HW8

April 29, 2020

To model the LDA for a set of documents, we want to find the probability a word belongs to a topic given the set of words in the document and the set of topics. This can be written as (All symbols are based on slide 18):

$P(w|\beta, z, \theta, \alpha, \eta)$

where η and α are paramaters for the diralect distributions of β and θ. This means to find the probability of a topic β_i or a distribution of topics over document d we simply calculate $P(\beta_i|\eta)$ and $P(\theta_d|\alpha)$ respectfully. This means if we want to find the probability a document d belongs to a topic i, we calculate $P(\beta_i|\eta)P(\theta_d|\alpha)$.

To find the overall distribution we calculate

$\prod_{i=1}^{K} P(\beta_i|\eta) \prod_{d=1}^{D} P(\theta_d|\alpha)$.

Now that we know how to model the distribution of topics over the documents, we can rewrite

$P(w|\beta, z, \theta, \alpha, \eta)$ as

$P(\beta_i|\eta)P(\theta_d|\alpha)P(w_{d,n}|\beta, z_{d,n}, \theta_d)$.

The last probability is finding the probability a word belongs to a topic given the set of topics in the document, the words belonging to that topic, and the entire set of topics. The probability a topic containing the word $w_{d,n}$ is in the document can be found by $P(z_{d,n}|\theta_d)$. If we multiply this by the probability the word $w_{d,n}$ belongs to the topic $z_{d,n}$ which is a topic in β, we get the probability $P(z_{d,n}|\theta_d)P(w_{d,n}|\beta, z_{d,n})$. If we want to model the entire word distribution for a given document d and topic i, we calculate

$\prod_{n=1}^{N} P(z_{d,n}|\theta_d)P(w_{d,n}|\beta, z_{d,n})$.

Add this back to the main equation and we get

$P(\beta_i|\eta)P(\theta_d|\alpha)P(z_{d,n}|\theta_d)P(w_{d,n}|\beta, z_{d,n})$.

To create a distribution over the entire corpus, we include the product symbols giving us the distribution:

$\prod_{i=1}^{K} P(\beta_i|\eta) \prod_{d=1}^{D} P(\theta_d|\alpha) \prod_{n=1}^{N} P(z_{d,n}|\theta_d)P(w_{d,n}|\beta, z_{d,n})$.

1 Topic Modeling using LDA

This notebook will model news topics using Latent Diralecht Analysis. First we import necessary packages and download english stopwords, as well as the dataset. We only look at articles related to christianity, hockey, the middle east, and motercycles.

[1]:
```
import sys
# !{sys.executable} -m spacy download en
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')
stop_words.extend(['from', 'subject', 're', 'edu', 'use', 'not', 'would',␣
↪'say', 'could', '_', 'be', 'know',
                   'good', 'go', 'get', 'do', 'done', 'try', 'many', 'some',␣
↪'nice', 'thank', 'think', 'see',
                   'rather', 'easy', 'easily', 'lot', 'lack', 'make', 'want',␣
↪'seem', 'run', 'need', 'even',
                   'right', 'line', 'even', 'also', 'may', 'take', 'come'])

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

[2]:
```
# Import Dataset
df = pd.read_json('https://raw.githubusercontent.com/BHill96/datasets/master/
↪newsgroups.json')
df = df.loc[df.target_names.isin(['soc.religion.christian', 'rec.sport.hockey',␣
↪'talk.politics.mideast',
                                  'rec.motorcycles']) , :]
print(df.shape)  #> (2361, 3)
df.head()
```

```
(2361, 3)
```

[2]:
```
                                              content  target  \
10  From: irwin@cmptrc.lonestar.org (Irwin Arnstei…       8
21  From: leunggm@odin.control.utoronto.ca (Gary L…      10
28  From: jonh@david.wheaton.edu (Jonathan Hayward…      15
33  From: ayr1@cunixa.cc.columbia.edu (Amir Y Rose…      17
35  From: dchhabra@stpl.ists.ca (Deepak Chhabra)\n…      10

             target_names
```

```
10         rec.motorcycles
21        rec.sport.hockey
28  soc.religion.christian
33   talk.politics.mideast
35        rec.sport.hockey
```

Now we tokenize each article so the computer can understand what a word is.

[3]:
```python
"""
We tokenize the sentences by removing emails, new line characters, and spaces.
"""
def sent_to_words(sentences):
    for sent in sentences:
        sent = re.sub('\S*@\S*\s?', '', sent)  # remove emails
        sent = re.sub('\s+', ' ', sent)  # remove newline chars
        sent = re.sub("\'", "", sent)  # remove single quotes
        sent = gensim.utils.simple_preprocess(str(sent), deacc=True)
        yield(sent)

