Machine Learning Algorithms for Solving Real-World Classification and Clustering Problems

Aryalakshmi Nellippillipathil Babu, Jeyatrathan Ramalingam

Faculty of Engineering Environment and Computing

Coventry University, England

Abstract: This paper focuses on how similar documents from a large text corpus can be categorized or grouped together by applying machine learning algorithms. The bag of words dataset used for this project is downloaded from UCI Machine Learning Repository. Since the dataset is not labeled, two clustering algorithms are applied to cluster the documents into different groups. Data labeling for around thousand records are done manually and two classification algorithms are applied to this dataset. The data preparation and machine learning algorithms are implemented using python language, nltk libraries and scikit-learn machine learning libraries.

Keywords: machine learning; natural language processing; document clustering; document classification; nltk; k-means; latent dirichlet allocation; naïve bayes; support vector machines; python.

INTRODUCTION

Nowadays most of the communication happens via audio, video and text data. These text data are generated in huge amount from various sources like social media, blogging, website operations, industry operations, digital payments etc. Most of the organizations need these text data to be organized into categories so that they can get rid of duplicate records, or they can extract a specified set of documents from a huge number of documents. Features of a new document can be compared with the features of already identified document groups so that the newly arrived document can be placed in its specified group. These tasks are hard to be done by humans and there comes the use of machine learning techniques.

When dataset is not labeled, unsupervised learning needs to be implemented and clustering is an unsupervised learning method. When dataset is labeled, supervised learning needs to be implemented and classification is a supervised learning method.

The text data should be preprocessed first by removing stop words, noisy and unwanted data. This process can be achieved by using nltk libraries.

The text data is unstructured data and to feed this data into clustering and classification model algorithms, it need to be converted into structured data. For this task Tfidf Vectorizer or Count Vectorizer can be used. A collection of text documents is converted into a matrix of token counts by using Count Vectorizer. A collection of text documents is converted into a matrix of TF-IDF features by using Tfidf Vectorizer.

DESCRIPTION OF THE DATASET

The dataset for this project is downloaded from UCI Machine Learning Repository. The dataset contains 299752 New York Times news articles, and the data is already in cleaned form by performing tokenization and removal of stopwords. This dataset has no class labels. The cleaned data is provided in two files named ‘docword.nytimes.txt’ and ‘vocab.nytimes.txt’, where the first file contains number of times each word occurs in each document and the second file is the vocabulary file which lists all the words that appear in the document collection.

The format of the ‘docword.nytimes.txt’ contains three header lines followed by document id (docID), word id (wordID) and word count (count). The three header lines will be omitted. The data format looks like below:

#D

#W

#NNZ

#docID wordID count

#docID wordID count

#...

#...

#docID wordID count

#docID wordID count

Each line in ‘vocab.nytimes.txt’ contains a unique word, where the line number is equal to word id (wordID).

PREPARATION OF THE DATASET

A. Dataset Merging and Cleaning

The dataset needs to be in a vectorized format to feed into the machine learning algorithms, where each row in the dataset will be each document having the vector representation of words containing in it. To create the final dataset in this format ‘docword.nytimes.txt’ and ‘vocab.nytimes.txt’ are joined together by ‘wordID’ for each document. The vocabulary data don’t have the ‘wordID’ attribute, so a new column named ‘n’ is added to the data having value line number which is equal to ‘wordID’. To reduce dimensionality of features, only the words which have word count greater than two are selected for the final dataset. Since stopwrods are already removed in the initial dataset, there is no need to check for stopwords. All words in each document is lemmatized using the nltk’s WordNetLemmatizer library. To get proper lemmatization of words in each document, each word is tagged with nltk’s pos_tag library. Some words in data start with ‘zzz_’ which don’t provide any meaning to the context are removed. Finally, each word is multiplied by its count and again the whole set of words are randomized to form the whole text for each document.

The initial dataset has 299752 documents. Because this will take a lot of hours to preprocess and train the data for machine learning, this project took only 5000 records for the clustering experiment. After preprocessing step done for 5000 records, only 4915 records are found to be usable. Each row in the final dataset represents each document and each cell in the column represents the text in each document or article.

B. Dataset Labeling

The initial dataset is not labeled and for classification task the data need to be labeled. 1083 records are manually labeled and used for classification task. Since manually analyzing the words in each document to categorize each record is a time-consuming process, a free machine learning web service named ‘uClassify’ is used to classify text for these 1083 records.

C. Data Vectorization

Text data is unstructured data and to feed this data into machine learning algorithms, it need to be converted into vectorized format. The process of converting a collection of text documents into numerical feature vectors is known as vectorization. Count Vectorizer creates a matrix of unique words for each document where each row represents a document in data and each column represents each unique word in respective documents, each cell contains the word count for that document. TF-IDF (Term Frequency – Inverse Document Frequency) Vectorizer too creates a matrix of unique words for each document by considering overall document weightage of a word. It weights and normalizes the word counts by analyzing how often the words appear in the documents which makes TF-IDF vectorizer better useful than Count Vectorizer.

In this project TF-IDF vectorizer is used for K-Means clustering algorithm, Naïve Bayes classification algorithm and Support Vector Machines classification algorithm. Count Vectorizer is used in Latent Dirichlet Allocation clustering algorithm because this algorithm is based on term count and document count.

