Data Analysis of Text

Subhadarshi Panda

- Topic Modeling using LDA
- Clustering using neural models

Data Analysis of Text

Subhadarshi Panda

- Topic Modeling using LDA

Topic Modeling

Topic Modeling is a technique to extract the hidden topics from large volumes of text.

LDA

LDA is a topic modeling technique:

- Each text sample is collection of topics in a certain proportion
- Each topic is a collection of keywords in a certain proportion

Clustering

Clustering is the task of dividing the data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups.

Import Packages

In [101]:
```
import sys
import re, numpy as np, pandas as pd
from pprint import pprint

# Gensim
import gensim, spacy, logging, warnings
import gensim.corpora as corpora
from gensim.utils import lemmatize, simple_preprocess
from gensim.models import CoherenceModel
import matplotlib.pyplot as plt

# NLTK Stop words
from nltk.corpus import stopwords
stop_words = stopwords.words('english')

# tqdm
from tqdm import tqdm_notebook as tqdm

%matplotlib inline
warnings.filterwarnings("ignore",category=DeprecationWarning)
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.ER
ROR)
```

Load Dataset

Load Dataset

A Million News Headlines

News headlines from 2003 to 2017

https://www.kaggle.com/therohk/million-headlines/data#
(https://www.kaggle.com/therohk/million-headlines/data#)

In [102]:
```
df_all = pd.read_csv('million-headlines/abcnews-date-text.csv')

# get year from date and put that in a new column "year"
df_all['year'] = df_all['publish_date'].apply(lambda s: str(s)[:4])

print('shape of dataframe:', df_all.shape)
```

```
shape of dataframe: (1103663, 3)
```

```
In [51]: print('number of news headlines in different years:')
         print(df_all.groupby('year')['year'].count())
```

```
number of news headlines in different years:
year
2003    64003
2004    72674
2005    73124
2006    66912
2007    77192
2008    80014
2009    76454
2010    74948
2011    77829
2012    89109
2013    92337
2014    82330
2015    77941
2016    54614
2017    44182
Name: year, dtype: int64
```

In [52]:
```
# get news headlines from 2017 only
df = df_all.loc[df_all['year'] == "2017"].sample(300)

print('shape of dataframe:', df.shape)
```

```
shape of dataframe: (300, 3)
```

```
In [53]: df.head()
```

```
Out[53]:
```

	publish_date	headline_text	year
1067099	20170301	former silver fern dalton dies after brain aneurysm	2017
1087844	20170816	priscilla musical auditions reveal cut throat industry	2017
1071078	20170329	federal government announces sugar industry code of conduct	2017
1076214	20170511	budget 2017 labor medicare levy limited to high income earners	2017
1062694	20170127	wall street posts mixed session as investors digest earnings	2017

Preprocessing

Preprocessing

- Tokenization
- Cleaning
- Lemmatization

Tokenize sentences and clean

```
In [7]: def sent_to_words(sentences):
            for sent in sentences:
                sent = re.sub("\'", "", sent)  # remove single quotes
                sent = gensim.utils.simple_preprocess(str(sent), deacc=True)
                yield(sent)

        # Convert to list
        data = df["headline_text"].values.tolist()
        data_words = list(sent_to_words(data))
        print(data_words[:1])

        [['tech', 'slide', 'and', 'north', 'korea', 'weigh', 'on', 'wall', 'street']]
```

Build the bigram, trigram models and lemmatize

In [54]:

```
# Build the bigram and trigram models
bigram = gensim.models.Phrases(data_words, min_count=5, threshold=100) # higher threshol
d fewer phrases.
trigram = gensim.models.Phrases(bigram[data_words], threshold=100)
bigram_mod = gensim.models.phrases.Phraser(bigram)
trigram_mod = gensim.models.phrases.Phraser(trigram)
```

In [55]:
```
def process_words(texts, stop_words=stop_words, allowed_postags=('NOUN', 'ADJ', 'VERB',
'ADV')):
    """Remove Stopwords, Form Bigrams, Trigrams and Lemmatization"""
    texts = [[word for word in simple_preprocess(str(doc)) if word not in stop_words] fo
r doc in texts]
    texts = [bigram_mod[doc] for doc in texts]
    texts = [trigram_mod[bigram_mod[doc]] for doc in texts]
    texts_out = []
    nlp = spacy.load('en_core_web_sm', disable=['parser', 'ner'])
    for sent in texts:
        doc = nlp(" ".join(sent))
        texts_out.append([token.lemma_ for token in doc if token.pos_ in allowed_postags
])
    # remove stopwords once more after lemmatization
    texts_out = [[word for word in simple_preprocess(str(doc)) if word not in stop_words
] for doc in texts_out]
    return texts_out

data_ready = process_words(data_words)

print("processed text data!")
```

```
processed text data!
```