# Convert to list
data = df.content.values.tolist()
data_words = list(sent_to_words(data))
print(data_words[:1])
# [['from', 'irwin', 'arnstein', 'subject', 're', 'recommendation', 'on',␣
↪'duc', 'summary', 'whats', 'it',
# 'worth', 'distribution', 'usa', 'expires', 'sat', 'may', 'gmt', ...trucated...
↪]]
```

```
[['from', 'irwin', 'arnstein', 'subject', 're', 'recommendation', 'on', 'duc',
'summary', 'whats', 'it', 'worth', 'distribution', 'usa', 'expires', 'sat',
'may', 'gmt', 'organization', 'computrac', 'inc', 'richardson', 'tx',
'keywords', 'ducati', 'gts', 'how', 'much', 'lines', 'have', 'line', 'on',
'ducati', 'gts', 'model', 'with', 'on', 'the', 'clock', 'runs', 'very', 'well',
'paint', 'is', 'the', 'bronze', 'brown', 'orange', 'faded', 'out', 'leaks',
'bit', 'of', 'oil', 'and', 'pops', 'out', 'of', 'st', 'with', 'hard', 'accel',
'the', 'shop', 'will', 'fix', 'trans', 'and', 'oil', 'leak', 'they', 'sold',
'the', 'bike', 'to', 'the', 'and', 'only', 'owner', 'they', 'want', 'and', 'am',
'thinking', 'more', 'like', 'any', 'opinions', 'out', 'there', 'please',
'email', 'me', 'thanks', 'it', 'would', 'be', 'nice', 'stable', 'mate', 'to',
'the', 'beemer', 'then', 'ill', 'get', 'jap', 'bike', 'and', 'call', 'myself',
'axis', 'motors', 'tuba', 'irwin', 'honk', 'therefore', 'am', 'computrac',
'richardson', 'tx', 'dod']]
```

In order to understand context, we create bigrams and trigrams. We also lemmatize the words since many words are basically the same with different pre/suffixes.

[4]:
```python
# Build the bigram and trigram models
# Group two and three adjacent tokens together
```

```
bigram = gensim.models.Phrases(data_words, min_count=5, threshold=100) # higher␣
↪threshold 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)

"""
Lemmatization is returning a word to it's root (ex. running -> run)
"""
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] for 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')
    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)  # processed Text Data!
```

Now we map the words to integer ids and create a bag of words with their term-document frequency. Using the dictionary and the bag of wrds, we finally create the LDA model. Below we print out each topic (represented by an integer) and the weights of the most important words to that topic.

[5]:
```
# Create Dictionary
id2word = corpora.Dictionary(data_ready)

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

# Build LDA model
# chunksize represents the number of important words to the topic
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)
```

```
pprint(lda_model.print_topics())
```

```
[(0,
  '0.019*"armenian" + 0.017*"greek" + 0.014*"turk" + 0.013*"government" + '
  '0.011*"turkish" + 0.010*"soldier" + 0.010*"people" + 0.009*"turkey" + '
  '0.008*"greece" + 0.007*"village"'),
 (1,
  '0.010*"write" + 0.009*"time" + 0.009*"article" + 0.008*"organization" + '
  '0.006*"work" + 0.006*"year" + 0.006*"number" + 0.005*"well" + 0.005*"kill" '
  '+ 0.005*"leave"'),
 (2,
  '0.013*"people" + 0.012*"god" + 0.009*"write" + 0.008*"believe" + '
  '0.007*"christian" + 0.007*"reason" + 0.006*"organization" + 0.006*"thing" + '
  '0.006*"way" + 0.006*"israel"'),
 (3,
  '0.017*"team" + 0.013*"game" + 0.012*"organization" + 0.010*"hockey" + '
  '0.009*"bike" + 0.007*"play" + 0.007*"win" + 0.006*"player" + 0.006*"write" '
  '+ 0.006*"year"')]
```

Since we primarily describe documents as having a single topic, we extract the most dominant topic for each document.

[6]:
```
def format_topics_sentences(ldamodel=None, corpus=corpus, texts=data):
    # Init output
    sent_topics_df = pd.DataFrame()

