k-means

April 16, 2019

0.1 # Gahan Saraiya (18MCEC10)

0.2 # AIM: Unsupervised Learning - K means clustering

K-means clustering is a type of unsupervised learning, which is used when data are unlabeled. The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K. The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on feature similarity. The results of the K-means clustering algorithm are:

The centroids of the K clusters, which can be used to label new data Labels for the training data (each data point is assigned to a single cluster)

```
In [1]: # Modules
        import os
        import matplotlib.pyplot as plt
        from matplotlib.image import imread
        import pandas as pd
        import seaborn as sns
        from sklearn.datasets.samples_generator import (make_blobs,
                                                         make_circles,
                                                         make_moons)
        from sklearn.cluster import KMeans, SpectralClustering
        from sklearn.preprocessing import StandardScaler
        from sklearn.metrics import silhouette_samples, silhouette_score

        %matplotlib inline

In [2]: import numpy as np
        from numpy.linalg import norm


        class Kmeans:
            '''Implementing Kmeans algorithm.'''
            def __init__(self, n_clusters, max_iter=100, random_state=123, plot_on_every_iter=
                self.n_clusters = n_clusters
                self.max_iter = max_iter
                self.random_state = random_state
        #         self.plot_on_every_iter = seld.max_iter  # 10 else plot_on_every_iter
```

```python
def initializ_centroids(self, X):
    np.random.RandomState(self.random_state)
    random_idx = np.random.permutation(X.shape[0])
    centroids = X[random_idx[:self.n_clusters]]
    return centroids

def compute_centroids(self, X, labels):
    centroids = np.zeros((self.n_clusters, X.shape[1]))
    for k in range(self.n_clusters):
        centroids[k, :] = np.mean(X[labels == k, :], axis=0)
    return centroids

def compute_distance(self, X, centroids):
    distance = np.zeros((X.shape[0], self.n_clusters))
    for k in range(self.n_clusters):
        row_norm = norm(X - centroids[k, :], axis=1)
        distance[:, k] = np.square(row_norm)
    return distance

def find_closest_cluster(self, distance):
    return np.argmin(distance, axis=1)

def compute_sse(self, X, labels, centroids):
    distance = np.zeros(X.shape[0])
    for k in range(self.n_clusters):
        distance[labels == k] = norm(X[labels == k] - centroids[k], axis=1)
    return np.sum(np.square(distance))

def plot(self):
    # Plot the clustered data
    fig, ax = plt.subplots(figsize=(6, 6))
    plt.scatter(X_std[self.labels == 0, 0], X_std[self.labels == 0, 1],
                c='green', label='cluster 1')
    plt.scatter(X_std[self.labels == 1, 0], X_std[self.labels == 1, 1],
                c='blue', label='cluster 2')
    plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=300,
                c='r', label='centroid')
    plt.legend()
    plt.xlim([-2, 2])
    plt.ylim([-2, 2])
    plt.xlabel('Eruption time in mins')
    plt.ylabel('Waiting time to next eruption')
    plt.title('Visualization of clustered data', fontweight='bold')
    ax.set_aspect('equal');

def fit(self, X):
    self.centroids = self.initializ_centroids(X)
    for i in range(self.max_iter):
```