EXPERIMENTAL SETUP

This project implements the data preparation, machine learning algorithms and visualizations using python as the main programming language. The python code is written and tested in Jupyter Notebook which is a web-based interactive computing platform. The operating system used for this project is 64-bit Windows 10 having 8 GB ram. The whole python package can be installed by downloading and installing Anaconda Distribution. The Python version used in this project is 3.7.3 and the version for Jupyter Notebook is 4.4.0. Additional libraries needs to be installed are nltk, pandas, numpy, sklearn, wordcloud and matplotlib.

A. Clustering

Clustering is an unsupervised learning approach. The dataset used in this project has no labels and hence to group the similar documents together clustering machine learning algorithms are applied. This project took only 5000 records for the clustering experiment and after preprocessing steps like vocabulary merging with document id and lemmatization done for 5000 records, only 4915 records are found to be usable. These text records are vectorized using TF-IDF vectorizer and fed into K-Means clustering machine learning algorithm. Elbow method is used to find the optimal number of clusters, but from the plot of sum of squared distances vs number of clusters an elbow shape couldn't be found. After referring to New York Times website, it is found that the news articles are mainly divided into 18 categories like Education, Weather, Politics, Business etc. So the number of clusters is set as 18 for K-means algorithm.

One more clustering algorithm named Latent Dirichlet Allocation is applied to vectorized dataset, but this time Count vectorizer is used because this algorithm is based on term count and document count. The vectorization of data and implementation of clustering machine learning algorithms are done by pre-define scikit-learn machine learning libraries.

The results of K-Means clustering algorithm is evaluated by Davies Bouldin Score, Silhoutte Score and word cloud plots. The Latent Dirichlet Allocation algorithm is evaluated by plotting top 20 words found for each document group as a horizontal bar graph.

The time taken to execute data pre-processing steps is one and half hour.

The time taken to vectorize dataset using TF-IDF vectorizer is 1.02 seconds.

The time taken to train K-Means model is 18.11 seconds.

The time taken to vectorize dataset using Count vectorizer is 0.52 seconds.

The time taken to train Latent Dirichlet Allocation model is 4.97 seconds.

B. Classification

Classification is a supervised learning approach. For classification task the data need to be labeled and since this dataset is not labeled, 1083 records are manually labeled and used for this project's classification task. The data is split into training and test set using the train_test_split function from scikit-learn. These records are vectorized using TF-IDF vectorizer and fed into two classification algorithms named Naïve Bayes and Support Vector Machines. The vectorization and implementation of machine learning algorithms are assembled into a pipeline using scikit-learn pipeline function. The results of both classification algorithms are evaluated by accuracy score, f1-score and jaccard-score.

The total time taken to vectorize dataset using TF-IDF and to train Naïve Bayes model is 0.09 seconds.

The total time taken to vectorize the dataset using TF-IDF and to train the Support Vector Machines model is 0.07 seconds.

MACHINE LEARNING CLUSTERING TECHNIQUES

A. K-Means

The K-means clustering is a vector quantization method, originally from signal

processing, and it is popular for cluster analysis in data mining. This clustering algorithm partitions n observations into k clusters and each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

B. Latent Dirichlet Allocation (LDA)

Latent Dirichlet Allocation (LDA) is a widely used text mining technique which is a probabilistic topic model and it processes documents as the probability distribution of topics. In this model, the topics follow the Dirichlet distribution which considers that the occurrence of a topic has nothing to do with that of other topics.

MACHINE LEARNING CLASSIFICATION TECHNIQUES

A. Naïve Bayes

The Naive Bayes classifiers are simple probabilistic classifiers with their foundation on Bayes theorem with the assumption of strong independence among the features.

B. Support Vector Machines (SVMs)

Support vector machine is a non-probabilistic binary linear classifier and it analyzes data and recognizes patterns used for classification and regression tasks. SVM model is a representation of the examples as points in space, mapped, so that the examples of separate categories are divided by a clear gap, that is wide as possible.

EXPERIMENT RESULTS

A. K-Means

Using python and scikit-learn libraries K-Means clustering algorithm applied on the selected dataset and results are presented in Fig. 1 and Fig. 2.

```
Davies Bouldin Score: 4.5031274856130254
Silhoutte Score: 0.03075543284802214
```

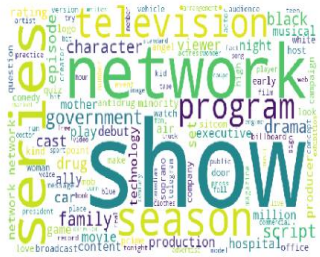
Fig. 1 K-Means model Scores

Davies Bouldin score is the average similarity measure of each cluster with its most similar cluster. The lower values of this score indicate better clustering. Here this measure has the value 4.5 which is not considered as a better result. The Silhoutte Score is a measure of similarity of a data point within a cluster compared to other clusters. Here this measure has the value 0.03 which means there are overlapping clusters.

10

4

8



15

Fig. 2 Word Cloud for K-Means Clustering (Cluster 10, 8, 4 and 15)

Word clouds for words in each cluster are plotted. From Fig. 2, it can be concluded as cluster 10 is related to Sports, cluster 8 is related to Arts, cluster 4 is related to Business and cluster 15 is related to Technology. It should be noted that all these clusters have a few words which are not related to the corresponding category and hence it cannot be concluded as a best performed clustering algorithm.

B. Latent Dirichlet Allocation (LDA)

Using python and scikit-learn libraries Latent Dirichlet Allocation clustering algorithm is applied on the selected dataset and results are presented in the appendix.

