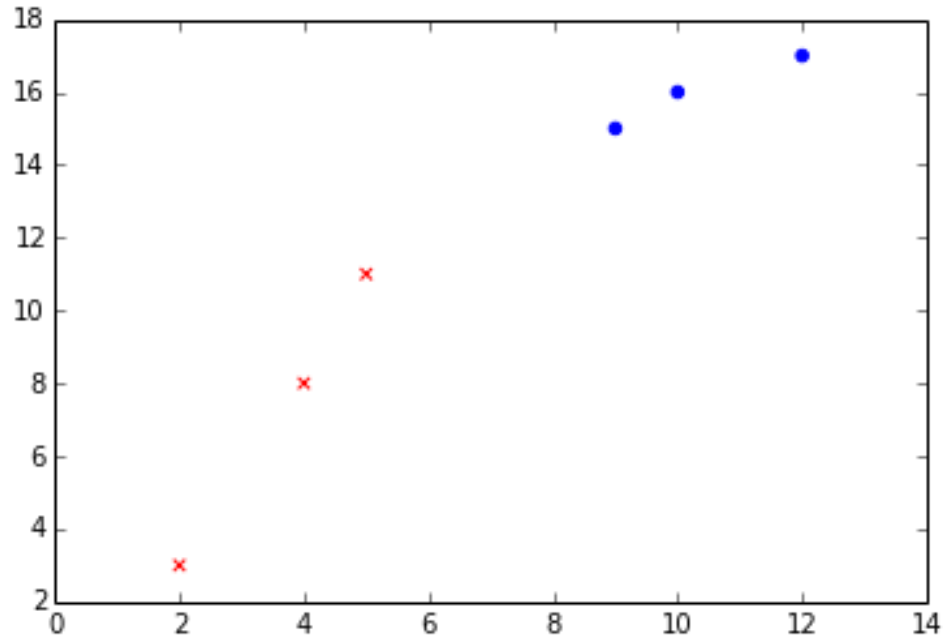

Figure 2: Clusters found by running k-means with 5 iterations and 2 centers

3. compactness $C = 4 * \Pi * A/P^2$,
4. length of kernel
5. width of kernel
6. asymmetry coefficient
7. length of kernel groove

All of these parameters are real-valued continuous.

3.2 Feature Selection

Feature selection is an important process in machine learning, this section describes how our attributes were selected for the experiment.

Features are selected by variance. The intuition is that if the variance is higher, it is probably easier to separate the samples into different groups than if the values are very close to each other.

Figure 3 demonstrates how the values of 8 attributes change over the samples of the data set. The blue line on the bottom of the chart suggests that there are indeed 3 types of samples. Although determining the value of k is possible to some extent with experimentation, it is not always successful. We have an easier task with this data set, since k is known. 3 was created by the following snippet:

```
plt.plot(data.values)
plt.legend(["Area", "Perimeter", "Compactness", "Length", "Width", "Assymetry", "GrooveLength"], l
```

An alternative way of visualizing the attributes is by sample. In Figure 4 each line represents a data sample, so the differences in the variance of the attributes is more visible. Figure 4 was created with the following Python code:

```
for idx, row in enumerate(data.iloc[:].values):
    plt.plot(row, label='row')
```

For this experiment, the 0^{th} and 5^{th} attribute were selected as values for 2-dimensional data because it is expected that attributes with greater variance will help us create good clusters.