Build the topic model

In [56]:

```
# Create Dictionary
id2word = corpora.Dictionary(data_ready)

# Create Corpus: Term Document Frequency
corpus = [id2word.doc2bow(text) for text in data_ready]

# Build LDA model
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                            id2word=id2word,
                                            num_topics=4,
                                            random_state=100,
                                            update_every=1,
                                            chunksize=10,
                                            passes=10,
                                            alpha='symmetric',
                                            iterations=100,
                                            per_word_topics=True)
```

```
In [57]: pprint(lda_model.print_topics())
```

```
[(0,
  '0.014*"remain" + 0.010*"doctor" + 0.010*"cctv" + 0.009*"market" + '
  '0.009*"france" + 0.009*"risk" + 0.007*"employee" + 0.007*"holden" + '
  '0.007*"use" + 0.007*"mugabe"'),
 (1,
  '0.019*"court" + 0.013*"man" + 0.010*"australian" + 0.009*"melbourne" + '
  '0.008*"law" + 0.008*"baby" + 0.007*"tony" + 0.007*"outbreak" + '
  '0.007*"party" + 0.007*"interview"'),
 (2,
  '0.023*"australia" + 0.014*"discuss" + 0.011*"say" + 0.011*"gun" + '
  '0.009*"great" + 0.009*"win" + 0.009*"russia" + 0.008*"press" + '
  '0.008*"world" + 0.008*"call"'),
 (3,
  '0.016*"trump" + 0.014*"future" + 0.012*"cancer" + 0.011*"lawyer" + '
  '0.010*"charge" + 0.010*"kill" + 0.008*"woman" + 0.007*"manslaughter" + '
  '0.007*"case" + 0.007*"attacker"')]
```

What is the dominant topic and its percentage contribution in each headline?

```
In [59]: def format_topics_sentences(ldamodel=None, corpus=corpus, texts=data):
             # Init output
             sent_topics_df = pd.DataFrame()

             # Get main topic in each headline
             for i, row_list in enumerate(ldamodel[corpus]):
                 row = row_list[0] if ldamodel.per_word_topics else row_list
                 # print(row)
                 row = sorted(row, key=lambda x: (x[1]), reverse=True)
                 # Get the Dominant topic, Perc Contribution and Keywords for each headline
                 for j, (topic_num, prop_topic) in enumerate(row):
                     if j == 0:  # => dominant topic
                         wp = ldamodel.show_topic(topic_num)
                         topic_keywords = ", ".join([word for word, prop in wp])
                         sent_topics_df = sent_topics_df.append(pd.Series([int(topic_num), round(
         prop_topic,4), topic_keywords]), ignore_index=True)
                     else:
                         break
             sent_topics_df.columns = ['Dominant_Topic', 'Perc_Contribution', 'Topic_Keywords']

             # Add original text to the end of the output
             contents = pd.Series(texts)
             sent_topics_df = pd.concat([sent_topics_df, contents], axis=1)
             return(sent_topics_df)

         df_topic_sents_keywords = format_topics_sentences(ldamodel=lda_model, corpus=corpus, tex
         ts=data_ready)

         # Format
         df_dominant_topic = df_topic_sents_keywords.reset_index()
         df_dominant_topic.columns = ['Headline_No', 'Dominant_Topic', 'Topic_Perc_Contrib', 'Key
         words', 'Text']
```

In [60]:

```
df_dominant_topic.head(10)
```

Out[60]:

	Headline_No	Dominant_Topic	Topic_Perc_Contrib	Keywords	Text
0	0	3.0	0.9033	trump, future, cancer, lawyer, charge, kill, woman, manslaughter, case, attacker	[tech, slide, north, korea, weigh, wall, street]
1	1	2.0	0.7254	australia, discuss, say, gun, great, win, russia, press, world, call	[victorian, printing, business, owner, threaten, abc, journalist]
2	2	3.0	0.4226	trump, future, cancer, lawyer, charge, kill, woman, manslaughter, case, attacker	[police, border, force, seize, drug]
3	3	0.0	0.6013	remain, doctor, cctv, market, france, risk, employee, holden, use, mugabe	[teen, alcohol, drink, family, home, backfire, study, say]
4	4	0.0	0.9033	remain, doctor, cctv, market, france, risk, employee, holden, use, mugabe	[barcelona, play, empty, nou, camp, catalonia, referendum]
5	5	0.0	0.7207	remain, doctor, cctv, market, france, risk, employee, holden, use, mugabe	[shark, bite, kayaker, moreton, bay, water, police, rescue]
6	6	1.0	0.6714	court, man, australian, melbourne, law, baby, tony, outbreak, party, interview	[louis, ck, issue, apology, allegation, sexual, misconduct]
7	7	0.0	0.8099	remain, doctor, cctv, market, france, risk, employee, holden, use, mugabe	[news, quiz, september]
8	8	1.0	0.3308	court, man, australian, melbourne, law, baby, tony, outbreak, party, interview	[state, debt, blow, treasury, say]
9	9	3.0	0.5068	trump, future, cancer, lawyer, charge, kill, woman, manslaughter, case, attacker	[rock, roll, dad, riot, revolution]

The most representative headline for each topic

In [61]:
```
# Display setting to show more characters in column
pd.options.display.max_colwidth = 100

sent_topics_sorteddf_mallet = pd.DataFrame()
sent_topics_outdf_grpd = df_topic_sents_keywords.groupby('Dominant_Topic')

for i, grp in sent_topics_outdf_grpd:
    sent_topics_sorteddf_mallet = pd.concat([sent_topics_sorteddf_mallet,
                                             grp.sort_values(['Perc_Contribution'], asce
nding=False).head(1)],
                                            axis=0)

# Reset Index
sent_topics_sorteddf_mallet.reset_index(drop=True, inplace=True)

# Format
sent_topics_sorteddf_mallet.columns = ['Topic_Num', "Topic_Perc_Contrib", "Keywords", "R
epresentative Text"]
```

```
In [62]: # Show
         sent_topics_sorteddf_mallet
```

