Topic Modelling using Word Embeddings and Latent Dirichlet Allocation

Saket Garodia

Jan 18 · 10 min read

*Extract **topics** from a million headlines using clustering (on embeddings) and **LDA** techniques*

Media, journals and newspapers around the world every day have to cluster all the data they have into specific topics to show the articles or news in a structured manner under specific **topics**. All the social network companies like **Facebook**, **Twitter**, etc do some sort of topic modeling on the posts and advertisements before using the recommendation engines to recommend stuff to users. Even **Google** runs topic modeling in their search to identify the documents relevant to the user search. Imagine having a digital library where the books are randomly placed irrespective of their topics. How difficult it will be to search for them or search for the books that belong to specific topics we are interested in. Fortunately, we have deep learning and analytical tools to rescue us from these situations.

In this project, I am going to extract topics from a million news headlines sourced from the reputable Australian news source ABC (Australian Broadcasting Corp.). The dataset is available in Kaggle.

https://www.kaggle.com/therohk/million-headlines

Dataset Content

publish_date: Date of publishing for the article in yyyyMMdd format

headline_text: Text of the headline in Ascii, English, lowercase

Start Date: 2003–02–19; End Date: 2019–12–31

We will explore this in two ways:

1) In the first case, we will create embeddings for each headlines using '**Google News 'wordtovec' embeddings**' which takes care of the semantic and meaning and **cluster** the headlines into 8 clusters and see the most frequent words in the different clusters

2) In the second case, we will use the **LDA** (Latent Dirichlet Allocation) method to model the topics from these headlines. LDA assumes that each headline is taken from several topics and each topic consists fo several words.

Now, let us start with importing some of the libraries.

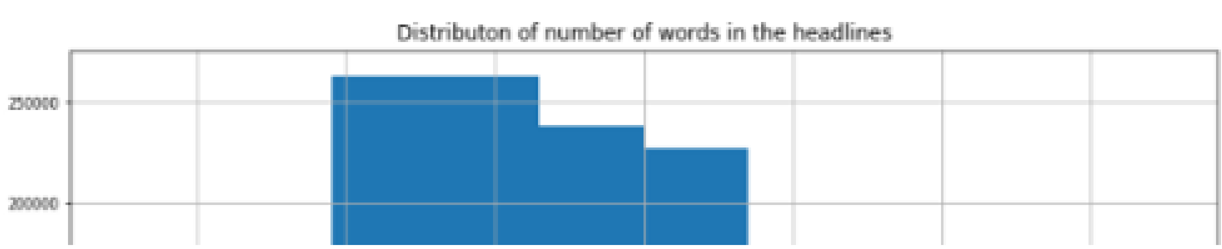
```
#importing libraries
```

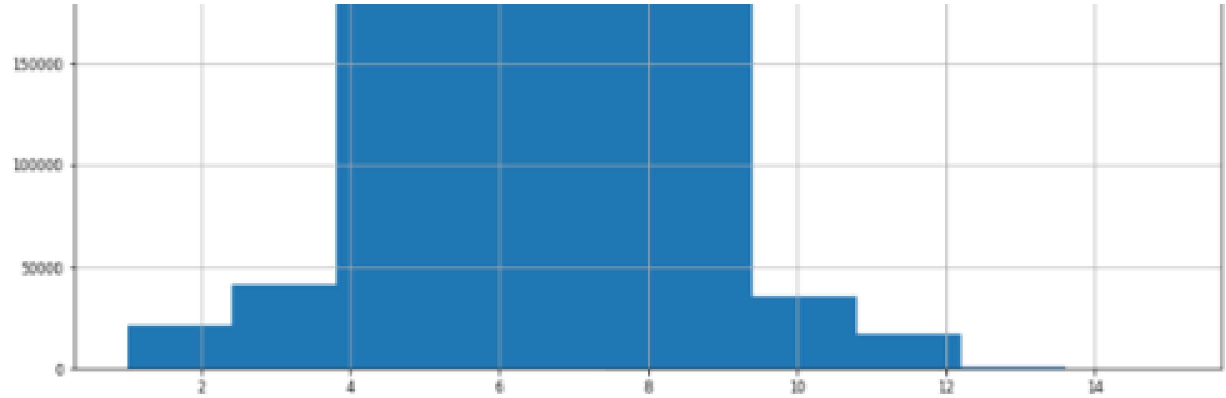
```
import numpy as np
import pandas as pd
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

headlines = pd.read_csv('abcnews-date-text.csv',parse_dates=[0],
infer_datetime_format=True)

headlines.head()
```

	publish_date	headline_text
0	2003-02-19	aba decides against community broadcasting lic...
1	2003-02-19	act fire witnesses must be aware of defamation
2	2003-02-19	a g calls for infrastructure protection summit
3	2003-02-19	air nz staff in aust strike for pay rise
4	2003-02-19	air nz strike to affect australian travellers

Now that we have imported our data, we will start will exploratory data analysis so that we have all the intuitions about what our data contains. Let’s start by creating a column that contains the length of each headline to get an intuition on the average number of words used in a headline.

```
headlines['NumWords'] = headlines['headline_text'].apply(lambda x:
len(x.split()))
headlines[['NumWords']].hist(figsize=(12, 6), bins=10, xlabelsize=8,
ylabelsize=8);
plt.title("Distributon of number of words in the headlines")
```

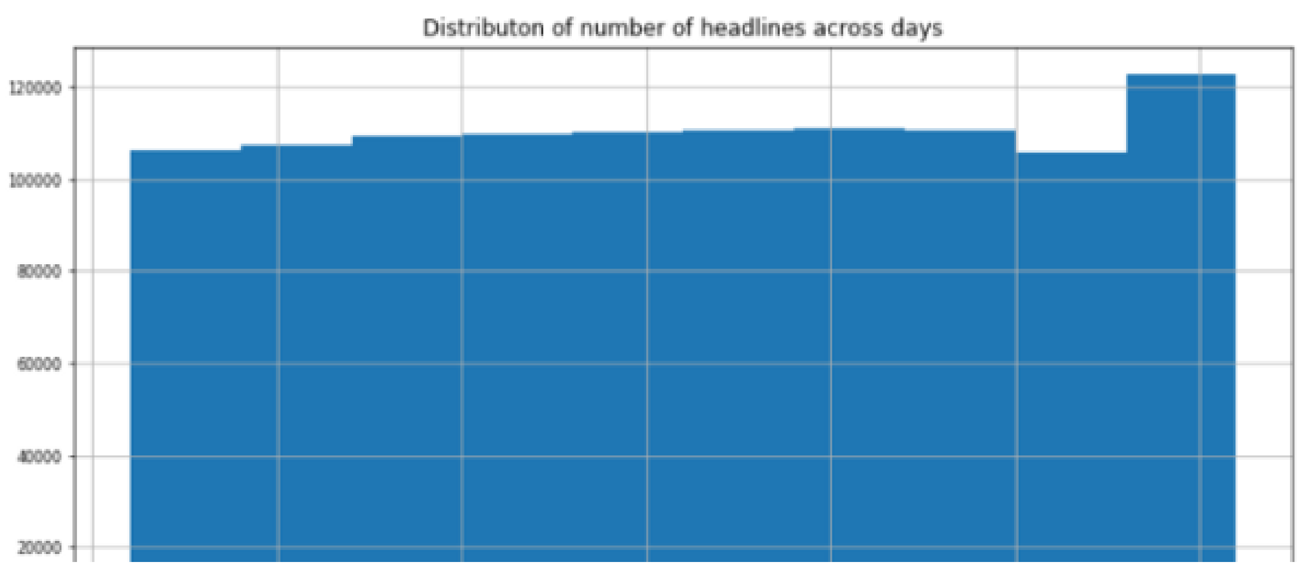


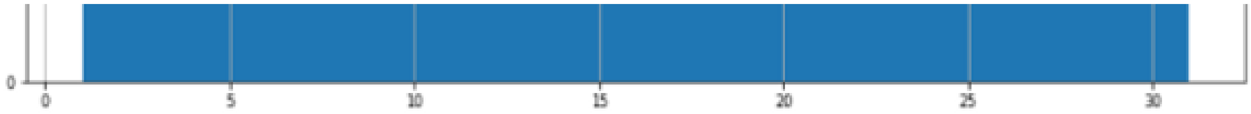


We can see that most of the headlines have around 4–6 words, Now let's also make a year, month and a day field to see the distribution of headlines across these attributes.