It can be concluded as cluster 1 is related to Society, cluster 2 is related to Business, cluster 3 is related to Sports, cluster 4 is related to Games, cluster 5 is related to Society and Government, cluster 6 is related to Arts. It should be noted that all these clusters have a few words which is not related to the corresponding category and hence it cannot be concluded as a best performed clustering algorithm.

C. Naïve Bayes

Using python and scikit-learn libraries **Naïve Bayes** classification algorithm is applied on the selected dataset and results are presented in Fig. 4 and Fig.5.

Accuracy Score: 0.7188940092165899
Classification Report

	precision	recall	f1-score	support
Arts	1.00	0.42	0.59	24
Business	0.94	0.62	0.75	24
Computers	0.93	0.81	0.87	16
Games	0.00	0.00	0.00	2
Health	1.00	0.14	0.25	7
Home	1.00	0.40	0.57	5
Recreation	0.00	0.00	0.00	13
Science	0.00	0.00	0.00	3
Society	0.60	1.00	0.75	82
Sports	0.87	0.80	0.84	41
accuracy			0.72	217
macro avg	0.63	0.42	0.46	217
weighted avg	0.73	0.72	0.68	217

Fig. 4 Classification Report for Naïve Bayes

```
#f1-score
print('f1-score')
f1_score(y_test, predicted_values, av
```

```
f1-score

0.6752658734289885
```

```
#jaccard-score
print('jaccard-score')
jaccard_score(y_test, predicted_value
```

```
jaccard-score

0.5460346270345445
```

Fig. 5 f1-score and jaccard-score for Naïve Bayes classification

Naïve Bayes classification resulted in an accuracy score of 72%, f1-score of 68% and jaccard-score of 55%.

D. Support Vector Machines (SVMs)

Using python and scikit-learn libraries Support Vector Machines classification algorithm is applied on the selected dataset and results are presented in Fig. 6 and Fig.7.

```
Accuracy Score: 0.8064516129032258
Classification Report
              precision    recall  f1

       Arts       0.67      0.75
   Business       0.88      0.92
  Computers       0.94      0.94
      Games       0.50      0.50
     Health       0.56      0.71
       Home       0.67      0.40
 Recreation       0.60      0.23
    Science       0.60      1.00
    Society       0.83      0.91
     Sports       0.89      0.76

   accuracy
  macro avg       0.71      0.71
weighted avg       0.80      0.81
```

Fig. 6 Classification Report for Support Vector Machines

```
#f1-score
print('f1-score')
f1_score(y_test, predicted_values, average='wei
```

```
f1-score

0.7968180946549042
```

```
#jaccard-score
print('jaccard-score')
jaccard_score(y_test, predicted_values, average
```

```
jaccard-score

0.6835288917997892
```

Fig. 7 f1-score and jaccard-score for Support Vector Machines

Support Vector Machines classification resulted in an accuracy score of 81%, f1-score of 80% and jaccard-score of 68%.

Compared to Naïve Bayes results, Support Vector machines performed better in the classification task.

CONCLUSION

For clustering tasks, both K-Means and LDA have overlapping document clusters. And for classification tasks, Support vector machines have best performance than Naive Bayes.

FUTURE RESEARCH

Clustering algorithms have categorized the documents up to some percentage, but it has some overlapping clusters too. To get better accurate results, deep learning neural networks or hybrid deep learning algorithms could be used. Classification algorithms used in this project have performed well and to get a more accurate performance other classification algorithms could be used and some deep learning methods can also be tried to get an accuracy around 99%. If the dataset is not labeled an automation algorithm needs to be set up to scrape classification labels from free text classification apis like ‘uClassify’.

REFERENCES

Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

Mohammed, M., Bashier, Eihab Bashier Mohammed, & Khan, Muhammad Badruddin. (2017). Machine learning : Algorithms and applications (1st ed.).

B. Wang, Y. Liu, Z. Liu, M. Li and M. Qi, "Topic selection in latent dirichlet allocation," *2014 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, 2014, pp. 756-760, doi: 10.1109/FSKD.2014.6980931.

Dr. Omid Chatrabgoun, 7072 CEM, Machine Learning, Coventry University.

Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

CONTRIBUTION

This project is done by Aryalakshmi Nellippillipathil Babu and Jeyatrathan Ramalingam. The contributions by both are explained below.

The selection of the dataset, code for dataset cleaning and merging are done by both.

Since the dataset wasn't labeled, we decided to manually label around a thousand numbers of data for classification tasks. Around five hundred numbers of data is labeled by Jeyatrathan and another five hundred numbers of data is labeled by Aryalakshmi.

Two clustering algorithms are applied on the data. K-Means clustering algorithm along with its data vectorization steps are done by Aryalakshmi. Latent Dirichlet Allocation along with its data vectorization steps are done by Jeyatrathan.

Two classification algorithms are applied on the data. Naïve Bayes classification algorithm along with its data vectorization steps are done by Aryalakshmi. Support Vector Machines classification algorithm along with its data vectorization steps are done by Jeyatrathan.

