Computation Tools for Data Science

Bjørn Hansen[1], Zuzia Rowinska[1], Jakub Solis[1], and Maria Kalimantzali[1]

[1]Department of Applied Mathematics and Computer Science, Technical University of Denmark, Kgs. Lyngby, Denmark

1 Introduction

Text classification is a common objective within the field of natural language processing with a wide variety of applications. With the growing size of digital libraries, Wikipedia articles and even posts/tweets on social media, the need for a fast and effective approach to text classification is more relevant than ever. Classifying threats and hate speech, categorizing books by genre or articles by topic is no longer a feat for humans due to the sheer volume. This is where computer science and machine learning enters the picture.

The main objective of our project is to perform classification on a text-corpus solely according to the word content. The corpus we have investigated is the BBC News data set[3] which consists of articles from 5 different categories. We will perform unsupervised clustering and compare the results to supervised classification. This will be achieved using the two unsupervised algorithms K-means and agglomerate clustering, and the two supervised algorithms K-Nearest Neighbours and a dense neural Network. The performance will be evaluated on metrics such as accuracy and Rand-index[5]. We will also discuss some of the notable outliers in our analysis and elaborate on why these might have occurred.

Note that we will use the term "document" and "article" interchangeably in this report.

2 Materials and Methods

2.1 Data

The data set used is the public BBC News data set[3] consisting of 2225 articles, each labeled as one of the five categories: `business`, `entertainment`, `politics`, `sport` or `tech`. The articles in the data set contains at maximum 4757 words, a minimum of 323 words and have an average of 412 words. The distribution of article word count and category is visualized in Fig. 1. All stopwords in the english `nltk corpus` module have been removed from the articles to avoid redundancy and prevent potential representational issue with the word vectorization described below.

2.2 Word vectorization

The discrete representation of words by letters is not useful when it comes to utilizing a computer, and thus a continuous representation is needed. A common approach to word vectorization is the `term-frequency times inverse document-frequency` (TF-IDF). For a term t in document d, the TF-IDF is given by:

$$\text{TF-IDF}(t,d) = \text{TF}(t,d) \cdot \text{IDF}(t) \tag{1}$$

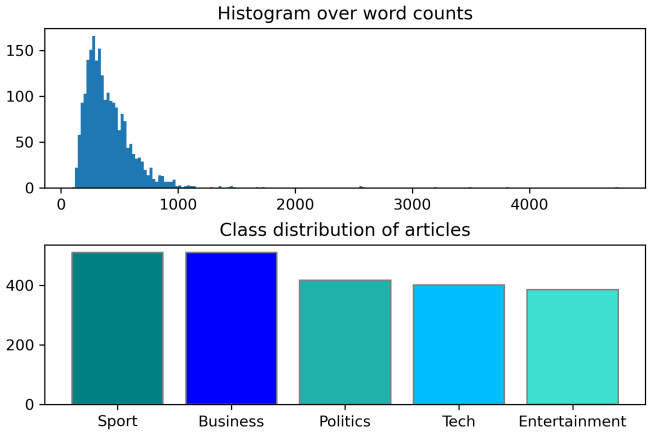


Figure 1: Top: Histogram over number of words per article. Bottom: Bar plot of class distribution

Here $\text{TF}(t,d)$ is the term frequency for term t in document d and the inverse document frequency is given by:

$$\text{IDF}(t) = \log(\frac{n}{\text{DF}}) + 1 \tag{2}$$

Where n is the total amount of documents and DF is the document frequency. We note that the IDF in Eq. (2) differs slightly to the notation used by Leskovec et al. [4, chap 1, p. 10] as they do not include the " + 1". This term however, ensures that even words occurring in every single document will not be entirely removed. We argue later that this does not matter when selecting our optimal hyperparameters for the vectorization.

The idea behind the inverse document frequency is to scale down the impact of word-tokens that occur frequently in a given corpus since they provide less empirically information in a classification-setting. This is done by utilizing a "one-hot encoding"-like scheme with frequencies for all the words in a corpus. This means the matrix representing the word encoding for all the documents is of size 2225 $\times$ No. unique words. Essentially each row represents a document, each column a specific word and the value of the entry is given by the TF-IDF.

2.3 Clustering

Clustering is a class of unsupervised machine learning methods that groups the data. The methods group the data points based on metrics such as similarity or distance.

Consider a data set with 2-dimensional scalar attributes: $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, where $n \in \mathbb{N}$ is the number of observations. When visualized as a scatter plot, the data set should form clusters if the magnitude of the attributes significantly differ across groups but not within them.

The criterion used for assigning a data point to a clusters varies between different clustering algorithms. Below we outline the approach of the two we have used in our project, name K-means and agglomerative clustering.

2.3.1 K-means algorithm

One of the most common clustering methods is the K-means algorithm. The algorithm follows Algorithm 1 outlined below.

Algorithm 1 K-means algorithm

Require: $\mathcal{D}$: Data set of size $n \times m$
Require: K: Number of clusters
Require: δ: Maximum number of iterations
1: **procedure** K-MEANS($\mathcal{D}, K, \delta$)
2: Initialize cluster centroids $c_1, c_2, \ldots, c_K \in \mathbb{R}^m$
3: **for** ite $\leq \delta$ **do**
4: Set: $c_{old} \leftarrow [c_1, c_2, \ldots, c_K]$
5: For $i \in n$: $\mu^{(i)} \leftarrow \arg\min_j \left\| x^{(i)} - c_j \right\|^2$
6: Update centroids: $c_j \leftarrow \frac{\sum_{i=1}^{n} 1\{\mu^{(i)}=j\}\mathcal{D}^{(i)}}{\sum_{i=1}^{n} 1\{\mu^{(i)}=j\}}$
7: **if** $c_{old} = c$ **then**
8: **break**

The K in K-means indicates how many centroids the algorithm fits. It is noted that if convergence is achieved, i.e. $c_{old} = c$ then the algorithm stops before the maximum number of iterations is reached. This is only the case if none of the n data points are assigned to a new cluster durring an iteration. We also note that K-means with random centroid initialization returns clusters with great variation in quality. More advance version of the algorithm such as the one by Arthur and Vassilvitskii [2] determines the inertia of the point cloud $\mathcal{D}$ and applies an optimized initialization.

2.3.2 Agglomerative clustering

Agglomerative clustering is a hierarchical clustering method based on building a tree structure from data points, called dendrograms. Agglomerative algorithms are widely applied for clustering different types of inputs, such as numerical, categorical, and text data. [1]

To find clusters with the agglomerative method, the dataset of n points is divided into n clusters, where each point is assigned to a single cluster. In the next steps, clusters are merged, based on distance or similarity. Merging is repeated until all of the points are assigned to one cluster, or the process is stopped when the given number of clusters is found. The algorithm and its results can vary, based on a chosen distance or similarity measure. Single and complete linkage techniques are one of the simplest and most often used. Distance between clusters is defined as the distance between the two closest data points for a single linkage, or the two furthest points for a complete linkage. [6]

We used an agglomerative algorithm with Ward's linkage method, with euclidean distance, to find the clustering of the BBC articles. In Ward's technique clusters are merged based on the criterion of minimum variance.

2.3.3 Clustering Metrics

The quality of a clustering can be assessed with a wide range of metrics and scores. The most simple however is the Rand-index (RI) given by

$$\text{RI} = \frac{\text{No. Agreeing Pairs}}{\text{No. Pairs}} = \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}} \tag{3}$$

The RI is frequently used in unsupervised tasks where the ground truth labels are known, which coincide with our assumptions about the clusters. The RI computes a similarity between pairs of data points and is thus very similar to accuracy in a binary setting. The main discrepancy between accuracy and Rand comes from the invariant property of Rand. Accuracy is sensitive to the ground truth label whereas Rand is a measure of "homogeneity" in the clusters. This means Rand solely requires information about whether a pair of points belongs to the same cluster whereas accuracy requires information if the points are assigned to the cluster corresponding to the correct class. From the definition in Eq. (3) it is clear that the output values ranges from 0 to 1, which corresponds to a perfect clustering.

