[1]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
import wikipedia
import random

CONVERGENCE_DELTA = 1e-5
MAXIMUM_ITERATIONS = 200
```

[2]:
```
given_articles = [
    'Linear algebra',
    'Data Science',
    'Artificial intelligence',
    'European Central Bank',
    'Financial technology',
    'International Monetary Fund',
    'Basketball',
    'Swimming',
    'Cricket'
]
```

[3]:
```
class KMeans():
    def __init__(
        self,
        x_train,
        y_train,
        num_clusters=3,
        seed: str = "random",
    ):
        self.dataset = x_train
        self.targets = y_train
        self.k = num_clusters
        self.num_features = x_train.shape[1]
        self.num_samples = x_train.shape[0]
        if seed == "random":
            self.centroids = self.random_initialise_centroids()
        elif seed == "custom":
            self.centroids = self.initialise_from_data()
        else:
            raise ValueError("Cohoose a seed between ['random', 'custom']")
        self.old_centroids = np.copy(self.centroids)
        self.cluster_labels = np.zeros(self.num_samples, dtype=int)
        for i in range(self.num_samples):
            self.cluster_labels[i] = np.argmin(
                np.linalg.norm(self.dataset[i]-self.centroids, ord=2, axis=1))


    def random_initialise_centroids(self):
```

```
        mean = np.mean(self.dataset, axis = 0)
        std = np.std(self.dataset, axis = 0)
        return np.random.randn(self.k, self.num_features)*std + mean

    def initialise_from_data(self):
        centroids = np.copy(self.dataset[np.random.choice(
                self.num_samples, self.k, replace=(False if self.k <= self.
↪num_samples else True))])
        return centroids

    def get_centroid_labels(self):
        centroid_labels = np.zeros(self.k)
        for i in range(self.k):
            count = np.bincount(self.targets[self.cluster_labels == i])
            if len(count) > 0:
                centroid_labels[i] = np.argmax(count)
        return centroid_labels

    def calculate_loss(self):
        loss = np.mean(np.linalg.norm(
            self.dataset - self.centroids[self.cluster_labels], ord=2, axis=1),␣
↪axis=0)
        return loss

    def fit(self):
        for i in range(MAXIMUM_ITERATIONS):
            # assigning clusters to all data points
            for i in range(self.num_samples):
                self.cluster_labels[i] = np.argmin(
                    np.linalg.norm(self.dataset[i]-self.centroids, ord=2,␣
↪axis=1))
            prev_centers = np.copy(self.centroids)
            converged = True
            for i in range(self.k):
                alloted = self.dataset[self.cluster_labels == i]
                if len(alloted) > 0:
                    self.centroids[i] = np.mean(alloted, axis=0)
                else:
                    self.centroids[i] = np.zeros(self.num_features)
                if np.linalg.norm(prev_centers[i] - self.centroids[i]) >␣
↪CONVERGENCE_DELTA:
                    converged = False
            loss = self.calculate_loss()
            if converged is True:
                print(f"TOTAL ITERATIONS = {i}")
                break
            self.old_centroids = np.copy(self.centroids)
```

[4]:
```
articles = []
for article_name in given_articles:
    text = wikipedia.page(article_name, preload=True).content
    articles.append(text)
vectorizer = TfidfVectorizer(stop_words={'english'})
x_train = vectorizer.fit_transform(articles).toarray()
y_train = np.arange(len(given_articles))
k = [4, 8, 12]
losses = []
```

[21]:
```
random.seed(60)
np.random.seed(60)
```

[45]:
```
for num_clusters in k:
    kmeans = KMeans(x_train, y_train, num_clusters=num_clusters, seed='custom')
    print(f"\n----------------------------------\nNUMBER OF CLUSTERS =␣
 ↪{num_clusters}")
    kmeans.fit()
    losses.append(kmeans.calculate_loss())
    clusters = [[] for i in range(num_clusters)]
    for i, article in enumerate(given_articles):
        index = kmeans.cluster_labels[i]
        clusters[index].append(article)
    for i, cluster in enumerate(clusters):
        print("Cluster no. {}: {}".format(i+1, cluster))
```

```
----------------------------------
NUMBER OF CLUSTERS = 4
TOTAL ITERATIONS = 3
Cluster no. 1: ['Data Science']
Cluster no. 2: ['Basketball', 'Swimming', 'Cricket']
Cluster no. 3: ['European Central Bank', 'Financial technology', 'International
Monetary Fund']
Cluster no. 4: ['Linear algebra', 'Artificial intelligence']

----------------------------------
NUMBER OF CLUSTERS = 8
TOTAL ITERATIONS = 7
Cluster no. 1: ['Financial technology']
Cluster no. 2: ['Artificial intelligence']
Cluster no. 3: ['Data Science']
Cluster no. 4: ['International Monetary Fund']
Cluster no. 5: ['Swimming']
Cluster no. 6: ['Linear algebra']
Cluster no. 7: ['European Central Bank']
Cluster no. 8: ['Basketball', 'Cricket']
```

```
---------------------------------
NUMBER OF CLUSTERS = 12
TOTAL ITERATIONS = 11
Cluster no. 1: ['Financial technology']
Cluster no. 2: ['Basketball', 'Cricket']
Cluster no. 3: ['Data Science']
Cluster no. 4: ['European Central Bank']
Cluster no. 5: []
Cluster no. 6: ['Swimming']
Cluster no. 7: ['Linear algebra']
Cluster no. 8: []
Cluster no. 9: ['Artificial intelligence']
Cluster no. 10: []
Cluster no. 11: ['International Monetary Fund']
Cluster no. 12: []
```