Project on text clustering: Unsupervised Learning

In this project, I will use the unsupervised learning on text data and perform text clustering by using different algorithms. I will use the Health-Tweets news dataset which can be found in the following link: https://archive.ics.uci.edu/ml/datasets/Health+News+in+Twitter. Once one downloads the zip file, one can see that there are different text files containing health news tweets. I choose the "bbchealth.txt" file in my analysis below.

Objective of the analysis

The main objective of my analysis is to use the health news tweets and cluster them to see the most common trend topics. So, the main analysis is focused on **text clustering**. I will use the BBC health news tweets but one can use other news agency tweets as well such CNN or Bloomberg.

Brief description of the dataset and summary attributes

The first step in my analysis is to use to import some of the main Python libraries that will be used extensively below. They are the following:

```
In [ ]:
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(context="notebook")
```

Now I specify the path of the file on my computer to import it as a Pandas dataframe

```
In [ ]:
PATH = "/..../bbchealth.txt"

df = pd.read_csv(PATH, delimiter="|", header=None)

df
```

	0	1	2
0	585978391360221184	Thu Apr 09 01:31:50 +0000 2015	Breast cancer risk test devised http://bbc.in/...
1	585947808772960257	Wed Apr 08 23:30:18 +0000 2015	GP workload harming care - BMA poll http://bbc...
2	585947807816650752	Wed Apr 08 23:30:18 +0000 2015	Short people's 'heart risk greater' http://bbc...
3	585866060991078401	Wed Apr 08 18:05:28 +0000 2015	New approach against HIV 'promising' http://bb...
4	585794106170839041	Wed Apr 08 13:19:33 +0000 2015	Coalition 'undermined NHS' - doctors http://bb...
...	...	...	...
3924	384766023120871424	Mon Sep 30 19:45:43 +0000 2013	Baby born after ovaries 'reawakened' http://bb...
3925	384700230920175617	Mon Sep 30 15:24:17 +0000 2013	Identical triplets born against odds http://bb...
3926	384678543088562178	Mon Sep 30 13:58:06 +0000 2013	Hospital failed to make improvements http://bb...
3927	384678542455222273	Mon Sep 30 13:58:06 +0000 2013	New patient targets pledge for NHS http://bbc....
3928	384569546108964864	Mon Sep 30 06:44:59 +0000 2013	C. diff 'manslaughter' inquiry call http://bbc...

3929 rows × 3 columns

Fig. 1. The BBC health tweet news dataset as a Pandas dataframe is shown. The dataset has only three columns.

As one can see from Fig. 1, the dataset has three main columns and 3929 rows. The column names have not been yet spacified what the are but one can easily understand their nature since now. The following Pandas command gives important information about the dataframe in Fig. 1

```
In [ ]:
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3929 entries, 0 to 3928
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   0       3929 non-null   int64 
 1   1       3929 non-null   object
 2   2       3929 non-null   object
dtypes: int64(1), object(2)
memory usage: 92.2+ KB
```

Fig. 2. Summary information about the dataset.

As one can see in Fig.2, the dataset has three columns which two are of 'object' type while one is of 'int64' type. The dataset has no null values. The first column contains only unique values, the second column is a date-time type column which gives the time stamp of each tweet, and the third column contains text elements. To see for example the the first column has only unique values, one can run the following Python command:

```
In [ ]:
df[0].value_counts().sum()
```

3929

Brief summary EDA, data cleaning and feature engineering

In this section, I will perform some data cleaning and feature engineering in order to get a version of the dataset ready for clustering. This first step is to rename the columns and give them descriptive names. As already mentioned above, the first column has only unique values and in reality each value is a tweet ID. The second column is a the tweet date and time, and the third column is the tweet text. The columns can be renamed with the following code:

```
In [ ]:
df.columns=["TweetID", "TweetDate", "TweetText"]

df
```