3 Experiments and Results

Several experiments with varying hyperparameters for the TF-IDF word vectorization were conducted. The best performing vectorization with regards to K-means clustering was achieved with a sublinear TF i.e. TF is replaced by 1+log TF, a maximum document frequency of 0.95 and a minimum document frequency of 5. A maximum document frequency of 0.95 means that a term can at most be present in 95% of documents, and a minimum document frequency of 5 means that the term have to be present in at least 5 documents. The total number of terms satisfying the cut-off frequency thresholds amounted to 8984 for all 2225 articles which means each article in the data set is represented by a vector $v \in \mathbb{R}_+^{8984}$ constrained by $\|v\|_2 = 1$.

We will in the following sections interpret the clusters obtained by K-means as representations of the different article classes. We find the class of the K-means clusters by looking at the 10 most frequent keywords. From the keywords, it is immediately clear what class the clusters represent. This interpretation is obviously wrong since K-means is an unsupervised learning algorithm, however it makes the analysis more convenient. Additionally, we will argue that this assumption is not too far from reality further down.

3.1 Evaluation of clustering

To evaluate and compare the performance of K-means and agglomerative clustering we use the Rand metric described in Section 2.3.3. The agglomerative clustering achieved a $RI = 0.924$ whereas K-means achieved $RI = 0.965$. This means that the K-means algorithm achieves a higher level of homogeneity within each cluster than the agglomerative algorithm when compared to the ground truth. The clustering results are visualized in Figs. 3 and 4.

Of the 2225 articles in the data set only 100 were misclassified by the K-means algorithm, and 220 by the agglomerative clustering algorithm. Notably, the business cluster contained 62% and 70% of the incorrectly classified articles respectively for K-means and agglomerative algorithm. Looking at Fig. 5 it is clear that articles from the politics and tech category are by far the most often misclassified when using K-means, compared to politics, entertainment and tech when using the agglomerative method.

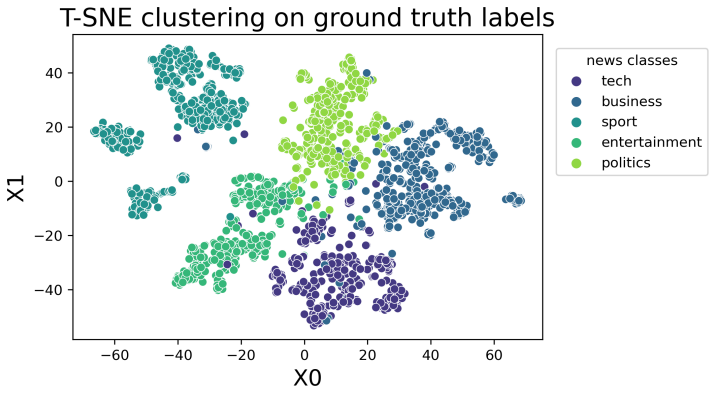


Figure 2: 2-Dimensional T-SNE projection applied on the data projected onto first 50 principal components. Classes are based on the ground truth

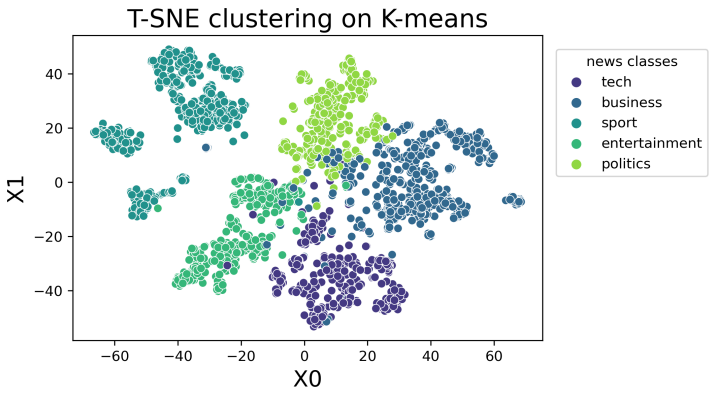


Figure 3: 2-Dimensional T-SNE projection applied on the data projected onto first 50 principal components. Classes are based on K-means clusters

When inspecting the word content of the 62 articles wrongly classified as a business using the K-means algorithm, the reason becomes somewhat apparent. Among the top 10 keywords from the articles are "Law", "EU", "US", "Year", "Government", "New" and "Committee". One could argue that most of these keywords semantically could belong to either the politics, tech or business class when considered in an isolated context.

3.2 Supervised methods

In addition to the K-means and agglomerative clustering we trained and tested a K-nearest neighbours classifier (KNN) and a dense neural network (NN) on our data. In contradiction to the clustering methods, the KNN and NN are supervised methods, meaning they have access to and learns from the ground truth. For both the KNN and NN we split the data 80/20 into respectively train and test sets. The split was done randomly due to the uniformity of the class distribution and approximate normality of word counts seen in Fig. 1. It is notable to mention that the data was projected onto the first 300 principal components for the NN as having an input layer with almost 9000 neurons is excessive. The KNN was trained with 5 nearest neighbors and the 2-norm as distance metric. The NN was trained for 15 epochs on the entire training set, and had $\approx$ 63k parameters which likely could be reduced dramatically without significant loss of performance. The KNN and NN classifiers achieved an accuracy of respectively 76.2% and 98.4% on the test set. The classification results of the KNN is illustrated in Fig. 6 in the Appendix.

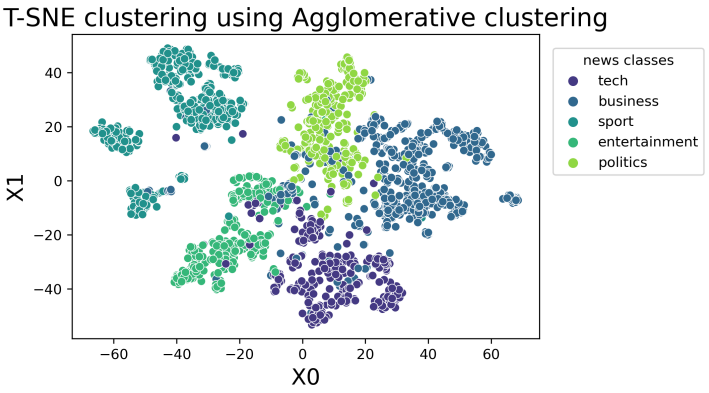


Figure 4: 2-Dimensional T-SNE projection applied on the data projected onto first 50 principal components. Classes are based on agglomerative clusters

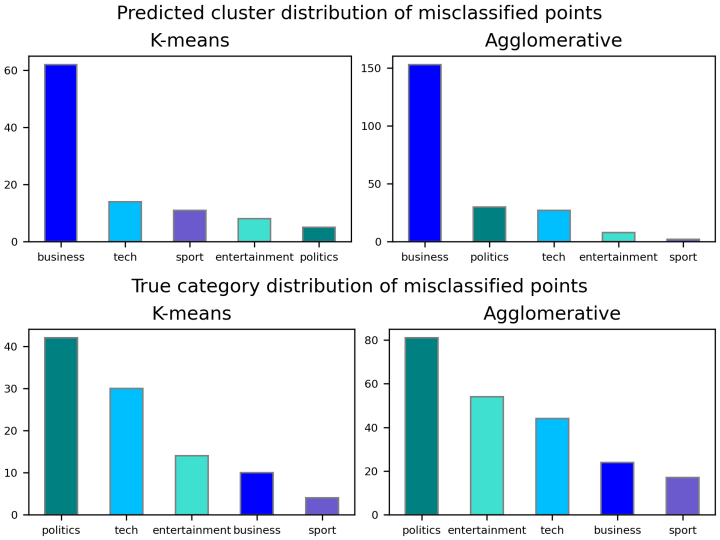


Figure 5: Distribution of ground truth category (bottom) and predicted class (top) of misclassified articles

4 Discussion

The performance of K-means and agglomerative clustering is quite similar when evaluated with the Rand-index. A $RI = 0.965$ for K-means compared to $RI = 0.924$ for agglomerative clustering is merely a performance difference of about 4.2%. This comes as no surprise since the RI is a measure of class-homogeneity within the clusters as previously mentioned, and we found the agglomerative cluster to have 220 misclassified articles compared to the 100 by K-means. This translate to a 5.4% point reduction in miss classification for K-means, which is similar to the results from RI.