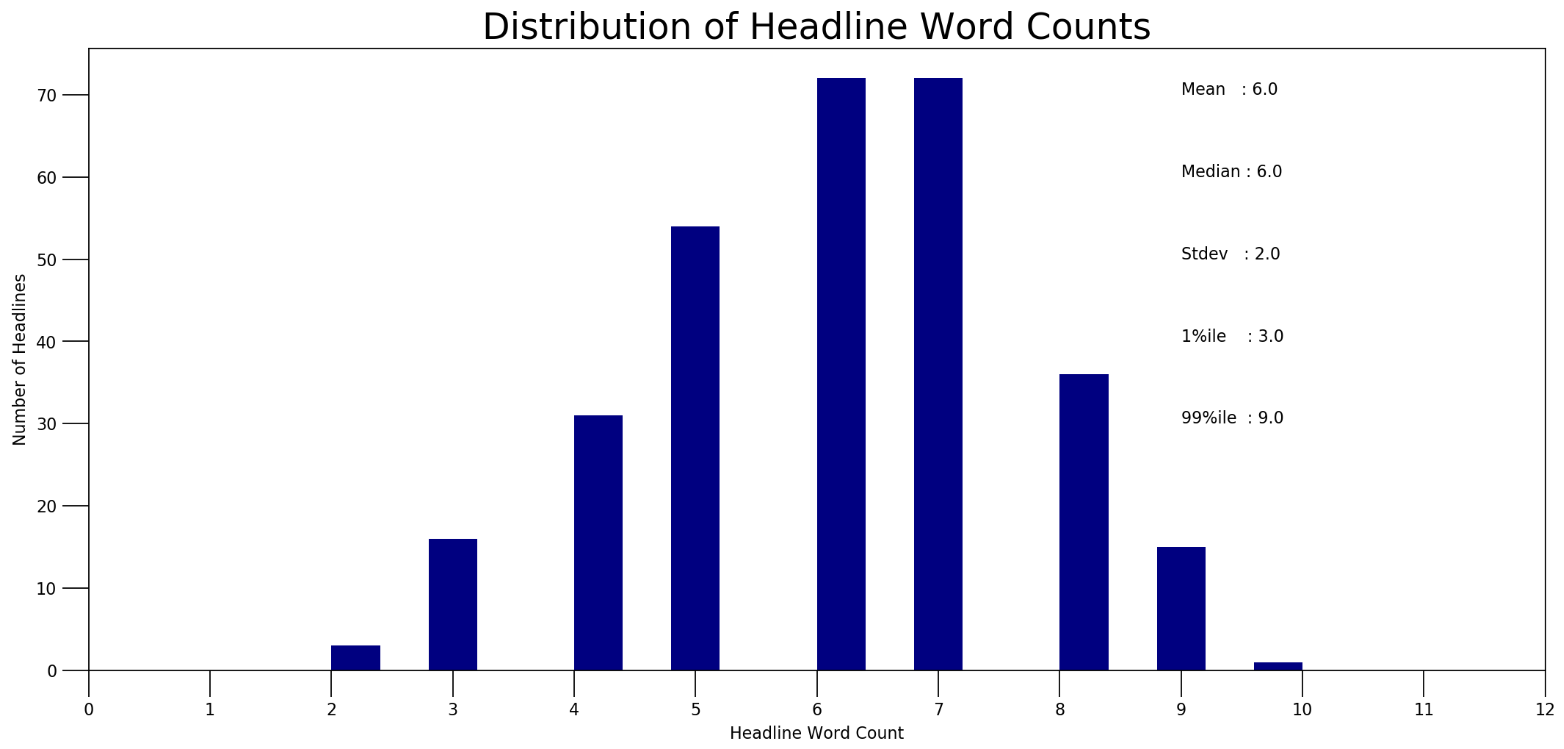
Out[62]:

	Topic_Num	Topic_Perc_Contrib	Keywords	Representative Text
0	0.0	0.9165	remain, doctor, cctv, market, france, risk, employee, holden, use, mugabe	[rodeo, ring, wedding, complete, nt, couple, love, story]
1	1.0	0.9248	court, man, australian, melbourne, law, baby, tony, outbreak, party, interview	[scott, morrison, greg, hunt, medicare, federal, politic, party, room]
2	2.0	0.9249	australia, discuss, say, gun, great, win, russia, press, world, call	[murray, ball, fan, pay, tribute, late, footrot, flat, cartoonist]
3	3.0	0.9249	trump, future, cancer, lawyer, charge, kill, woman, manslaughter, case, attacker	[fuel, tanker, explosion, kill, dozen, north, syrian, city, azaz]