	TweetID	TweetDate	TweetText
0	585978391360221184	Thu Apr 09 01:31:50 +0000 2015	Breast cancer risk test devised http://bbc.in/...
1	585947808772960257	Wed Apr 08 23:30:18 +0000 2015	GP workload harming care - BMA poll http://bbc...
2	585947807816650752	Wed Apr 08 23:30:18 +0000 2015	Short people's 'heart risk greater' http://bbc...
3	585866060991078401	Wed Apr 08 18:05:28 +0000 2015	New approach against HIV 'promising' http://bb...
4	585794106170839041	Wed Apr 08 13:19:33 +0000 2015	Coalition 'undermined NHS' - doctors http://bb...
...	...	...	...
3924	384766023120871424	Mon Sep 30 19:45:43 +0000 2013	Baby born after ovaries 'reawakened' http://bb...
3925	384700230920175617	Mon Sep 30 15:24:17 +0000 2013	Identical triplets born against odds http://bb...
3926	384678543088562178	Mon Sep 30 13:58:06 +0000 2013	Hospital failed to make improvements http://bb...
3927	384678542455222273	Mon Sep 30 13:58:06 +0000 2013	New patient targets pledge for NHS http://bbc....
3928	384569546108964864	Mon Sep 30 06:44:59 +0000 2013	C. diff 'manslaughter' inquiry call http://bbc...

3929 rows × 3 columns

Fig. 3. Renamed dataframe columns.

Now that each column has been renamed with an appropriate descriptive name, I focus my attention on the **TweetText** column since it is this column that contains the text of each tweet. The other columns are irrelevant for the purpose of my analysis and will not be considered. For example, one can see what type of text has the first row by running the following command:

```
In [ ]:
df.TweetText[0]
```

'Breast cancer risk test devised http://bbc.in/1CimpJF'

As one can see from the first row tweet text, there is a text part of the tweet and the link related to that tweet. For the purpose of text clustering the link part text is unnecessary and it must be removed. To remove the text associated to the link part in each row, it is necessary to use a regular expression to remove the link text and replace it with white space. This can be achieved by using the following code:

```
In [ ]:
df["TweetText"].replace("http://.*", regex=True, value="", inplace=True)

df.TweetText   # Display the tweet after link text removal
```

```
0             Breast cancer risk test devised
1         GP workload harming care - BMA poll
2         Short people's 'heart risk greater'
3        New approach against HIV 'promising'
4        Coalition 'undermined NHS' - doctors
                         ...                 
3924     Baby born after ovaries 'reawakened'
3925     Identical triplets born against odds
3926     Hospital failed to make improvements
3927       New patient targets pledge for NHS
3928      C. diff 'manslaughter' inquiry call
Name: TweetText, Length: 3929, dtype: object
```

Fig. 4. The TweetText column with the link text removed.

As one can see from Fig. 4, the text associated to the tweet link has been completely removed in each row. Now the next step is to perform some feature extraction from the **TweetText** column and transform it to a readable sklearn object. Indeed, now one needs to transform the text in each row into a numerical feature and this can be done in different ways. Here I use the **TfidfVectorizer** method which takes text features and transform them into matrices.

The first step toward feature extraction and engineering is to import the **TfidfVectorizer** module and initiate it

```
In [ ]:
from sklearn.feature_extraction.text import TfidfVectorizer

tf = TfidfVectorizer(stop_words="english")
tf_transform = tf.fit_transform(df.TweetText)
```

One can type "**tf?**" to see what are the parameters that **TfidfVectorizer** takes and what are its attributes. In short, **TfidfVectorizer** transforms the text into lowercase, then it tokenizes it, then it uses the stop_words option to remove common words of the English language and after it transforms the text into a normalized numerical feature. By applying **TfidfVectorizer** to the TweetText, one gets the following dataframe for the TweetText column:

```
pd.DataFrame(tf_transform.toarray(), columns=tf.get_feature_names_out())
```

	00	000	10	100	10m	11	111	113	12	13	...	young	youngest	youth	youtube	yuk	zap	zeneca	zero	zone	zones
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3924	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3925	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3926	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3927	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3928	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

3929 rows × 4259 columns

Fig. 5. Feature extraction and engineering of the TweetText column by using TfidfVectorizer.

As one can see from Fig. 5, the TweetText column has been transformed into a multi-column dataframe with 4259 columns. This is a consequence of tokenization of a text feature where for each word relevant text word creates a numerical feature. So, from a one column and 3929 rows dataframe now we have 4259 columns and 3929 rows. Each feature is normalized by default by using the *L2* metric.

Summary of training at least three unsupervised learning models

Now that the TweetText feature has been engineered, it is the time to model each of the features in Fig. 5 and obtain predictions by using at least 3 different clustering techniques or hyperparameters. Here we do not have a target variable like in supervised learning and so it is not necessary to split the data for train and test purposes. **In what follows below, I will use only the KMeans clustering method and its variations and hyper-parameters since this method is more adapted for text clustering**.