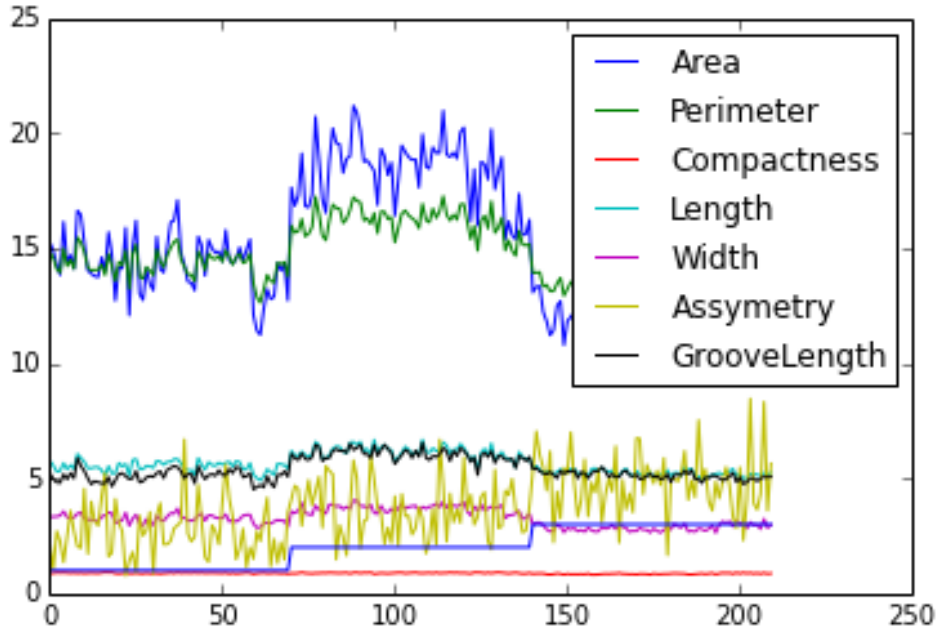

Figure 3: Visualization of the 8 features of seed data, each line corresponds to 1 attribute of the data series

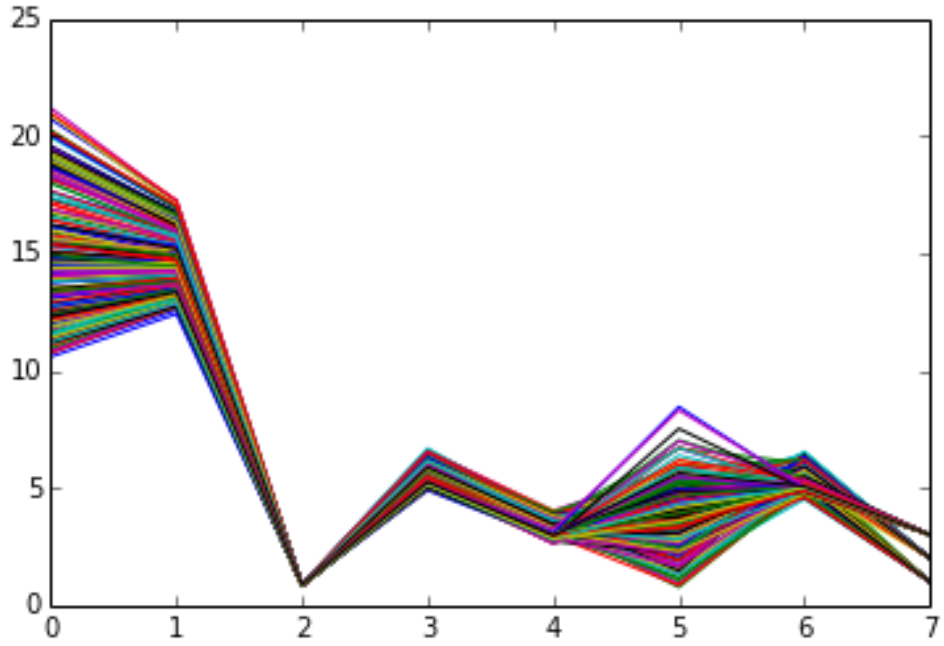

Figure 4: Alternative visualization of the attributes of seed data, each line corresponds to one data sample

3.3 Clusters

This section describes the clusters, which were found using *k-means*.