Frequency Distribution of Word Counts in Headlines

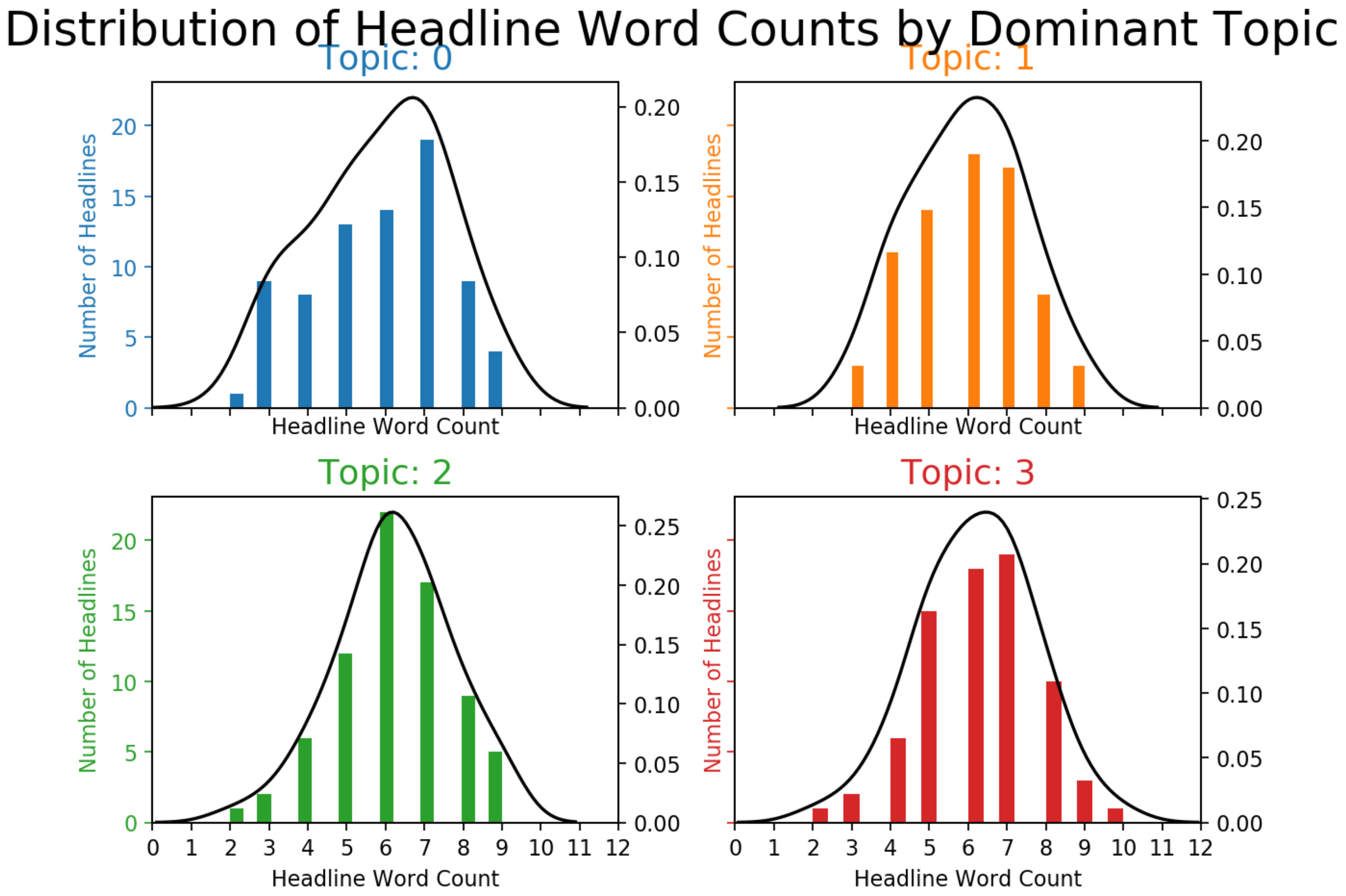
In [63]:
```
doc_lens = [len(d) for d in df_dominant_topic.Text]

def plot_func():
    # Plot
    plt.figure(figsize=(16,7), dpi=160)
    plt.hist(doc_lens, bins = 20, color='navy')
    plt.text(9, 70, "Mean   : " + str(round(np.mean(doc_lens))))
    plt.text(9, 60, "Median : " + str(round(np.median(doc_lens))))
    plt.text(9, 50, "Stdev   : " + str(round(np.std(doc_lens))))
    plt.text(9, 40, "1%ile    : " + str(round(np.quantile(doc_lens, q=0.01))))
    plt.text(9, 30, "99%ile  : " + str(round(np.quantile(doc_lens, q=0.99))))

    plt.gca().set(xlim=(0, 12), ylabel='Number of Headlines', xlabel='Headline Word Coun
t')
    plt.tick_params(size=16)
    plt.xticks(np.linspace(0,12,13))
    plt.title('Distribution of Headline Word Counts', fontdict=dict(size=22))
    plt.show()
```

In [64]:
```
plot_func()
```



In [104]:
```
import seaborn as sns
import matplotlib.colors as mcolors
cols = [color for name, color in mcolors.TABLEAU_COLORS.items()]  # more colors: 'mcolor
s.XKCD_COLORS'

def plot_func():
    fig, axes = plt.subplots(2,2,figsize=(8,6), dpi=160, sharex=True, sharey=True)

    for i, ax in enumerate(axes.flatten()):
        df_dominant_topic_sub = df_dominant_topic.loc[df_dominant_topic.Dominant_Topic =
= i, :]
        doc_lens = [len(d) for d in df_dominant_topic_sub.Text]
        ax.hist(doc_lens, bins = 20, color=cols[i])
        ax.tick_params(axis='y', labelcolor=cols[i], color=cols[i])
        sns.kdeplot(doc_lens, color="black", shade=False, ax=ax.twinx())
        ax.set(xlim=(0, 12), xlabel='Headline Word Count')
        ax.set_ylabel('Number of Headlines', color=cols[i])
        ax.set_title('Topic: '+str(i), fontdict=dict(size=16, color=cols[i]))

    fig.tight_layout()
    fig.subplots_adjust(top=0.90)
    plt.xticks(np.linspace(0,12,13))
    fig.suptitle('Distribution of Headline Word Counts by Dominant Topic', fontsize=22)
    plt.show()
```