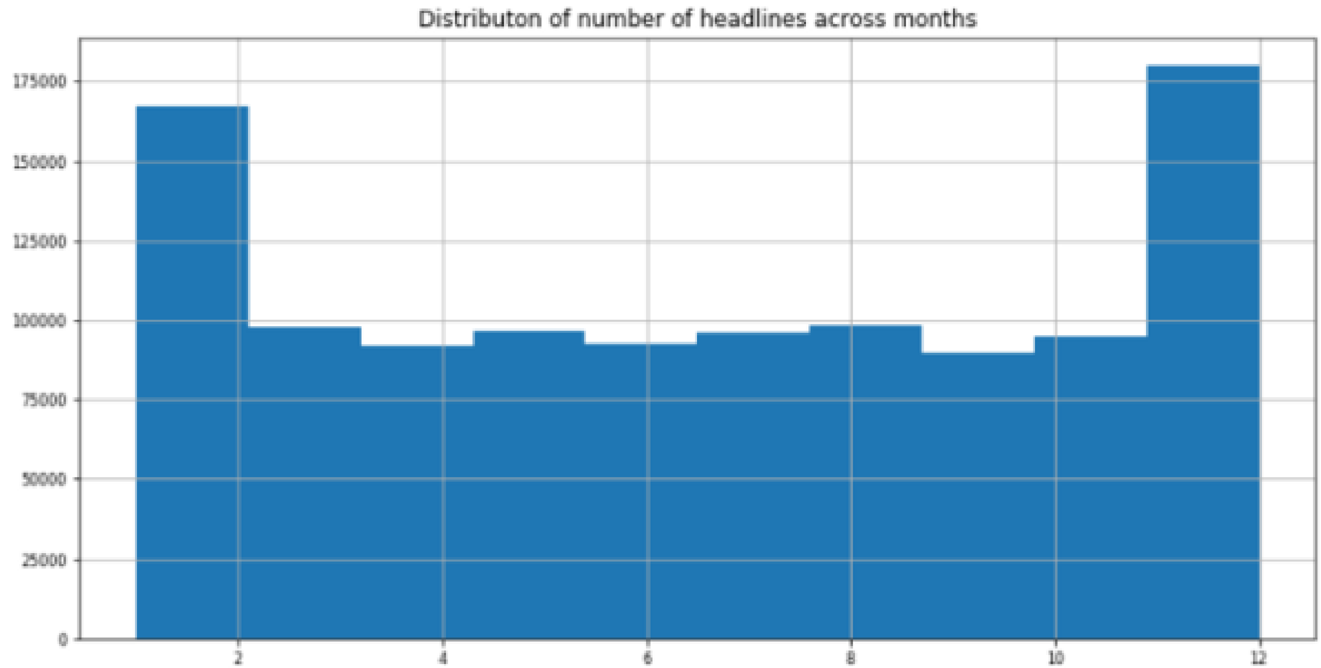
```
headlines['year'] = pd.DatetimeIndex(headlines['publish_date']).year
headlines['month'] =
pd.DatetimeIndex(headlines['publish_date']).month
headlines['day'] = pd.DatetimeIndex(headlines['publish_date']).day
```

	publish_date	headline_text	NumWords	year	month	day
0	2003-02-19	aba decides against community broadcasting lic...	6	2003	2	19
1	2003-02-19	act fire witnesses must be aware of defamation	8	2003	2	19
2	2003-02-19	a g calls for infrastructure protection summit	7	2003	2	19
3	2003-02-19	air nz staff in aust strike for pay rise	9	2003	2	19
4	2003-02-19	air nz strike to affect australian travellers	7	2003	2	19

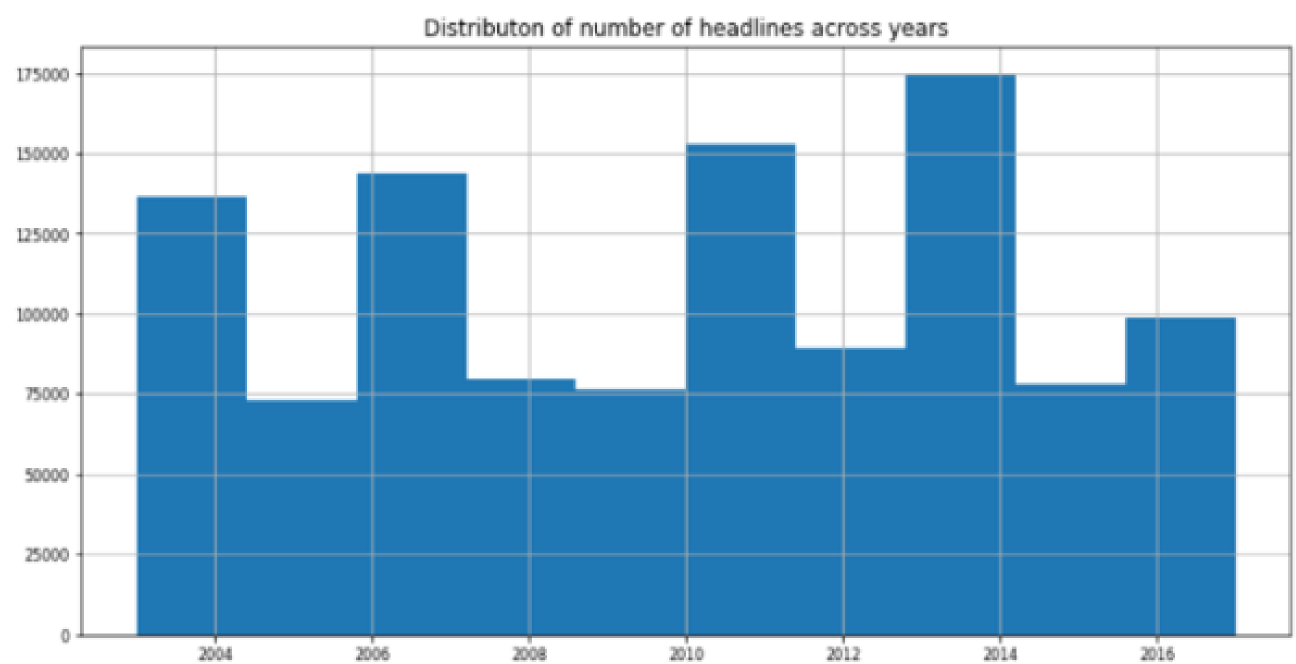




Seems like the distribution is uniform across the days.



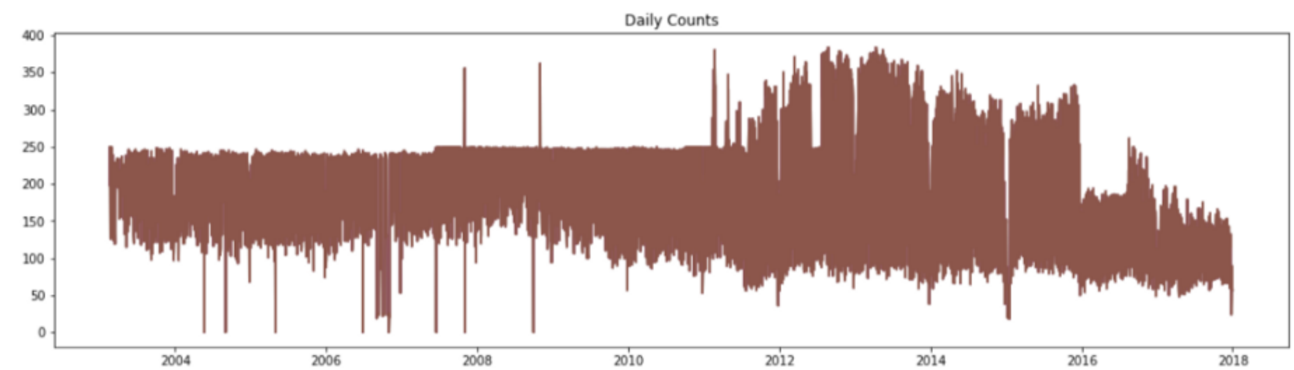
The start and the end of the year have contributed to most of the headlines in the dataset.