APPENDIX

Dataset link

https://archive.ics.uci.edu/ml/datasets/Bag+of+Words

Website link for labeling text for classification task

Topics Classifier | uClassify

Code with output

K-Means Clustering

In [1]:
```
import pandas as pd
import random
from nltk.stem import WordNetLemmatizer
from nltk import pos_tag
```

In [2]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn import metrics
```

In [3]:
```
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from time import time
```

In [4]:
```
#Read document data
col_names=['docID', 'wordID', 'count']
df_dataText=pd.read_csv('NewYorkTimes/docword.nytimes/docword.nytimes.txt',
                        header=None, delim_whitespace=True, skiprows=3,names=col_names)
#check the first 5 rows of data
df_dataText.head()
```

Out[4]:

	docID	wordID	count
0	1	413	1
1	1	534	1
2	1	2340	1
3	1	2806	1
4	1	3059	1

In [5]:
```
print('Maximum count of words:',df_dataText['count'].max(),
      'Minimum count of words:',df_dataText['count'].min())
```

```
Maximum count of words: 212 Minimum count of words: 1
```

In [6]:
```
#check the no. of rows and columns in dataset
df_dataText.shape
```

Out[6]: (69679427, 3)

In [7]:
```
#check null values
df_dataText[df_dataText['wordID'].isna()==True]['wordID'].count()
```

Out[7]: 0

In [8]:
```
#check the number of documents
len(df_dataText['docID'].unique())
```

Out[8]: 299752

In [9]:
```
#check the number of unique words
len(df_dataText['wordID'].unique())
```

Out[9]: 101636

In [10]:
```
#Find if the wordID starts with 0 or 1 and max wordID
print('wordID starts with: ',df_dataText['wordID'].min())
print('wordID ends with: ',df_dataText['wordID'].max())
```

```
wordID starts with:  1
wordID ends with:  102660
```

In [11]:
```
#Read vocabulary data
col_Vnames=['vocab_word']
df_dataVocab=pd.read_csv('NewYorkTimes/vocab.nytimes.txt',
                         header=None, delim_whitespace=True,names=col_Vnames)
df_dataVocab.head()
```

Out[11]:

	vocab_word
0	aah
1	aahed
2	aaron
3	aback
4	abacus

In [12]:
```
#check null values
df_dataVocab[df_dataVocab['vocab_word'].isna()==True].count()
```

Out[12]:
```
vocab_word    0
dtype: int64
```

In [13]:
```
#check the no. of rows and columns in dataset
df_dataVocab.shape
```

Out[13]:
```
(102660, 1)
```

In [14]:
```
#check the number of unique words
len(df_dataVocab['vocab_word'].unique())
```

Out[14]:
```
102659
```

In [15]:
```
#Add n(id number) column to vocab data
df_dataVocab['n']=df_dataVocab.index+1
df_dataVocab.head()
```

Out[15]:

	vocab_word	n
0	aah	1
1	aahed	2
2	aaron	3
3	aback	4
4	abacus	5

In [16]:
```
#function to collect words for each document
def get_docWords(docId):
    c1=df_dataText['docID']==docId
    c2=df_dataText['count']>=3#count greater than 3
    df_filtered=df_dataText[(c1)&(c2)][['wordID','count']].reset_index(drop=True)
    df_filtered=df_filtered.merge(df_dataVocab,left_on=['wordID'],
                                  right_on=['n'],how='inner')[['vocab_word','count']]
    return df_filtered
```

In [17]:
```
#Lemmatize each word
def lemmatize_words(lst_tag_words):
    lst_words_lemmatized=[]
    for w,t in lst_tag_words:
        tg=t[0].lower()
        if tg not in ['a', 'r', 'n', 'v']:
            lst_words_lemmatized.append(w)
        else:
            lst_words_lemmatized.append(obj_lemmatizer.lemmatize(w,tg))
    return lst_words_lemmatized
```

In [18]:
```
#Create cleaned text dataset in order to feed into clustering algorithms
obj_lemmatizer=WordNetLemmatizer()
train_corpus=[]
for docId in range(1,5001):
    isSuccess=True
    #get words in a document
    df_doc_words=get_docWords(docId)
    try:
        #Assign POS tag for each word, Lemmatizer requires correct POS tag to accurately lemmatize words
        df_doc_words['pos_tag']= pos_tag(df_doc_words['vocab_word'])
        #Group different inflections of the same word.
        df_doc_words['lemmatized_words']=lemmatize_words(df_doc_words['pos_tag'].tolist())
    except:
        isSuccess=False
    if isSuccess:
        txt_sentnc=''
        for i,row in df_doc_words.iterrows():
            if row['lemmatized_words']!='':
                w=row['lemmatized_words']
                if w[:4]!='zzz_':
                    txt_sentnc+=' '+(w+' ')*row['count']
        txt_sentnc=txt_sentnc.strip()
        if txt_sentnc!='':
            txt_sentnc=txt_sentnc.split()
            random.shuffle(txt_sentnc)
            txt_sentnc=' '.join(txt_sentnc)
            train_corpus.append(txt_sentnc)
```

In [19]:
```
#Print first five sentences
train_corpus[0:5]
```

Out[19]:
```
['scored scored play player game win play win point team play game half night game scored team player rank point point scored n
ight rank night team half half scored game rank play team play point half rank player team win play',
 'issue paper paper front error issue number number number issue issue issue issue number paper error number issue error front
number paper issue front number',
 'light midnight celebration midnight leave leave firework celebration crowd celebration security firework light friend friend
friend crowd celebration security crowd terrorism crowd crowd terrorism security light midnight firework leave terrorism',
```

In [20]:
```
#print no. of elements in train_corpus
len(train_corpus)
```

Out[20]:
```
4915
```

In [21]:
```
#########################   Vectorization  ########################
```

In [22]:
```
#Create vector of words for each document using TfidfVectorizer
time_start=time()
obj_vectorizer=TfidfVectorizer()
train_corpus_vectorized = obj_vectorizer.fit_transform(train_corpus)
time_taken=time()-time_start
print('Time(seconds) taken to vectorize text data:',time_taken)
train_corpus_vectorized
```