K-Means clustering

In this section, I use the K-Means clustering method to perform text clustering. To use this method, it is more elegant and compact create a pipeline to do the necessary steps toward text clustering. So, I import the following modules

```
In [ ]:
from sklearn.cluster import KMeans
from sklearn.pipeline import Pipeline
```

auto and full algorithms

In this section, I use the **auto and full algorithms** of KMeans. In addition, I also use different initialization methods as well for these algorithms. Before implementing the Pipeline, it is important to establish which is the right number k of clustering the data. Here I will use the Elbow method as an example and plot the inertia vs. the cluster number k. The following code gives the desired results:

```
In [ ]:
fig, axs = plt.subplots(2, 2, figsize=(20, 13))
fig.subplots_adjust(hspace=.8, wspace=.7)
algs = ["auto", "full"]
for alg, i in zip(algs, range(2)):
    intls = ["k-means++", "random"]
    for intl, j in zip(intls, range(2)):
        inertia_score = []
        K = list(range(2, 20))
        for k in K:
            km = KMeans(init=intl, n_clusters=k, random_state=0, algorithm=alg).fit(X)
            inertia_score.append(km.inertia_)
        axs[i, j].plot(K, inertia_score, marker="s")
        axs[i, j].set_xlabel("Number of clusters $k$")
        axs[i, j].set_ylabel("Inertia")
        axs[i, j].set_title(f"Algorithm {alg} and initialialization {intl}")
fig.suptitle('Elbow method for different algorithms and initializations', fontweight='bold')
fig.tight_layout()
plt.show()
```