    # Get main topic in each document
    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␣
↪document
        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, texts=data_ready)

# Format
df_dominant_topic = df_topic_sents_keywords.reset_index()
df_dominant_topic.columns = ['Document_No', 'Dominant_Topic',␣
 ↪'Topic_Perc_Contrib', 'Keywords', 'Text']
df_dominant_topic.head(10)
```

[6]:

```
   Document_No  Dominant_Topic  Topic_Perc_Contrib  \
0            0             3.0              0.7699
1            1             3.0              0.8189
2            2             2.0              0.8304
3            3             1.0              0.5091
4            4             3.0              0.5090
5            5             1.0              0.5663
6            6             3.0              0.4600
7            7             1.0              0.5432
8            8             2.0              0.9906
9            9             0.0              0.5934

                                            Keywords  \
0  team, game, organization, hockey, bike, play, …
1  team, game, organization, hockey, bike, play, …
2  people, god, write, believe, christian, reason…
3  write, time, article, organization, work, year…
4  team, game, organization, hockey, bike, play, …
5  write, time, article, organization, work, year…
6  team, game, organization, hockey, bike, play, …
7  write, time, article, organization, work, year…
8  people, god, write, believe, christian, reason…
9  armenian, greek, turk, government, turkish, so…

                                                Text
0  [irwin, arnstein, recommendation, duc, summary…
1  [gary, leung, organization, university, system…
2  [jonathan, hayward, pantheism, organization, w…
3  [amir_rosenblatt, reply, amir_rosenblatt, orga…
4  [deepak_chhabra, goalie_mask, ists_ca, organiz…
5  [joe, ehrlich, bmw_moa_member, read, organizat…
6  [chris_behanna, require, organization, article…
7  [speedy_mercer, look, movie, bike, organizatio…
8  [darius_lecointe, organization, florida_state,…
9  [serdar_argic, day, night, armenian, round, ma…
```

We can also find the most appropriate sentence for each topic

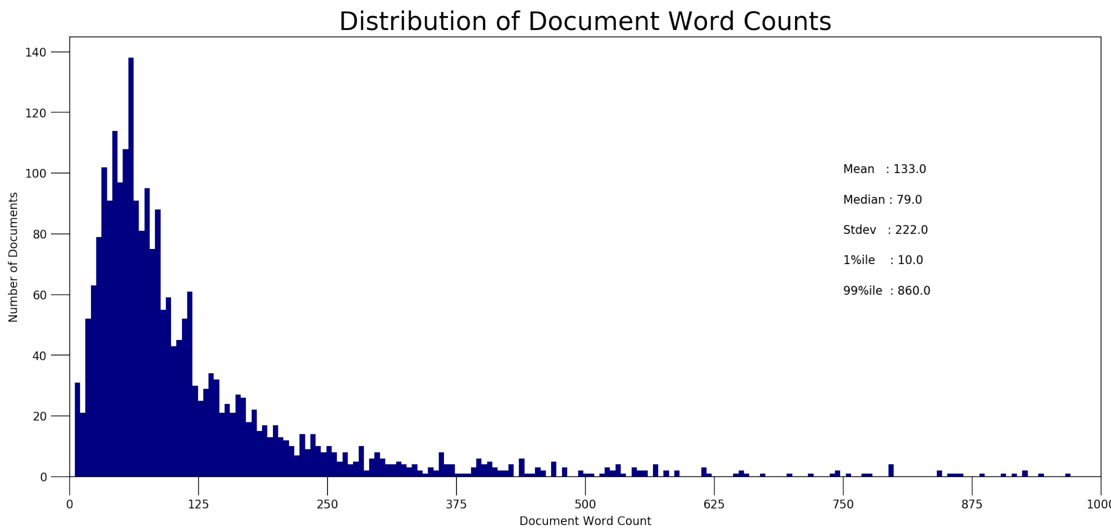
[7]:
```python
# Display setting to show more characters in column
pd.options.display.max_colwidth = 100