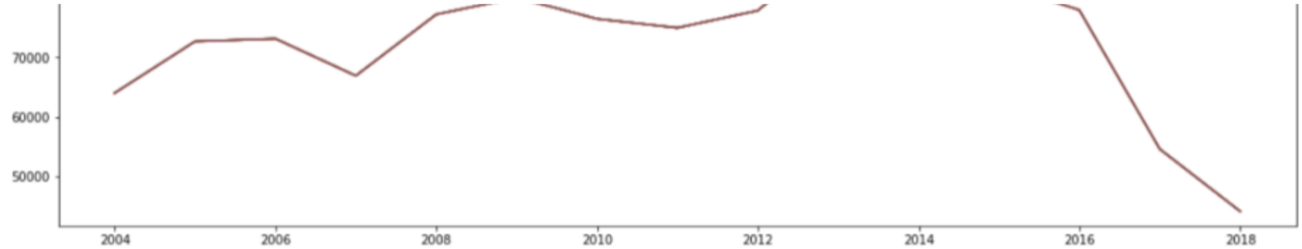


Similarly, we can see how the number of headlines is distributed across the years.

Now, let's visualize in a time-series pattern to see the everyday changes in the number of headlines. This will give us a better intuition and will be interesting to think about all the reasons that could have contributed to the spike during some years.

```
monthly_counts = headlines.resample('M').count()
yearly_counts = headlines.resample('A').count()
daily_counts = headlines.resample('D').count()

fig, ax = plt.subplots(3, figsize=(18,16))
ax[0].plot(daily_counts);
ax[0].set_title('Daily Counts');
ax[1].plot(monthly_counts);
ax[1].set_title('Monthly Counts');
ax[2].plot(yearly_counts);
ax[2].set_title('Yearly Counts');
plt.show()
```

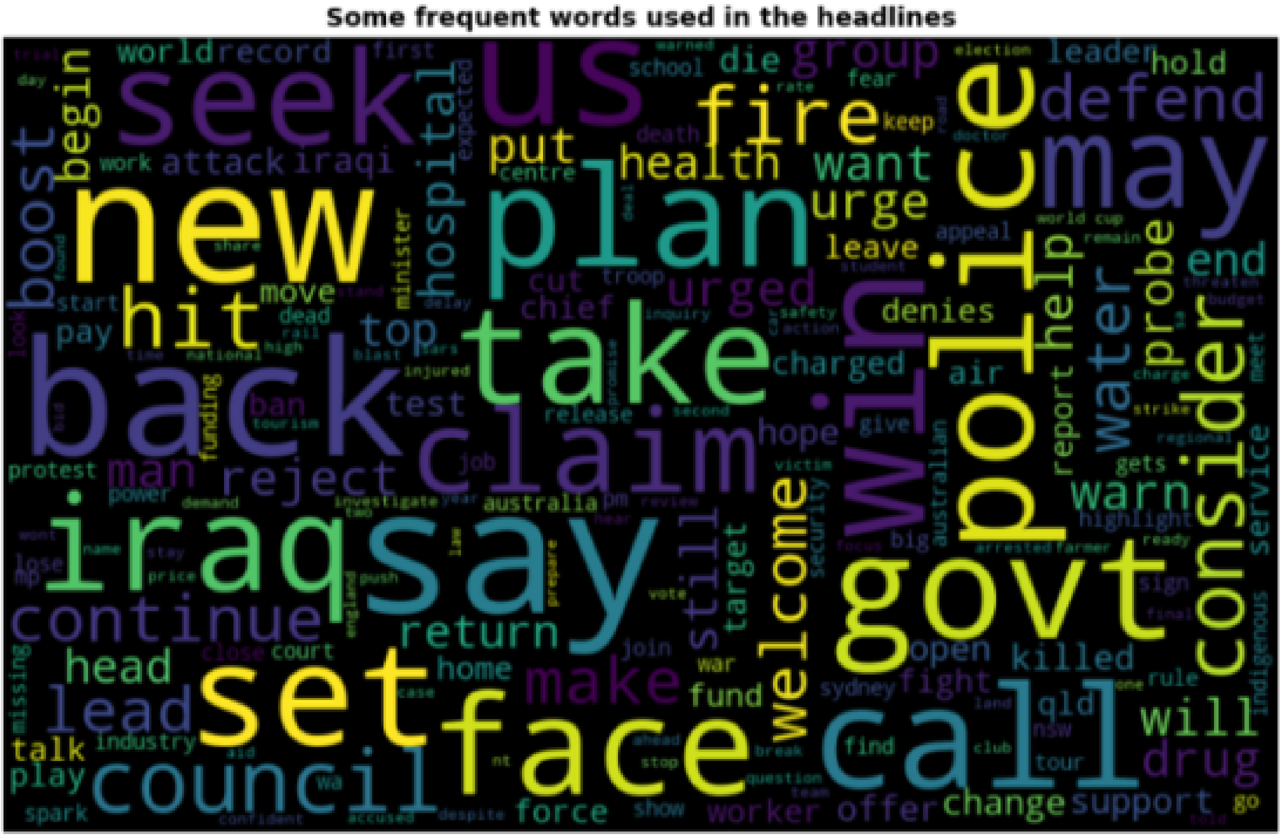




Now, let us build a word cloud to see the most frequent words that are being used in the headlines over the years.

```
from wordcloud import WordCloud
all_words = ''.join([word for word in headlines['headline_text']
[0:100000]])
all_words
wordcloud = WordCloud(width=800, height=500, random_state=21,
max_font_size=110).generate(all_words)
plt.figure(figsize=(15, 8))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.title("Some frequent words used in the headlines", weight='bold',
fontsize=14)
plt.show()
```

Some frequent words used in the headlines



The word cloud seems so interesting. In spite of the news channel belonging to Australia, we can see some frequent words like 'Iraq' and some other words like 'police', 'plan', 'health', 'council', etc.