Despite the assumption that the K-means and agglomerative clusters representing news classes is quite unorthodox we believe it in this case to be true to a high degree. Comparing the T-SNE projection of the ground truth data in Fig. 2 to the K-means and agglomerative clustering in Figs. 3 and 4 it becomes clear that the clusters actually do somewhat represent article classes. The correct interpretation of the clusters would be that the word-usage within each cluster is highly similar. However it is very apparent by the high performance

of both K-means and agglomerative clustering, that the word-usage of articles is deeply connected to the class. This should come as no surprise, as the content of articles logically depends on the topic.

When comparing the two supervised methods, KNN and NN, there is no doubt which one had better performance. The accuracy of the KNN amounted to 76.2% on the test set, while the NN scored a staggering 98.4% accuracy. This is quit surprising as the KNN was trained on the whole, high-dimensional data representation, whereas the NN was trained on the data projected onto the first 300 principal components which only accounted for 42.3% of the total variance.

Comparing unsupervised and supervised learning schemes are like comparing apples and oranges. Not only are the circumstances of their training widely different, but their evaluation is based on different metrics. Nonetheless the performance of K-means and the NN seems to be similar.

The TF-IDF word vectorization turned out to be a surprisingly strong representation of the articles. Despite the algorithm being essentially a one-hot encoding of keyword frequencies without any contextual information, it performed much better than early experiments with Word2Vec. A reason for this could be that the multi-layer-perceptrons in the Word2Vec network were not trained properly and thus the word-mapping gave us bad results.

We note that our own implementation of K-means at times produced results almost identical to the implementation in the `sklearn` library. However, it all came down to the initialization of the centroids. The `sklearn` implementation utilizes an optimized initialization-scheme based on point cloud inertia, and ours simply picked 5 random data points. This does not suffices if time and consistency are key values, and thus we chose to use the implementation in `sklearn`.

5 Conclusion

Our work shows that the TF-IDF word vectorization with proper hyperparameters in combination with either agglomerative clustering or K-means produces accurate clusters representing the true class of the articles in the BBC news data set.

The performance of the unsupervised clustering methods were quite identical, however a clear discrepancy in performance was observed among the supervised classification methods. The NN outperformed the KNN by a far margin and achieves the highest overall accuracy of all the methods.

To no surprise the state-of-the-art supervised classification method (NN) performed best on an inherently supervised task. To our surprise however, the performance of the K-means method came quite close. In data where clusters are naturally appearing, one should definitely consider applying K-means as a classification method rather than NN to avoid training time.

References

[1] C. C. Aggarwal and C. Zhai. *A Survey of Text Classification Algorithms*, pages 163–222. Springer US, Boston, MA, 2012. ISBN 978-1-4614-3223-4. doi: 10.1007/978-1-4614-3223-4_6. URL `https://doi.org/10.1007/978-1-4614-3223-4_6`.

[2] D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 1027–1035, USA, 2007. Society for Industrial and Applied Mathematics. ISBN 9780898716245.

[3] D. Greene and P. Cunningham. Practical solutions to the problem of diagonal dominance in kernel document clustering. In *Proc. 23rd International Conference on Machine learning (ICML'06)*, pages 377–384. ACM Press, 2006.

[4] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, USA, 2nd edition, 2014. ISBN 1107077230.

[5] W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971. doi: 10.1080/01621459.1971.10482356. URL `https://www.tandfonline.com/doi/abs/10.1080/01621459.1971.10482356`.

[6] R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005. doi: 10.1109/TNN.2005.845141.

A Appendix

A.1 Contribution to this project

Part	Bjørn Hansen	Zuzanna Rowinska	Jakub Solis	Maria Kalimantzali
Text vectorization and preprocessing	25%	0%	50%	25%
Clustering methods	25%	50%	0%	25%
Supervised methods	50%	0%	50%	0%
Results analysis	20%	50%	0%	30%
Report	25%	25%	25%	25%

A.2 Trained KNN classifier on test set

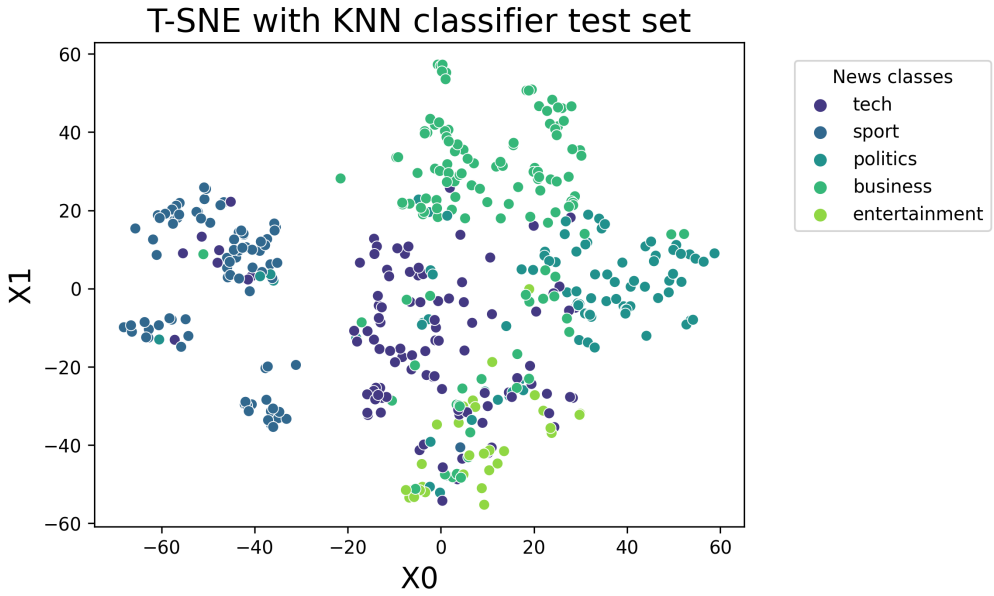


Figure 6: Visualization of KNN predictions on the T-SNE projection of the test set

A.3 Project file

All project files are available on GitHub:

Table of Contents

1 BBC Articles Clustering

In [1]:

```
# Libraries for the notebook

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from tqdm import tqdm
import re
import os

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.stem import PorterStemmer
import nltk

nltk.download('stopwords')
nltk.download('punkt')

from collections import Counter

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import DBSCAN
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn import metrics
from sklearn.cluster import AgglomerativeClustering
import scipy
from scipy.cluster import hierarchy

import time
```

...