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'], ascending=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", "Representative Text"]

# Show
sent_topics_sorteddf_mallet.head(10)
```

[7]:
```
   Topic_Num  Topic_Perc_Contrib  \
0        0.0              0.9743
1        1.0              0.8996
2        2.0              0.9906
3        3.0              0.9984

Keywords  \
0  armenian, greek, turk, government, turkish, soldier, people, turkey, greece,
village
1              write, time, article, organization, work, year, number, well,
kill, leave
2      people, god, write, believe, christian, reason, organization, thing, way,
israel
3                  team, game, organization, hockey, bike, play, win, player,
write, year

   Representative Text
0  [serdar_argic, armenian, genocide, muslim, people, article, reply, article,
panos_tamamidi, writ…
1  [gillian, runcie, bar, organization, comp_sci, dept, strathclyde, univ,
glasgow, scotland, liste…
2  [darius_lecointe, organization, florida_state, university, follow, thread,
talk, religion, bible…
```

```
3  [result, game, play, sit, april, cook_charlie, organization, university,
new_brunswick, tampa_ba…
```

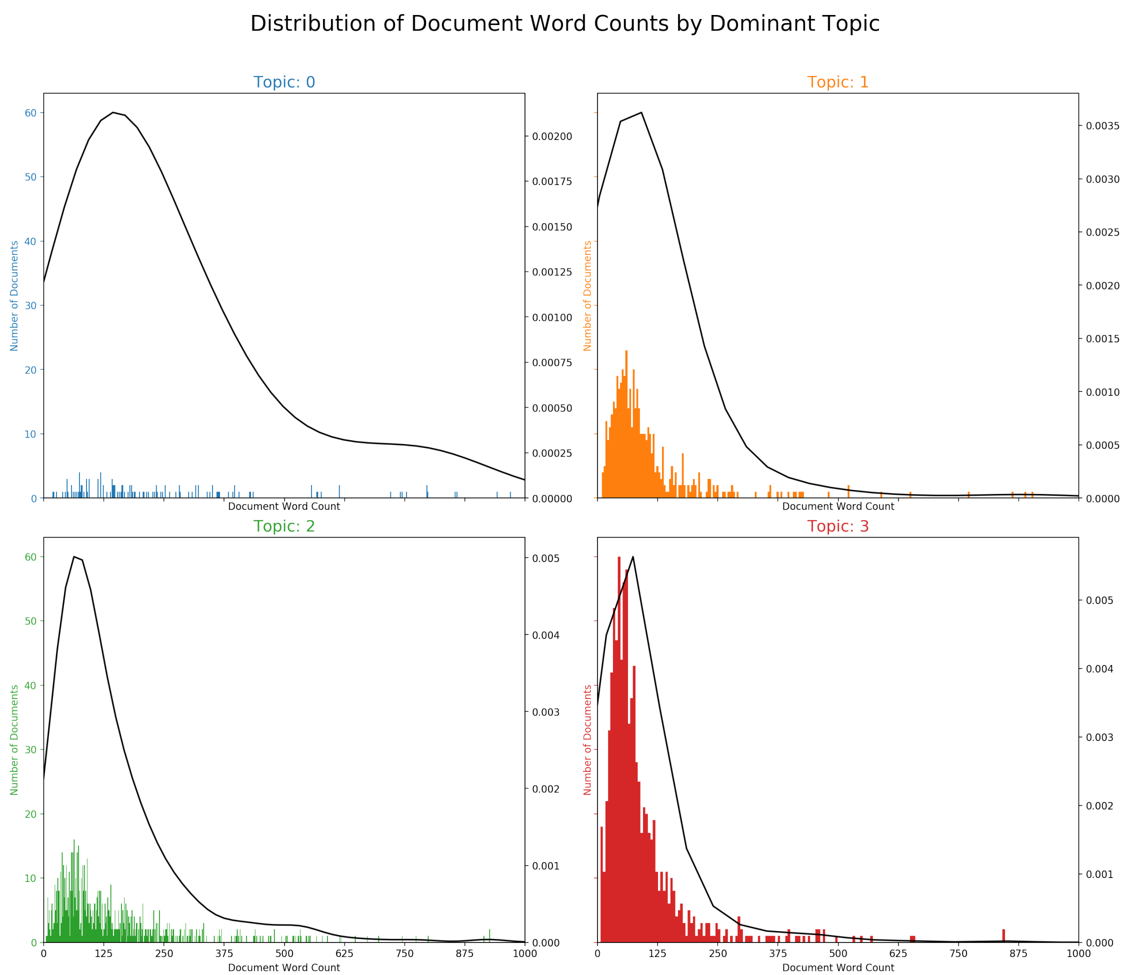
The next couple of graphs give us an idea on how common our words our within our corpus and within each topic.

[8]:
```python
doc_lens = [len(d) for d in df_dominant_topic.Text]