```
Time(seconds) taken to vectorize text data: 1.0237939357757568
```

Out[22]:
```
<4915x5807 sparse matrix of type '<class 'numpy.float64'>'
        with 64115 stored elements in Compressed Sparse Row format>
```

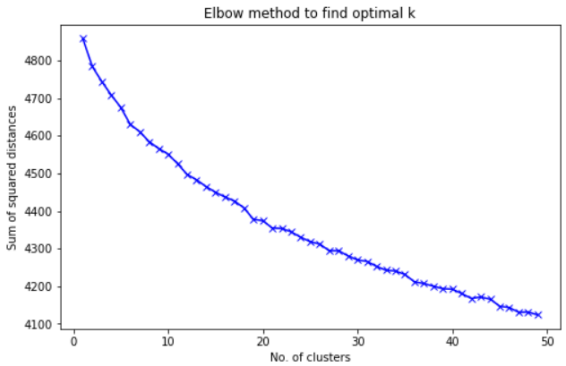
In [ ]:
```
######################### Clustering k-means  ########################
```

In [ ]:
```
'''
News categories from NewYork times
----------------------------------
Education, Weather, Politics, Business, Technology,
Science, Health, Arts, Sports, Automobiles,
Books, Movies, Dining & Wine, Home & Garden, Fashion & Style,
Travel
'''
```

In [34]:
```
#Find optimal k
lst_sum_sqrdDstnc = []
lst_k = range(1,50)
for k in lst_k:
    obj_kmeansModel = KMeans(n_clusters=k, init='k-means++')
    obj_kmeansModel = obj_kmeansModel.fit(train_corpus_vectorized)
    lst_sum_sqrdDstnc.append(obj_kmeansModel.inertia_)
print('Sum of squared distances: ',lst_sum_sqrdDstnc)
```

```
Sum of squared distances:  [4858.022112308217, 4783.700603017065, 4743.844907043461, 4707.964672423966, 4675.293187861651, 463
0.132621396857, 4610.6225275703755, 4584.061781476942, 4565.843602893804, 4550.96159179005, 4525.533705755692, 4497.14095192715
6, 4482.852057613711, 4464.830335647557, 4449.67878814535, 4437.041548241703, 4426.433356856204, 4408.248763793084, 4377.915354
819804, 4374.156981634963, 4354.423578674616, 4353.5948723112815, 4344.3695229363675, 4330.126413107971, 4318.244391445674, 431
2.186024251035, 4294.3528720738, 4294.297583782735, 4280.532056516204, 4269.855831415304, 4266.005124715298, 4252.663088120542,
4242.55606881093, 4240.708422877047, 4231.046028191113, 4211.78479729948, 4207.052422954881, 4200.25754350748, 4193.49086153954
1, 4192.3125926276625, 4180.430683327662, 4167.720778481529, 4171.393817502214, 4165.762117794812, 4146.956956392174, 4143.4583
46303323, 4130.870506733675, 4131.513367686824, 4123.858891901067]
```

In [35]:
```
#Plot to find optimal k
plt.plot(lst_k, lst_sum_sqrdDstnc, 'bx-')
plt.rcParams["figure.figsize"] = (8, 5)
plt.xlabel('No. of clusters')
plt.ylabel('Sum of squared distances')
plt.title('Elbow method to find optimal k')
plt.show()
```



In [36]:
```
#KMeans clustering model
time_start=time()
optimalK=18
obj_kmeansModel = KMeans(n_clusters=optimalK, init='k-means++')
obj_kmeansModel.fit(train_corpus_vectorized)
time_taken=time()-time_start
print('Time(seconds) taken to train KMeans model:',time_taken)
```

```
Time(seconds) taken to train KMeans model: 18.1162371635437
```

In [37]:
```
pred_labels=obj_kmeansModel.labels_
pred_labels
```

Out[37]:
```
array([10,  4,  4, ...,  4,  4,  4])
```

In [38]:
```
df_result=pd.DataFrame({"text":train_corpus,"labels":pred_labels})
df_result.head(10)
```

Out[38]:

	text	labels
0	scored scored play player game win play win po...	10
1	issue paper paper front error issue number num...	4
2	light midnight celebration midnight leave leav...	4
3	passenger thousand aviation air midnight passe...	4
4	show program program television show televisio...	8
5	red village village computer government red re...	15
6	celebrate million million problem firework com...	4
7	thousand midnight western western calendar rel...	4
8	night music show firework midnight millennium ...	4
9	parade night millennium family millennium mill...	4

In [ ]:
```
######################## Evaluate Results ########################
```

In [39]:
```
dbi_score = metrics.davies_bouldin_score(train_corpus_vectorized.toarray(), pred_labels)
print("Davies Bouldin Score:", dbi_score)
silhouet_score = metrics.silhouette_score(train_corpus_vectorized.toarray(), pred_labels ,
                                          metric='euclidean')
print("Silhoutte Score:", silhouet_score)
```