2 Data

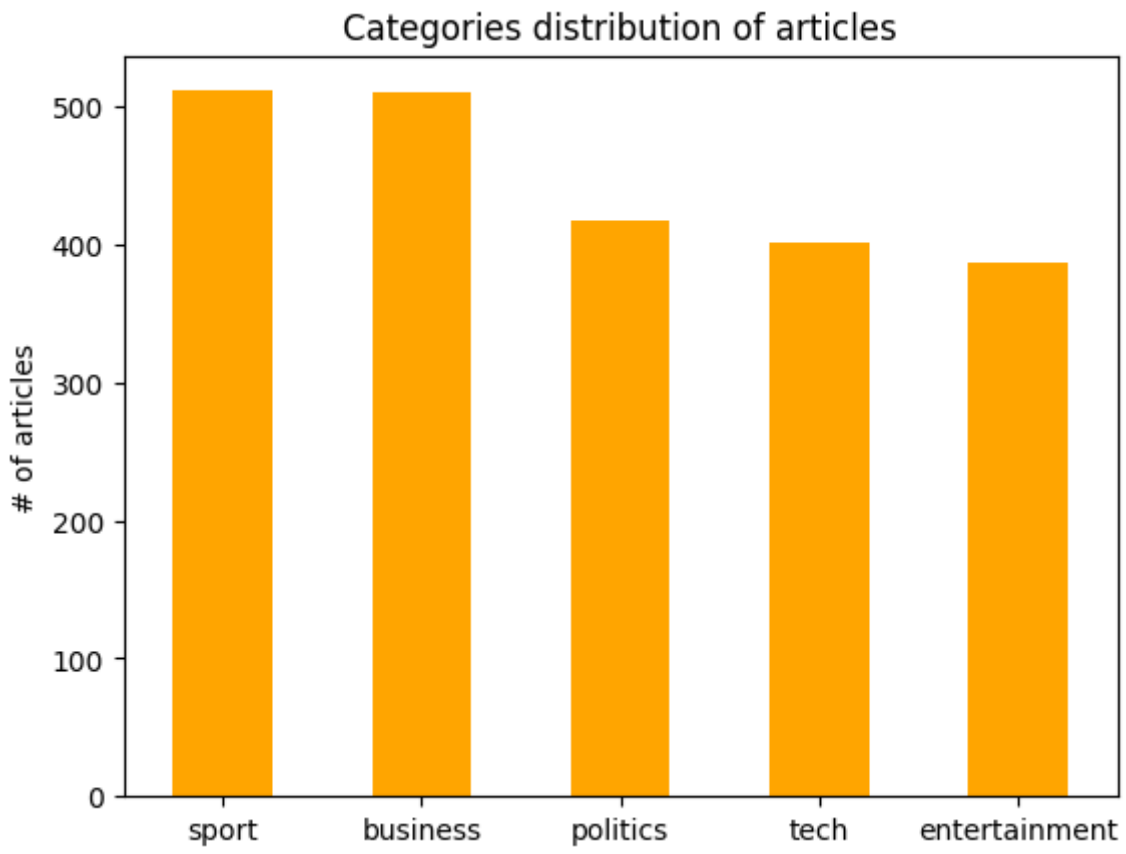
The data set used is the public BBC News data set consisting of 2225 articles, each labeled as one of the five categories: business, entertainment, politics, sport or tech.

Original dataset source: [http://mlg.ucd.ie/datasets/bbc.html (http://mlg.ucd.ie/datasets/bbc.html)](http://mlg.ucd.ie/datasets/bbc.html) *Download file:* [https://raw.githubusercontent.com/donglinchen/text_classification/master/bbc-text.csv (https://raw.githubusercontent.com/donglinchen/text_classification/master/bbc-text.csv)](https://raw.githubusercontent.com/donglinchen/text_classification/master/bbc-text.csv)

In [2]:

```
#Upload dataset

#df = pd.read_csv("/work/category.csv")

df= pd.read_csv('https://raw.githubusercontent.com/donglinchen/text_classification/master/b

print(
    f"Dataset BBC with {len(df)} articles, divided to {len(df['category'].unique())} catego
    )

#Plot categories distribution
df['category'].value_counts().plot.bar(rot=0, color='orange')

plt.title('Categories distribution of articles')
plt.ylabel('# of articles')
None
```

```
Dataset BBC with 2225 articles, divided to 5 categories
```



3 Text preprocessing

3.1 Data cleaning

In [3]:

```
#Function for data cleaning


def preprocess_text(text, remove_stopwords):
    """This utility function sanitizes a string by:
        - removing links
        - removing special characters
        - removing numbers
        - removing stopwords
        - transforming in lowercase
        - removing excessive whitespaces
        Args:
            text (str): the input text you want to clean
            remove_stopwords (bool): whether or not to remove stopwords
        Returns:
            str: the cleaned text
        """
    # remove links
    text = re.sub(r"http\S+", "", text)
    # remove special chars and numbers
    text = re.sub("[^A-Za-z]+", " ", text)
    # remove stopwords
    if remove_stopwords:
        # 1. tokenize
        tokens = word_tokenize(text)
        # 2. check if stopword
        tokens = [
            w for w in tokens if not w.lower() in stopwords.words("english")
        ]
        # 3. join back together
        text = " ".join(tokens)
    # return text in lower case and stripped of whitespaces
    text = text.lower().strip()
    return text
```

In [4]:

```
t1 = time.time()
print('Starting cleaning of data')
tqdm.pandas(dynamic_ncols=True, smoothing=0.01)
#Use function preprocess_text() for every row, assign results to new column
df['cleaned'] = df['text'].progress_apply(lambda x: preprocess_text(x, remove_stopwords=Tru
print('Finished cleaning of data')
t2 = time.time()
print(f'Elapsed time for initilization: {t2-t1:.2f}s')
```

...

Vectorization

Vectorize the text cleaned in a previous step.

In [5]:

```
vectorizer = TfidfVectorizer(sublinear_tf=True, min_df=5, max_df=0.95)

# fit_transform applies TF-IDF to clean texts - we save the array of vectors in X
X = vectorizer.fit_transform(df['cleaned'])
```

4 Clustering

In [6]:

```
def get_top_keywords(X, clusters, vectorizer, n_terms):
    """This function returns the keywords for each centroid of the KMeans"""
    dff = pd.DataFrame(X.todense()).groupby(
        clusters).mean()  # groups the TF-IDF vector by cluster
    terms = vectorizer.get_feature_names_out()  # access tf-idf terms
    for i, r in dff.iterrows():
        print('\nCluster {}'.format(i))
         # for each row of the dataframe, find the n terms that have the highest tf idf sco
        print(','.join([terms[t] for t in np.argsort(r)[-n_terms:]]))
```

Reduce the data dimensionality for visualization. At first we use PCA on a full dataset and use only first 50 principal components. Afterwards we reduce dimensionality to two dimensions with T-SNE algorithm.

Classification is performed on a full dimensionality.

In [7]:

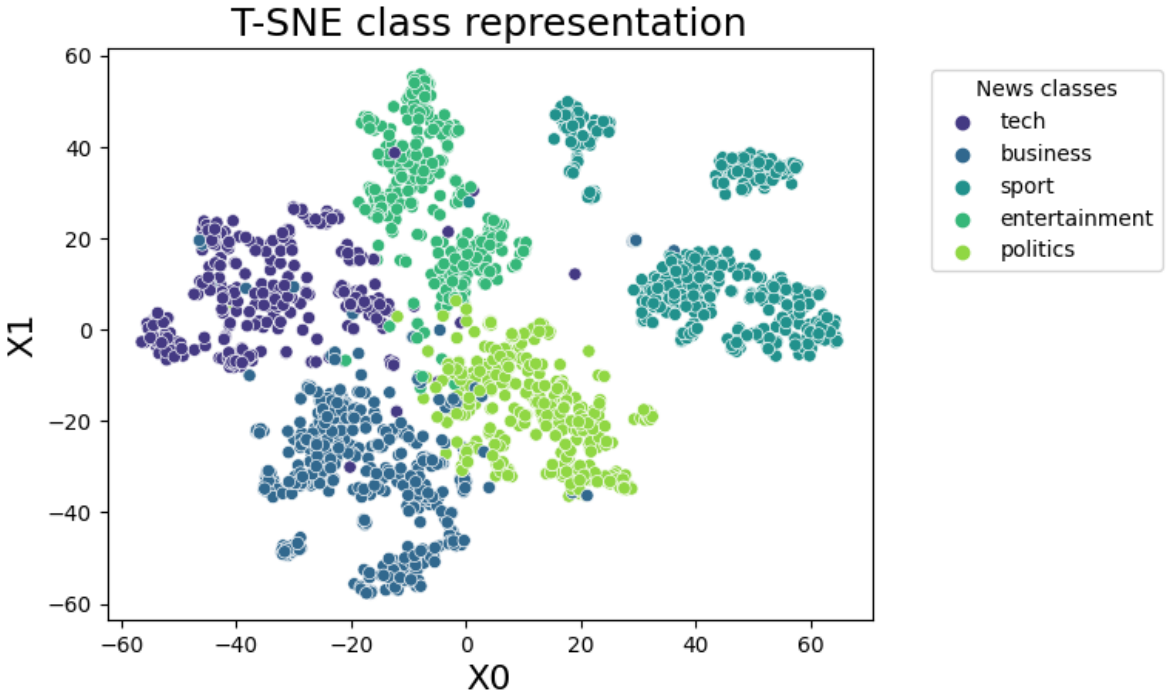
```
pca = PCA(n_components=50, random_state=42)
# pass our X to the pca and store the reduced vectors into pca_vecs
pca_vecs = pca.fit_transform(X.toarray())

X_embedded = TSNE(n_components=2, learning_rate="auto",
                  init="random").fit_transform(pca_vecs)
```

4.1 True categories analysis

Plot of the datapoints, with reduced dimensionality by using T-SNE. Colours corespond to true categories.