# Plot
plt.figure(figsize=(16,7), dpi=160)
plt.hist(doc_lens, bins = 1000, color='navy')
plt.text(750, 100, "Mean   : " + str(round(np.mean(doc_lens))))
plt.text(750,  90, "Median : " + str(round(np.median(doc_lens))))
plt.text(750,  80, "Stdev   : " + str(round(np.std(doc_lens))))
plt.text(750,  70, "1%ile    : " + str(round(np.quantile(doc_lens, q=0.01))))
plt.text(750,  60, "99%ile  : " + str(round(np.quantile(doc_lens, q=0.99))))

plt.gca().set(xlim=(0, 1000), ylabel='Number of Documents', xlabel='Document␣
↪Word Count')
plt.tick_params(size=16)
plt.xticks(np.linspace(0,1000,9))
plt.title('Distribution of Document Word Counts', fontdict=dict(size=22))
plt.show()
```



[10]:
```python
import seaborn as sns
import matplotlib.colors as mcolors
cols = [color for name, color in mcolors.TABLEAU_COLORS.items()]  # more colors:
↪ 'mcolors.XKCD_COLORS'
```

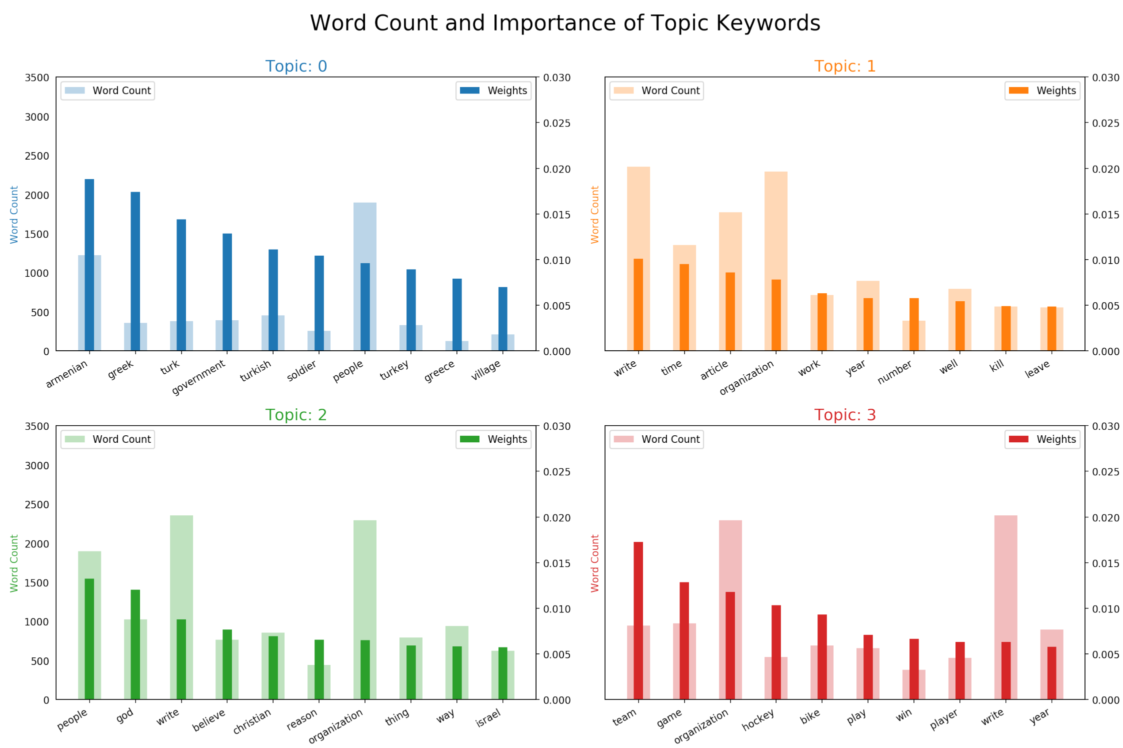
```python
fig, axes = plt.subplots(2,2,figsize=(16,14), dpi=160, sharex=True, sharey=True)