```
Davies Bouldin Score: 4.5031274856130254
Silhoutte Score: 0.03075543284802214
```

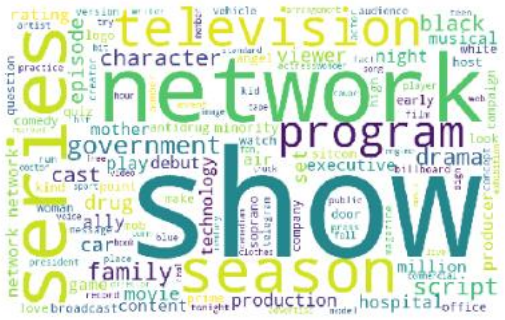
In [40]:
```
#Plot wordcloud
def plot_wordcloud(text):
    wordCloud = WordCloud(width = 800, height = 500, min_font_size = 14,
                          background_color ='white').generate(text)
    plt.figure(figsize = (8, 6))
    plt.imshow(wordCloud)
    plt.axis("off")
    plt.show()
```

In [41]:
```
#Plot wordcloud
for i in df_result.labels.unique():
    df_plt=df_result[df_result.labels==i]
    text=" ".join(df_plt.text.tolist())
    print(i)
    print('-  -  -  -  -  -  -  -  -  -  -  -')
    plot_wordcloud(text)
```

```
10
-  -  -  -  -  -  -  -  -  -  -  -
```


```
4
-  -  -  -  -  -  -  -  -  -  -  -
```

8



15

Latent Dirichlet Allocation (LDA)

In [1]:
```
import pandas as pd
import random
from time import time
```

In [2]:
```
from nltk.stem import WordNetLemmatizer
from nltk import pos_tag
```

In [3]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
```

In [4]:
```
import matplotlib.pyplot as plt
```

In [5]:
```
#Read document data
col_names=['docID', 'wordID', 'count']
df_dataText=pd.read_csv('NewYorkTimes/docword.nytimes/docword.nytimes.txt', header=None, delim_whitespace=True, skiprows=3,
                        names=col_names)
#check the first 5 rows of data
df_dataText.head()
```

Out[5]:

	docID	wordID	count
0	1	413	1
1	1	534	1
2	1	2340	1
3	1	2806	1
4	1	3059	1

In [6]:
```
#Read vocabulary data
col_Vnames=['vocab_word']
df_dataVocab=pd.read_csv('NewYorkTimes/vocab.nytimes.txt',header=None, delim_whitespace=True,names=col_Vnames)
df_dataVocab.head()
```

Out[6]:

	vocab_word
0	aah
1	aahed
2	aaron
3	aback
4	abacus

In [7]:
```
#Add n(id number) column to vocab data
df_dataVocab['n']=df_dataVocab.index+1
df_dataVocab.head()
```

Out[7]:

	vocab_word	n
0	aah	1
1	aahed	2
2	aaron	3
3	aback	4
4	abacus	5

In [8]:
```
#function to collect words for each document
def get_docWords(docId):
    c1=df_dataText['docID']==docId
    c2=df_dataText['count']>=3#count greater than 3
    df_filtered=df_dataText[(c1)&(c2)][['wordID','count']].reset_index(drop=True)
    df_filtered=df_filtered.merge(df_dataVocab,left_on=['wordID'], right_on=['n'],how='inner')[['vocab_word','count']]
    return df_filtered
```

In [9]:
```
#Lemmatize each word
def lemmatize_words(lst_tag_words):
    lst_words_lemmatized=[]
    for w,t in lst_tag_words:
        tg=t[0].lower()
        if tg not in ['a', 'r', 'n', 'v']:
            lst_words_lemmatized.append(w)
        else:
            lst_words_lemmatized.append(obj_lemmatizer.lemmatize(w,tg))
    return lst_words_lemmatized
```

In [10]:
```
#Create cleaned text dataset in order to feed into clustering algorithms
obj_lemmatizer=WordNetLemmatizer()
train_corpus=[]
for docId in range(1,5001):
    isSuccess=True
    #get words in a document
    df_doc_words=get_docWords(docId)
    try:
        #Assign POS tag for each word, Lemmatizer requires correct POS tag to accurately lemmatize words
        df_doc_words['pos_tag']= pos_tag(df_doc_words['vocab_word'])
        #Group different inflections of the same word.
        df_doc_words['lemmatized_words']=lemmatize_words(df_doc_words['pos_tag'].tolist())
    except:
        isSuccess=False
    if isSuccess:
        txt_sentnc=''
        for i,row in df_doc_words.iterrows():
            if row['lemmatized_words']!='':
                w=row['lemmatized_words']
                if w[:4]!='zzz_':
                    txt_sentnc+=' '+(w+' ')*row['count']
        txt_sentnc=txt_sentnc.strip()
        if txt_sentnc!='':
            txt_sentnc=txt_sentnc.split()
            random.shuffle(txt_sentnc)
            txt_sentnc=' '.join(txt_sentnc)
            train_corpus.append(txt_sentnc)
```

In [11]:
```
#Print first five sentences
train_corpus[0:5]
```

Out[11]:
```
['rank win team team scored scored play point scored half game half player play point point play scored game team team win scor
ed rank rank night play rank play win player game night team game night point half player half play',
 'number paper front issue error number issue error issue number issue issue front number paper error paper number paper number
number issue issue issue front',
 'security celebration terrorism crowd light friend friend firework friend crowd leave terrorism celebration celebration securi
ty midnight leave celebration light crowd firework leave midnight security midnight crowd terrorism light crowd firework',
```