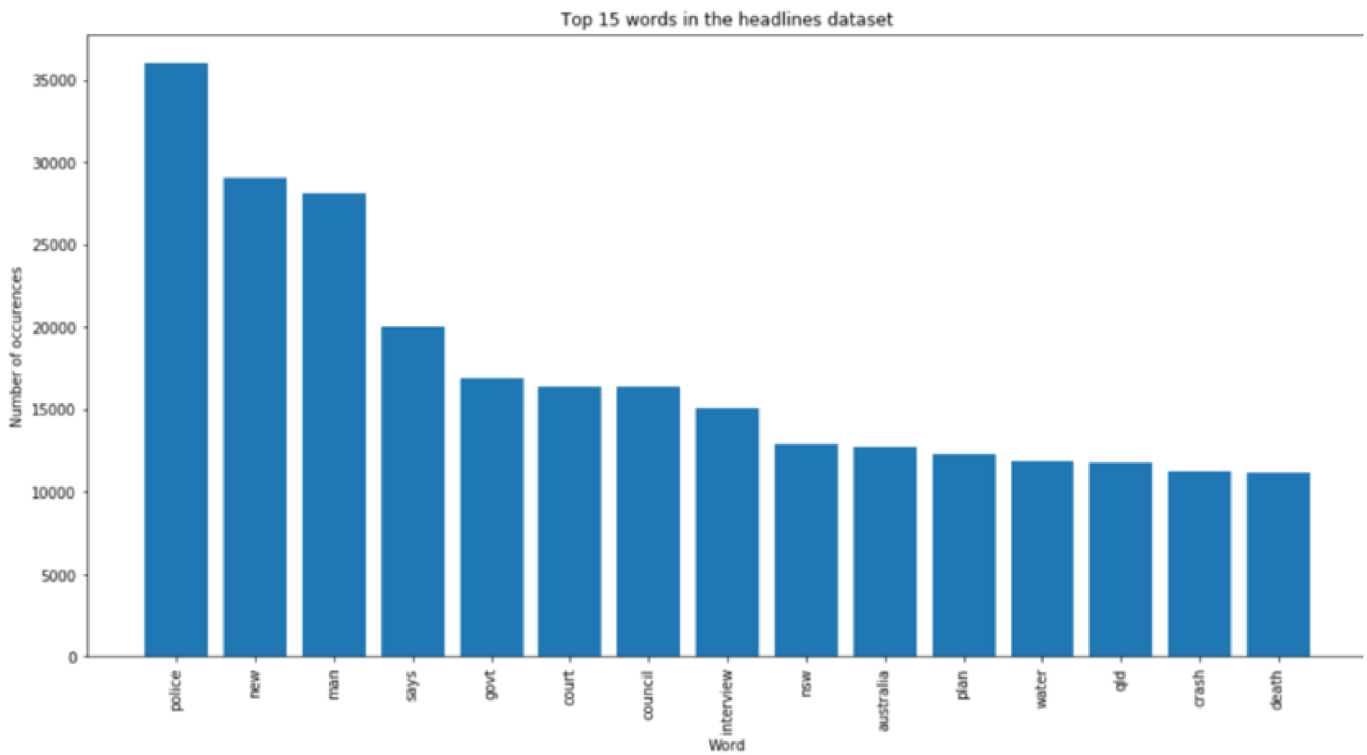
Now, let us move forward with performing some cleaning operations like turning each word to the lowercase font, removing the punctuations and non-ASCII characters which are irrelevant for modeling topics out of these headlines.

```
import re
NON_ALPHANUM = re.compile(r'[\W]')
NON_ASCII = re.compile(r'[^a-z0-1\s]')
def normalize_texts(texts):
  normalized_texts = ''
  lower = texts.lower()
  no_punctuation = NON_ALPHANUM.sub(r' ', lower)
  no_non_ascii = NON_ASCII.sub(r'', no_punctuation)
  return no_non_ascii

headlines['headline_text'] =
headlines['headline_text'].apply(normalize_texts)
headlines.head()
headlines['headline_text'] = headlines['headline_text'].apply(lambda
x: ' '.join([w for w in x.split() if len(w)>2]))
```

Let us draw one last plot for the top 15 words used with their frequencies.

```
def get_top_n_words(corpus, n=10):
  vec = CountVectorizer(stop_words='english').fit(corpus)
  bag_of_words = vec.transform(corpus)
  sum_words = bag_of_words.sum(axis=0)
  words_freq = [(word, sum_words[0, idx]) for word, idx in
vec.vocabulary_.items()]
  words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
  return words_freq[:n]

words = []
word_values = []
for i,j in get_top_n_words(headlines['headline_text'],15):
  words.append(i)
  word_values.append(j)
fig, ax = plt.subplots(figsize=(16,8))
ax.bar(range(len(words)), word_values);
ax.set_xticks(range(len(words)));
ax.set_xticklabels(words, rotation='vertical');
ax.set_title('Top 15 words in the headlines dataset');
```

```
ax.set_xlabel('Word');
ax.set_ylabel('Number of occurences');
plt.show()
```



Method 1: Clustering using 'wordtovec' embeddings

Now, let's start with our first method. We will import the word embeddings from the pre-trained deep NN on google news and then represent each headline with the mean of word embeddings for each word in that headline. Hold on if it seems complicated.

```
pip install --upgrade gensim
#importing wordtovec embeddings
from gensim.models import KeyedVectors
pretrained_embeddings_path = "https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz"

word2vec = 
KeyedVectors.load_word2vec_format(pretrained_embeddings_path, 
binary=True)
```

Let us see how a word is represented in its embedding format.

```
word = 'iraq'
print('Word: {}'.format(word))
print('First 20 values of embedding:\n{}'.format(word2vec[word]
[:20]))
```

```
Word: iraq
First 20 values of embedding:
[-0.27539062  0.13574219 -0.15332031  0.11962891 -0.25585938  0.00793457
  0.04638672 -0.35546875 -0.11474609  0.32617188  0.05859375 -0.33203125
 -0.36914062  0.04321289  0.25585938  0.18261719 -0.15527344 -0.171875
 -0.11230469 -0.20507812]
```

Let me show you the beauty of the word embeddings first. It captures synonyms, antonyms and all the logical analogies which humans can understand. If someone asks you "What is woman+king-man, the first thing which comes to our mind will be queen. Now let's see what 'wordtovec' embeddings give as the most similar answers to this question.

```
print(word2vec.most_similar(positive=['woman', 'king'], negative=
['man'], topn=3))

print(word2vec.most_similar(positive=['Tennis', 'Ronaldo'], negative=
['Soccer'], topn=3))
```

```
[('queen', 0.7118192911148071), ('monarch', 0.6189674139022827), ('princess', 0.5902431011199951)]
[('Nadal', 0.6514425277709961), ('Safin', 0.6181677579879761), ('Federer', 0.6156208515167236)]
```

Well, it is quite intelligent to give queen.

Now, we will randomly sample 20% of the data because of the memory constraints and then build the clustering model using the word embeddings we just imported.

```
X_train = pd.DataFrame(X)
headlines_smaller = X_train.sample(frac = 0.2, random_state= 423)
headlines_smaller.columns = ['head_line']

class WordVecVectorizer(object):
    def __init__(self, word2vec):
```

```
        self.word2vec = word2vec
        self.dim = 300

    def fit(self, X, y):
        return self

    def transform(self, X):
        return np.array([
            np.mean([self.word2vec[w] for w in texts.split() if w in
self.word2vec]
                    or [np.zeros(self.dim)], axis=0)
            for texts in X
        ])

#representing each headline by the mean of word embeddings for the
words used in the headlines.

wtv_vect = WordVecVectorizer(word2vec)
X_train_wtv = wtv_vect.transform(headlines_smaller.head_line)
print(X_train_wtv.shape)
```

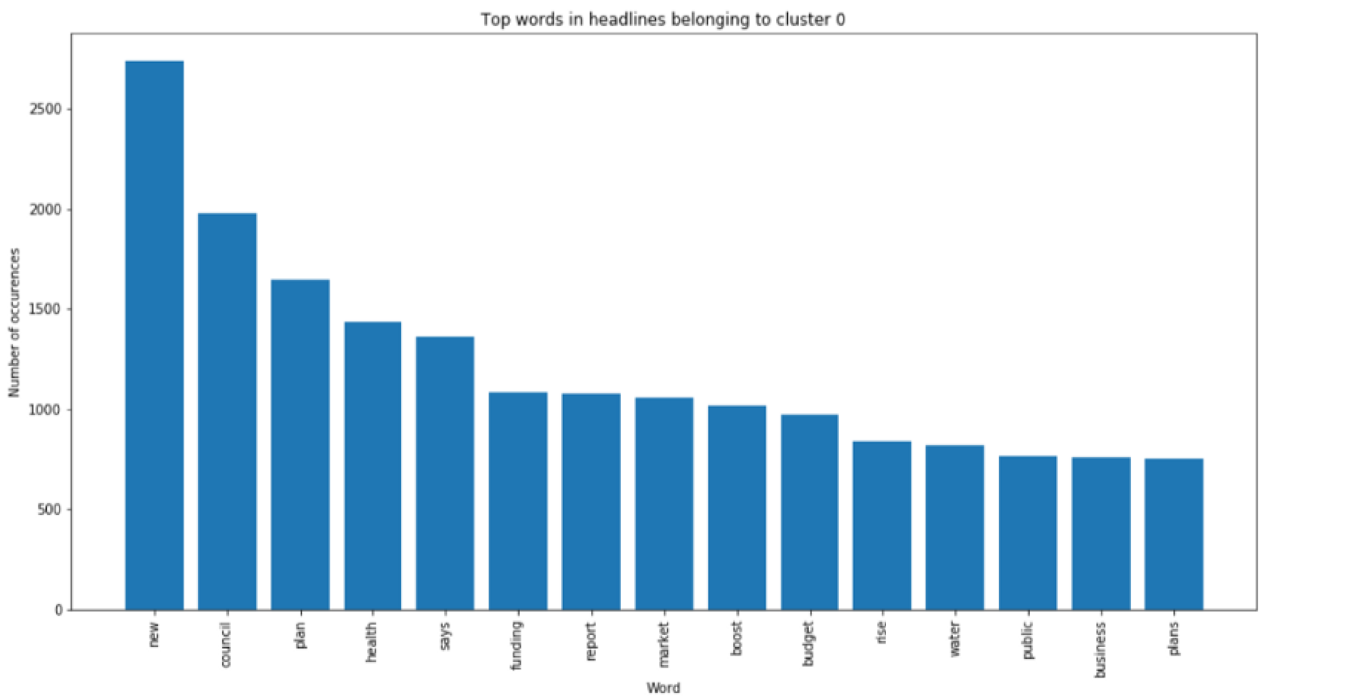
Now we have 220733 headlines and each headline has 300 features. Let us use KMeans CLustering to cluster them into 8 clusters.

```
from sklearn.cluster import KMeans

km = KMeans(
    n_clusters=8, init='random',
    n_init=10, max_iter=300,
    tol=1e-04, random_state=0
)
y_km = km.fit_predict(X_train_wtv)

df = pd.DataFrame({'headlines' :headlines_smaller.head_line,
'topic_cluster' :y_km })
```