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 = 1000, 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, 1000), xlabel='Document Word Count')
    ax.set_ylabel('Number of Documents', 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(0,1000,9))
fig.suptitle('Distribution of Document Word Counts by Dominant Topic', fontsize=22)
plt.show()
```



[12]:
```python
# 1. 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:
↪ 'mcolors.XKCD_COLORS'

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()
```

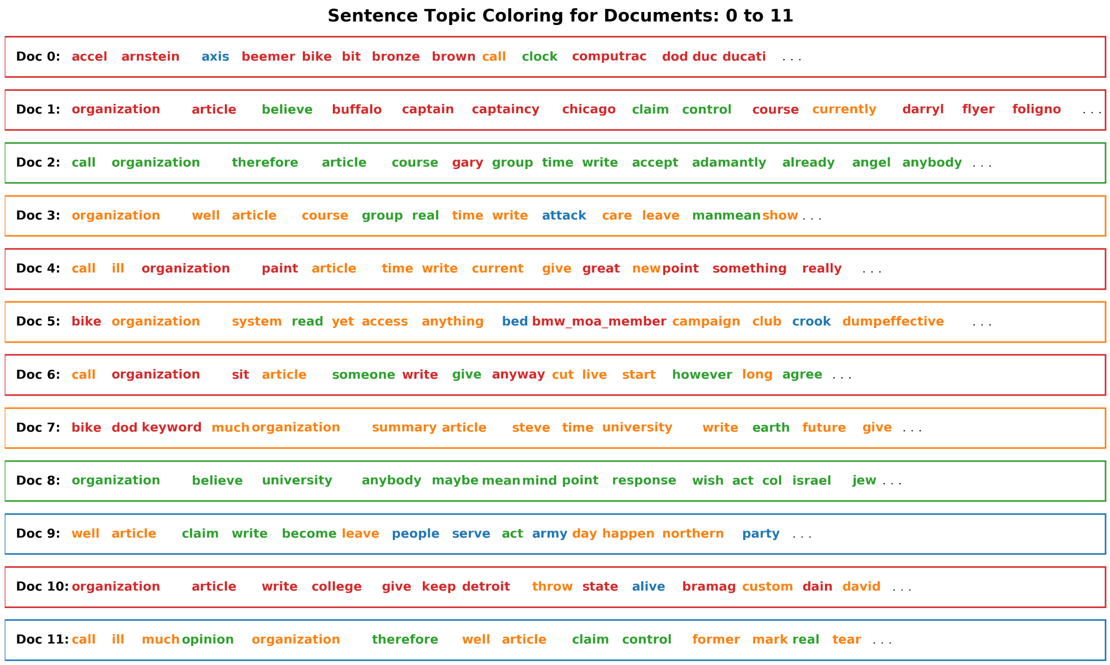
[13]:
```python
from collections import Counter
topics = lda_model.show_topics(formatted=False)
data_flat = [w for w_list in data_ready for w in w_list]
counter = Counter(data_flat)

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'])

# 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, 3500)
    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, horizontalalignment= '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()
```



[14]:
```
# Sentence Coloring of N Sentences
from matplotlib.patches import Rectangle

