WEB MINING

16BCE1184
SOUMOK DUTTA

CODE :

```
import string
import numpy as np
import requests
import re
from bs4 import BeautifulSoup
from bs4.element import Comment
from nltk.stem import PorterStemmer

# Function to filter the HTML tags and text
def visible_text(element):
    if element.parent.name in ['style', 'title', 'script', 'head', '[document]', 'class', 'a', 'li']:
        return False
    elif isinstance(element, Comment):
        return False
    elif re.match(r"[\s\r\n]+",str(element)):
        return False
    elif re.match(r"www.", str(element)):
        return False
    return True

class document_clustering(object):
    """Implementing the document clustering class.
    It creates the vector space model of the passed documents and then  creates K-Means Clustering to
organize them.
    Parameters:
    ----------
    file_dict: dictionary
        Contains the path to the different files to be read.
        Format: {file_index: path}
    word_list: list
        Contains the list of words using which the vector space model is to  be created.
    k: int
        Number of clusters to be created from the documents.
    Attributes:
        listing_dict_: dictionary
        Contains the frequency of the words in each document as file_index
```

```
        as key and frequency as value.
    distance_matrix_: 2D array
        Contains the square matrix of documents containing the pairwise   distance between them.
    centroids_: dictionary
        Contains the centroids of k-means clustering
    classes_: dictionary
        Contains the cluster index as index of the document and documents   assigned to them as value in the form of list
    features_: dictionary
        Contains the coordinates of the points assigned to a cluster in a   list    """


    def __init__(self, file_dict, word_list, k):
        self.file_dict = file_dict
        self.word_list = word_list
        self.k = k

    def tokenize_document(self, document):
        """Returns a list of words contained in the document after converting  it to lowercase and striping the punctuation marks"""

        ps = PorterStemmer()
        terms = []
        for i in document:
            temp = i.lower().replace('vehicle', 'car').replace('automobile', 'car').split()
            for j in temp:
                terms.append(j)
        return [ps.stem(term.strip(string.punctuation)) for term in terms]

    def create_word_listing(self):
        """Function to create the word listing of the objects"""

        # Dictionary to hold the frequency of the words in word_list
        # with file_index as key
        self.listing_dict_ = {}

        for id in self.file_dict:
            temp_word_list = []
            response = requests.get(self.file_dict[id])
            soup = BeautifulSoup(response.text, 'html.parser')
            text = soup.find_all(text = True)
            text = list(filter(visible_text, text))
            terms = self.tokenize_document(text)
            for term in self.word_list[:500]:
                temp_word_list.append(terms.count(term.lower()))
            self.listing_dict_[id] = temp_word_list

        print('Word listing of each document')
        for id in self.listing_dict_:
            print('%d\t%s' % (id, self.listing_dict_[id]))
```

```
    def create_document_matrix(self):
        """Function to create the document matrix based on Manhattan Distance"""

        self.distance_matrix_ = []
        for id1 in self.file_dict:
            temp_list = []
            for id2 in self.file_dict:
                dist = 0
                for term1, term2 in zip(self.listing_dict_[id1], self.listing_dict_[id2]):
                    dist += abs(term1 - term2)
                temp_list.append(dist)
            self.distance_matrix_.append(temp_list)

        print('\nDistance Matrix')
        for i in self.distance_matrix_:
            print(i)

    def find_centroid(self, feature):
        """Function to find the centroid to which the document belongs"""

        distances = []
        for centroid in self.centroids_:
            dist = 0
            # print(type(self.centroids_[centroid]))
            # print(type(feature))
            for term1, term2 in zip(self.centroids_[centroid], feature):
                dist += abs(term1 - term2)
            distances.append(dist)

        return np.argmin(distances)

    def kmeans_clustering(self):
        """Function to perform k-means clustring of the documents based on   the k value passed during initialisation"""

        self.centroids_ = {}

        # Initialize the centroids with the first k documents as initial centroids
        for i in range(self.k):
            self.centroids_[i] = self.listing_dict_[i + 1]

        for i in range(500):
            self.classes_ = {}
            self.features_ = {}

            for i in range(self.k):
                self.classes_[i] = [i+1]
                self.features_[i] = [self.centroids_[i]]

            for id in self.listing_dict_:
                if id > self.k:
```

```
                classification = self.find_centroid(self.listing_dict_[id])
                self.classes_[classification].append(id)
                self.features_[classification].append(self.listing_dict_[id])
        previous = dict(self.centroids_)

        # Recalculate the cluster centroid based on the documents alloted
        for i in self.features_:
            self.centroids_[i] = np.average(self.features_[i], axis = 0)
        isOptimal = True

        for centroid in self.centroids_:
            original_centroid = np.array(previous[centroid])
            curr_centroid = self.centroids_[centroid]

            if np.sum(original_centroid - curr_centroid) != 0:
                isOptimal = False

        # Breaking the results if the centroids found are optimal
        if isOptimal:
            break

    def print_clusters(self):
        """Function to print the final clusters"""

        print('\nFinal Clusters')
        for i in self.classes_:
            print('%d:-->%s' % (i+1, self.classes_[i]))


# Dictionary containing the web_address and path
file_dict = {1: 'https://www.zigwheels.com/newcars/Tesla',
             2: 'https://www.financialexpress.com/auto/car-news/mahindra-to-launch-indias-first-electric-suv-in-2019-all-new-e-verito-sedan-on-cards/1266853/',
             3: 'https://en.wikipedia.org/wiki/Toyota_Prius',
             4: 'https://economictimes.indiatimes.com/industry/auto/auto-news/government-plans-new-policy-to-promote-electric-vehicles/articleshow/65237123.cms',
             5: 'https://indianexpress.com/article/india/india-news-india/demonetisation-hits-electric-vehicles-industry-society-of-manufacturers-of-electric-vehicles-4395104/',
             6: 'https://www.livemint.com/Politics/ySbMKTIC4MINsz1btccBJO/How-demonetisation-affected-the-Indian-economy-in-10-charts.html',
             7: 'https://www.hrblock.in/blog/impact-gst-automobile-industry-2/',
             8: 'https://inc42.com/buzz/electric-vehicles-this-week-centre-reduces-gst-on-lithium-ion-batteries-hyundai-to-launch-electric-suv-in-india-and-more/',
             9: 'https://www.youthkiawaaz.com/2017/12/impact-of-demonetisation-on-the-indian-economy/',
             10: 'https://indianexpress.com/article/india/demonetisation-effects-cash-crisis-mobile-wallets-internet-banking-4406005/',
             11: 'https://www.news18.com/news/business/how-gst-will-curb-tax-evasion-1446035.html',
             12: 'https://economictimes.indiatimes.com/small-biz/policy-trends/is-gst-helping-the-indian-economy-for-the-better/articleshow/65319874.cms'}
# List containig the words using which the vector space model is to be created
```

```
word_list = ['Tesla', 'Electric', 'Car', 'pollution', 'de-monetisation', 'GST' ,'black money']

# Creating class instance and calling appropriate functions
document_cluster = document_clustering(file_dict = file_dict, word_list = word_list, k = 4)
document_cluster.create_word_listing()
document_cluster.create_document_matrix()
document_cluster.kmeans_clustering()
document_cluster.print_clusters()
```