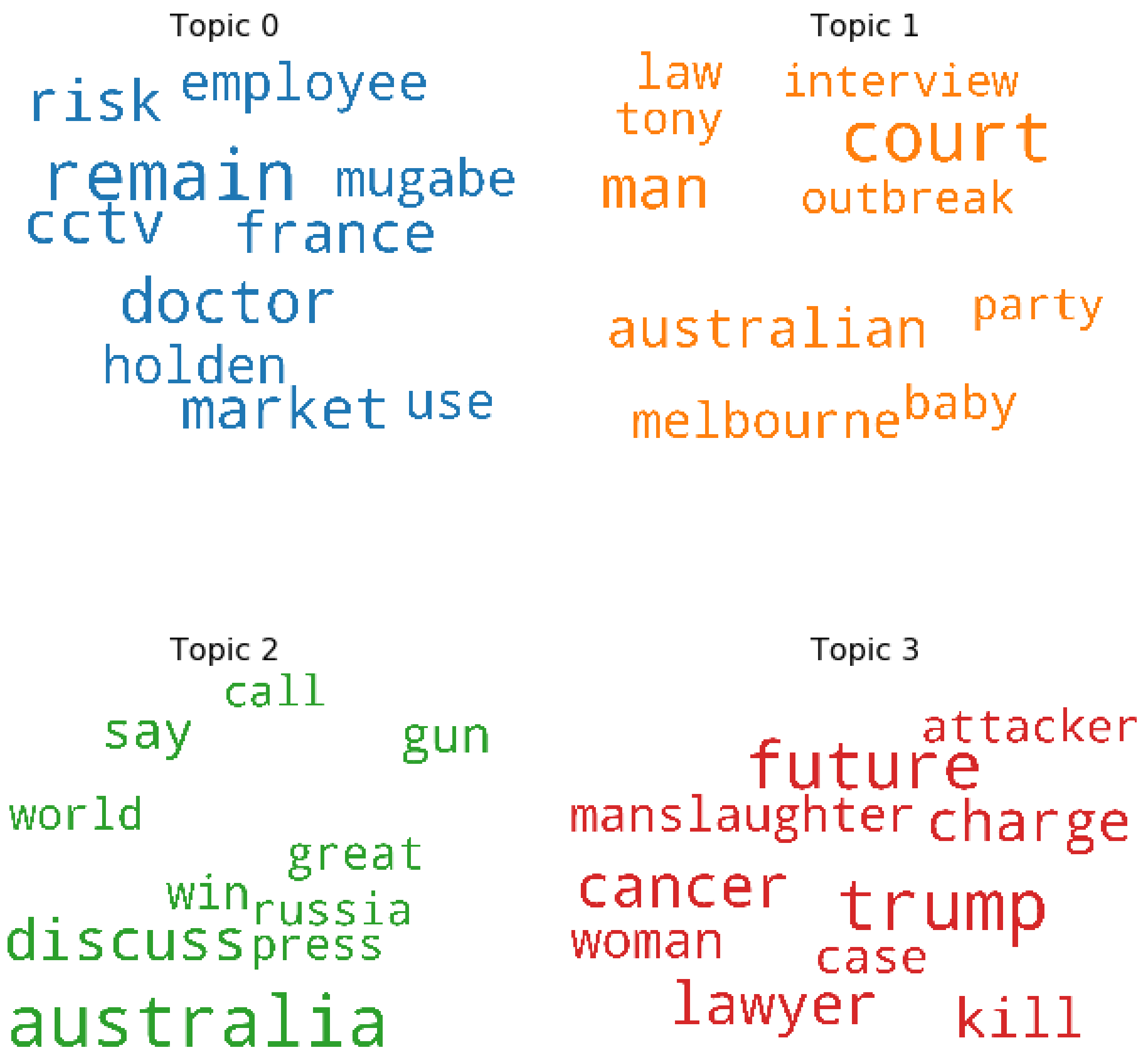
```
In [105]: plot_func()
```



Word Clouds of Top N Keywords in Each Topic

In [76]:
```
# Wordcloud of Top N words in each topic
from matplotlib import pyplot as plt
from wordcloud import WordCloud, STOPWORDS
import matplotlib.colors as mcolors

cols = [color for name, color in mcolors.TABLEAU_COLORS.items()]  # more colors: 'mcolor
s.XKCD_COLORS'

def show_wordcloud():
    cloud = WordCloud(stopwords=stop_words,
                      background_color='white',
                      width=2500,
                      height=1800,
                      max_words=10,
                      colormap='tab10',
                      color_func=lambda *args, **kwargs: cols[i],
                      prefer_horizontal=1.0)

    topics = lda_model.show_topics(formatted=False)
    fig, axes = plt.subplots(2, 2, figsize=(10,10), sharex=True, sharey=True)

    for i, ax in enumerate(axes.flatten()):
        fig.add_subplot(ax)
        topic_words = dict(topics[i][1])
        cloud.generate_from_frequencies(topic_words, max_font_size=300)
        plt.gca().imshow(cloud)
        plt.gca().set_title('Topic ' + str(i), fontdict=dict(size=16))
        plt.gca().axis('off')


    plt.subplots_adjust(wspace=0, hspace=0)
    plt.axis('off')
    plt.margins(x=0, y=0)
    plt.tight_layout()
    plt.show()
```

```
In [77]: show_wordcloud()
```