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()]
```

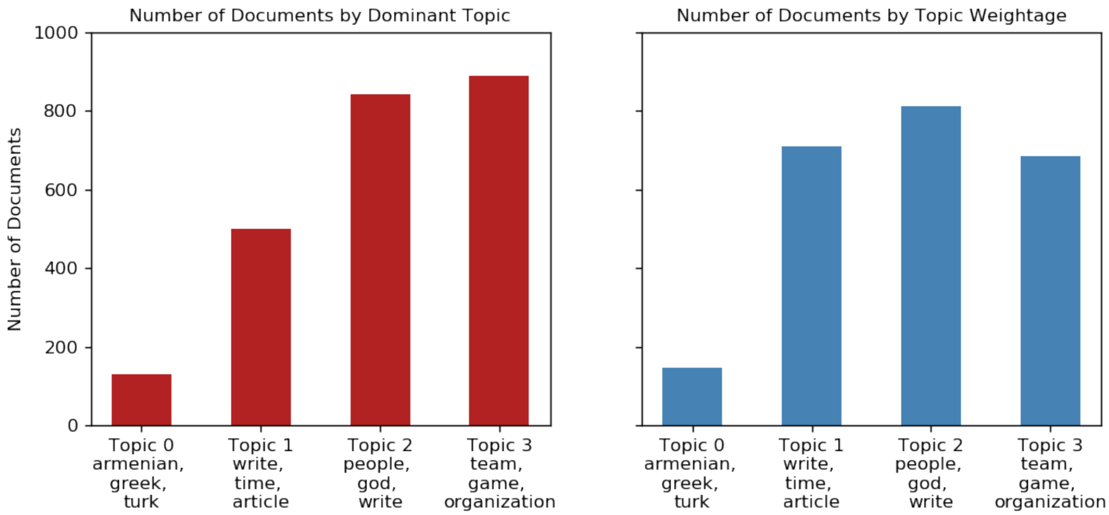
```
    fig, axes = plt.subplots(end-start, 1, figsize=(20, (end-start)*0.95),␣
↪dpi=160)
    axes[0].axis('off')
    for i, ax in enumerate(axes):
        if i > 0:
            corp_cur = corp[i-1]
            topic_percs, wordid_topics, wordid_phivalues = lda_model[corp_cur]
            word_dominanttopic = [(lda_model.id2word[wd], topic[0]) for wd,␣
↪topic in wordid_topics]
            ax.text(0.01, 0.5, "Doc " + 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)

    plt.subplots_adjust(wspace=0, hspace=0)
    plt.suptitle('Sentence Topic Coloring for Documents: ' + str(start) + ' to␣
↪' + str(end-2), fontsize=22, y=0.95, fontweight=700)
    plt.tight_layout()
    plt.show()

sentences_chart()
```



[15]:
```python
# Sentence Coloring of N Sentences
def topics_per_document(model, corpus, start=0, end=1):
    corpus_sel = corpus[start:end]
    dominant_topics = []
    topic_percentages = []
    for i, corp in enumerate(corpus_sel):
        topic_percs, wordid_topics, wordid_phivalues = model[corp]
        dominant_topic = sorted(topic_percs, key = lambda x: x[1], reverse=True)[0][0]
        dominant_topics.append((i, dominant_topic))
        topic_percentages.append(topic_percs)
    return(dominant_topics, topic_percentages)

dominant_topics, topic_percentages = topics_per_document(model=lda_model, corpus=corpus, end=-1)

# Distribution of Dominant Topics in Each Document
df = pd.DataFrame(dominant_topics, columns=['Document_Id', 'Dominant_Topic'])
dominant_topic_in_each_doc = df.groupby('Dominant_Topic').size()
df_dominant_topic_in_each_doc = dominant_topic_in_each_doc.to_frame(name='count').reset_index()

# Total Topic Distribution by actual weight
topic_weightage_by_doc = pd.DataFrame([dict(t) for t in topic_percentages])
```

```
df_topic_weightage_by_doc = topic_weightage_by_doc.sum().to_frame(name='count').
↪reset_index()

# Top 3 Keywords for each Topic
topic_top3words = [(i, topic) for i, topics in lda_model.
↪show_topics(formatted=False)
                                 for j, (topic, wt) in enumerate(topics) if j <␣
↪3]

df_top3words_stacked = pd.DataFrame(topic_top3words, columns=['topic_id',␣
↪'words'])
df_top3words = df_top3words_stacked.groupby('topic_id').agg(', \n'.join)
df_top3words.reset_index(level=0,inplace=True)
```

[16]:
```
from matplotlib.ticker import FuncFormatter