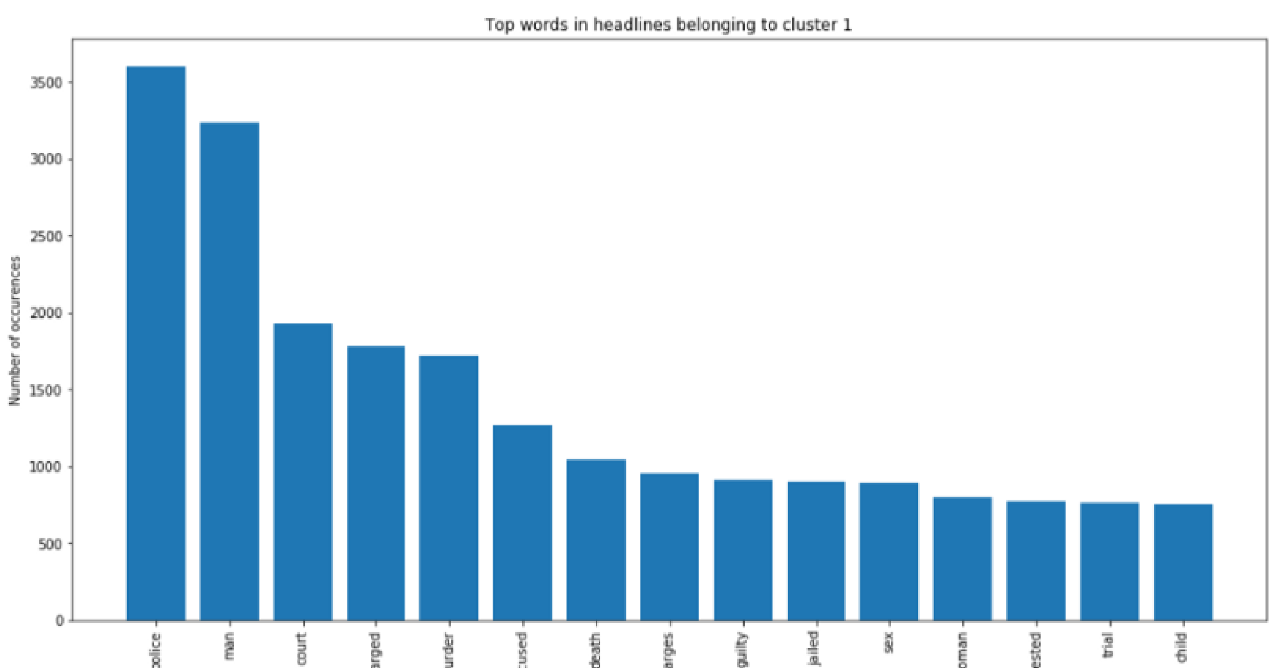
	headlines	topic_cluster
35301	man pleads guilty possessing child porn	1
559511	victorian ski resorts enjoy record snowfalls	4
338538	rod stewarts son pleads not guilty assault	1
285024	crusaders hold off committed reds	7

285024	crusaders hold off committed reds	7
171356	williams signs one year deal with swans	7

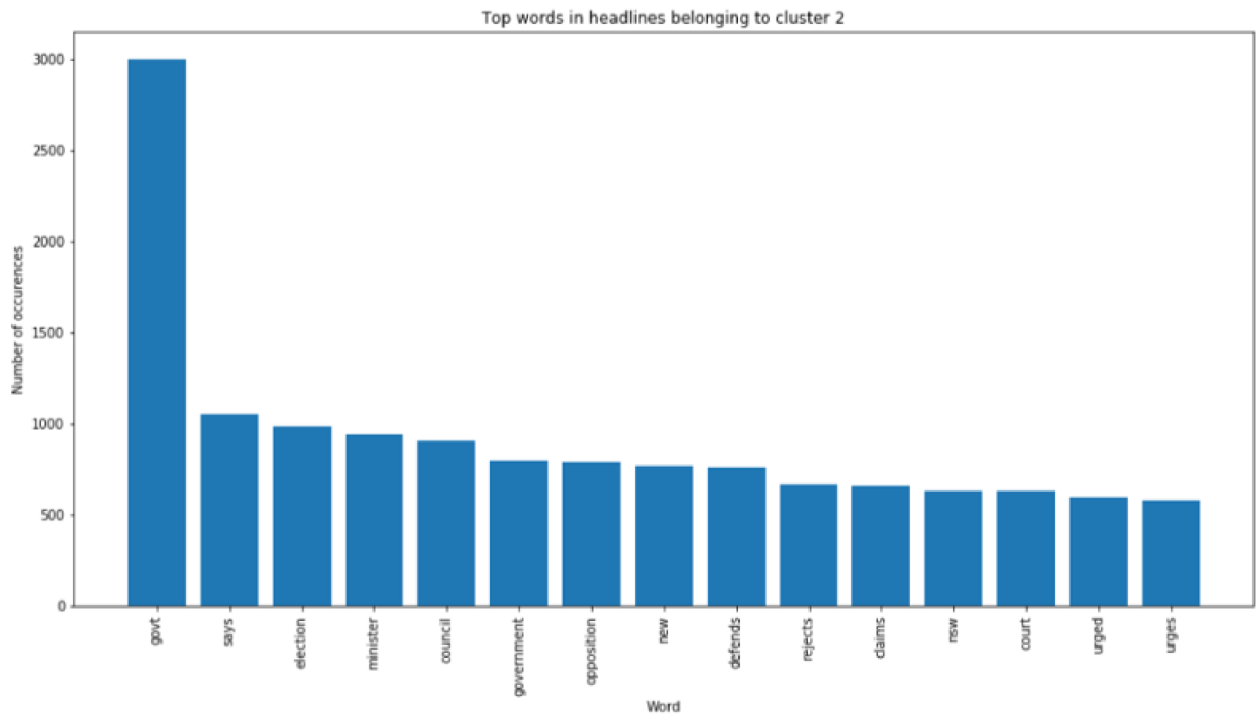
Let us visualize the top 15 words in each cluster and think about the topic they might be representing.