In [12]:
```
#print no. of elements in train_corpus
len(train_corpus)
```

Out[12]:
```
4915
```

In [26]:
```
#Plot topics found with top number of words in it
def plot_top_words(model, feature_names, n_top_words):
    fig, axes = plt.subplots(3, 6, figsize=(30, 20), sharex=True)
    axes = axes.flatten()
    for topc_indx, topic in enumerate(model.components_):
        top_features_ind = topic.argsort()[: -n_top_words - 1 : -1]
        top_features = [feature_names[i] for i in top_features_ind]
        weights = topic[top_features_ind]
        ax = axes[topc_indx]
        ax.barh(top_features, weights, height=0.7)
        ax.set_title(f"Topic {topc_indx +1}", fontdict={"fontsize": 30})
        ax.invert_yaxis()
        ax.tick_params(axis="both", which="major", labelsize=20)
        for i in "top right left".split():
            ax.spines[i].set_visible(False)
        fig.suptitle("Topics found with LDA model", fontsize=40)
    plt.subplots_adjust(top=0.90, bottom=0.05, wspace=0.90, hspace=0.3)
    plt.show()
```

In [14]:
```
########################   Vectorization  ########################
```

In [15]:
```
#Create vector of words for each document using CountVectorizer
#max_df=0.95; Ignore terms that appear in more than 95% of the documents
#min_df=2; Ignore terms that appear in less than 2 document
time_start=time()
num_features = 5000
obj_CountVectorizer = CountVectorizer(max_df=0.95, min_df=2, max_features=num_features)
train_corpus_vectorized = obj_CountVectorizer.fit_transform(train_corpus)
time_taken=time()-time_start
print('Time(seconds) taken to vectorize text data:',time_taken)
```

```
Time(seconds) taken to vectorize text data: 0.528977632522583
```

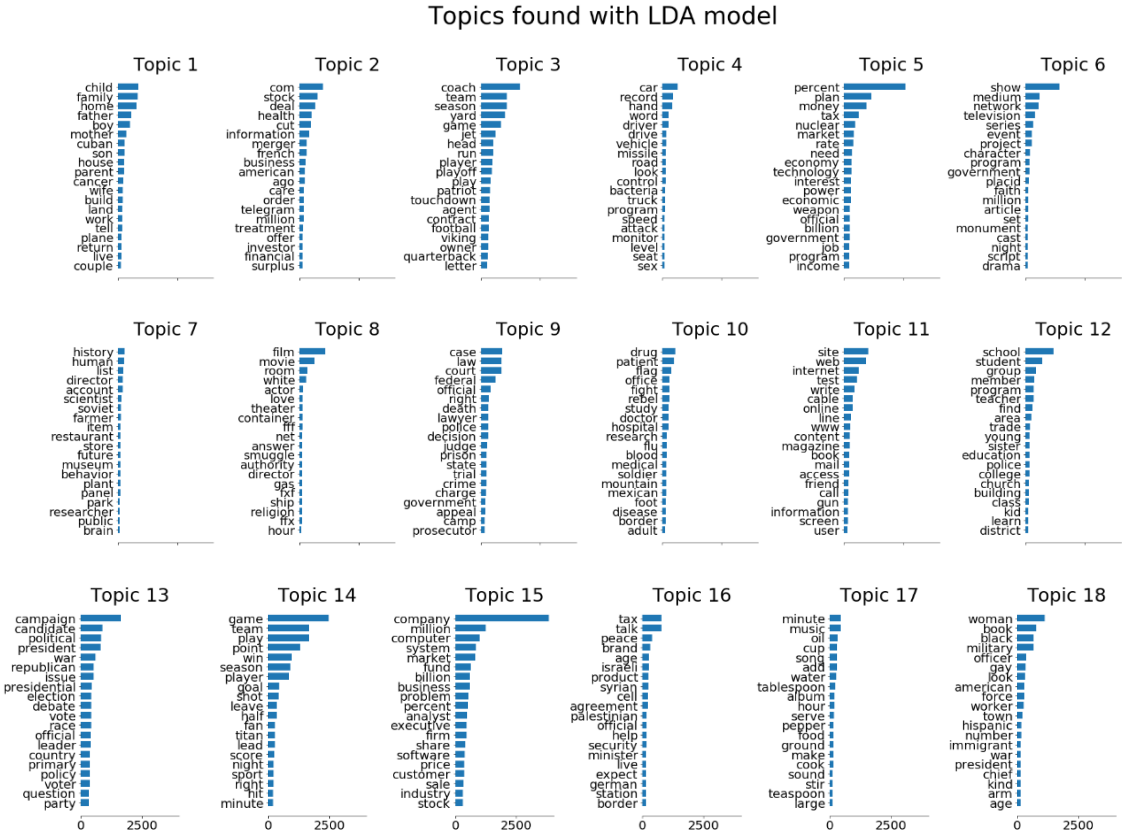
In [16]:
```
######################## Clustering Latent Dirichlet Allocation ########################
```

In [17]:
```
#Latent Dirichlet Allocation clustering model
time_start=time()
num_components = 18
objLDA_model = LatentDirichletAllocation(n_components=num_components, max_iter=5, learning_method="online",
    learning_offset=50.0, random_state=0)
objLDA_model.fit(train_corpus_vectorized)
time_taken=time()-time_start
print('Time(seconds) taken to train LDA model:',time_taken)
```

```
Time(seconds) taken to train LDA model: 4.977417707443237
```

In [18]:
```
######################## Evaluate Results ########################
```