# Plot
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4), dpi=120, sharey=True)

# Topic Distribution by Dominant Topics
ax1.bar(x='Dominant_Topic', height='count', data=df_dominant_topic_in_each_doc,␣
↪width=.5, color='firebrick')
ax1.set_xticks(range(df_dominant_topic_in_each_doc.Dominant_Topic.unique().
↪__len__()))
tick_formatter = FuncFormatter(lambda x, pos: 'Topic ' + str(x)+ '\n' +␣
↪df_top3words.loc[df_top3words.topic_id==x, 'words'].values[0])
ax1.xaxis.set_major_formatter(tick_formatter)
ax1.set_title('Number of Documents by Dominant Topic', fontdict=dict(size=10))
ax1.set_ylabel('Number of Documents')
ax1.set_ylim(0, 1000)

# Topic Distribution by Topic Weights
ax2.bar(x='index', height='count', data=df_topic_weightage_by_doc, width=.5,␣
↪color='steelblue')
ax2.set_xticks(range(df_topic_weightage_by_doc.index.unique().__len__()))
ax2.xaxis.set_major_formatter(tick_formatter)
ax2.set_title('Number of Documents by Topic Weightage', fontdict=dict(size=10))

plt.show()
```



[19]:
```
# 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)

# 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()])
```

```python
plot = figure(title="t-SNE Clustering of {} LDA Topics".format(n_topics),
              plot_width=900, plot_height=700)
plot.scatter(x=tsne_lda[:,0], y=tsne_lda[:,1], color=mycolors[topic_num])
show(plot)
```

```
[t-SNE] Computing 91 nearest neighbors…
[t-SNE] Indexed 2351 samples in 0.001s…
[t-SNE] Computed neighbors for 2351 samples in 0.029s…
[t-SNE] Computed conditional probabilities for sample 1000 / 2351
[t-SNE] Computed conditional probabilities for sample 2000 / 2351
[t-SNE] Computed conditional probabilities for sample 2351 / 2351
[t-SNE] Mean sigma: 0.027656
[t-SNE] KL divergence after 250 iterations with early exaggeration: 58.121723
[t-SNE] KL divergence after 1000 iterations: 0.518520
```

[20]:
```python
import pyLDAvis.gensim
pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim.prepare(lda_model, corpus, dictionary=lda_model.id2word)
vis
```

[20]:
```
PreparedData(topic_coordinates=              x         y  topics  cluster
Freq
topic
2      0.115209 -0.255526       1        1  36.772083
1     -0.113986 -0.021604       2        1  29.800037
3     -0.221225  0.077041       3        1  22.042894
0      0.220003  0.200088       4        1  11.384983, topic_info=          Term
Freq         Total Category  logprob  loglift
1417      team  1193.000000  1193.000000  Default  30.0000  30.0000
619   armenian   671.000000   671.000000  Default  29.0000  29.0000
178     people  1900.000000  1900.000000  Default  28.0000  28.0000
146        god  1389.000000  1389.000000  Default  27.0000  27.0000
1045     greek   665.000000   665.000000  Default  26.0000  26.0000
…          …          …          …        …        …      …
1045     greek   621.695801   665.134827   Topic4  -4.0495   2.1053
112     attack   231.081909   429.865906   Topic4  -5.0392   1.5522
178     people   341.783081  1900.957764   Topic4  -4.6477   0.4569
930    country   139.371780   274.920319   Topic4  -5.5448   1.4935
763        way   152.372498   984.874268   Topic4  -5.4556   0.3067

[228 rows x 6 columns], token_table=      Topic      Freq      Term
term
794        2  0.996929      able
3538       2  0.991358     allow
812        2  0.997907  american
565        1  0.998456    answer
224        1  0.997413      arab
```

```
…     …        …        …
105        1  0.423480     write
105        2  0.394217     write
105        3  0.182268     write
395        2  0.574411      year
395        3  0.425214      year

[272 rows x 3 columns], R=30, lambda_step=0.01, plot_opts={'xlab': 'PC1',
'ylab': 'PC2'}, topic_order=[3, 2, 4, 1])
```