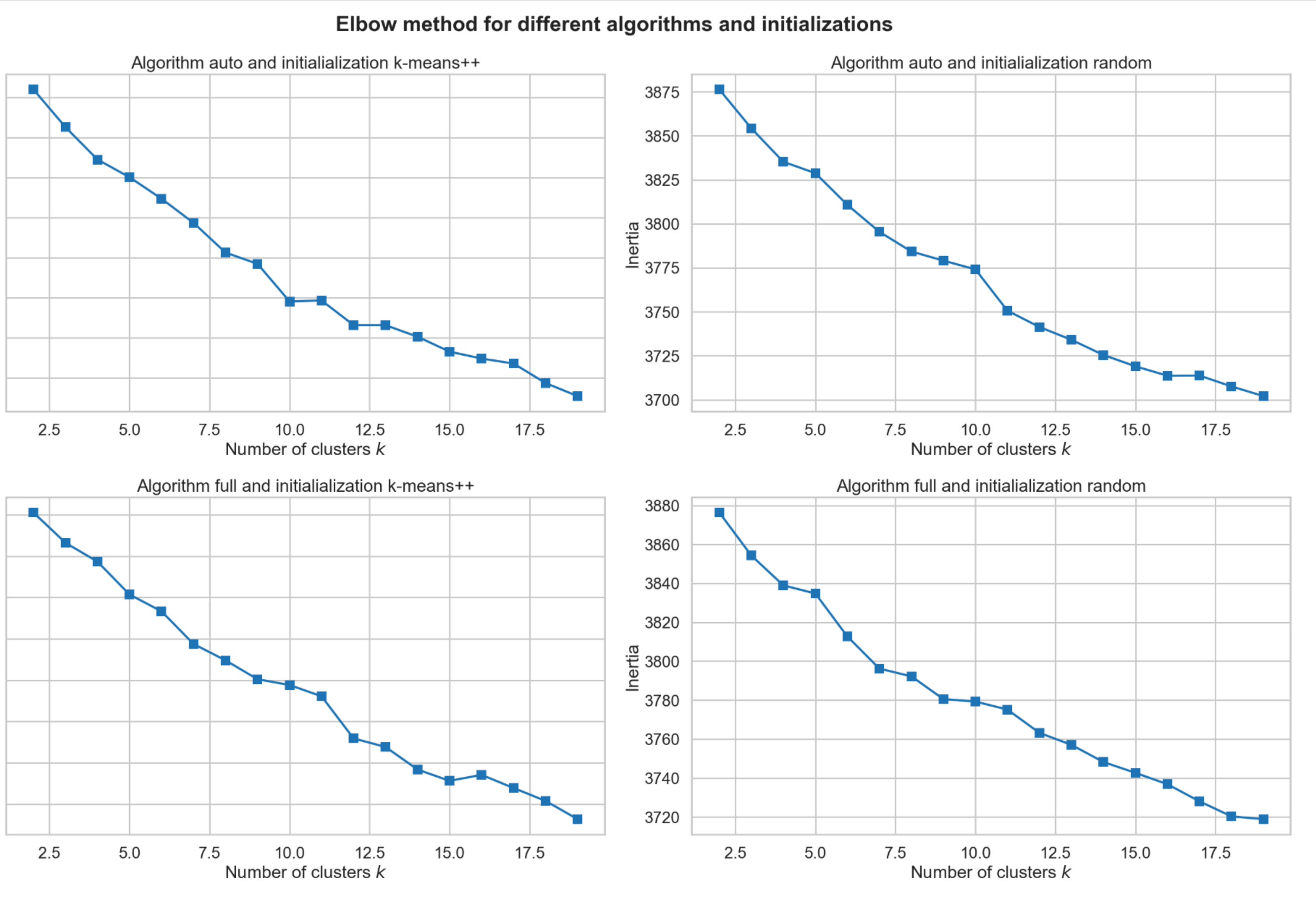

Fig. 6. The inertia versus the cluster number k is shown.

The Elbow method is a good method to establish the cluster number that gives the best compromise between model simplicity and complexity. The key point here is to find where is the elbow located in each plot in Fig. 6 and calculate the inertia for each elbow k. When all pairs of k and inertia have been collected, one needs to choose the that cluster number with the smallest inertia value.

By running a specific Python code which do not show here explicitly and which involves for loops, the values of k and inertia pairs are:

```
[(10, 3747.775545779579),
 (8, 3784.273755652402),
 (9, 3775.5263732080557),
 (9, 3780.5705548972883)]
```

Fig. 7. Values of inertia vs. k for the algorithms and initializations of Fig. 6.

As one can see from Fig. 7 the lowest value of inertia for the optimal value of k is **inertia=3747.77 for** $k = 10$. Therefore, I choose $k = 10$ to proceed in what follows since this value gives the lowest value of inertia. These value pairs are obtained for the algorithm **auto** and initialization **k-means++** which essentially correspond to the first plot on the upper left hand side of Fig. 6. Indeed, it is visually evident the location of the elbow in the first plot on the upper left hand side of Fig. 6 at $k = 10$.

Now I choose the algorithms that give the best value of inertia for the optimal k and I build the following pipeline:

```
In [ ]:
k = 10

pipeline = Pipeline([('feature_extraction', tf), ('clusterer', KMeans(init="k-means++",
n_clusters=k, random_state=0, algorithm="auto"))])

pipeline.fit(df["TweetText"])
labels = pipeline.predict(df["TweetText"])  # Cluster lables ranging from 0 to 9.
```

Now I run a code to display the number of word samples for each cluster where the number of clusters is set to $k = 10$ from the Elbow method used above. The code is the following:

```
In [ ]:
k = 10
from collections import Counter

c = Counter(labels)
for cluster_number in range(k):
    print(f"Clusters {cluster_number} contains {c[cluster_number]} samples")
```

```
Clusters 0 contains 2037 samples
Clusters 1 contains 156 samples
Clusters 2 contains 101 samples
Clusters 3 contains 137 samples
Clusters 4 contains 341 samples
Clusters 5 contains 528 samples
Clusters 6 contains 328 samples
Clusters 7 contains 59 samples
Clusters 8 contains 89 samples
Clusters 9 contains 153 samples
```

Fig. 8. The number of word samples for each cluster. The first cluster has the label equal to 0.

It is also possible to show the most common words for each cluster and their relative score. The code to show the plot is quite long and I do not show it here but only the final result. By considering **the five most common words** for each cluster, I get the following words(features) for each cluster:

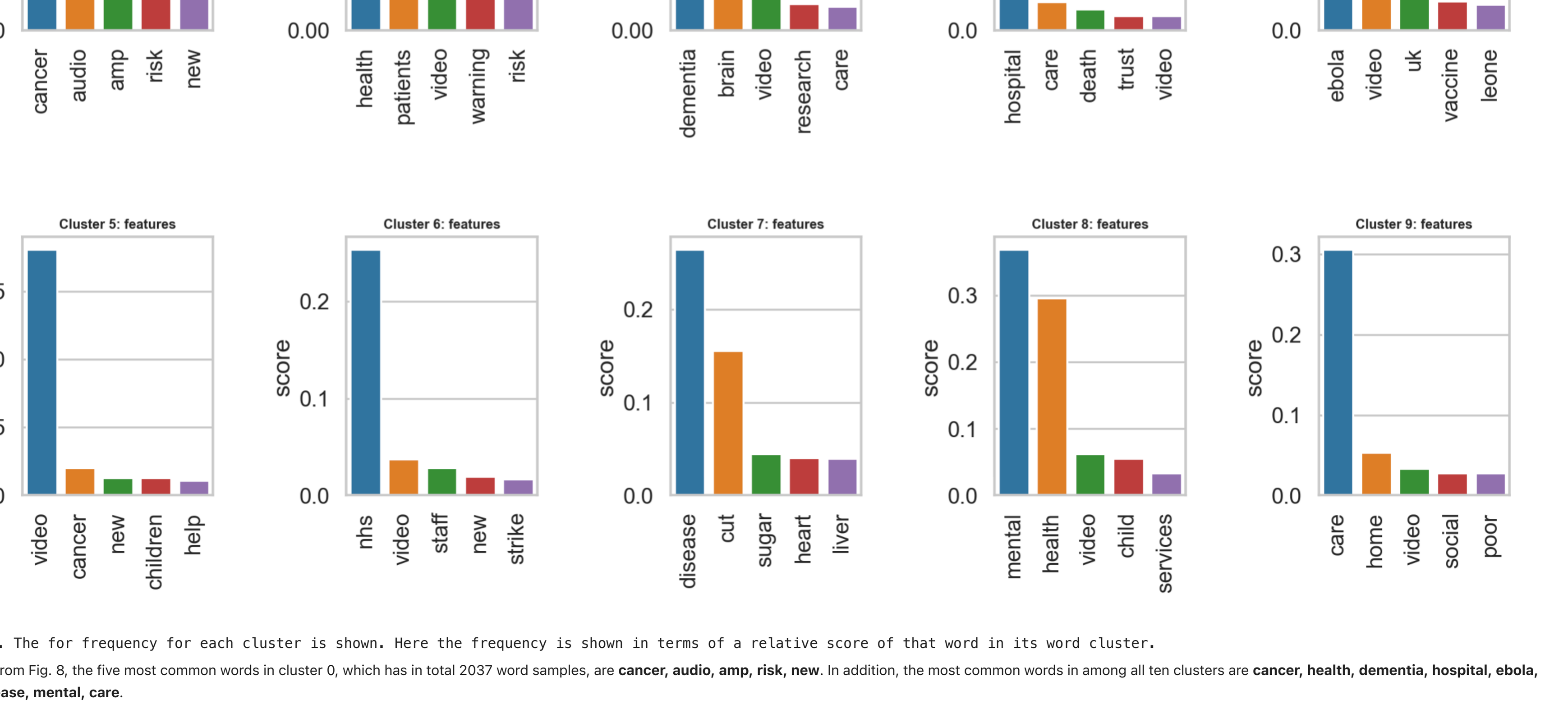

Fig.9. The for frequency for each cluster is shown. Here the frequency is shown in terms of a relative score of that word in its word cluster.

As one can see from Fig. 8, the five most common words in cluster 0, which has in total 2037 word samples, are **cancer, audio, amp, risk, new**. In addition, the most common words in among all ten clusters are **cancer, health, dementia, hospital, ebola, video, nhs, disease, mental, care**.

Best clustering model selection

From the considerations that I have showed above, where I used several KMeans algorithms and initializations, the best model that I choose is for the algorithm **auto** which is equivalently called the Elkan algorithm and initialization **k-means++**. It is this combination that give the optimal value of cluster number with the smallest value of inertia.

Key findings and next steps

The key findings of this project have been to cluster the most common tweet keywords of BBC health news articles. As I have shown in Fig. 9, the first cluster with 2037 samples, the largest one, has as main keyword **cancer**. This is something that one would expect because cancer is the most prevalent and deadliest diseases that exists and it usually dominates the health news titles.

The second largest cluster is cluster number five where the most common word is **video**. Even this keyword is to be expected to dominate the news headlines because very often in these tweet headlines there is a link to a video and the word video is often associated with it. So, my Unsupervised text classification has given me the most common words that dominated the BBC tweet headlines at a particular time period in the past. The third most common word is **ebola** which is located in cluster 4.

The next step for the analysis of this project would be to use other clustering methods such as DBSCAN, Mean shift, etc. and compare their results with the KMeans results obtained in this study.