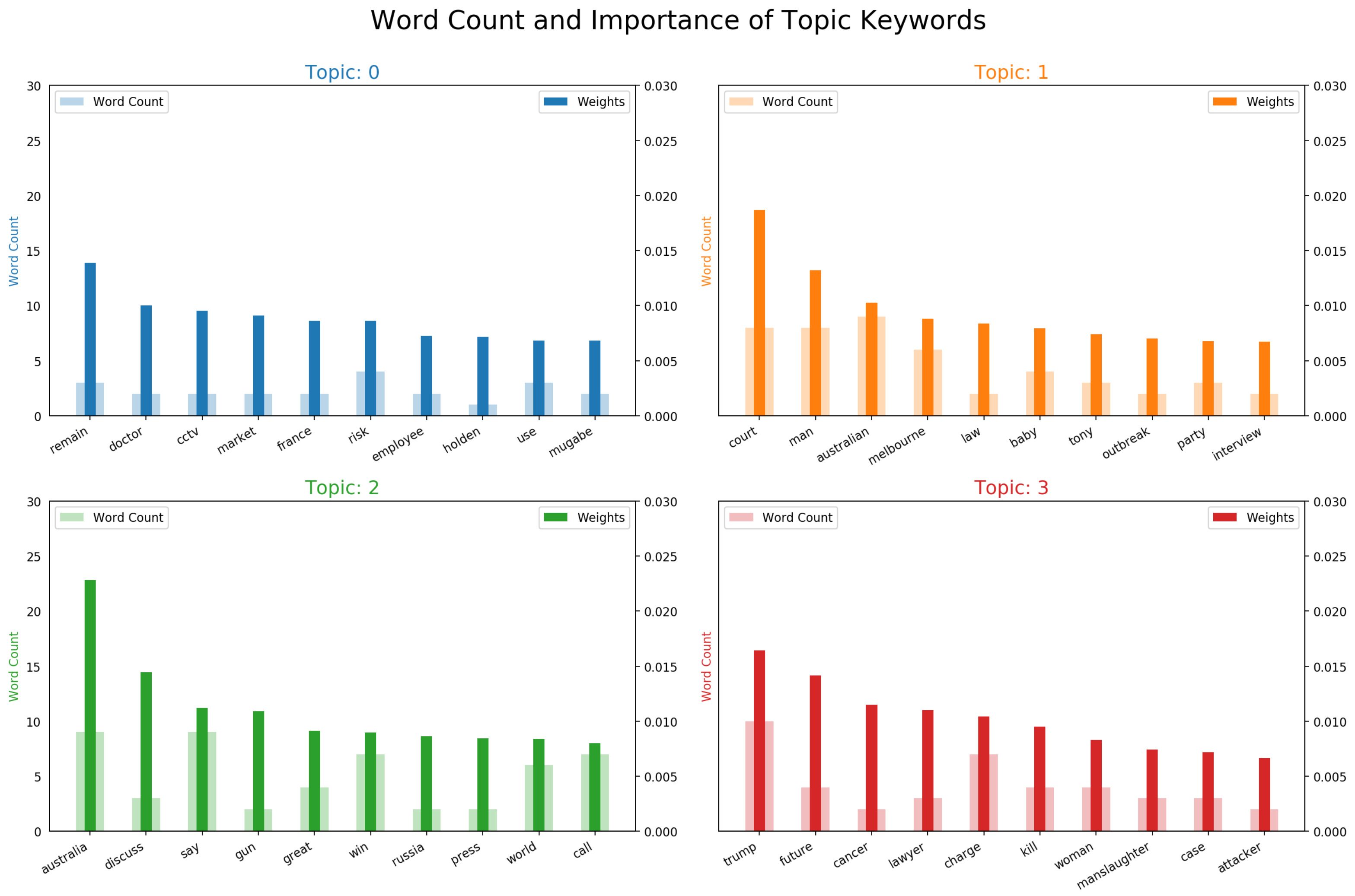
Word Counts of Topic Keywords

In [78]:

```python
from collections import Counter
topics = lda_model.show_topics(formatted=False)
data_flat = [w for w_list in data_ready for w in w_list]
counter = Counter(data_flat)

out = []
for i, topic in topics:
    for word, weight in topic:
        out.append([word, i , weight, counter[word]])

df = pd.DataFrame(out, columns=['word', 'topic_id', 'importance', 'word_count'])
```

```
In [81]: def show_wordcounts_and_importance():

             # Plot Word Count and Weights of Topic Keywords
             fig, axes = plt.subplots(2, 2, figsize=(16,10), sharey=True, dpi=160)
             cols = [color for name, color in mcolors.TABLEAU_COLORS.items()]
             for i, ax in enumerate(axes.flatten()):
                 ax.bar(x='word', height="word_count", data=df.loc[df.topic_id==i, :], color=cols
         [i], width=0.5, alpha=0.3, label='Word Count')
                 ax_twin = ax.twinx()
                 ax_twin.bar(x='word', height="importance", data=df.loc[df.topic_id==i, :], color
         =cols[i], width=0.2, label='Weights')
                 ax.set_ylabel('Word Count', color=cols[i])
                 ax_twin.set_ylim(0, 0.030); ax.set_ylim(0, 30)
                 ax.set_title('Topic: ' + str(i), color=cols[i], fontsize=16)
                 ax.tick_params(axis='y', left=False)
                 ax.set_xticklabels(df.loc[df.topic_id==i, 'word'], rotation=30, horizontalalignm
         ent= 'right')
                 ax.legend(loc='upper left'); ax_twin.legend(loc='upper right')

             fig.tight_layout(w_pad=2)
             fig.suptitle('Word Count and Importance of Topic Keywords', fontsize=22, y=1.05)
             plt.show()
```

```
In [82]: show_wordcounts_and_importance()
```