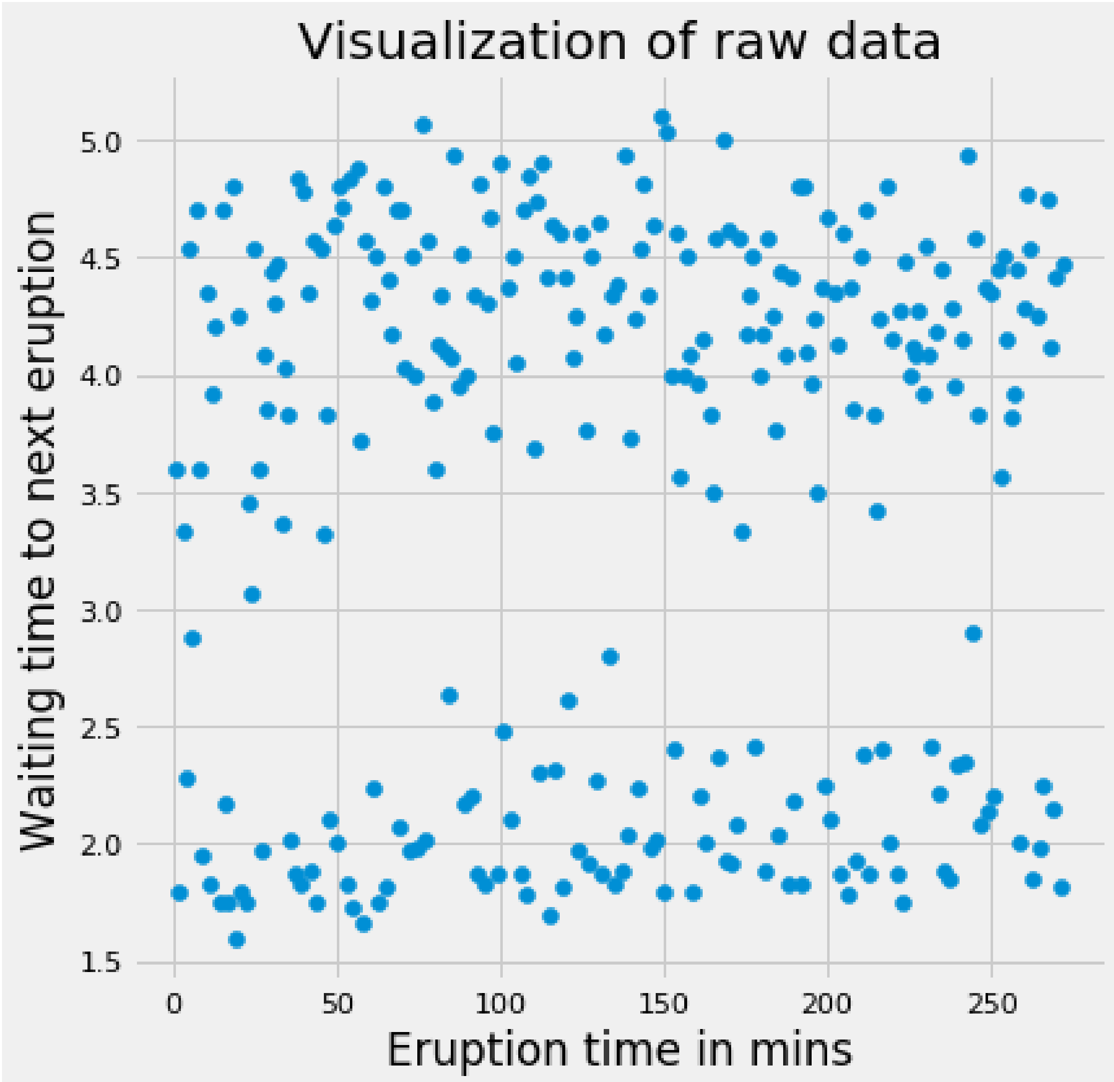
```
                old_centroids = self.centroids
                distance = self.compute_distance(X, old_centroids)
                self.labels = self.find_closest_cluster(distance)
                self.centroids = self.compute_centroids(X, self.labels)
                if np.all(old_centroids == self.centroids):
                    break
            self.error = self.compute_sse(X, self.labels, self.centroids)

        def predict(self, X):
            distance = self.compute_distance(X, old_centroids)
            return self.find_closest_cluster(distance)
```

```
In [3]: sns.set_context('notebook')
        plt.style.use('fivethirtyeight')
        from warnings import filterwarnings
        filterwarnings('ignore')

        # Import the data
        DATA_PATH = os.path.join('datasets', 'faithful.csv')
        df = pd.read_csv(DATA_PATH)

        # Plot the data
        plt.figure(figsize=(6, 6))
        plt.scatter(df.iloc[:, 0], df.iloc[:, 1])
        plt.xlabel('Eruption time in mins')
        plt.ylabel('Waiting time to next eruption')
        plt.title('Visualization of raw data');
```