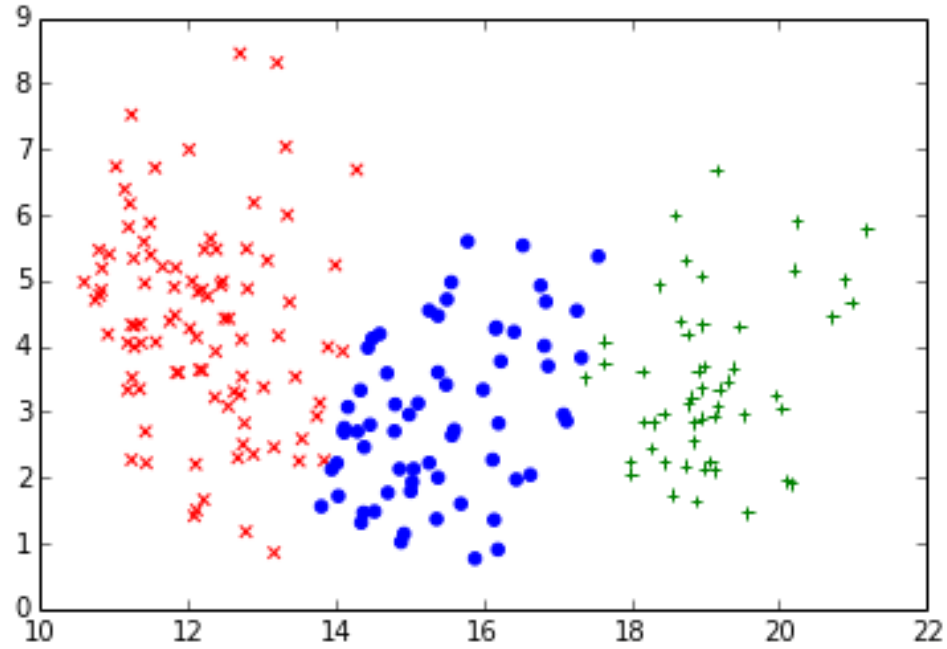

Figure 5: Clusters found by running k-means with 5 iterations and 3 centers (there are 3 types of seeds

Although, the colors and shapes of the plot can naturally make the viewer believe that this is the correct grouping of the data samples, there were some variations in the results when the k-means algorithm was run. These clusters look natural as there are some patches of empty areas that separate the groups. The code that produced the plot:

```
col1 = 0;
col2 = 5;
planar_data = np.vstack((data.values[:,col1], data.values[:,col2])).T
centers = kmeans(planar_data, 3, 5)
cent_idx = closest(centers, planar_data)
markers = ['x', 'o', '+', 'D']
colors = ['red', 'blue', 'green', 'purple']
plt.close()
for idx, cen in enumerate(close):
    m = markers[cen];
    c = colors[cen];
    plt.scatter(planar_data[idx,0], planar_data[idx,1], marker=m, color=c)
```

3.4 Number of iterations and cost (J-value)

```
iters = [i for i in range(1,10)]
j_vec = list()
for it in iters:
    it_centers = kmeans(planar_data, 3, it)
    j_vec.append(calcJ(planar_data, it_centers))

plt.close()
plt.xlabel("Iterations")
```

```
plt.ylabel("J value")
plt.plot(iters, j_vec, '-')
```