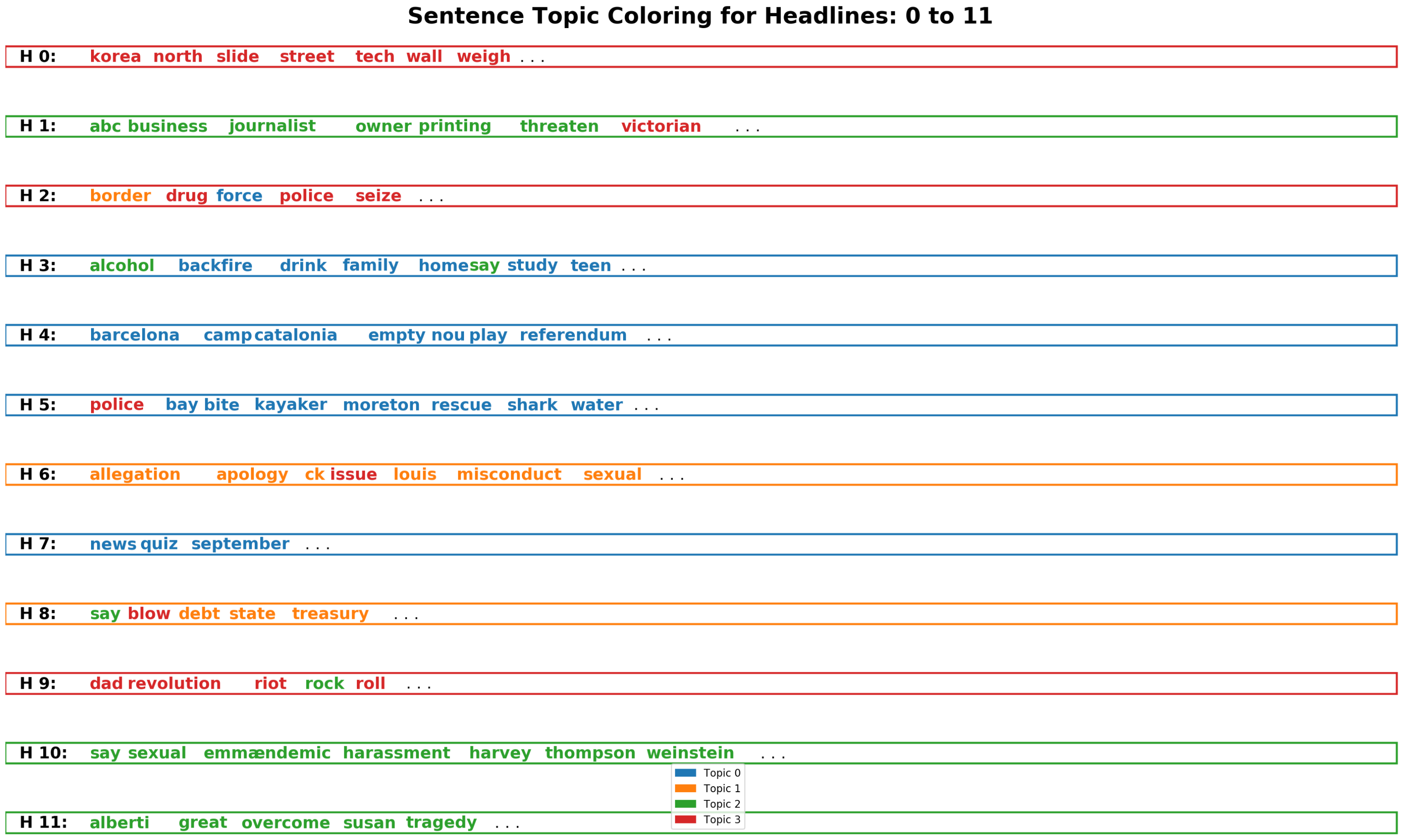
Headline Chart Colored by Topic

In [98]:

```
# Sentence Coloring of N Sentences
from matplotlib.patches import Rectangle
import matplotlib.patches as mpatches
```

```
In [99]: def sentences_chart(lda_model=lda_model, corpus=corpus, start = 0, end = 13):
             corp = corpus[start:end]
             mycolors = [color for name, color in mcolors.TABLEAU_COLORS.items()]

             fig, axes = plt.subplots(end-start, 1, figsize=(20, (end-start)*0.95), dpi=160)
             axes[0].axis('off')
             for i, ax in enumerate(axes):
                 if i > 0:
                     corp_cur = corp[i-1]
                     topic_percs, wordid_topics, wordid_phivalues = lda_model[corp_cur]
                     word_dominanttopic = [(lda_model.id2word[wd], topic[0]) for wd, topic in wor
         did_topics]
                     ax.text(0.01, 0.5, "H " + str(i-1) + ": ", verticalalignment='center',
                             fontsize=16, color='black', transform=ax.transAxes, fontweight=700)

                     # Draw Rectange
                     topic_percs_sorted = sorted(topic_percs, key=lambda x: (x[1]), reverse=True)
                     ax.add_patch(Rectangle((0.0, 0.05), 0.99, 0.90, fill=None, alpha=1,
                                            color=mycolors[topic_percs_sorted[0][0]], linewidth=2
         ))

                     word_pos = 0.06
                     for j, (word, topics) in enumerate(word_dominanttopic):
                         if j < 14:
                             ax.text(word_pos, 0.5, word,
                                     horizontalalignment='left',
                                     verticalalignment='center',
                                     fontsize=16, color=mycolors[topics],
                                     transform=ax.transAxes, fontweight=700)
                             word_pos += .009 * len(word)  # to move the word for the next iter
                             ax.axis('off')
                     ax.text(word_pos, 0.5, '. . .',
                             horizontalalignment='left',
                             verticalalignment='center',
                             fontsize=16, color='black',
                             transform=ax.transAxes)
```

```
In [100]: sentences_chart()
```



t-SNE

In [94]:
```
# Get topic weights and dominant topics ------------
from sklearn.manifold import TSNE
from bokeh.plotting import figure, output_file, show
from bokeh.models import Label
from bokeh.io import output_notebook

# Get topic weights
topic_weights = []
for i, row_list in enumerate(lda_model[corpus]):
    topic_weights.append([w for i, w in row_list[0]])

# Array of topic weights
arr = pd.DataFrame(topic_weights).fillna(0).values

# Keep the well separated points (optional)
arr = arr[np.amax(arr, axis=1) > 0.35]

# Dominant topic number in each doc
topic_num = np.argmax(arr, axis=1)
```