SOLUTION:

Word listing of each document

```
1    [16, 0, 24, 0, 0, 0, 0]
2    [0, 0, 12, 0, 0, 0, 0]
3    [0, 0, 94, 0, 0, 0, 0]
4    [1, 0, 13, 0, 0, 0, 0]
5    [0, 0, 9, 0, 0, 0, 0]
6    [0, 0, 1, 0, 0, 0, 0]
7    [0, 0, 22, 0, 0, 32, 0]
8    [0, 0, 21, 0, 0, 6, 0]
9    [0, 0, 0, 0, 0, 0, 0]
10   [0, 0, 0, 0, 0, 0, 0]
11   [0, 0, 0, 0, 0, 2, 0]
12   [0, 0, 0, 0, 0, 19, 0]
```

Distance Matrix

```
[0, 28, 86, 26, 31, 39, 50, 25, 40, 40, 42, 59]
[28, 0, 82, 2, 3, 11, 42, 15, 12, 12, 14, 31]
[86, 82, 0, 82, 85, 93, 104, 79, 94, 94, 96, 113]
[26, 2, 82, 0, 5, 13, 42, 15, 14, 14, 16, 33]
[31, 3, 85, 5, 0, 8, 45, 18, 9, 9, 11, 28]
[39, 11, 93, 13, 8, 0, 53, 26, 1, 1, 3, 20]
[50, 42, 104, 42, 45, 53, 0, 27, 54, 54, 52, 35]
[25, 15, 79, 15, 18, 26, 27, 0, 27, 27, 25, 34]
[40, 12, 94, 14, 9, 1, 54, 27, 0, 0, 2, 19]
[40, 12, 94, 14, 9, 1, 54, 27, 0, 0, 2, 19]
[42, 14, 96, 16, 11, 3, 52, 25, 2, 2, 0, 17]
```

[59, 31, 113, 33, 28, 20, 35, 34, 19, 19, 17, 0]

Final Clusters
1:-->[1]
2:-->[2, 7, 12]
3:-->[3]
4:-->[4, 5, 6, 8, 9, 10, 11]