In [8]:

```
plt.title("T-SNE class representation", fontdict={"fontsize": 18})
# set axes names
plt.xlabel("X0", fontdict={"fontsize": 16})
plt.ylabel("X1", fontdict={"fontsize": 16})
# create scatter plot with seaborn, where hue is the class used to group the data
sns.scatterplot(data=df,
                x=X_embedded[:, 0],
                y=X_embedded[:, 1],
                hue='category',
                palette="viridis")
#plt.legend(loc='upper right')
plt.legend(bbox_to_anchor=(1.05, 1),
           loc='upper left',
           borderaxespad=1,
           title='News classes')
plt.show()
```



In [9]:

```
print('The most central words for each category')
get_top_keywords(X, df['category'], vectorizer, 10)
```

```
The most central words for each category

Cluster business
shares,firm,economy,year,company,market,growth,us,said,bn

Cluster entertainment
actor,year,star,said,award,music,awards,show,best,film

Cluster politics
people,minister,would,blair,party,government,said,election,labour,mr

Cluster sport
play,players,first,england,team,match,cup,said,win,game

Cluster tech
net,mobile,digital,use,computer,software,said,technology,users,people
```

4.2 K-means clustering method

In [10]:

```
class KMeans:

    def __init__(self, n_clusters, max_iter=300, random_state=1312):
        """
        Parameters
        ----------
        n_clusters : INT
            Number of clusters for K-means
        max_iter : INT, optional
            Number of iterations run by K-means. The default is 300.
        random_state : INT, optional
            Random state for initilization. Used for replication. The default is 1312.

        Returns
        -------
        None.
        """
        self.n_clusters = n_clusters
        self.max_iter = max_iter
        self.random_state = random_state

    def initCentroids(self, X):
        np.random.RandomState(self.random_state)
        random_idx = np.random.permutation(X.shape[0])
        centroids = X[random_idx[:self.n_clusters]]
        return centroids

    def getCentroids(self, X, labels):
        centroids = np.zeros((self.n_clusters, X.shape[1]))
        for k in range(self.n_clusters):
            centroids[k, :] = np.mean(X[labels == k, :], axis=0)
        return centroids

    def getDist(self, X, centroids):
        distance = np.zeros((X.shape[0], self.n_clusters))
        for k in range(self.n_clusters):
            row_norm = np.linalg.norm(X - centroids[k, :], axis=1) #default is frobenius no
            distance[:, k] = np.square(row_norm)
        return distance

    def fit(self, X):
        self.centroids = self.initCentroids(X)
        for i in range(self.max_iter):
            old_centroids = self.centroids
            distance = self.getDist(X, old_centroids)
            self.labels = np.argmin(distance, axis=1)
            self.centroids = self.getCentroids(X, self.labels)
            if np.all(old_centroids == self.centroids): #If no updates are done
                break

    def predict(self, X):
        distance = self.getDist(X, self.centroids)
        return np.argmin(distance,axis=1)
```

In [11]:

```
kmeans = KMeans(n_clusters=5, max_iter=300, random_state=42)

# fit the model
kmeans.fit(X.todense())

# store cluster labels in a variable
clusters_kmeans = kmeans.predict(X)
```

In [12]:

```
print(clusters_kmeans.shape)
#add a column with clusters assigned by kmeans
df['cluster_kmeans_own'] = clusters_kmeans
```

```
(2225,)
```

In [13]:

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=5, random_state=42)
kmeans.fit(X)
df['cluster_kmeans'] = kmeans.labels_
```

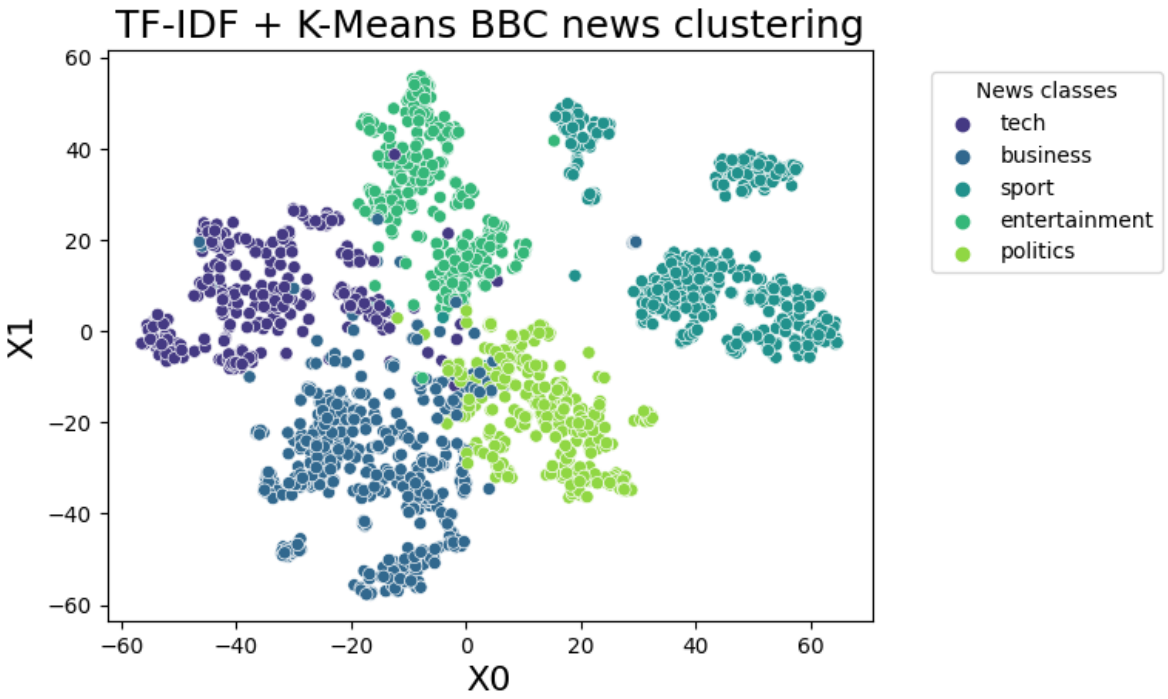
In [14]:

```
get_top_keywords(X, kmeans.labels_, vectorizer, 10)
```

```
Cluster 0
tory,minister,would,said,government,blair,party,election,labour,mr

Cluster 1
play,players,first,england,match,team,cup,said,win,game

Cluster 2
computer,net,software,use,mobile,digital,said,technology,users,people

Cluster 3
shares,firm,economy,growth,year,market,company,us,said,bn

Cluster 4
actor,year,said,music,star,show,award,awards,best,film
```

In [15]:

```
cluster_map_kmeans = {
    0: "politics",
    1: "sport",
    2: "tech",
    3: "business",
    4: "entertainment"
}
# apply mapping
df['cluster_mapped_kmeans'] = df['cluster_kmeans'].map(cluster_map_kmeans)

plt.title("TF-IDF + K-Means BBC news clustering", fontdict={"fontsize": 18})
# set axes names
plt.xlabel("X0", fontdict={"fontsize": 16})
plt.ylabel("X1", fontdict={"fontsize": 16})
# create scatter plot with seaborn, where hue is the class used to group the data
sns.scatterplot(data=df,
                x=X_embedded[:, 0],
                y=X_embedded[:, 1],
                hue='cluster_mapped_kmeans',
                palette="viridis")
#plt.legend(loc='upper right')
plt.legend(bbox_to_anchor=(1.05, 1),
           loc='upper left',
           borderaxespad=1,
           title='News classes')
plt.show()
```



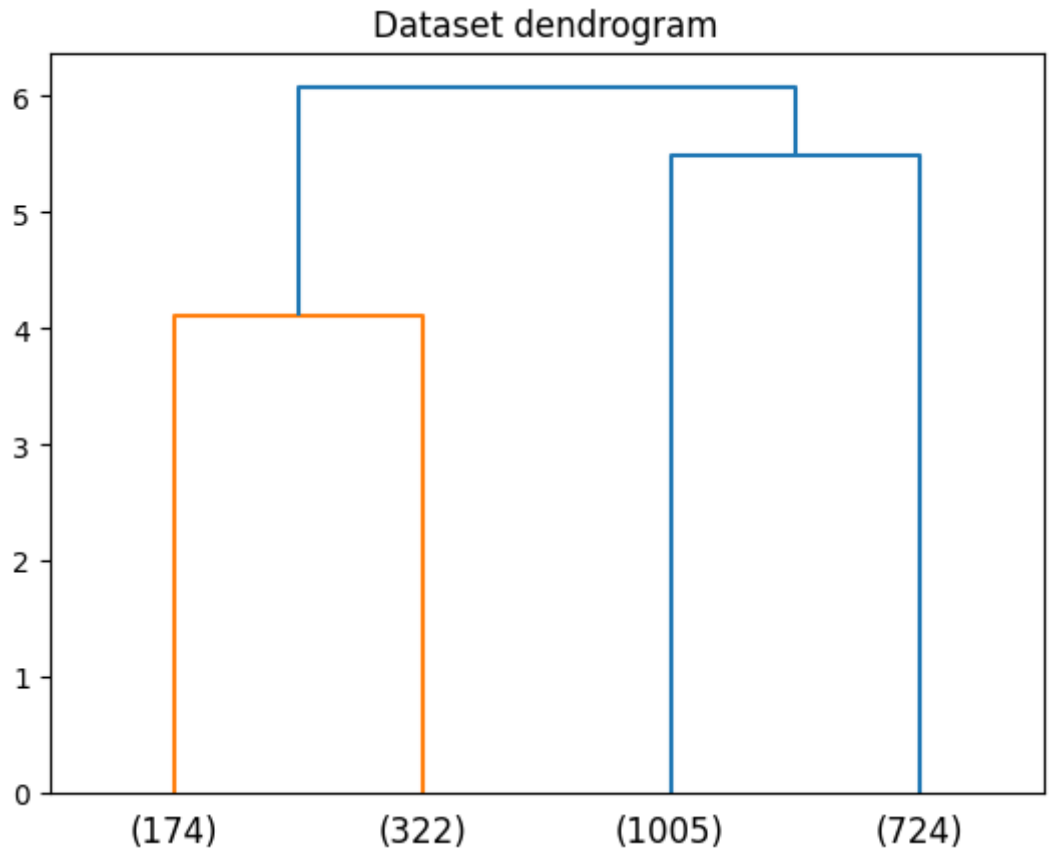
4.3 Agglomerative clustering

In [16]:

```
dendro = hierarchy.dendrogram(hierarchy.linkage(X.toarray(), method='ward'),
                              truncate_mode='level',
                              p=1)
plt.title('Dataset dendrogram')
```

Out[16]:

```
Text(0.5, 1.0, 'Dataset dendrogram')
```



In [17]:

```
#Find clusters with agglomerative algoritm from SKLEARN library
agg = AgglomerativeClustering(n_clusters=5,
                              affinity='euclidean',
                              linkage='ward')
agg.fit(X.toarray())
agg_clusters = agg.labels_

#add a column with clusters assigned by kmeans
df['cluster_agg'] = agg_clusters
```

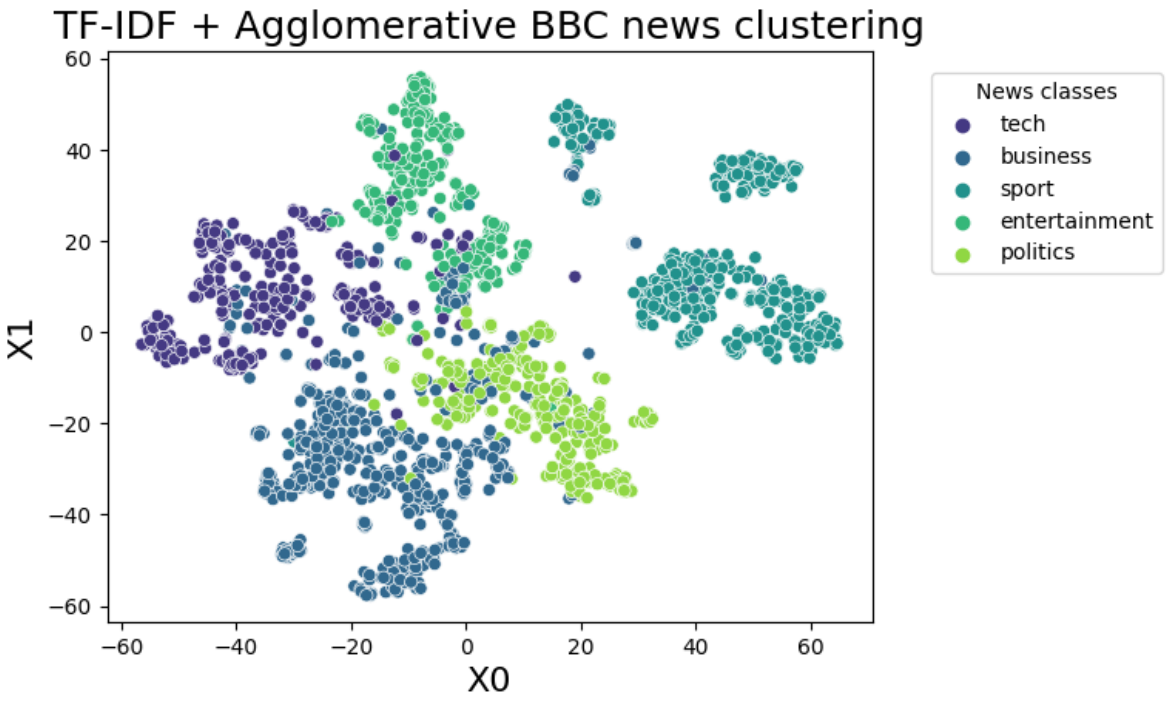
In [18]:

```
get_top_keywords(X, agg_clusters, vectorizer, 10)
```

```
Cluster 0
players,season,first,team,england,match,cup,said,win,game

Cluster 1
firm,economy,market,growth,company,mr,year,us,bn,said

Cluster 2
net,computer,software,use,digital,mobile,said,technology,users,people

Cluster 3
films,said,year,actor,star,show,award,awards,best,film

Cluster 4
tory,minister,blair,would,said,government,party,election,mr,labour
```

In [19]:

```
cluster_map_agg = {
    0: "sport",
    1: "business",
    2: "tech",
    3: "entertainment",
    4: "politics"
}
# apply mapping
df['cluster_mapped_agg'] = df['cluster_agg'].map(cluster_map_agg)

plt.title("TF-IDF + Agglomerative BBC news clustering", fontdict={"fontsize": 18})
# set axes names
plt.xlabel("X0", fontdict={"fontsize": 16})
plt.ylabel("X1", fontdict={"fontsize": 16})
# create scatter plot with seaborn, where hue is the class used to group the data
sns.scatterplot(data=df,
                x=X_embedded[:, 0],
                y=X_embedded[:, 1],
                hue='cluster_mapped_agg',
                palette="viridis")
#plt.legend(loc='upper right')
plt.legend(bbox_to_anchor=(1.05, 1),
           loc='upper left',
           borderaxespad=1,
           title='News classes')
plt.show()
```



4.4 Results analysis

Comparison of results obtaied with different clustering methods

In [20]:

```
labels_true = df['category']
labels = df['cluster_agg']

print('Estimated values for agglomerative clustering')

print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_true, labels))
print("Completeness: %0.3f" % metrics.completeness_score(labels_true, labels))
print("V-measure: %0.3f" % metrics.v_measure_score(labels_true, labels))
print("Davies-Bouldin index (full dimensionality): %0.3f" %
      metrics.davies_bouldin_score(X.toarray(), labels))
print("Davies-Bouldin index (T-SNE projection): %0.3f" %
      metrics.davies_bouldin_score(X_embedded, labels))

print("Silhouette Coefficient (full dimensionality): %0.3f" %
      metrics.silhouette_score(X.toarray(), labels))
print("Silhouette Coefficient (T-SNE projection): %0.3f" %
      metrics.silhouette_score(X_embedded, labels))
print("Rand-index: %0.3f" % metrics.rand_score(labels_true, labels))
print("Mutual-information: %0.3f" % metrics.mutual_info_score(labels_true, labels))

labels = df['cluster_kmeans']

print('\nEstimated values for KMeans clustering')

print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_true, labels))
print("Completeness: %0.3f" % metrics.completeness_score(labels_true, labels))
print("V-measure: %0.3f" % metrics.v_measure_score(labels_true, labels))
print("Davies-Bouldin index: %0.3f" %
      metrics.davies_bouldin_score(X.toarray(), labels))
print("Silhouette Coefficient: %0.3f" %
      metrics.silhouette_score(X.toarray(), labels))
#sklearn.metrics.rand_score(labels_true, labels_pred)
print("Rand-index: %0.3f" % metrics.rand_score(labels_true, labels))
print("Mutual-information: %0.3f" % metrics.mutual_info_score(labels_true, labels))
```

```
Estimated values for agglomerative clustering
Homogeneity: 0.760
Completeness: 0.771
V-measure: 0.766
Davies-Bouldin index (full dimensionality): 8.162
Davies-Bouldin index (T-SNE projection): 1.012
Silhouette Coefficient (full dimensionality): 0.013
Silhouette Coefficient (T-SNE projection): 0.328
Rand-index: 0.924
Mutual-information: 1.218

Estimated values for KMeans clustering
Homogeneity: 0.867
Completeness: 0.872
V-measure: 0.870
Davies-Bouldin index: 7.736
Silhouette Coefficient: 0.014
Rand-index: 0.965
Mutual-information: 1.390
```

Word Clouds for clusters found using K Means method

In [21]:

```
!pip install wordcloud
```

...

In [22]:

```
from wordcloud import WordCloud

fig = plt.figure(figsize=(8, 9))
fig.suptitle('Word Clouds')

k = 0
for idx in df['cluster_mapped_kmeans'].unique():

    plt.subplot(3, 2, k + 1)
    df_cluster = df[df['cluster_mapped_kmeans'] == idx]

    word_cloud = WordCloud(width=800,
                           height=500,
                           background_color='white',
                           min_font_size=14).generate(''.join(
                               df_cluster.cleaned.values.tolist()))
    plt.imshow(word_cloud)
    plt.title(str(idx))
    plt.axis('off')

    k += 1

plt.show()
```

4.5 Misclassified points analysis

In [23]:

```
#Find the list of misclassified indexes

diff_list_agg = np.where(df["category"]!=df['cluster_mapped_agg'])
diff_list_kmeans = np.where(df["category"]!=df['cluster_mapped_kmeans'])



misclassified_agg = df[df.index.isin(diff_list_agg[0])]
misclassified_kmeans = df[df.index.isin(diff_list_kmeans[0])]
```

In [24]:

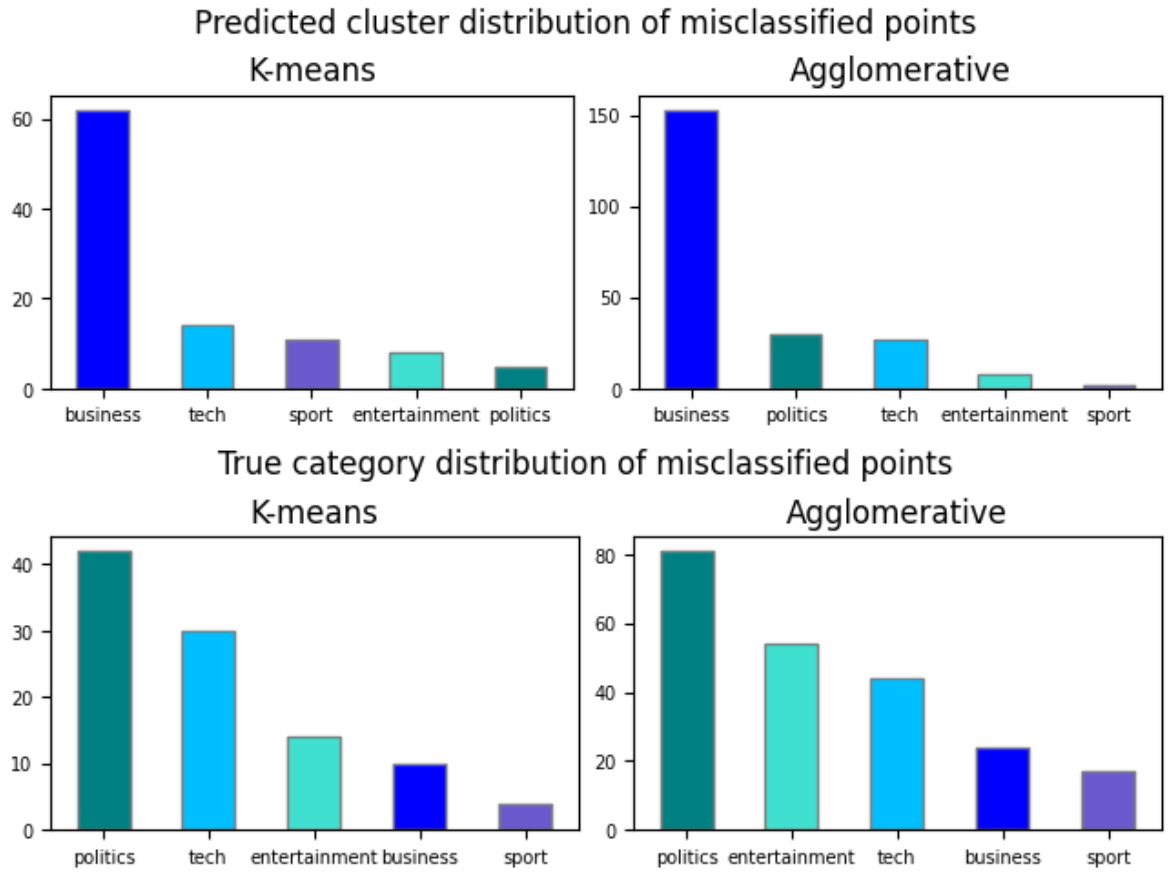
```
print(len(diff_list_agg[0]))
print(len(diff_list_kmeans[0]))
```

```
220
100
```

In [29]:

```
#Politics and tech plots



fig = plt.figure(constrained_layout=True)

plt.rc('xtick', labelsize=10)

subfigs = fig.subfigures(nrows=2, ncols=1)
subfigs[0].suptitle(f'Predicted cluster distribution of misclassified points')
ax1 = subfigs[0].subplots(nrows=1, ncols=2)

#Distribution of misclassified points - true category
misclassified_kmeans['cluster_mapped_kmeans'].value_counts().plot.bar(ax=ax1[0],
                                    #kind='bar',
                                    rot=0,
                                    color=["blue","deepskyblue",'slateblue', "turquoise", "t
                                    edgecolor=["gray"],
                                    title = 'K-means',
                                    fontsize = 7 )


#Distribution (of true ) of misclassified as bussiness
misclassified_agg['cluster_mapped_agg'].value_counts().plot(ax=ax1[1],
                                    kind='bar',
                                    rot=0,
                                    color=["blue","teal", "deepskyblue","turquoise", 'slateb
                                    edgecolor=["gray"],
                                    title =  'Agglomerative',
                                    fontsize = 7)

subfigs[1].suptitle(f'True category distribution of misclassified points')
ax2 = subfigs[1].subplots(nrows=1, ncols=2)