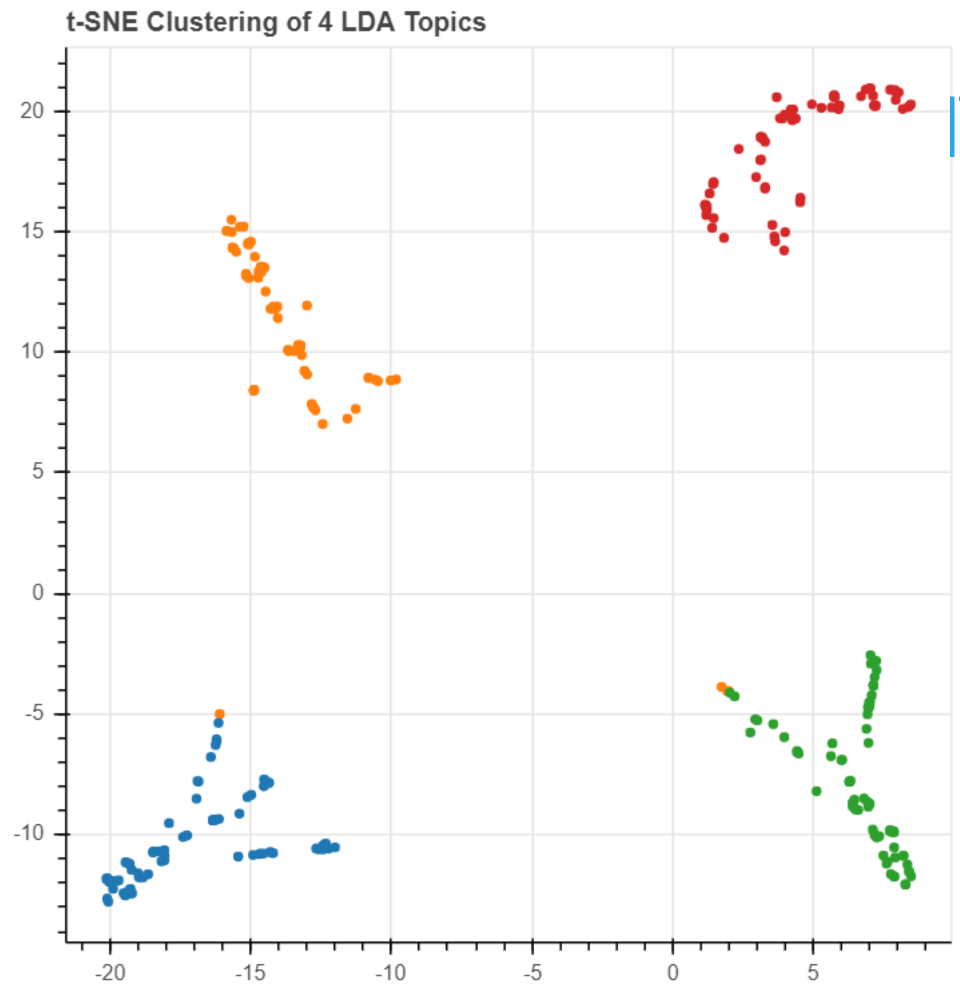
```
In [95]: # tSNE Dimension Reduction
         tsne_model = TSNE(n_components=2, verbose=1, random_state=0, angle=.99, init='pca')
         tsne_lda = tsne_model.fit_transform(arr)

         # Plot the Topic Clusters using Bokeh
         output_notebook()
         n_topics = 4
         mycolors = np.array([color for name, color in mcolors.TABLEAU_COLORS.items()])
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 295 samples in 0.001s...
[t-SNE] Computed neighbors for 295 samples in 0.020s...
[t-SNE] Computed conditional probabilities for sample 295 / 295
[t-SNE] Mean sigma: 0.032562
[t-SNE] KL divergence after 250 iterations with early exaggeration: 49.399616
[t-SNE] KL divergence after 1000 iterations: 0.123616
```

(https://bokeh.pydata.org) BokehJS 1.3.4 successfully loaded.

In [96]:
```
def show_tsne_plot():
    plot = figure(title="t-SNE Clustering of {} LDA Topics".format(n_topics),
                  plot_width=500, plot_height=500)
    plot.scatter(x=tsne_lda[:,0], y=tsne_lda[:,1], color=mycolors[topic_num])
    show(plot)
```

In [103]:
```
show_tsne_plot()
```



(ht

Challenges / Issues

- Use more data (more than 1,000,000 samples)
- What is a good number of topics?
- Find topics of *new* text samples

Next Steps

- Above challenges
- Clustering using neural models

Useful Sources:

https://radimrehurek.com/gensim/apiref.html
(https://radimrehurek.com/gensim/apiref.html)

Thank You

#

Questions?