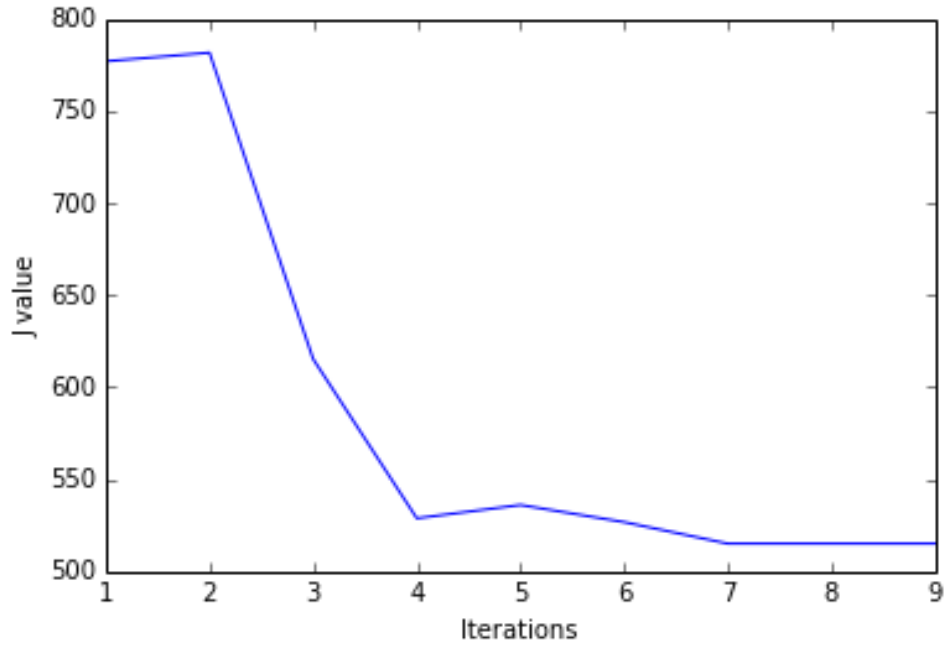

Figure 6: Number of iterations versus cost - no significant improvement after the 4th iteration

The number of iterations determine how much the data is converged at the time the algorithm completes the task of assignments and refinements of the data centers. From Figure 6, it is clear that it does not make sense to choose large numbers for n. Although there is some variation of value J starting from iteration 5, a number greater than 5 is probably not going to help us achieve better results. The reason for not having an improvement for more iterations could be that after the randomly selected centers have been corrected, there is less room for a dramatic improvement.

3.5 Number of centers and cost (k)

```
iters = [i for i in range(1,10)]
j_vec = list()
for it in iters:
    it_centers = kmeans(planar_data, it, 5)
    j_vec.append(calcJ(planar_data, it_centers))

plt.close()
plt.xlabel("k")
plt.ylabel("J value")
plt.plot(iters, j_vec, '-')
```

The kernel data set includes 3 types of seeds. Therefore k is known. To demonstrate the proper process, *k-means* was executed using various k values, and the result is plotted. Figure 7 shows how the increase of k reduces J. This makes sense because the smaller the group, there are fewer distances to sum, and those distances are smaller. When k equals the number of samples, J is 0. For this dataset, the value of 3 for k seems like a good choice because J does not significantly decrease with values greater than 4.

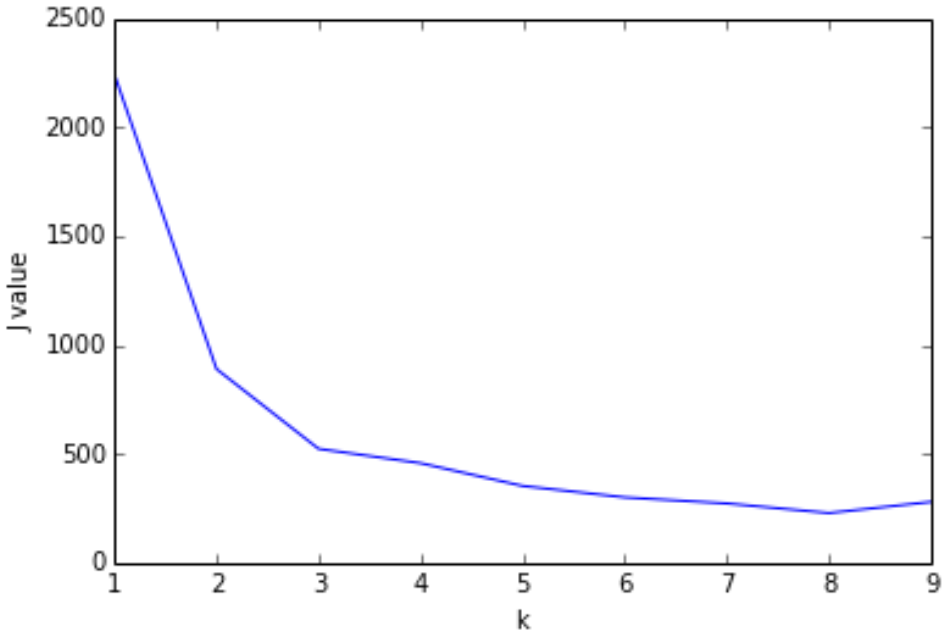

Figure 7: Number of centers versus cost - it is known that there are 3 types of seed in the data set, but increasing k does not significantly reduce cost

References

Christopher Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.