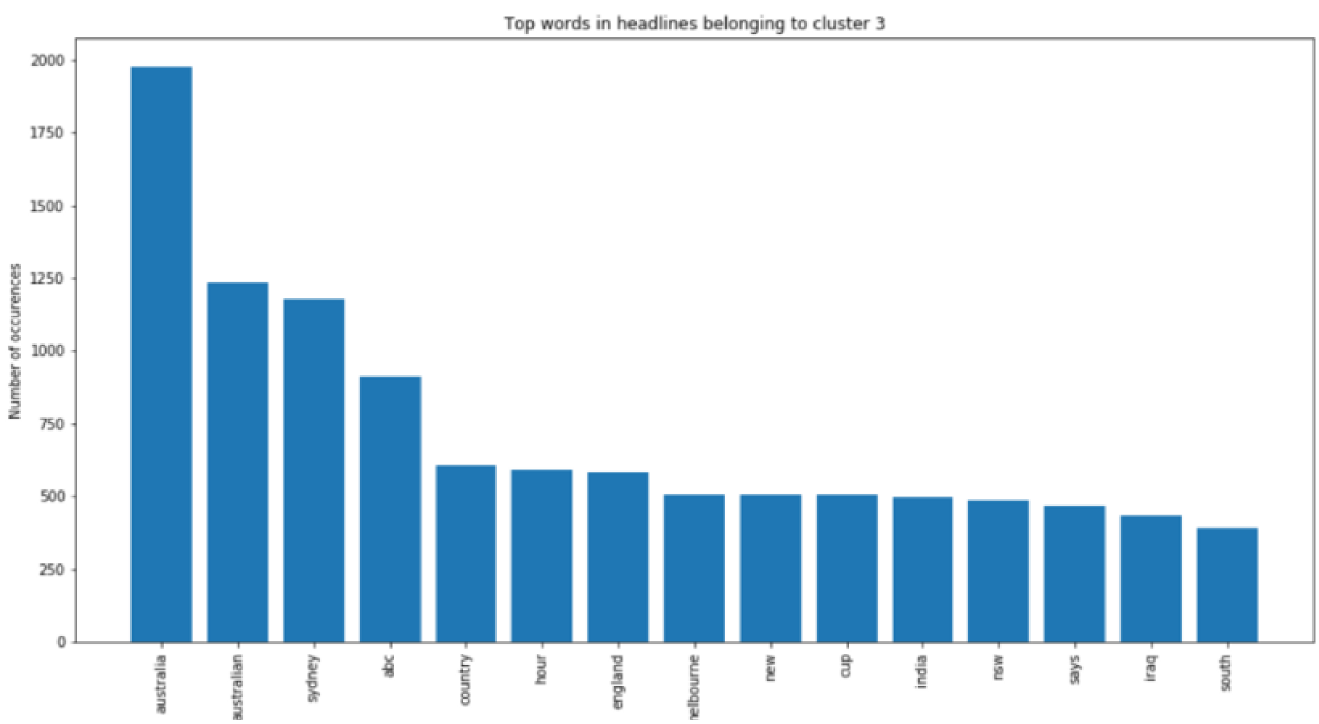
Cluster 0 represents words like ‘plan’, ‘health’, ‘budget’, etc and it seems like this belongs to the Economics or Business section of the news.



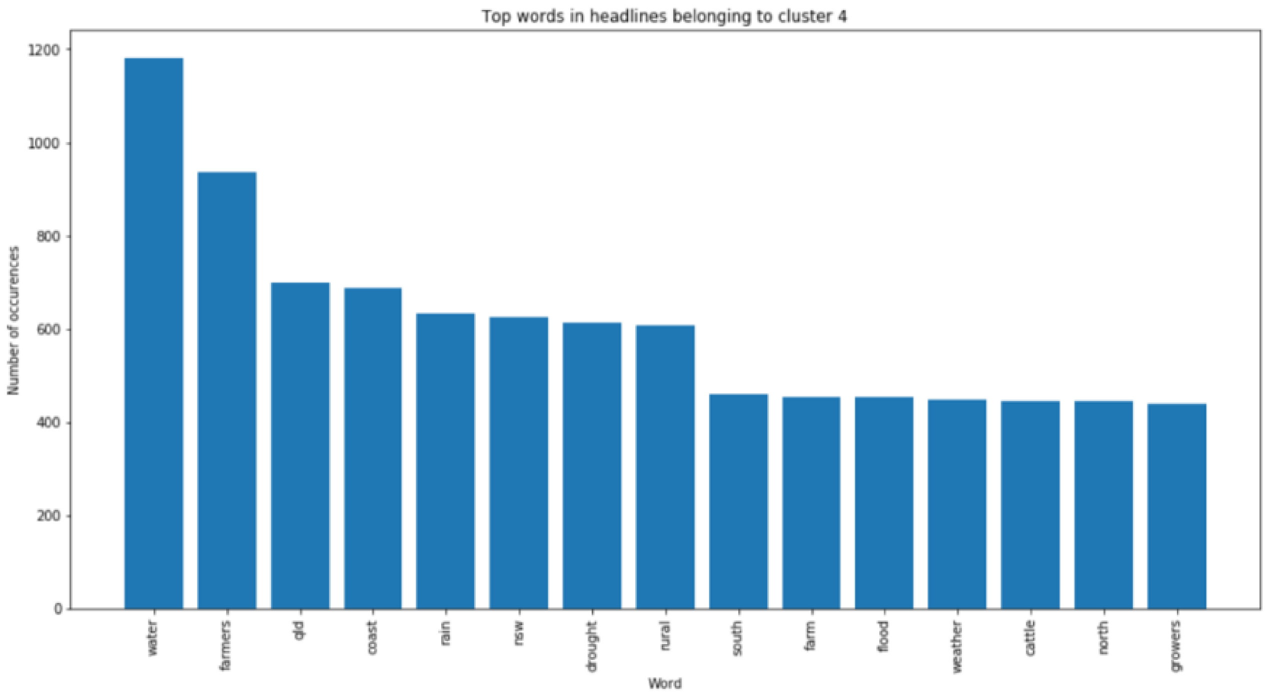
Cluster 1 represents words like ‘police’, ‘murder’, ‘death’, etc and it seems like this belongs to the Crime section of the news.



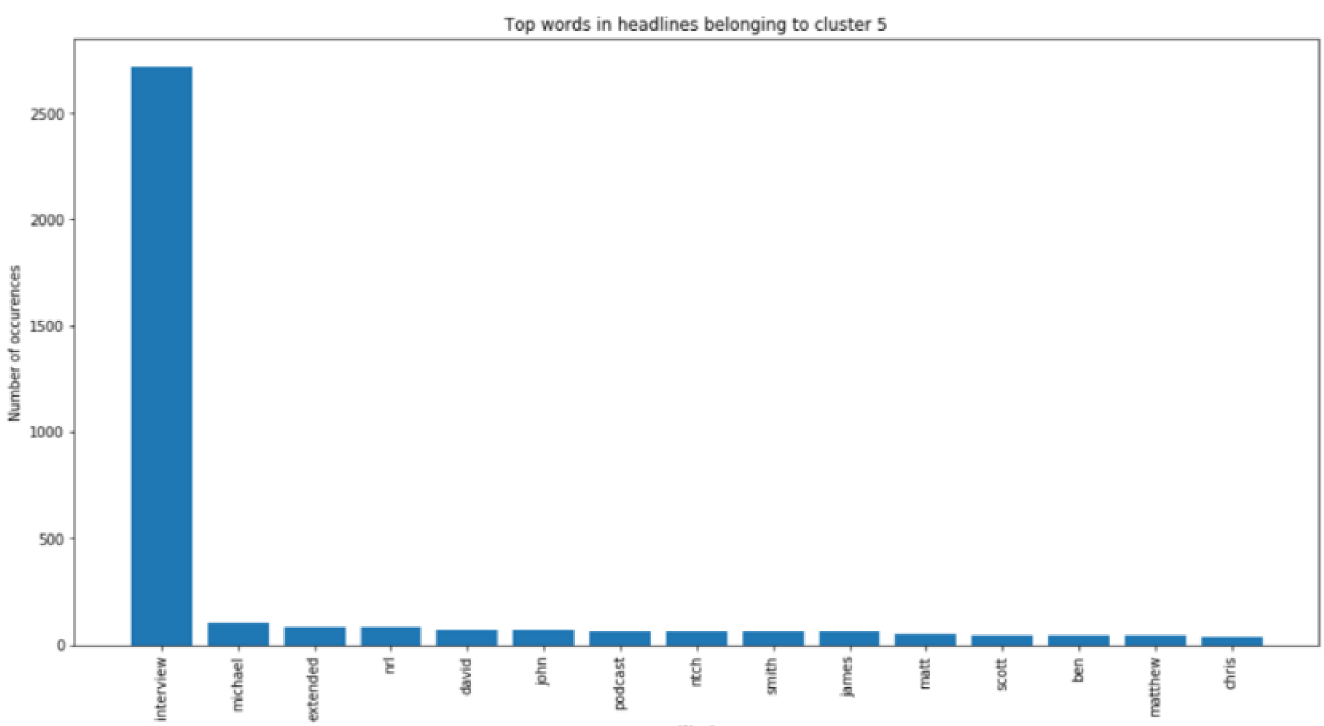
Cluster 2 represents words like ‘election’, ‘minister’, ‘court’ etc and it seems like this belongs to the Politics section of the news.