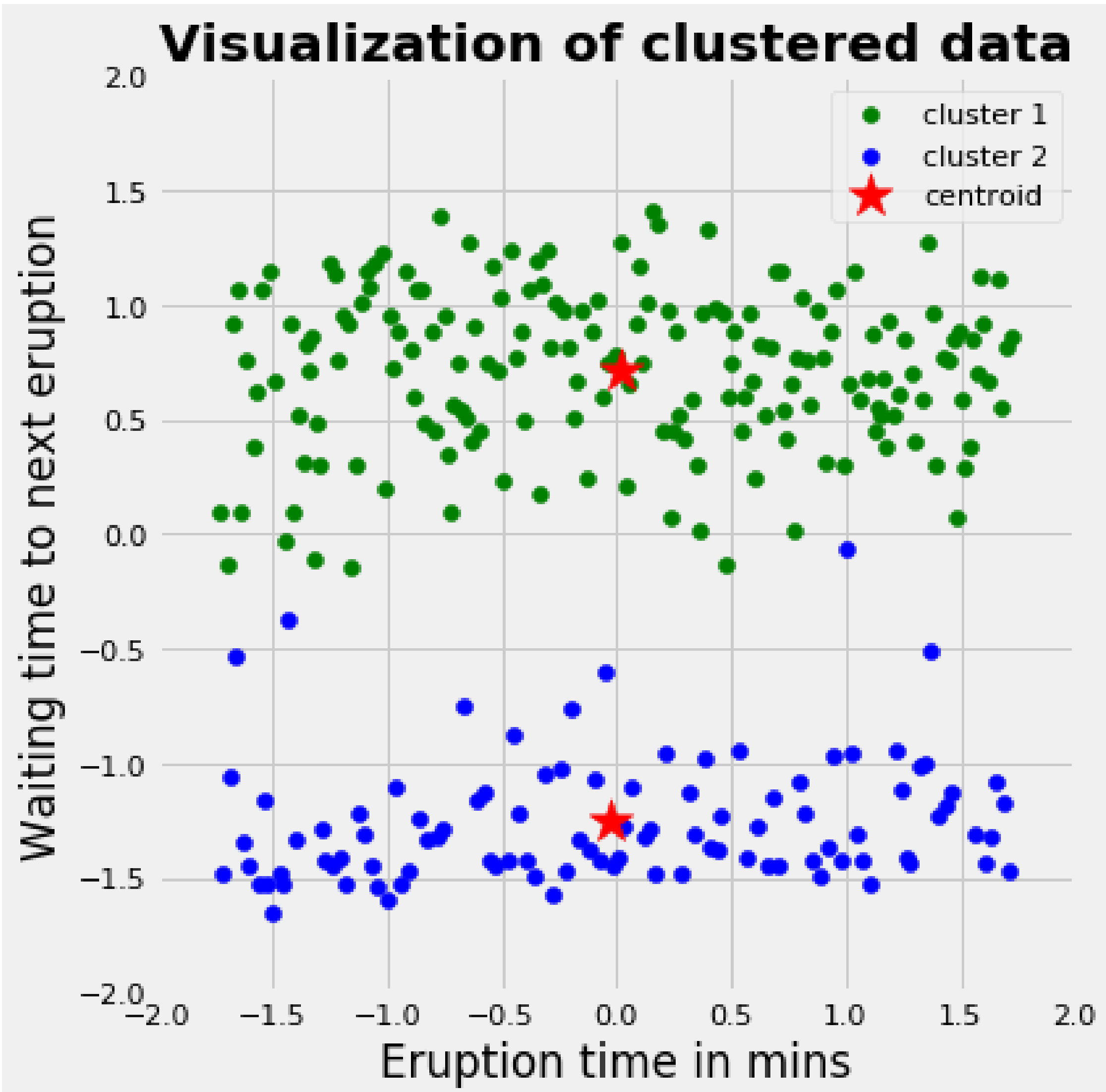


In [4]:
```
# Standardize the data
X_std = StandardScaler().fit_transform(df)

# Run local implementation of kmeans
km = Kmeans(n_clusters=2, max_iter=100)
km.fit(X_std)
centroids = km.centroids

# Plot the clustered data
fig, ax = plt.subplots(figsize=(6, 6))
plt.scatter(X_std[km.labels == 0, 0], X_std[km.labels == 0, 1],
            c='green', label='cluster 1')
plt.scatter(X_std[km.labels == 1, 0], X_std[km.labels == 1, 1],
            c='blue', label='cluster 2')
```

```
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=300,
            c='r', label='centroid')
plt.legend()
plt.xlim([-2, 2])
plt.ylim([-2, 2])
plt.xlabel('Eruption time in mins')
plt.ylabel('Waiting time to next eruption')
plt.title('Visualization of clustered data', fontweight='bold')
ax.set_aspect('equal');
```

