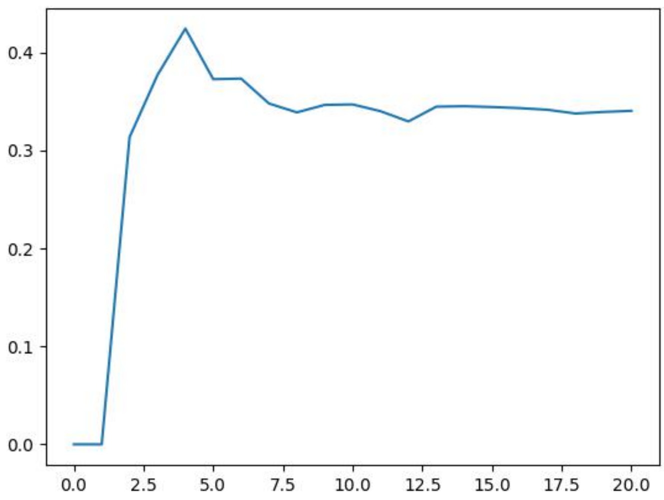
Data Science Lab

Lab#5 report

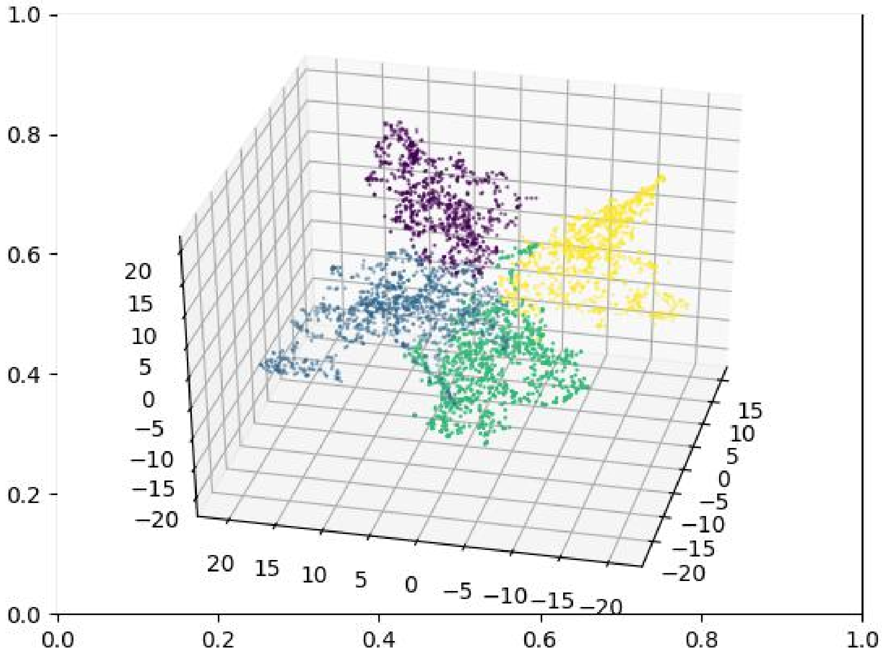
Newsgroups clustering

In order to load the dataset I used the os.litdir function to get the file names and I added them to a list in numerical order.
For what concerns the data preparation step, I decided to start from the basic TFIDF implementation provided. I used the suggested nltk function to remove stop words, then proceeded to try different values for the minimum and maximum threshold for document frequency.
I supposed the minimum frequency to be good at around 2/4000: a word needs to appear at least in two documents to be considered a valid sample. For the max_df a value of 0.2 was used: from that point on, words started to become too common and cluster quality decreased.
For what concerns dimensionality reduction I first tried to use the PCA algorithm via SVD decomposition as suggested, reducing the dimensions to 3 in order to visualize the clusters with matplotlib's 3D plotting functions later on.
Once i had my reduced TFIDF matrix I applied the kmeans algorithm to it, and i calculated the clusters labels for different values of K, ranging from 2 to 20. I also computed the silhouettes for those labels.
Plotting the silhouette values on a chart suggested that the right value for K was 4:



Silhouette values for different number of clusters

Uploading the labels for K = 4 on the submission platform gave me a score of about 20 points. Playing around with variables made the score go up to 32, but I never managed to go over that value just by using this technique.
Then I decided to use the TSNE algorithm for the dimensionality reduction. TSNE converts similarities between data points to a probability distribution, it defines another probability distribution over the low-dimensional map, and then tries to reduce the divergence between the two distributions.
This algorithm improves results dramatically. However, it's recommended to use truncatedSDV in order to reduce the number of dimensions up to a reasonable amount before applying TSNE. This is made to speed up the computation and to suppress some noise.
After applying the TSNE function the score on the leaderboard improved significantly, reaching a maximum of 78.67.
Below we can see the 3D representation of clusters after reducing to 3 dimensions with the help of the TSNE function:

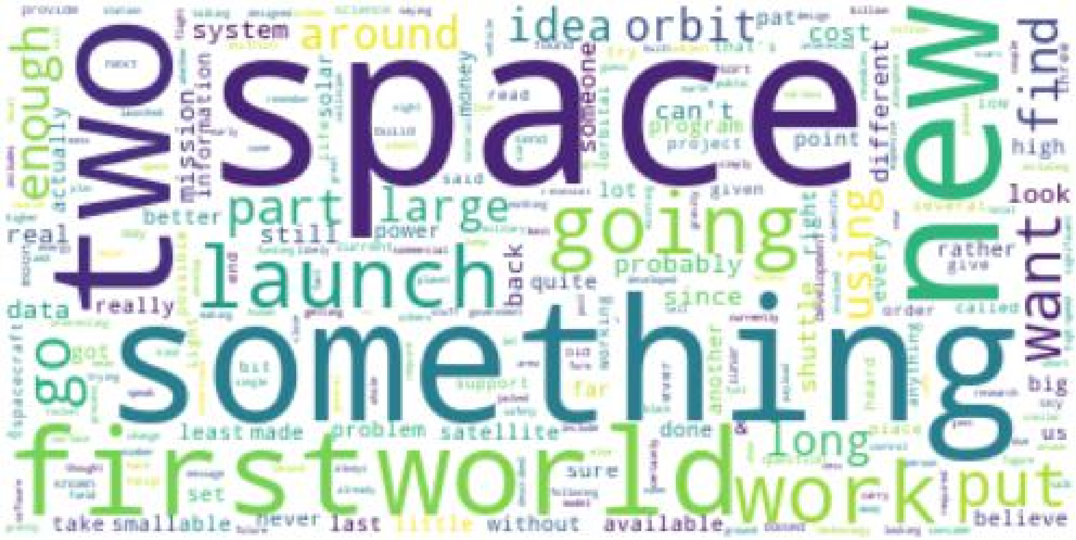


3d representation of clusters

Cluster characterization by means of word clouds

After splitting the data in 4 different chunks, I used the wordcloud library in order to visualize the most common words for each cluster. I discarded all the words that were not present in the TFIDF matrix, in order to avoid to see common words in the English language.
In these images we can see the results:

Cluster 1: “Space”

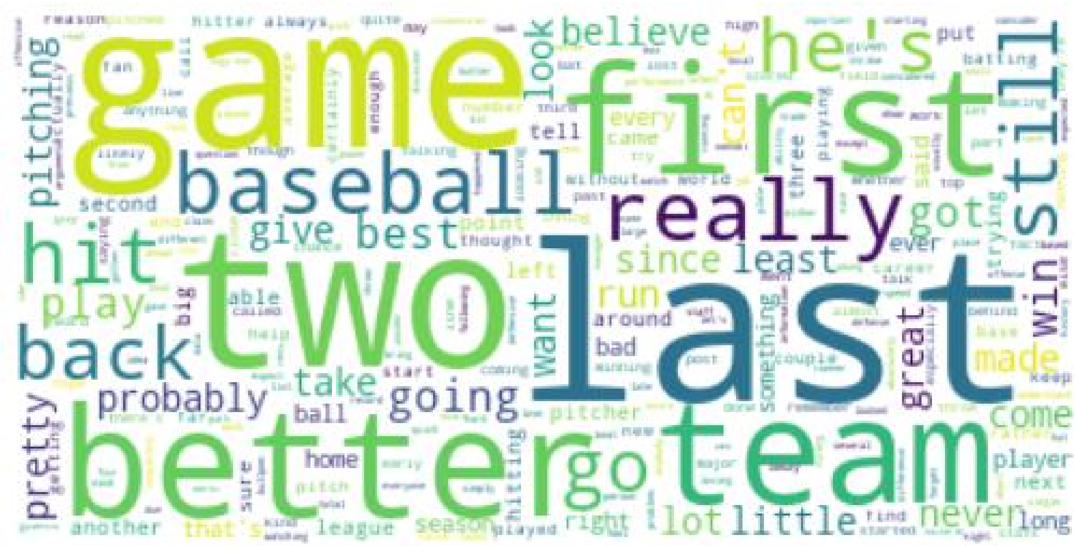


We can safely assume that the first cluster it’s composed of scientific articles, regarding astronomy and space exploration. We can see the words “space”, “launch”, “orbit”, “mission”, “system” and “satellite”.

Cluster 2: “Gun policies and laws”

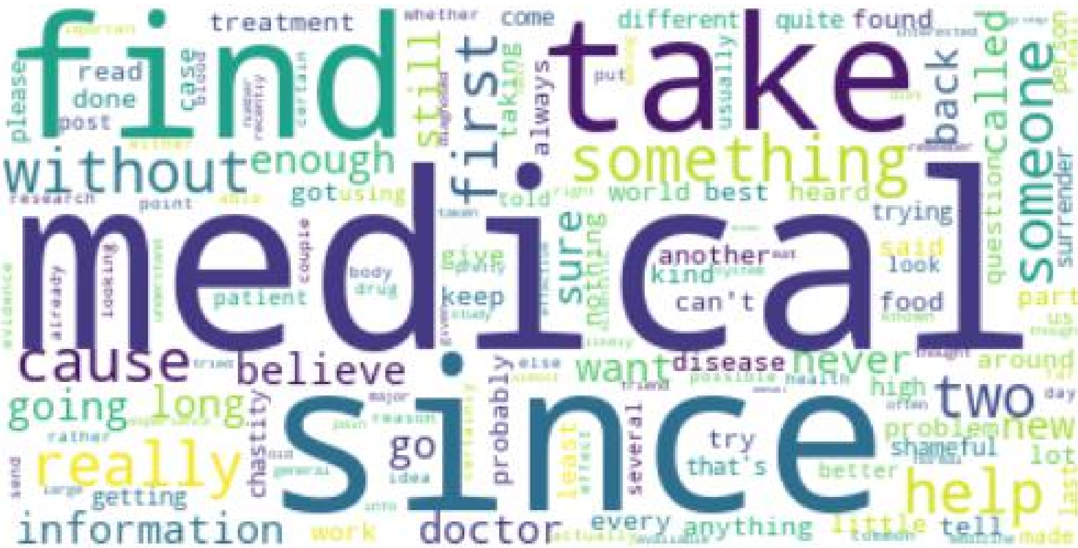
The second group of articles is probably about government policies and laws. There are a lot of articles about gun control, wich is a really debated topic in the US.

Cluster 3: “baseball”



We can easily see that the main topic of these articles is related to sports, in particular we can see a lot of terms related to basebell, like “hit”, “run”, “pitching” and “game”.

Cluster 4: “medicine”



We can see a lot of medicine related terms, we can find a lot of words like “medical”, “disease”, “treatment”, “doctor”...

PYTHON CODE:

```python
import os
import sys
import time
import csv
import re
import numpy as np
import nltk
import sklearn
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.tokenize import word_tokenize
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import stopwords as sw
from sklearn.cluster import KMeans
from sklearn.decomposition import TruncatedSVD
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.manifold import TSNE
import os
from wordcloud import WordCloud
import multidict as multidict


class LemmaTokenizer(object):
    def __init__(self):
        self.lemmatizer = WordNetLemmatizer()

    def __call__(self, documents):
        lemmas = []
        for t in word_tokenize(documents):
            t = t.strip()
            # Skip words containing numbers or unwanted characters
            if any((char.isdigit() or char=="." or char=="*") for char in t) == True:
                continue
            lemma = self.lemmatizer.lemmatize(t)
            lemmas.append(lemma)
        return lemmas


def dump_to_file(labels):
    # Dump the evaluated labels to a CSV file.
    with open("lab5_sample_submission.csv", "w+") as f:
        writer = csv.writer(f)
        writer.writerow(['Id', 'Predicted'])
        for index, a_label in enumerate(labels):
            writer.writerow([index, a_label])


def readfiles():
    # Reading files from T-newsgroups folder. Returns list of all articles as strings
    dataset = []
    filenames = os.listdir(path='T-newsgroups')
    filenames = [int(x) for x in filenames]  # Converting filenames to integers
```

```
    filenames.sort()  # Sorting filenames
    for i in range(len(filenames)):  # Appending files content to a
list
        with open('T-newsgroups/' + str(filenames[i]), "r") as f:
            content = f.read()
            dataset.append(content)
    return dataset


def tokenize(documents):
    stopwords = sw.words('english')
    stopwords.extend(
        ["'d", "'ll", "'re", "'s", "'ve", 'could', 'doe', 'ha', 'might',
'must', "n't", 'need', 'sha', 'wa', 'wo',
         'would'])
    lemmaTokenizer = LemmaTokenizer()
    vect = TfidfVectorizer(tokenizer=lemmaTokenizer,
encoding='utf-8', strip_accents='unicode', lowercase=True,
                           stop_words=stopwords,  min_df=0.0005,
max_df=0.2)
    tfidf = vect.fit_transform(documents)
    return tfidf, vect


def dim_red(tfidf):
    # Results improved significantly using TSNE dimensionality
reduction. However it's recommended to use truncatedSDV
    # in order to reduce the number of dimensions to a reasonable amount
before applying TSNE, this is made to speed up
    # the computation and to suppress some noise
    svd = TruncatedSVD(n_components=10, random_state=42)
    red_X_svd = svd.fit_transform(tfidf)  # red_X_svd will be:
np.array, shape (4000, 10)
    redx = TSNE(n_components=3).fit_transform(red_X_svd)  # red_X
will be: np.array, shape (4000, 3)
    return redx


def clusters_testing(max_clusters, matrix):
    # Computes clusters and silhouettes, starting from 2 centroids and
going up to the max_clusters value
    # It returns a list of the obtained clusters labels and silhouettes
    silhouettes = []
    clusters_labels = []
    # Since i start the computation from 2 clusters, i write zeros in
position 0 and 1 of the lists to ease debugging
    clusters_labels.extend([0, 0])
    silhouettes.extend([0, 0])
    for x in range(2, max_clusters):
        kmeans = KMeans(n_clusters=x,
max_iter=300).fit_predict(matrix)  # Kmeans calculations
        clusters_labels.append(kmeans)  # Appending found labels to
the list
        silhouettes.append(sklearn.metrics.silhouette_score(matrix,
kmeans, metric='euclidean'))  # appending silhouette
        # value to the list
    return clusters_labels, silhouettes
```

```
def print_clusters_and_silhouette_score(labels, matrix,
silhouettes):
    # Plot 3d cluster visualization of the given labels
    fig1, ax1 = plt.subplots()
    ax1 = fig1.add_subplot(111, projection='3d')
    ax1.scatter(matrix[:, 0], matrix[:, 1], zs=matrix[:, 2], s=0.5,
c=labels)
    # Plot silhouette score over the given range of possible cluster
numbers
    plt.figure()
    plt.plot(silhouettes)


def split_data(data, labels, n):
    # Split data based on cluster labels
    sd = [[] for x in range(n)]
    for i in range(len(data)):
        sd[labels[i]].append(data[i])
    return sd


def get_freq_dict_for_cluster(cluster):
    # Returns a dictionary representing the frequency of words in the
documents of the given cluster
    full_terms_dict = multidict.MultiDict()  # Creating a multidict
object to store word frequency
    tmp_dict = {}  # Creating temporary dictionary
    for d in range(len(cluster)):
        words = cluster[d].split()
        reduced_text = " ".join(sorted(set(words), key=words.index))
# Removing duplicate words in a document
        for text in reduced_text.split(" "):
            if text not in vectorizer.vocabulary_:  # Discarding all the
words that are not in the tfidf matrix.
                continue
            val = tmp_dict.get(text.lower(), 0)  # Getting the
frequency value of our word from the dictionary
            tmp_dict[text.lower()] = val + 1  # Increasing the frequency
value of the selected word by 1
    for key, value in tmp_dict.items():
        full_terms_dict.add(key, value)  # the dictionary keys
represent our words, while the  values represent the
        # number of documents that contain said word in them
    return full_terms_dict


def make_image(freq_dict):
    # Given in input the frequency dictionary, plots the word cloud
    wc = WordCloud(background_color="white", max_words=1000)
    wc.generate_from_frequencies(freq_dict)  # generate word cloud
    plt.imshow(wc, interpolation="bilinear")  # show
    plt.axis("off")
    plt.show()
```

```python
# MAIN PROGRAM:

news = readfiles()  # Creating a list with all documents

tfidf_X, vectorizer = tokenize(news)  # Tokenization and creation of the tf-idf matrix

red_X = dim_red(tfidf_X)  # Dimensionality reduction with truncatedSVD and TSNE

cl, sil = clusters_testing(21, red_X)  # Trying clusterization with different numbers of centroids

number_of_clusters = 4  # Cluster number that gives the highest silhouette

my_labels = cl[number_of_clusters]  # Final labels list

print_clusters_and_silhouette_score(my_labels, red_X, sil)  # Plotting 3D clusters and silhouette scores

dump_to_file(my_labels)  # Dump the evaluated labels to a CSV file.

splitted_data = split_data(news, my_labels, number_of_clusters)  # Splitting data based on found labels

for i in range(number_of_clusters):  # Plotting word clouds for each cluster
    plt.figure()
    freq_dictionary = get_freq_dict_for_cluster(splitted_data[i])
    make_image(freq_dictionary)
```