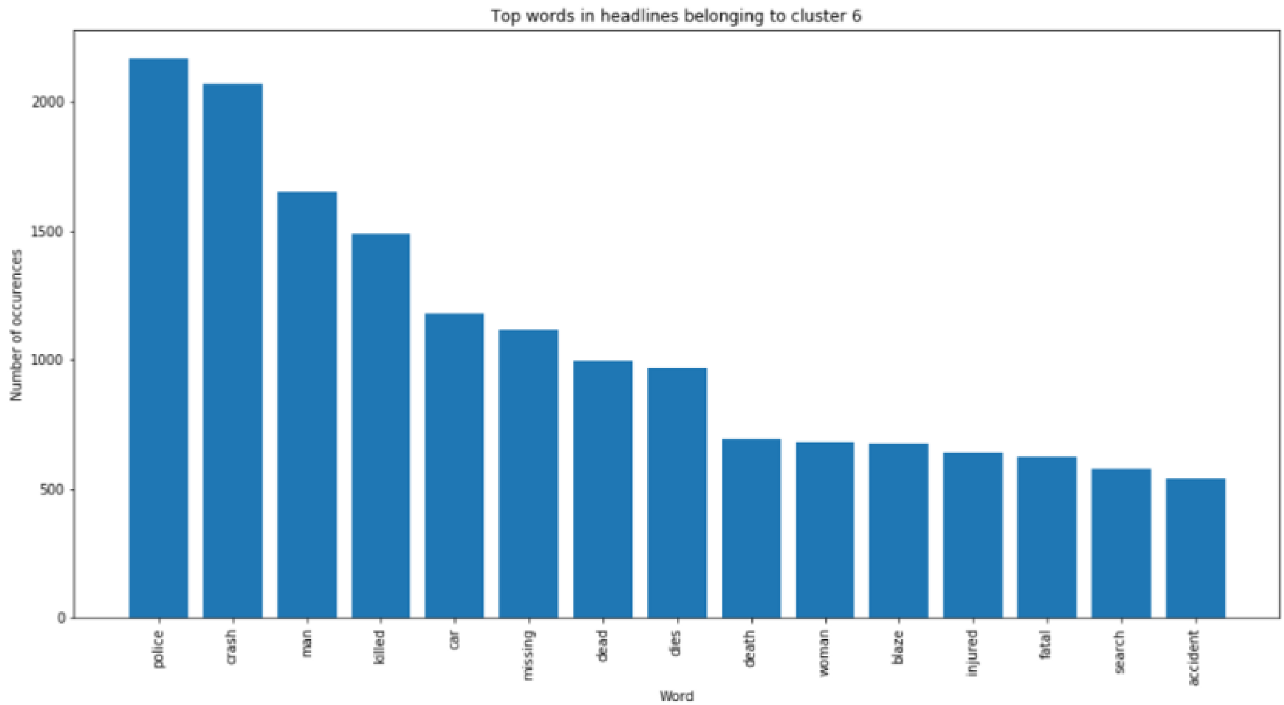
Cluster 3 represents words like ‘Australia’, ‘England’, ‘India’, etc and seems like this belongs to the Global section of the news.



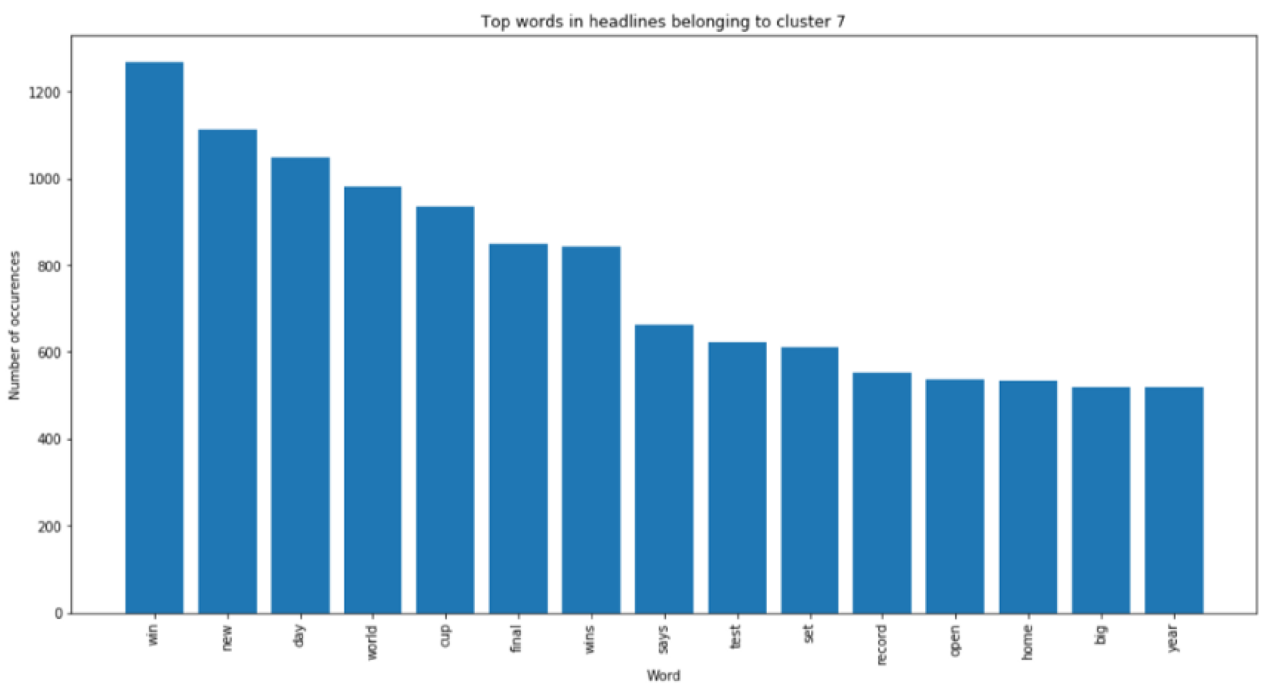
Cluster 4 represents words like ‘rural’, ‘water’, ‘flood’, etc and seems like this belongs to the Economics section but more towards micro issues.



Cluster 5 represents words like 'Michael', 'scoot', 'smith', etc and it seems like this belongs to the Interview or people section of the news.



Cluster 6 represents words like 'crash', 'killed', 'missing', etc and seems like this belongs to the accidents or Current happenings section of the news.



Cluster 7 represents words like 'open', 'wins', 'final', etc and it seems like this belongs to the Sports section of the news.

We can see that word2vec embeddings have led us to some random news to news belonging to specific topics in a very intelligent way. Now, let us move on to Method 2.

Method 2: Clustering using LDA (Latent Dirichlet Analysis)

LDA is a probabilistic method to extract the topics from documents. It assumes that each document is made up of several topics with a different probability distribution and each topic is made up of several words with a different distribution. So it works by initializing random topics to each word in each document and works in a reverse manner to discover the topics which would have generated these words in documents. To gain a high-level intuition, read this blog — https://medium.com/@pratikbarhate/latent-dirichlet-allocation-for-beginners-a-high-level-intuition-23f8a5cbad71

Since LDA has a lot of computations, we will sample 2% of the data and perform the analysis which might not lead to very intelligent topics but it will give us a high-level understanding of what LDA does.

```
news = headlines.sample(frac = 0.02, random_state= 423)
```

	index	headline_text	NumWords	year	month	day
0	35301	man pleads guilty possessing child porn	7	2003	8	7
1	559511	victorian ski resorts enjoy record snowfalls	6	2010	8	26
2	338538	rod stewarts son pleads not guilty assault	8	2007	10	19
3	285024	crusaders hold off committed reds	5	2007	2	10
4	171356	williams signs one year deal with swans	7	2005	6	22

We will build our features using TfIdf vectorizer which is similar to the bag-of-word model with the only difference being 'tfidf' penalizes the words which are present in several documents. Now, Let's fit the LDA model and see what topics LDA extracted using the top 15 words for each topic.

```
tf_vectorizer = TfidfVectorizer(stop_words='english',
max_features=50000)
news_matrix = tf_vectorizer.fit_transform(news['headline_text'])
#importing LDA

from gensim import corpora, models
from sklearn.decomposition import LatentDirichletAllocation

#Fitting LDA

lda = LatentDirichletAllocation(n_components=8,
learning_method='online',
                                          random_state=0, verbose=0,
n_jobs = -1)
lda_model = lda.fit(news_matrix)

lda_matrix = lda_model.transform(news_matrix)

lda_matrix
```

```
array([[0.03685454, 0.03685145, 0.03686212, ..., 0.03685514, 0.74201929,
        0.03685145],
       [0.03654849, 0.03654849, 0.03654848, ..., 0.03657858, 0.03654849,
        0.03654848],
       [0.03669439, 0.0366944 , 0.03669439, ..., 0.03669439, 0.2368973 ,
        0.03669438],
       ...,
       [0.04179878, 0.55889824, 0.04179878, ..., 0.04179878, 0.04179878,
        0.04179878],
       [0.39602019, 0.05226958, 0.29036235, ..., 0.05226958, 0.05226958,
        0.05226957],
       [0.03686059, 0.33620426, 0.31799028, ..., 0.16148106, 0.03686773,
        0.03686059]])
```

```
def print_topics(model, count_vectorizer, n_top_words):
    words = tf_vectorizer.get_feature_names()
    for topic_idx, topic in enumerate(model.components_):

        print("\nTopic #%d:" % topic_idx )
        print(" ".join([words[i]
```

```
                        for i in topic.argsort()[:-n_top_words -
1:-1]]))

# Print the topics found by the LDA model
print("Topics found via LDA:")
print_topics(lda_model, news_matrix, 15)
```

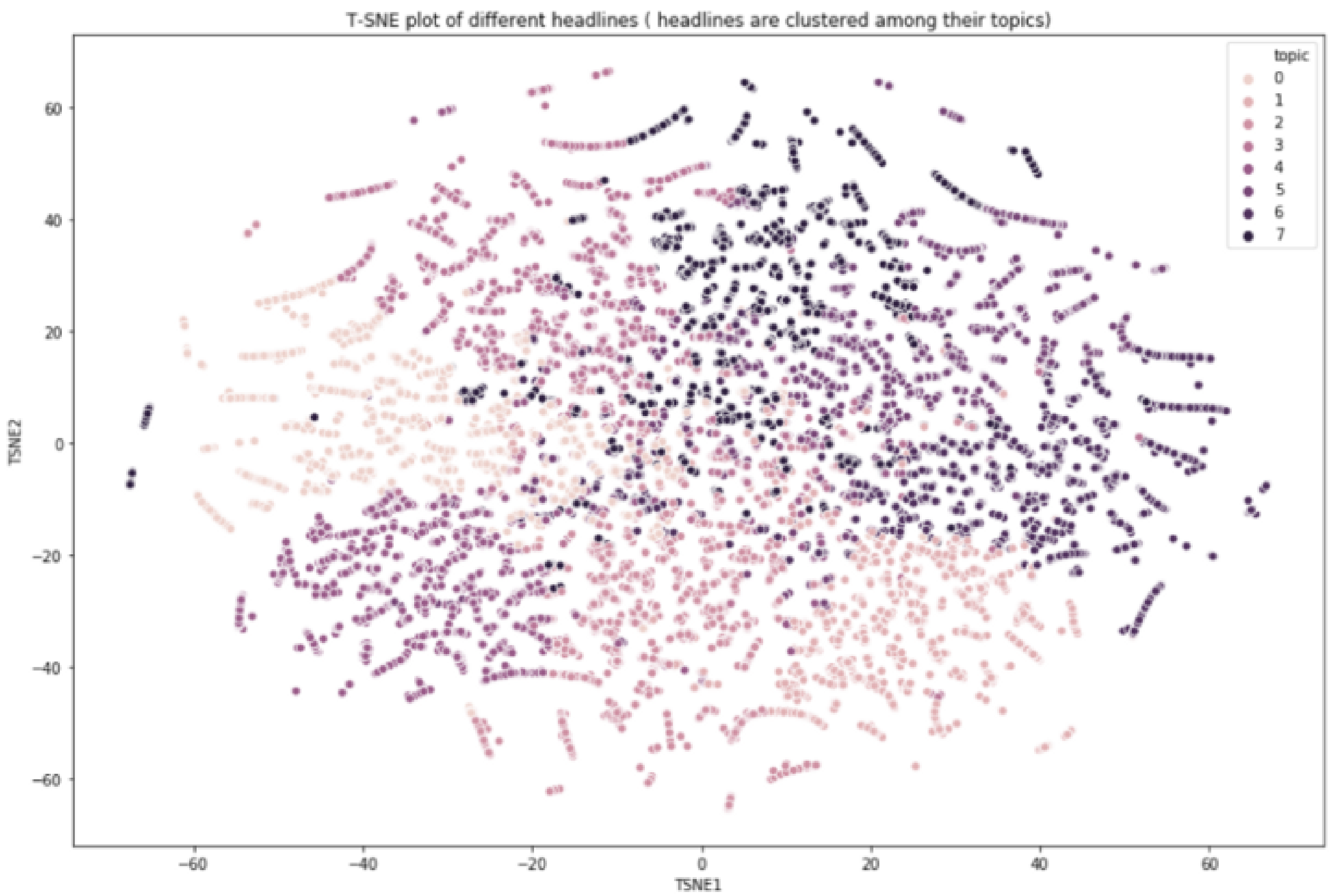
```
Topics found via LDA:

Topic #0:
accused council win wins boost national ban big tax protest warning appeal park services service

Topic #1:
australian road dies union iraq weather australia rural prices housing end bid faces hits anti

Topic #2:
police plan court missing man woman face crash deal car dead group inquiry perth cut

Topic #3:
interview says north labor rise public opposition house report fears election probe police cup calls

Topic #4:
sydney abc coast home plans south residents drug funds news continues jailed mining gold wants
```

```
Topic #5:
death govt nsw hospital killed urged high power qld market farmers china changes vic land

Topic #6:
charged country sex attack man budget child hour hit work gets government guilty concerns study

Topic #7:
day trial school health set minister funding support search fight new charges business adelaide water
```

You can try to understand what topics each of the above represent. once you start increasing the training sample, the topics will become more specific and the model will extract more intelligently. Now, let us use TSNE to plot all the documents and color them by the topics they represent according to LDA. Though we know that each document comes from several topics (an assumption for LDA), we will consider that for each document, the topic with the highest probability is the topic that document is representing.

```
from sklearn.manifold import TSNE
model = TSNE(n_components=2, perplexity=50, learning_rate=100,
                        n_iter=1000, verbose=1, random_state=0,
angle=0.75)

tsne_features = model.fit_transform(lda_matrix)
```

```
df = pd.DataFrame(tsne_features)
df['topic'] = lda_matrix.argmax(axis=1)
df.columns = ['TSNE1', 'TSNE2', 'topic']

import seaborn as sns
plt.figure(figsize=(15, 10))
plt.title('T-SNE plot of different headlines ( headlines are
clustered among their topics)')
ax = sns.scatterplot(x = 'TSNE1', y = 'TSNE2', hue = 'topic', data =
df, legend = 'full')
plt.show()
```



We can see how different documents belong to their cluster from the TSNE plot.

I hope you at least gain some insight into how topic modeling works. Now try scrapping data from twitter and try to get topics from those data.

Thank you.

Image reference: https://appliedmachinelearning.blog/2017/09/28/topic-modelling-part-2-discovering-topics-from-articles-with-latent-dirichlet-allocation/

Topic Modeling NLP Data Science Text Mining Analytics