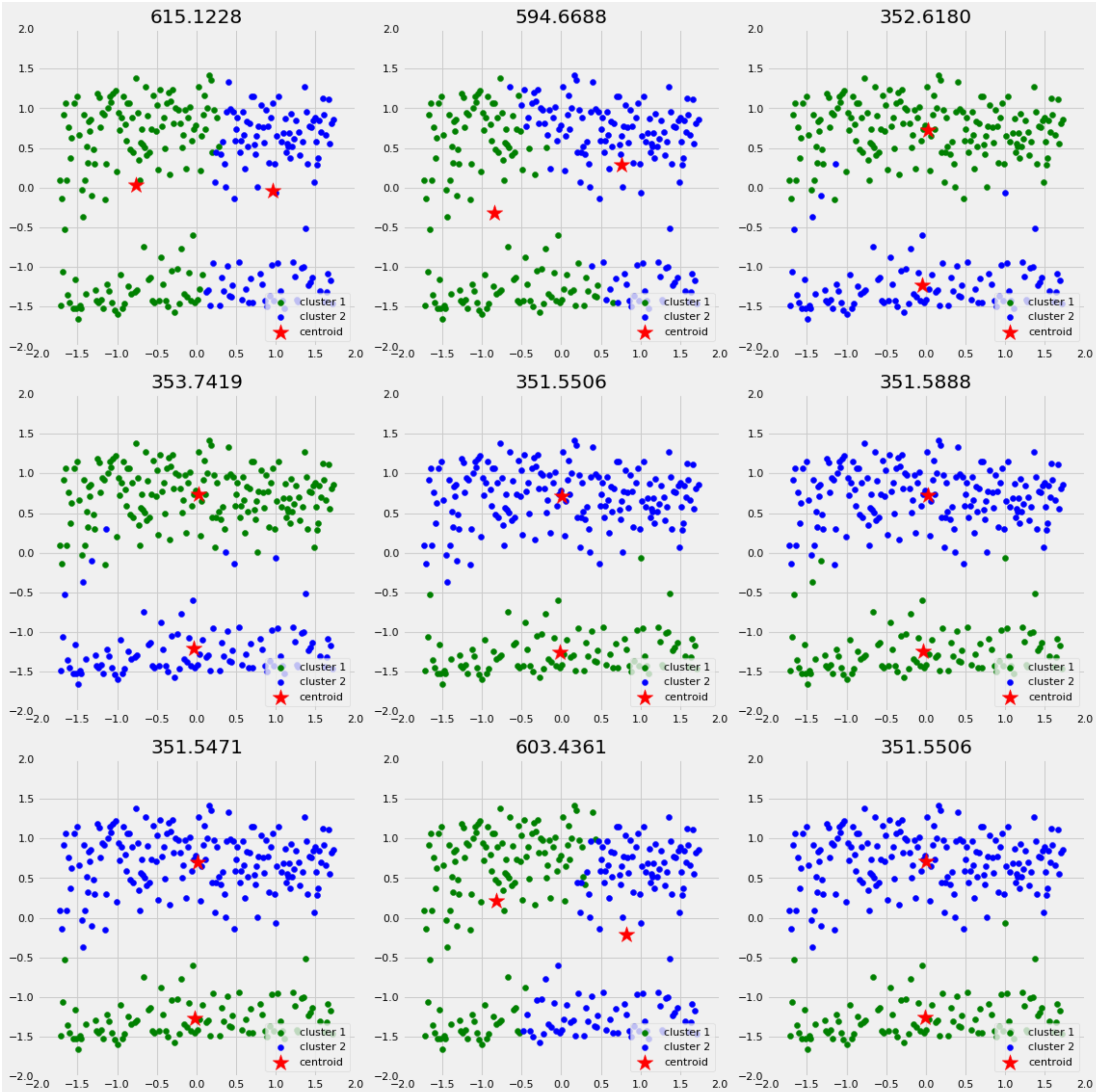



```
In [ ]:
```

0.3 different initializations of centroids

It may yield to different results. I'll use 9 different `random_state` to change the initialization of the centroids and plot the results. The title of each plot will be the sum of squared distance of each initialization.

```
In [5]: n_iter = 9
        fig, ax = plt.subplots(3, 3, figsize=(16, 16))
        ax = np.ravel(ax)
        centers = []
        for i in range(n_iter):
            # Run local implementation of kmeans
            km = Kmeans(n_clusters=2,
                        max_iter=3,
                        random_state=np.random.randint(0, 1000, size=1))
            km.fit(X_std)
            centroids = km.centroids
            centers.append(centroids)
            ax[i].scatter(X_std[km.labels == 0, 0], X_std[km.labels == 0, 1],
                          c='green', label='cluster 1')
            ax[i].scatter(X_std[km.labels == 1, 0], X_std[km.labels == 1, 1],
                          c='blue', label='cluster 2')
            ax[i].scatter(centroids[:, 0], centroids[:, 1],
                          c='r', marker='*', s=300, label='centroid')
            ax[i].set_xlim([-2, 2])
            ax[i].set_ylim([-2, 2])
            ax[i].legend(loc='lower right')
            ax[i].set_title('{:.4f}'.format(km.error))
            ax[i].set_aspect('equal')
        plt.tight_layout();
```