#Distribution of misclassified points - predicted cluster
misclassified_kmeans['category'].value_counts().plot(ax=ax2[0],
                                    kind='bar',
                                    rot=0,
                                    color=["teal", "deepskyblue","turquoise","blue", 'slateb
                                    edgecolor=["gray"],
                                    title = 'K-means',
                                    fontsize = 7
                                    )

#plt.savefig('outlier_anal.png', dpi=300)

#Distribution of misclassified points - true category




misclassified_agg['category'].value_counts().plot.bar(ax=ax2[1],
                                    #kind='bar',
                                    rot=0,
                                    color=[ "teal","turquoise","deepskyblue","blue", 'slateb
                                    edgecolor=["gray"],
                                    title =  'Agglomerative',
                                    fontsize = 7
```

```
                                    )

plt.savefig('sublot_missclasified.png', dpi=300)
None
```



5 Classification methods

5.1 K-NN classifier

In [30]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.neighbors import KNeighborsClassifier

X_train, X_test, y_train, y_test = train_test_split(X.toarray(),
                                                    df["category"],
                                                    test_size=0.20,
                                                    random_state=911)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
classifier = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
ac = accuracy_score(y_test, y_pred)
print(f"Confusion matrix: \n{cm}")
print(f"Accuracy: \n{ac}")
```

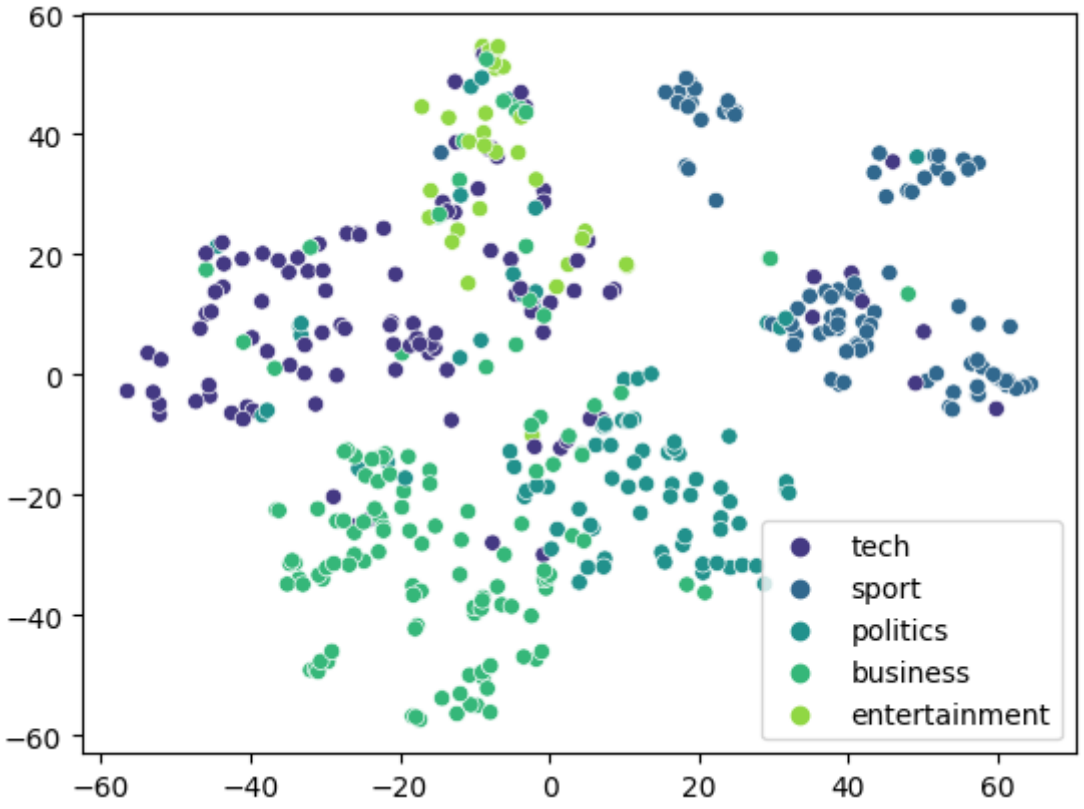
```
Confusion matrix:
[[97  0  6  0  5]
 [14 30 11  1 27]
 [12  1 65  0  5]
 [ 2  0  4 84  8]
 [ 4  0  6  0 63]]
Accuracy:
0.7617977528089888
```

In [31]:

```
sns.scatterplot(x=X_embedded[y_test.index, 0], y=X_embedded[y_test.index,1], hue=y_pred, pa

#sns.scatterplot(x=df1["x0"][y_test.index], y=df1["x1"][y_test.index], hue=y_pred, palette=
#something work with classes here
```

Out[31]:

<AxesSubplot: >



In [32]:

```
############
#Test 1 of Supervised Classification: Dense NN
############
import torch
import torchvision
from torchvision import transforms, datasets
import torch.nn as nn
import torch.nn.functional as F
from torch.nn import BCEWithLogitsLoss
import torch.optim as optim

cluster_map = {
    "politics": 0,
    "sport": 1,
    "tech": 2,
    "business": 3,
    "entertainment": 4
}
# apply mapping
df['category_nn'] = df['category'].map(cluster_map)

n = 300
pca = PCA(n_components=n, random_state=42)
# pass our X to the pca and store the reduced vectors into pca_vecs
pca_vecs = pca.fit_transform(X.toarray())

X_train, X_test, y_train, y_test = train_test_split(pca_vecs,
                                                    df["category_nn"],
                                                    test_size=0.20,
                                                    random_state=911)
```

```
/shared-libs/python3.9/py/lib/python3.9/site-packages/tqdm/auto.py:22: TqdmW
arning: IProgress not found. Please update jupyter and ipywidgets. See http
s://ipywidgets.readthedocs.io/en/stable/user_install.html (https://ipywidget
s.readthedocs.io/en/stable/user_install.html)
  from .autonotebook import tqdm as notebook_tqdm
```

In [33]:

```
#Neural Network architecture
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(300, 128)
        self.fc2 = nn.Linear(128, 128)
        self.fc3 = nn.Linear(128, 64)
        self.fc4 = nn.Linear(64, 5)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = self.fc4(x)

        return x


#Neural Network training loop

net = Net()
net = net.float()
optimizer = optim.Adam(net.parameters(), lr=0.001)
EPOCHS = 15
loss = BCEWithLogitsLoss()
for epoch in range(EPOCHS):
    for X, y in zip(X_train, y_train):
        X_nn = torch.tensor(X)
        y_nn = nn.functional.one_hot(torch.tensor(y), num_classes=5).float()
        output = net(X_nn.float())
        loss_fun = loss(output, y_nn).mean()
        net.zero_grad()
        loss_fun.backward()
        optimizer.step()
    print(loss_fun)
```

```
tensor(0.0166, grad_fn=<MeanBackward0>)
tensor(0.0005, grad_fn=<MeanBackward0>)
tensor(0.0003, grad_fn=<MeanBackward0>)
tensor(0.0001, grad_fn=<MeanBackward0>)
tensor(2.5009e-05, grad_fn=<MeanBackward0>)
tensor(1.7213e-05, grad_fn=<MeanBackward0>)
tensor(5.9605e-07, grad_fn=<MeanBackward0>)
tensor(3.3379e-07, grad_fn=<MeanBackward0>)
tensor(9.5367e-08, grad_fn=<MeanBackward0>)
tensor(0., grad_fn=<MeanBackward0>)
tensor(0., grad_fn=<MeanBackward0>)
tensor(0., grad_fn=<MeanBackward0>)
tensor(0., grad_fn=<MeanBackward0>)
tensor(0., grad_fn=<MeanBackward0>)
tensor(0., grad_fn=<MeanBackward0>)
```

In [34]:

```
correct = 0
total = 0
with torch.no_grad():
    for X, y in zip(X_test, y_test):
        X = torch.tensor(X)
        output = net(X.float())
        output = torch.sigmoid(output)
        if torch.argmax(output).item() == y:
            correct += 1
        total += 1

print("Accuracy: ", round(correct / total, 3))
```

```
Accuracy:  0.978
```