In [27]:
```
#Plot topics found with top number of words in it
num_top_words=20
feature_names = obj_CountVectorizer.get_feature_names()
plot_top_words(objLDA_model, feature_names, num_top_words)
```



In [ ]:
```

```

Naïve Bayes and Support Vector Machines

In [1]:
```
import pandas as pd
import random
import numpy as np
import itertools
import matplotlib.pyplot as plt
from time import time
```

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer,TfidfTransformer
```

In [3]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn import metrics
from sklearn.metrics import f1_score, jaccard_score, accuracy_score, confusion_matrix
from sklearn.svm import LinearSVC
```

In [4]:
```
#get labelled data
df_data=pd.read_csv('corpus_labels.csv')
df_data=df_data[['corpus','category']]
df_data.rename(columns={'category':'label'}, inplace=True)
df_data.head()
```

Out[4]:

	corpus	label
0	play game half half game point scored play ran...	Sports
1	paper front front paper issue paper error numb...	Computers
2	firework security leave midnight midnight ligh...	Society
3	commercial traffic flight passenger midnight t...	Recreation
4	television television program program show pro...	Arts

In [5]:
```
#Split data to training and testing data
X_train, X_test, y_train, y_test = train_test_split(df_data['corpus'], df_data['label'],
                                                    test_size=0.2, random_state=4)
print ('Number of records in training data:', X_train.shape,  y_train.shape)
print ('Number of records in testing data:', X_test.shape,  y_test.shape)
```

```
Number of records in training data: (866,) (866,)
Number of records in testing data: (217,) (217,)
```

In [6]:
```
######################### Classification NAIVE BAYES ########################
```

In [7]:
```
#Feed data vectorizer, transformer and naive bayes classifier object to a pipeline
time_start=time()
obj_pipeline = Pipeline([('vect', TfidfVectorizer()), ('tfidf', TfidfTransformer()),
                         ('clf', MultinomialNB())])
obj_pipeline.fit(X_train, y_train)
time_taken=time()-time_start
print('Time(seconds) taken to train naive bayes model:',time_taken)
#Predict values using test data
predicted_values = obj_pipeline.predict(X_test)
print('Accuracy Score: %s' % accuracy_score(predicted_values, y_test))
print('Classification Report')
print(metrics.classification_report(y_test, predicted_values))
```

```
Time(seconds) taken to train naive bayes model: 0.09142184257507324
Accuracy Score: 0.7188940092165899
Classification Report
              precision    recall  f1-score   support

        Arts       1.00      0.42      0.59        24
    Business       0.94      0.62      0.75        24
   Computers       0.93      0.81      0.87        16
       Games       0.00      0.00      0.00         2
      Health       1.00      0.14      0.25         7
        Home       1.00      0.40      0.57         5
  Recreation       0.00      0.00      0.00        13
     Science       0.00      0.00      0.00         3
     Society       0.60      1.00      0.75        82
      Sports       0.87      0.80      0.84        41

    accuracy                           0.72       217
   macro avg       0.63      0.42      0.46       217
weighted avg       0.73      0.72      0.68       217
```

In [8]:
```
#predicted values
predicted_values[0:5]
```

Out[8]:
```
array(['Arts', 'Business', 'Society', 'Society', 'Society'], dtype='<U10')
```

In [9]:
```
#f1-score
print('f1-score')
f1_score(y_test, predicted_values, average='weighted')
```

```
f1-score
```

Out[9]:
```
0.6752658734289885
```

In [10]:
```
#jaccard-score
print('jaccard-score')
jaccard_score(y_test, predicted_values, average='weighted')
```

```
jaccard-score
```

Out[10]:
```
0.5460346270345445
```

In [11]:
```
######################## Classification SVM ########################
```

In [12]:
```
#Feed data vectorizer, transformer and SVM classifier object to a pipeline
time_start=time()
objPipelineSvm = Pipeline([('vect', TfidfVectorizer()), ('tfidf', TfidfTransformer()),
                           ('clf', LinearSVC())])
objPipelineSvm.fit(X_train, y_train)
time_taken=time()-time_start
print('Time(seconds) taken to train SVM model:',time_taken)
#Predict values using test data
predicted_values = objPipelineSvm.predict(X_test)
print('Accuracy Score: %s' % accuracy_score(predicted_values, y_test))
print('Classification Report')
print(metrics.classification_report(y_test, predicted_values))
```

```
Time(seconds) taken to train SVM model: 0.07739925384521484
Accuracy Score: 0.8064516129032258
Classification Report
              precision    recall  f1-score   support

        Arts       0.67      0.75      0.71        24
    Business       0.88      0.92      0.90        24
   Computers       0.94      0.94      0.94        16
       Games       0.50      0.50      0.50         2
      Health       0.56      0.71      0.63         7
        Home       0.67      0.40      0.50         5
  Recreation       0.60      0.23      0.33        13
     Science       0.60      1.00      0.75         3
     Society       0.83      0.91      0.87        82
      Sports       0.89      0.76      0.82        41

    accuracy                           0.81       217
   macro avg       0.71      0.71      0.69       217
weighted avg       0.80      0.81      0.80       217
```

In [13]:
```
#f1-score
print('f1-score')
f1_score(y_test, predicted_values, average='weighted')
```

```
f1-score
```

Out[13]:
```
0.7968180946549042
```

In [14]:
```
#jaccard-score
print('jaccard-score')
jaccard_score(y_test, predicted_values, average='weighted')
```

```
jaccard-score
```

Out[14]:
```
0.6835288917997892
```