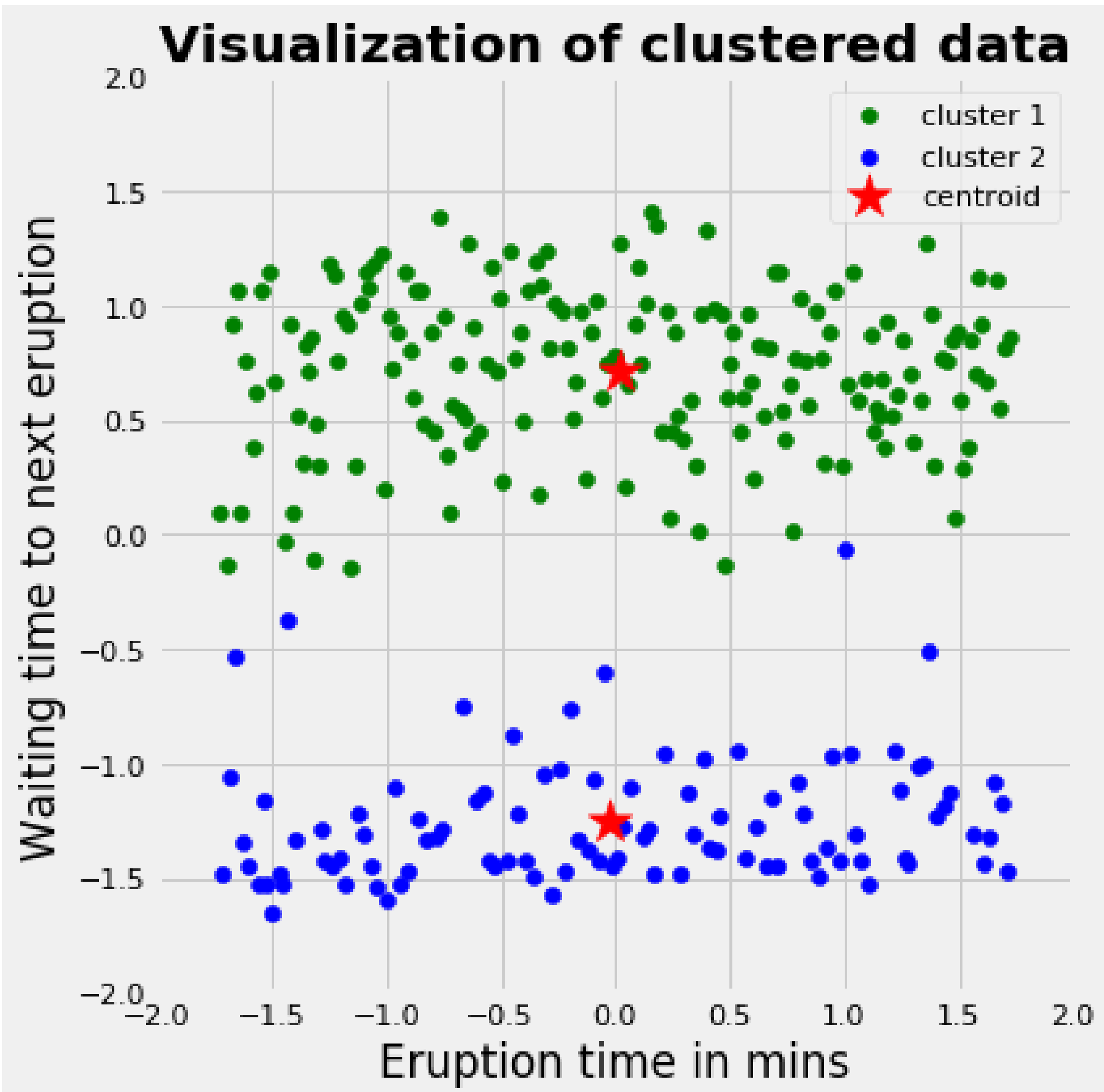

```
In [6]: ## For 1 iteration

In [11]: # Standardize the data
         X_std = StandardScaler().fit_transform(df)

         # Run local implementation of kmeans
         km = Kmeans(n_clusters=2, max_iter=9)
         km.fit(X_std)
         centroids = km.centroids

         # Plot the clustered data
         fig, ax = plt.subplots(figsize=(6, 6))
         plt.scatter(X_std[km.labels == 0, 0], X_std[km.labels == 0, 1],
```

```
             c='green', label='cluster 1')
plt.scatter(X_std[km.labels == 1, 0], X_std[km.labels == 1, 1],
            c='blue', label='cluster 2')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=300,
            c='r', label='centroid')
plt.legend()
plt.xlim([-2, 2])
plt.ylim([-2, 2])
plt.xlabel('Eruption time in mins')
plt.ylabel('Waiting time to next eruption')
plt.title('Visualization of clustered data', fontweight='bold')
ax.set_aspect('equal');
```



0.4 References

- https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a
- https://medium.com/machine-learning-algorithms-from-scratch/k-means-clustering-from-scratch-in-python-1675d38eee42
- https://www.dummies.com/programming/big-data/data-science/how-to-visualize-the-clusters-in-a-k-means-unsupervised-learning-model/
- https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html
- https://www.kaggle.com/dhanyajothimani/basic-visualization-and-clustering-in-python/data


```
In [ ]:
```