Document clustering using K Means algorithm

The following clusters are found with varying values of K for the given 9 documents:

1. K = 4

Cluster	Titles
1	Basketball, Cricket
2	Linear algebra, Data science, Artificial Intelligence
3	Financial technology, International Monetary fund, European Central Bank
4	Swimming

2. K = 6

Cluster	Titles
1	Linear Algebra
2	European Central Bank, International Monetary fund
3	Financial technology
4	Data Science
5	Basketball, Swimming, Cricket
6	Artificial intelligence

3. K = 8

Cluster	Titles
1	Cricket
2	Basketball
3	Swimming
4	Artificial Intelligence
5	Financial Technology, International Monetary Fund
6	Linear Algebra
7	Data Science
8	European Central Bank

4. K = 12

Cluster	Titles
1	Cricket
2	Basketball
3	Swimming
4	Artificial Intelligence
5	Financial Technology
6	Linear Algebra
7	Data Science
8	European Central Bank
9	International Monetary Fund
10	<Empty>
11	<Empty>
12	<Empty>

c) The optimal option for K from the above is k = 4 for the following reasons:

1) We can clearly see semantically similar concept documents from different categories, i.e AI, Finance and Sports,
2) For k = 6/8 we can see only very few concepts getting clustered together, Since we have only 9 concepts it is essential that we take k to be < N/2 at least since we can see clear semantics which are maintained by tf-idf vectorization as well

Note: All cluster centroids initializations are done from the dataset since it is very difficult to find vital seed initializations from random values as K-means converge to a local minima easily.

Kmeans_document

September 15, 2021

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

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

[ ]:
```
class KMeans():
    def __init__(
        self,
        x_train,
        y_train,
        num_clusters=3,
        max_iter=100,
        tol=1e-4,
        seed: str = None,
    ):
        """
        Initialize KMeans object.
        Arguments:
            dataset: numpy array of shape (n_samples, n_features)
            k: number of clusters
            max_iter: maximum number of iterations
            tol: tolerance for convergence
            seed: initial cluster centroids choice ['random','cluster']
        """
        self.dataset = x_train
        self.targets = y_train
```

```
        self.k = num_clusters
        self.max_iter = max_iter
        self.tol = tol

        self.num_features = x_train.shape[1]
        self.num_samples = x_train.shape[0]
        self.losses = []

        if seed == "random":
            self.centroids = np.random.uniform(
                size=(self.k, self.num_features))
        elif seed == "cluster":
            if (self.k > self.num_samples):  # hack for large k
                self.centroids = np.copy(self.dataset[np.random.choice(
                    self.num_samples, self.k, replace=True)])
            else:
                self.centroids = np.copy(self.dataset[np.random.choice(
                    self.num_samples, self.k, replace=False)])
        else:
            raise ValueError("seed must be in ['random', 'cluster']")
        # store old centroids for convergence check
        self.old_centroids = np.copy(self.centroids)
        # store cluster assignment indexes
        self.cluster_labels = np.zeros(self.num_samples, dtype=int)
        self.assign_clusters()

    def converged(self):
        return np.all(np.linalg.norm(self.centroids - self.old_centroids,␣
↪ord=2, axis=1) < self.tol)

    def assign_clusters(self):
        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 fit(self, verbose=False):
        for i in range(self.max_iter):
            self.assign_clusters()
            self.update_centroids()
            loss = self.calc_loss()
            self.losses.append(loss)
            if verbose:
                print(f"Iteration {i+1} Loss: {loss}")
                print("---------------------------")
            if self.converged():
                print(f"Total Iterations: {i+1}, Loss: {loss}")
```

```
                break
            self.old_centroids = np.copy(self.centroids)

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

    def update_centroids(self):
        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)
```

[ ]:
```
def load_data():
    articles = [wikipedia.page(
        title, preload=True).content for title in titles]
    vectorizer = TfidfVectorizer(stop_words={'english'})
    x_train = vectorizer.fit_transform(articles).toarray()
    y_train = np.arange(len(titles))

    return (x_train, y_train), vectorizer
```

[ ]:
```
def main():
    (x_train, y_train),_ = load_data()
    print("Data loaded, Finding Clusters ...")
    k = [6]
    losses = []
    for num_clusters in k:
        kmeans = KMeans(x_train, y_train, num_clusters=num_clusters,
                        seed='cluster', tol=1e-7, max_iter=100)
        kmeans.fit(verbose=False)
        print("Clusters found, printing results ...")
        losses.append(kmeans.calc_loss())
        clusters = [[] for i in range(num_clusters)]
        for i, title in enumerate(titles):
            index = kmeans.cluster_labels[i]
            clusters[index].append(title)
        print("Clusters:")
        for i, cluster in enumerate(clusters):
            print("Cluster {}: {}".format(i, cluster))
```

[ ]:
```
main()
```

[ ]:

[ ]: