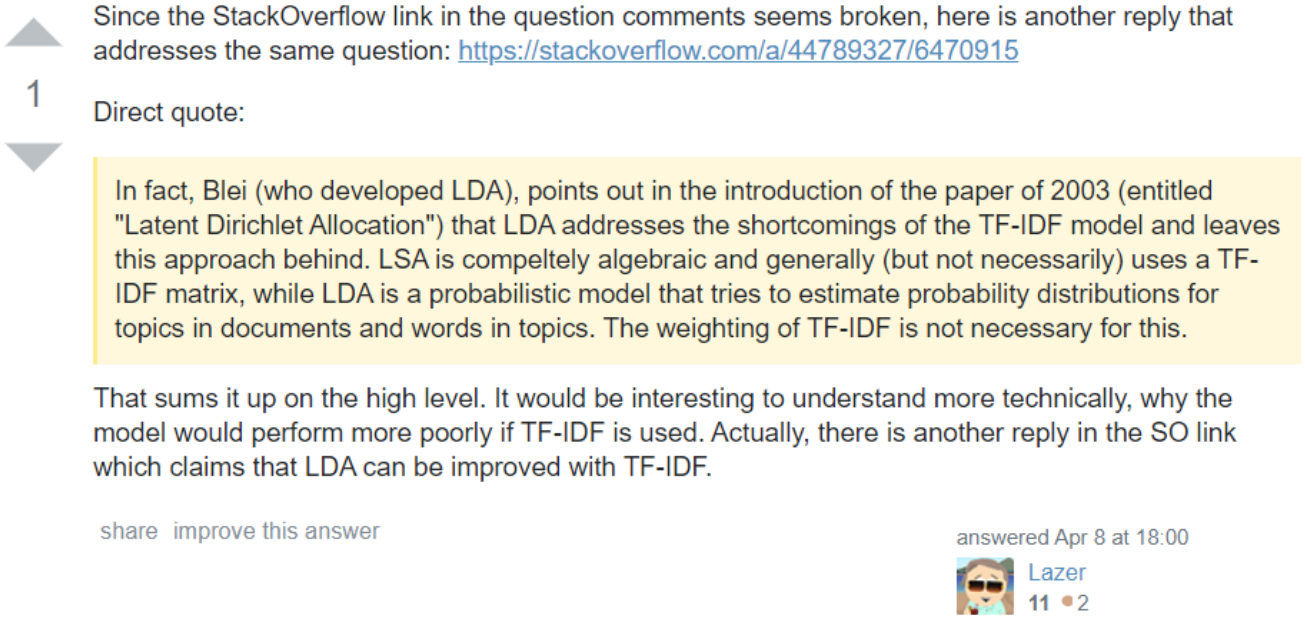
Term Frequency and Inverse Document Frequency

Necessity in LDA modeling?

In [1]:
```
from IPython.display import Image
Image(filename = "tf_idf_disclaimer.PNG", width=1000, height=1000)
```

Out[1]:



1

Since the StackOverflow link in the question comments seems broken, here is another reply that addresses the same question: https://stackoverflow.com/a/44789327/6470915

Direct quote:

> In fact, Blei (who developed LDA), points out in the introduction of the paper of 2003 (entitled "Latent Dirichlet Allocation") that LDA addresses the shortcomings of the TF-IDF model and leaves this approach behind. LSA is compeltely algebraic and generally (but not necessarily) uses a TF-IDF matrix, while LDA is a probabilistic model that tries to estimate probability distributions for topics in documents and words in topics. The weighting of TF-IDF is not necessary for this.

That sums it up on the high level. It would be interesting to understand more technically, why the model would perform more poorly if TF-IDF is used. Actually, there is another reply in the SO link which claims that LDA can be improved with TF-IDF.

share improve this answer

answered Apr 8 at 18:00

Lazer

11 2

add a comment

Latent Dirichlet Allocation

1. Import libraries

In [2]:
```
import re
import numpy as np
import pandas as pd
from pprint import pprint

# Gensim
import gensim
import gensim.corpora as corpora
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel

# Tokenizer and nltk
import nagisa
from nltk.corpus import stopwords

import spacy

# Plotting tools
import pyLDAvis
import pyLDAvis.gensim  # don't skip this
import matplotlib.pyplot as plt
%matplotlib inline

# Enable logging for gensim - optional
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.ERROR)

import warnings
warnings.filterwarnings("ignore",category=DeprecationWarning)
```

2. Prepare stopwords

In [3]:
```
jp_stopwords = pd.read_csv('stopwords.csv', encoding='utf-8')
all_stopwords = [x for x in jp_stopwords.word]

en_stopwords = stopwords.words('english')
all_stopwords.extend(en_stopwords)
```

3. Prepare data

In [4]:
```
tweets = pd.read_csv('DragonBallonTwitter.csv', encoding='utf-8')[['content']]

# Convert to list of strings
data = tweets.content.values.tolist()
data = [str(text).lower() for text in data]

# Remove urls
data = [re.sub(r'(https?:\/\/)?(www\.)?[-a-zA-Z0-9@:%._\+~#=]{1,256}\.[a-zA-Z0-9()]{1,6}\b([-a-zA-Z0-9()@:%_\+.~#?&//=]*)',
               '', text) for text in data]

# Remove distracting single quotes
data = [re.sub("\'", "", text) for text in data]

# Remove punctuations
jp_punctuations = r'[\u3000-\u303F]|[\u2605-\u2606]|[\u2190-\u2195]|[\uFF00-\uFFEF]|\u203B|`|´'
data = [re.sub(jp_punctuations, "", text) for text in data]

en_punctuations = r'[\-\!\(\)\[\];\:\'\"\\\,\<\>\.\/\@\#\$\%\^\&\*\_\~]'
data = [re.sub(en_punctuations, " ", text) for text in data]

# Remove new line characters
data = [re.sub(r'\s+', ' ', text) for text in data]

# Remove new line characters
data = [re.sub(r'(ドラゴンボール|dragonball|dragonballz|dbz)', '', text, flags = re.IGNORECASE) for text in data]
```

Note

- Words equivalent to the chosen hashtag ドラゴンボール have been removed

4. Tokenize Japanese words

In [5]:
```
def send_to_words(sentences):
    sentence_list = []
    doc_number = 0

    for sentence in sentences:
        tokens = nagisa.filter(sentence, filter_postags=['空白', '補助記号', '助詞'])
        sentence_list.append(tokens.words)

    return sentence_list

data_words = send_to_words(data)
```

In [6]:
```
def rejoin_verb_enders(texts_list):
    undesired_str = 'To-ReMoVe-VaLue'
    out_list = []

    for texts in texts_list:
        for i in range(1, len(texts)):
            if(texts[i] in ['た','る','て','てる','ている','てた','ていた','たい','ます','ました','ません','ない','すぎ','すぎる']):
                texts[i] = texts[i - 1] + texts[i]
                texts[i - 1] = undesired_str
            elif(texts[i] == 'バトル'):
                if(texts[i - 1] == 'ドッカン'):
                    texts[i] = texts[i - 1] + texts[i]
                    texts[i - 1] = undesired_str

        texts = list(filter(lambda a: a != undesired_str, texts))
        out_list.append(texts)

    return out_list

data_words = rejoin_verb_enders(data_words)
```

Notes

- There is no perfect tokenizer available for Japanese texts
- Issues exist for agglutinative language
- Verb-enders have been parsed into single characters

5. Creating N-gram Models

In [7]:
```
# Build the bigram and trigram models
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)

def process_words(texts, stop_words=all_stopwords, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']):

    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]

    return texts

data_ready = process_words(data_words)  # processed Text Data!
```

```
collecting all words and their counts
PROGRESS: at sentence #0, processed 0 words and 0 word types
collected 18986 word types from a corpus of 27144 words (unigram + bigrams) and 1393 sentences
using 18986 counts as vocab in Phrases<0 vocab, min_count=5, threshold=100, max_vocab_size=40000000>
collecting all words and their counts
PROGRESS: at sentence #0, processed 0 words and 0 word types
collected 19005 word types from a corpus of 25642 words (unigram + bigrams) and 1393 sentences
using 19005 counts as vocab in Phrases<0 vocab, min_count=5, threshold=100, max_vocab_size=40000000>
source_vocab length 18986
Phraser built with 101 phrasegrams
source_vocab length 19005
Phraser built with 112 phrasegrams
```

Note

- Further analysis for quadgram has been done but it did not make any significant difference with trigram

7. Dictionary and Corpus needed for Topic Modeling

In [8]:
```
# Create dictionary
id2word = corpora.Dictionary(data_ready)

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

texts = data_ready
```

```
adding document #0 to Dictionary(0 unique tokens: [])
built Dictionary(4559 unique tokens: ['add', 'arrived', 'collection', 'figures', 'hatsunemiku']...) from 1393 documents (total 166
```

In [9]:
```
# Human readable format of corpus (term-frequency)
readable_corpus = [[(id2word[id], freq) for id, freq in cp] for cp in corpus[:1]]
```

8. Find the optimal number of topics for LDA

Notes

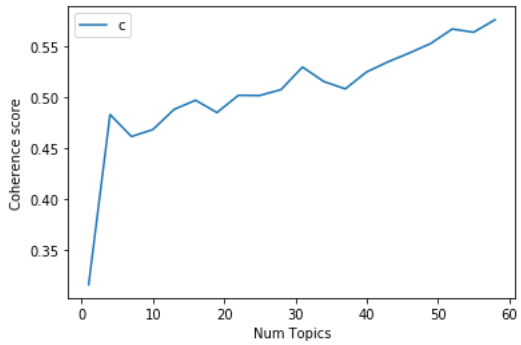
- Gensim package's ordinary LDA and wrapper for Mallet LDA have been used
- Mallet's version often gives better quality for topics
- Both functions below sweeps the hyperparameter 'num_topics' to get better models
- Topic Coherence is used to measure how good a set of topic models is. The higher the better model.

First, the word set t is segmented into a set of pairs of word subsets S. Second, word probabilities P are computed based on a given reference corpus. Both, the set of word subs probabilities P are consumed by the confirmation measure to calculate the agreements ϕ of pairs of S. Last, those values are aggregated to a single **coherence value**.

In [10]:
```
import os
os.environ.update({'MALLET_HOME':r'C:/Users/CD250050/Documents/jupyter-workspace/topic-modeling/mallet-2.0.8/'})

mallet_path = r'C:\Users\CD250050\Documents\jupyter-workspace\topic-modeling\mallet-2.0.8\bin\mallet' # update this path

def compute_coherence_values(dictionary, corpus, texts, limit, start=2, step=3):
    """
    Compute c_v coherence for various number of topics

    Parameters:
    ----------
    dictionary : Gensim dictionary
    corpus : Gensim corpus
    texts : List of input texts
    limit : Max num of topics

    Returns:
    -------
    model_list : List of LDA topic models
    coherence_values : Coherence values corresponding to the LDA model with respective number of topics
    """
    coherence_values = []
    model_list = []
    for num_topics in range(start, limit, step):
        model = gensim.models.wrappers.LdaMallet(mallet_path, corpus=corpus, num_topics=num_topics, id2word=id2word)
        model_list.append(model)
        coherencemodel = CoherenceModel(model=model, texts=texts, dictionary=dictionary, coherence='c_v')
        coherence_values.append(coherencemodel.get_coherence())

    return model_list, coherence_values

def ordinary_compute_coherence_values(dictionary, corpus, texts, limit, start=2, step=3):
    """
    Compute c_v coherence for various number of topics

    Parameters:
    ----------
    dictionary : Gensim dictionary
    corpus : Gensim corpus
    texts : List of input texts
    limit : Max num of topics

    Returns:
    -------
    model_list : List of LDA topic models
    coherence_values : Coherence values corresponding to the LDA model with respective number of topics
    """
    coherence_values = []
    model_list = []
    for num_topics in range(start, limit, step):
        model = gensim.models.ldamodel.LdaModel(corpus=corpus, id2word=id2word, num_topics=num_topics, random_state=100,
                                                update_every=1, chunksize=100, passes=10, alpha='symmetric', per_word_topics=True
        model_list.append(model)
        coherencemodel = CoherenceModel(model=model, texts=texts, dictionary=dictionary, coherence='c_v')
        coherence_values.append(coherencemodel.get_coherence())

    return model_list, coherence_values
```

In [11]:
```
# Can take a long time to run.
model_list_mallet, coherence_values = compute_coherence_values(dictionary=id2word, corpus=corpus, texts=texts, start=1, limit=60,
```

In [12]:
```
# Show graph
limit=60; start=1; step=3;
x = range(start, limit, step)
plt.plot(x, coherence_values)
plt.xlabel("Num Topics")
plt.ylabel("Coherence score")
plt.legend(("coherence_values"), loc='best')
plt.show()
```



Notes

- Picking an even higher value can sometimes provide more granular sub-topics
- But this would include repeated word/s inside a topic

In [13]:
```
# Print the coherence scores
for m, cv in zip(x, coherence_values):
    print("Num Topics =", m, " has Coherence Value of", round(cv, 4))
```

```
Num Topics = 1  has Coherence Value of 0.3159
Num Topics = 4  has Coherence Value of 0.4828
Num Topics = 7  has Coherence Value of 0.4612
Num Topics = 10  has Coherence Value of 0.468
Num Topics = 13  has Coherence Value of 0.4879
Num Topics = 16  has Coherence Value of 0.4968
Num Topics = 19  has Coherence Value of 0.4848
Num Topics = 22  has Coherence Value of 0.5017
Num Topics = 25  has Coherence Value of 0.5015
Num Topics = 28  has Coherence Value of 0.5073
Num Topics = 31  has Coherence Value of 0.5294
Num Topics = 34  has Coherence Value of 0.5152
Num Topics = 37  has Coherence Value of 0.508
Num Topics = 40  has Coherence Value of 0.5246
Num Topics = 43  has Coherence Value of 0.5346
Num Topics = 46  has Coherence Value of 0.5432
Num Topics = 49  has Coherence Value of 0.5526
Num Topics = 52  has Coherence Value of 0.5667
Num Topics = 55  has Coherence Value of 0.5636
Num Topics = 58  has Coherence Value of 0.5758
```

Note

- num_topic = 4 seems relatively high in the range [1,30]

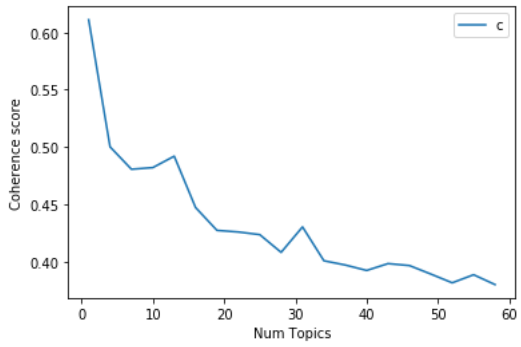
In [14]:
```
lda_model_mallet = model_list_mallet[1]
lda_model_mallet.print_topics()
```

```
topic #0 (12.500): 0.033*"悟空" + 0.027*"フィギュア" + 0.027*"ベジータ" + 0.023*"孫悟" + 0.011*"ください" + 0.009*"db" + 0.009*"登場"
topic #1 (12.500): 0.058*"ました" + 0.029*"レジェンズ" + 0.020*"たら" + 0.016*"ドカバト" + 0.012*"悟飯" + 0.011*"フリーザ" + 0.008*"動
6*"販売"
topic #2 (12.500): 0.131*"ドッカンバトル" + 0.049*"世界" + 0.048*"突破" + 0.048*"dl" + 0.046*"イベント" + 0.046*"公式" + 0.046*"サイト
*"盛り"
topic #3 (12.500): 0.031*"繋がりたい" + 0.027*"絵描き" + 0.026*"サイヤ" + 0.025*"イラスト" + 0.024*"好き" + 0.021*"ゴジータ" + 0.015*
013*"ヒーローズ"
```

Out[14]:
```
[(0,
  '0.033*"悟空" + 0.027*"フィギュア" + 0.027*"ベジータ" + 0.023*"孫悟" + 0.011*"ください" + 0.009*"db" + 0.009*"登場" + 0.009*"情報"'),
 (1,
  '0.058*"ました" + 0.029*"レジェンズ" + 0.020*"たら" + 0.016*"ドカバト" + 0.012*"悟飯" + 0.011*"フリーザ" + 0.008*"動画" + 0.008*"ド'),
 (2,
  '0.131*"ドッカンバトル" + 0.049*"世界" + 0.048*"突破" + 0.048*"dl" + 0.046*"イベント" + 0.046*"公式" + 0.046*"サイト" + 0.040*"開催'),
 (3,
  '0.031*"繋がりたい" + 0.027*"絵描き" + 0.026*"サイヤ" + 0.025*"イラスト" + 0.024*"好き" + 0.021*"ゴジータ" + 0.015*"ブロリー" + 0.01
ズ"')]
```

In [15]:
```
model_list_ord, coherence_values_ord = ordinary_compute_coherence_values(dictionary=id2word, corpus=corpus, texts=texts, start=1,
```

In [16]:
```
# Show graph
limit=60; start=1; step=3;
x = range(start, limit, step)
plt.plot(x, coherence_values_ord)
plt.xlabel("Num Topics")
plt.ylabel("Coherence score")
plt.legend(("coherence_values"), loc='best')
plt.show()
```



Notes

- Gensim's ordinary LDA gives degenerating models as the number of topics increases
- Similarly, num_topics = 4 seems not a bad choice

In [17]:
```
# Print the coherence scores
for m, cv in zip(x, coherence_values_ord):
    print("Num Topics =", m, " has Coherence Value of", round(cv, 4))
```

```
Num Topics = 1  has Coherence Value of 0.611
Num Topics = 4  has Coherence Value of 0.5002
Num Topics = 7  has Coherence Value of 0.4807
Num Topics = 10  has Coherence Value of 0.4821
Num Topics = 13  has Coherence Value of 0.4921
Num Topics = 16  has Coherence Value of 0.4473
Num Topics = 19  has Coherence Value of 0.4274
Num Topics = 22  has Coherence Value of 0.426
Num Topics = 25  has Coherence Value of 0.4237
Num Topics = 28  has Coherence Value of 0.4082
Num Topics = 31  has Coherence Value of 0.4304
Num Topics = 34  has Coherence Value of 0.4009
Num Topics = 37  has Coherence Value of 0.3972
Num Topics = 40  has Coherence Value of 0.3925
Num Topics = 43  has Coherence Value of 0.3984
Num Topics = 46  has Coherence Value of 0.3967
Num Topics = 49  has Coherence Value of 0.3893
Num Topics = 52  has Coherence Value of 0.3817
Num Topics = 55  has Coherence Value of 0.3887
Num Topics = 58  has Coherence Value of 0.3802
```

In [18]:
```
lda_model_ord = model_list_ord[1]
lda_model_ord.print_topics()
```

```
topic #0 (0.250): 0.122*"ドッカンバトル" + 0.046*"ガシャ" + 0.044*"世界" + 0.044*"公式" + 0.043*"サイト" + 0.043*"突破" + 0.043*"dl"
り"
topic #1 (0.250): 0.020*"ベジータ" + 0.010*"フリーザ" + 0.006*"dragon_ball" + 0.006*"リペイント" + 0.006*"ワン_ピース" + 0.005*"super
*"シーン" + 0.004*"ブルマ"
topic #2 (0.250): 0.028*"ました" + 0.020*"繋がりたい" + 0.017*"イラスト" + 0.016*"絵描き" + 0.014*"たら" + 0.013*"好き" + 0.013*"悟空
*"模写"
topic #3 (0.250): 0.020*"フィギュア" + 0.017*"孫悟" + 0.013*"レジェンズ" + 0.013*"ドカバト" + 0.012*"ました" + 0.010*"スーパー" + 0.0
い" + 0.008*"入荷"
```

Out[18]:
```
[(0,
  '0.122*"ドッカンバトル" + 0.046*"ガシャ" + 0.044*"世界" + 0.044*"公式" + 0.043*"サイト" + 0.043*"突破" + 0.043*"dl" + 0.042*"イベン
 (1,
  '0.020*"ベジータ" + 0.010*"フリーザ" + 0.006*"dragon_ball" + 0.006*"リペイント" + 0.006*"ワン_ピース" + 0.005*"super" + 0.005*"ベジ
*"ブルマ"'),
 (2,
  '0.028*"ました" + 0.020*"繋がりたい" + 0.017*"イラスト" + 0.016*"絵描き" + 0.014*"たら" + 0.013*"好き" + 0.013*"悟空" + 0.011*"ゴジ
 (3,
  '0.020*"フィギュア" + 0.017*"孫悟" + 0.013*"レジェンズ" + 0.013*"ドカバト" + 0.012*"ました" + 0.010*"スーパー" + 0.010*"ヒーローズ"
荷"')]
```

9. Finding the dominant topic in each sentence

In [19]:
```
def format_topics_sentences(ldamodel=None, corpus=corpus, texts=data, mallet=False):
    # Init output
    sent_topics_df = pd.DataFrame()

    # Get main topic in each document
    for i, row_list in enumerate(ldamodel[corpus]):
        if(mallet):
            row = row_list
        else:
            row = row_list[0] if ldamodel.per_word_topics else row_list

        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_i
            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, sort=True)
    return(sent_topics_df)

# Ordinary
df_topic_sents_keywords_ord = format_topics_sentences(ldamodel=lda_model_ord, corpus=corpus, texts=data_ready)
df_dominant_topic_ord = df_topic_sents_keywords_ord.reset_index()
df_dominant_topic_ord.columns = ['Document_No', 'Dominant_Topic', 'Topic_Perc_Contrib', 'Keywords', 'Text']
df_dominant_topic_ord.head(10)
```

Out[19]:

	Document_No	Dominant_Topic	Topic_Perc_Contrib	Keywords	
0	0	1.0	0.9305	ベジータ, フリーザ, dragon_ball, リペイント, ワン_ピース, super,...	[two
1	1	3.0	0.7468	フィギュア, 孫悟, レジェンズ, ドカバト, ました, スーパー, ヒーローズ, サイヤ,...	
2	2	3.0	0.6714	フィギュア, 孫悟, レジェンズ, ドカバト, ました, スーパー, ヒーローズ, サイヤ,...	[一番_くじ, 発売, greates
3	3	3.0	0.6902	フィギュア, 孫悟, レジェンズ, ドカバト, ました, スーパー, ヒーローズ, サイヤ,...	[キャンバスボード, クリア, ファイル
4	4	1.0	0.5793	ベジータ, フリーザ, dragon_ball, リペイント, ワン_ピース, super,...	[カッコ, エロイラスト, 終わり, ラスト
5	5	2.0	0.8566	ました, 繋がりたい, イラスト, 絵描き, たら, 好き, 悟空, ゴジータ, 今日, 模写	[一番, くじやり, ました, タオル, 当
6	6	3.0	0.6800	フィギュア, 孫悟, レジェンズ, ドカバト, ました, スーパー, ヒーローズ, サイヤ,...	[やっ, も
7	7	1.0	0.6848	ベジータ, フリーザ, dragon_ball, リペイント, ワン_ピース, super,...	[友人, イケ, ブロ, 当たる, ハプニ
8	8	2.0	0.7473	ました, 繋がりたい, イラスト, 絵描き, たら, 好き, 悟空, ゴジータ, 今日, 模写	
9	9	3.0	0.6139	フィギュア, 孫悟, レジェンズ, ドカバト, ました, スーパー, ヒーローズ, サイヤ,...	

Note

- When the number of words in a document increases, there is a match in the words of the most dominantly-assigned topic

In [20]:
```
# Ordinary
df_topic_sents_keywords_mallet = format_topics_sentences(ldamodel=lda_model_mallet,
                                                         corpus=corpus, texts=data_ready, mallet=True)
df_dominant_topic_mallet = df_topic_sents_keywords_mallet.reset_index()
df_dominant_topic_mallet.columns = ['Document_No', 'Dominant_Topic', 'Topic_Perc_Contrib', 'Keywords', 'Text']
df_dominant_topic_mallet.head(10)
```

```
serializing temporary corpus to C:\Users\CD250050\AppData\Local\Temp\b7df44_corpus.txt
converting temporary corpus to MALLET format with C:\Users\CD250050\Documents\jupyter-workspace\topic-modeling\mallet-2.0.8\bin\ma
uence --remove-stopwords --token-regex "\S+" --input C:\Users\CD250050\AppData\Local\Temp\b7df44_corpus.txt --output C:\Users\CD25
t.infer --use-pipe-from C:\Users\CD250050\AppData\Local\Temp\b7df44_corpus.mallet
inferring topics with MALLET LDA 'C:\Users\CD250050\Documents\jupyter-workspace\topic-modeling\mallet-2.0.8\bin\mallet infer-topic
emp\b7df44_corpus.mallet.infer --inferencer C:\Users\CD250050\AppData\Local\Temp\b7df44_inferencer.mallet --output-doc-topics C:\U
topics.txt.infer --num-iterations 100 --doc-topics-threshold 0.0 --random-seed 0'
```

Out[20]:

	Document_No	Dominant_Topic	Topic_Perc_Contrib	Keywords	
0	0	3.0	0.2979	繋がりたい, 絵描き, サイヤ, イラスト, 好き, ゴジータ, ブロリー, いい, この,...	[two, n
1	1	3.0	0.2596	繋がりたい, 絵描き, サイヤ, イラスト, 好き, ゴジータ, ブロリー, いい, この,...	
2	2	0.0	0.3095	悟空, フィギュア, ベジータ, 孫悟, ください, db, 登場, 情報, 入荷, 予定	[一番_くじ, 発売, greatest,
3	3	0.0	0.2797	悟空, フィギュア, ベジータ, 孫悟, ください, db, 登場, 情報, 入荷, 予定	[キャンバスボード, クリア, ファイル,
4	4	1.0	0.2702	ました, レジェンズ, たら, ドカバト, 悟飯, フリーザ, 動画, ドッカン, 漫画, 販売	[カッコ, エロイラスト, 終わり, ラスト,
5	5	3.0	0.2887	繋がりたい, 絵描き, サイヤ, イラスト, 好き, ゴジータ, ブロリー, いい, この,...	[一番, くじやり, ました, タオル, 当た
6	6	0.0	0.2827	悟空, フィギュア, ベジータ, 孫悟, ください, db, 登場, 情報, 入荷, 予定	[やっ, ちま
7	7	0.0	0.3145	悟空, フィギュア, ベジータ, 孫悟, ください, db, 登場, 情報, 入荷, 予定	[友人, イケ, ブロ, 当たる, ハプニ
8	8	0.0	0.2596	悟空, フィギュア, ベジータ, 孫悟, ください, db, 登場, 情報, 入荷, 予定	
9	9	2.0	0.2647	ドッカンバトル, 世界, 突破, dl, イベント, 公式, サイト, 開催, だくさん, 盛り	

11. The most representative sentence for each topic

In [21]:
```
# Display setting to show more characters in column (Mallet)
pd.options.display.max_colwidth = 100

sent_topics_sorteddf_mallet = pd.DataFrame()
sent_topics_outdf_grpd = df_topic_sents_keywords_mallet.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, sort=True)

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

# Format
sent_topics_sorteddf_mallet.columns = ['Topic_Num_Mallet', "Topic_Perc_Contrib", "Keywords", "Representative Text"]

# Show
sent_topics_sorteddf_mallet.head(10)
```

Out[21]:

	Topic_Num_Mallet	Topic_Perc_Contrib	Keywords	
0	0.0	0.4285	悟空, フィギュア, ベジータ, 孫悟, ください, db, 登場, 情報, 入荷, 予定	[入荷, 情報, materia, son_gokou, 素材, こだわった, materia, シ
1	1.0	0.4587	ました, レジェンズ, たら, ドカバト, 悟飯, フリーザ, 動画, ドッカン, 漫画, 販売	[ドッカンバトル, 販売, してます, 新孫, 悟飯, lr, 所持, 気軽, dm
2	2.0	0.4643	ドッカンバトル, 世界, 突破, dl, イベント, 公式, サイト, 開催, だくさん, 盛り	[ドッカンバトル, 世界, dl, 突破, イベント, 盛り, だくさん, 世界
3	3.0	0.5135	繋がりたい, 絵描き, サイヤ, イラスト, 好き, ゴジータ, ブロリー, いい, この, ヒーローズ	[おはよー, 今日, イラスト, 鳥山, 明風, 描い_みた, 鳥山, ドラゴ

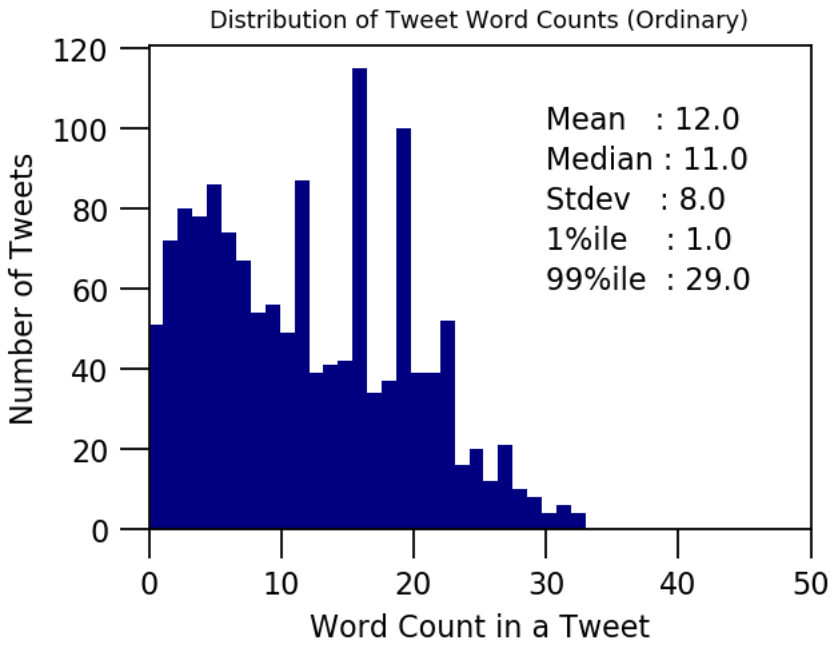
In [22]:
```
# Display setting to show more characters in column (Ordinary)
pd.options.display.max_colwidth = 100

sent_topics_sorteddf_ord = pd.DataFrame()
sent_topics_outdf_grpd_ord = df_topic_sents_keywords_ord.groupby('Dominant_Topic')

for i, grp in sent_topics_outdf_grpd_ord:
    sent_topics_sorteddf_ord = pd.concat([sent_topics_sorteddf_ord,
                                          grp.sort_values(['Perc_Contribution'], ascending=False).head(1)],
                                         axis=0, sort=True)

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

# Format
sent_topics_sorteddf_ord.columns = ['Topic_Num_Ord', "Topic_Perc_Contrib", "Keywords", "Representative Text"]

# Show
sent_topics_sorteddf_ord.head(10)
```

Out[22]:

	Topic_Num_Ord	Topic_Perc_Contrib	Keywords	
0	0.0	0.9658	ドッカンバトル, ガシャ, 世界, 公式, サイト, 突破, dl, イベント, 開催, 盛り	[ドッカンバトル, 世界, dl, 突破, イベント, 盛り, だくさん
1	1.0	0.9762	ベジータ, フリーザ, dragon_ball, リベイント, ワン_ピース, super, ベジット, goku, シーン, ブルマ	[さゆりんご, 軍団, 乃木坂, 研究, 時代, カンペ, スケッチ,
2	2.0	0.9698	ました, 繋がりたい, イラスト, 絵描き, たら, 好き, 悟空, ゴジータ, 今日, 模写	[塗る, めんど, くさく, なっ, 時間, かかる, 思った, 線画,
3	3.0	0.9766	フィギュア, 孫悟, レジェンズ, ドカバト, ました, スーパー, ヒーローズ, サイヤ, いい, 入荷	[神奈川, 店頭, 悟空, カメハメ, 当時, 珍しい, 商品, 入り,

12. Frequency Distribution of Word Counts in Documents

In [23]:
```
doc_lens = [len(d) for d in df_dominant_topic_ord.Text]

# Plot
plt.figure(figsize=(4,3), dpi=160)
plt.hist(doc_lens, bins = 30, color='navy')
plt.text(30, 100, "Mean   : " + str(round(np.mean(doc_lens))))
plt.text(30,  90, "Median : " + str(round(np.median(doc_lens))))
plt.text(30,  80, "Stdev   : " + str(round(np.std(doc_lens))))
plt.text(30,  70, "1%ile    : " + str(round(np.quantile(doc_lens, q=0.01))))
plt.text(30,  60, "99%ile  : " + str(round(np.quantile(doc_lens, q=0.99))))

plt.gca().set(xlim=(0,50), ylabel='Number of Tweets', xlabel='Word Count in a Tweet')
plt.tick_params(size=10)
plt.xticks(np.linspace(0,50,6))
plt.title('Distribution of Tweet Word Counts (Ordinary)', fontdict=dict(size=8))
plt.show()
```

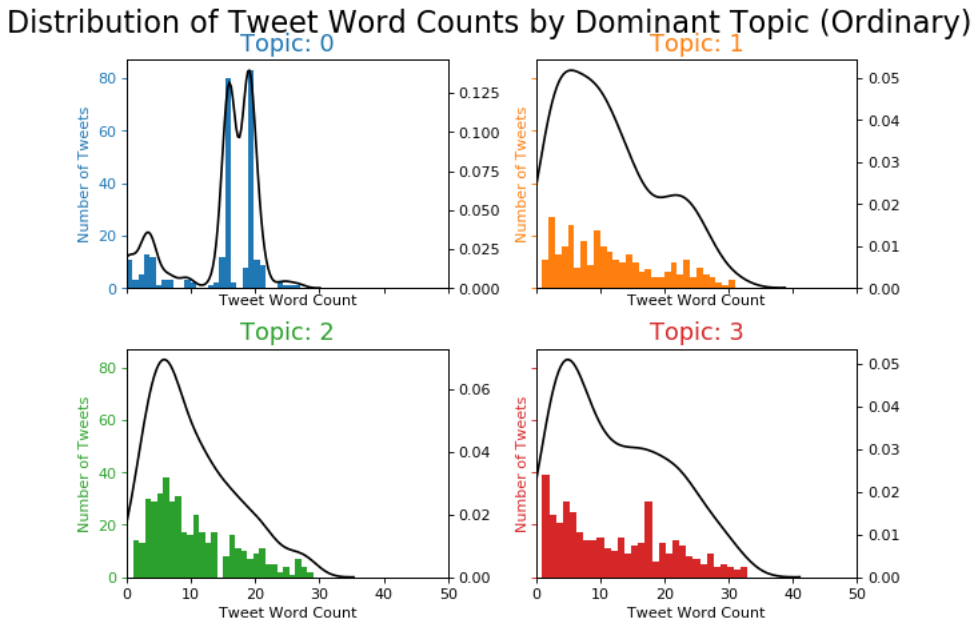


Notes

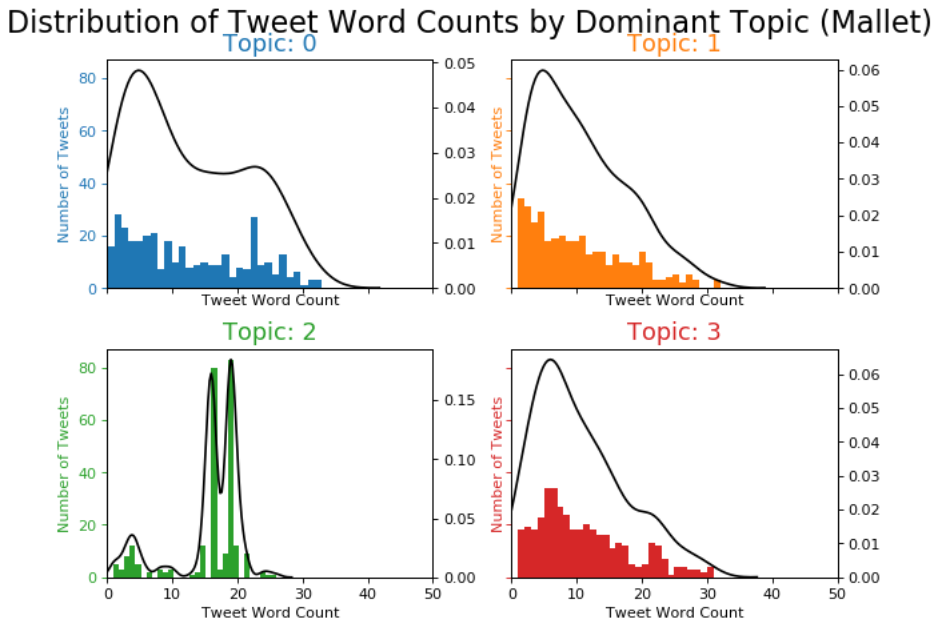
- More than 100 tweets are composed of around 16 meaningful words.
- Around 100 tweets with 19 words.
- On the average, a tweet is composed of 12 words

In [24]:

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

fig, axes = plt.subplots(2,2,figsize=(8,6), dpi=80, sharex=True, sharey=True)

for i, ax in enumerate(axes.flatten()):
    df_dominant_topic_sub = df_dominant_topic_ord.loc[df_dominant_topic_ord.Dominant_Topic == i, :]
    doc_lens = [len(d) for d in df_dominant_topic_sub.Text]
    ax.hist(doc_lens, bins = 30, 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, 50), xlabel='Tweet Word Count')
    ax.set_ylabel('Number of Tweets', 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(0,50,6))
fig.suptitle('Distribution of Tweet Word Counts by Dominant Topic (Ordinary)', fontsize=20)
plt.show()
```



In [25]:
```
cols = [color for name, color in mcolors.TABLEAU_COLORS.items()]  # more colors: 'mcolors.XKCD_COLORS'

fig, axes = plt.subplots(2,2,figsize=(8,6), dpi=80, sharex=True, sharey=True)

for i, ax in enumerate(axes.flatten()):
    df_dominant_topic_sub = df_dominant_topic_mallet.loc[df_dominant_topic_mallet.Dominant_Topic == i, :]
    doc_lens = [len(d) for d in df_dominant_topic_sub.Text]
    ax.hist(doc_lens, bins = 30, 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, 50), xlabel='Tweet Word Count')
    ax.set_ylabel('Number of Tweets', 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(0,50,6))
fig.suptitle('Distribution of Tweet Word Counts by Dominant Topic (Mallet)', fontsize=20)
plt.show()
```



13. Word Clouds of Top N Keywords in Each Topic (Ordinary LDA from here onwards)

In [26]:
```
# 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'

fpath = r"C:\Windows\Fonts\UDDigiKyokashoN-R.ttc"

cloud = WordCloud(stopwords=all_stopwords,
                  background_color='white',
                  font_path=fpath,
                  width=1500, #width=2500,
                  height=800, #height=1800,
                  max_words=10,
                  colormap='tab10',
                  prefer_horizontal=1.0)

def plot_wordcloud(lda_model):
    topics = lda_model.show_topics(formatted=False)

    fig, axes = plt.subplots(2, 2, figsize=(8,8), 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 [27]:
```
plot_wordcloud(lda_model_ord)
```

Above is for ordinary LDA by Gensims

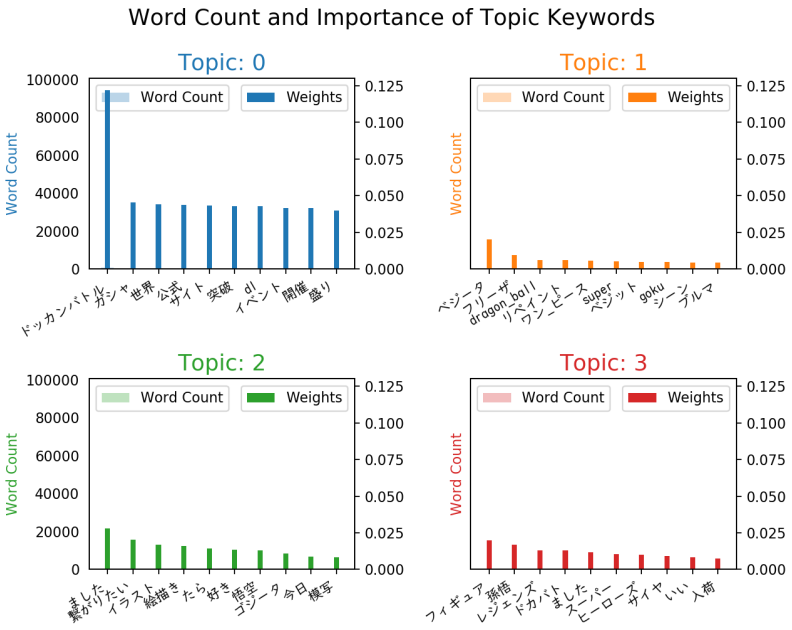
- Topic 0: Dokkan Battle, Gasha, Public, Site, Opening, DL, Peak, Opening, World, Breakthough, Event ⇒ **Dokkan Battle opening**
- Topic 1: Vejeta, Freeza, Scene, Repaint, Goku, Bulma, One_Piece ⇒ **Dragon Ball characters**
- Topic 2: Wanting to connect, Drawings, Gojita, Illustration, Copy, Likes, Goku ⇒ **Connecting artworks**
- Topic 3: Toy figure, Legends, DokaBat, Goku, Super saiyan, Heros, Arrival of goods ⇒ **Toy figures**

Below is Mallet's version

- Topic 0: Toy Figures, Goku, Vejeta, Introduction (into a market), Information ⇒ **Toy figures into market**
- Topic 1: Legends, Freeza, Gohan, Dokkan, Manga, Video, Selling ⇒ **DB Characters in Manga and Video**
- Topic 2: Dokkan Battle, Public, World, Peak, Opening, DL, Event, Site, Breakthrough ⇒ **Dokkan Battle opening**
- Topic 3: Wanting to connect, Drawings, Illustration, Like, Saiyan, Good, Gojita ⇒ **Desire to connect drawing and likings**

In [28]:
```
plot_wordcloud(lda_model_mallet)
```

In [29]:
```
from matplotlib.font_manager import FontProperties
from collections import Counter
topics = lda_model_ord.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=(8,6), sharey=True, dpi=160)
cols = [color for name, color in mcolors.TABLEAU_COLORS.items()]
font_prop = FontProperties(fname=fpath)


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, 0.130); ax.set_ylim(0, 100500)
    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', fontproperties=font_prop)
    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=16, y=1.05)
plt.show()
```

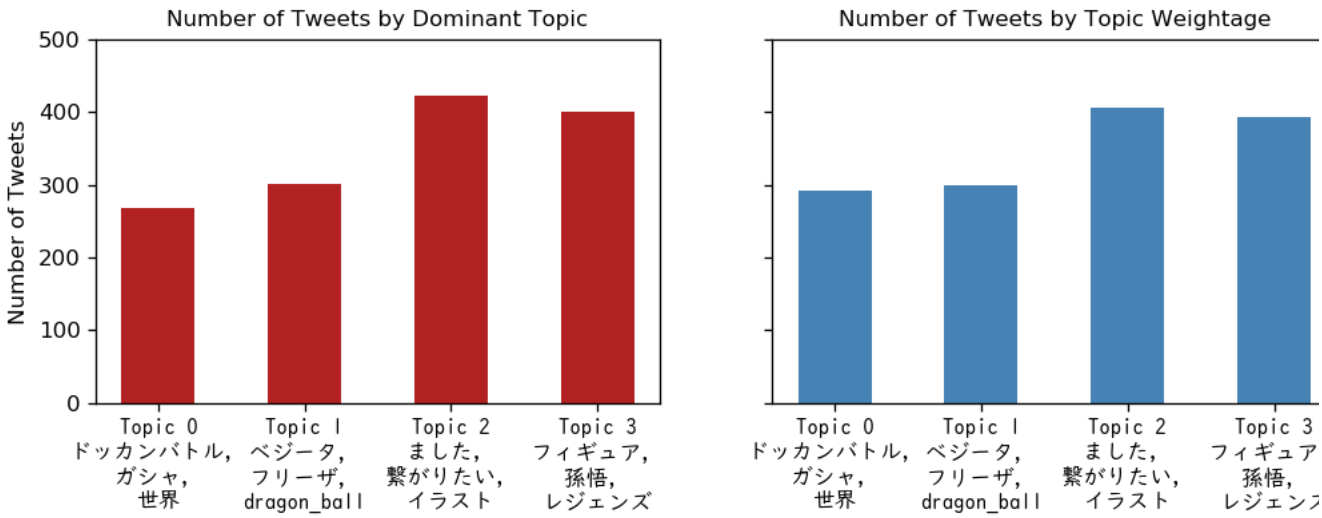


14. What are the most discussed topics in the documents?

In [30]:

```
# 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_ord, 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_ord.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)
```

In [31]:
```
from matplotlib.ticker import FuncFormatter

# Plot
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 3), 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'].value
ax1.xaxis.set_major_formatter(tick_formatter)

for tick in ax1.xaxis.get_major_ticks():
    tick.label.set_fontproperties(font_prop)

ax1.set_title('Number of Tweets by Dominant Topic', fontdict=dict(size=10))
ax1.set_ylabel('Number of Tweets')
ax1.set_ylim(0, 500)

# 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)

for tick in ax2.xaxis.get_major_ticks():
    tick.label.set_fontproperties(font_prop)

ax2.set_title('Number of Tweets by Topic Weightage', fontdict=dict(size=10))

plt.show()
```



Note: Above topics are from the Ordinary model

- Tweets belonging to Topic 2 (Connecting artworks) and 3 (Toy figures) are most frequent

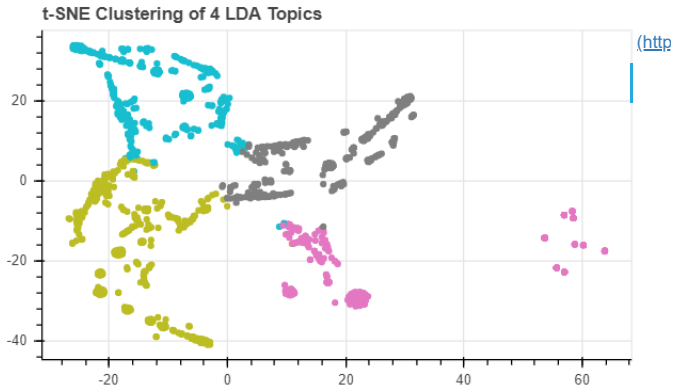
15. t-SNE Clustering Chart

In [32]:
```
# 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_ord[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()])
plot = figure(title="t-SNE Clustering of {} LDA Topics".format(n_topics),
              plot_width=500, plot_height=300)
plot.scatter(x=tsne_lda[:,0], y=tsne_lda[:,1], color=mycolors[topic_num + 6])
show(plot)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 1379 samples in 0.013s...
[t-SNE] Computed neighbors for 1379 samples in 0.023s...
[t-SNE] Computed conditional probabilities for sample 1000 / 1379
[t-SNE] Computed conditional probabilities for sample 1379 / 1379
[t-SNE] Mean sigma: 0.000000
[t-SNE] KL divergence after 250 iterations with early exaggeration: 53.735733
[t-SNE] KL divergence after 1000 iterations: 0.272303
```

(https://bokeh.pydata.org)

BokehJS 1.2.0 successfully loaded.



t-distributed Stochastic Neighbor Embedding

- t-SNE is a tool to visualize high-dimensional data. It converts similarities between data points to joint probabilities and tries to minimize the *Kullback-Leibler divergence* betw dimensional embedding and the high-dimensional data.
- The relevant information is in the relative distances between low dimensional points. t-SNE captures structure in the sense that neighboring points in the input space will ten dimensional space.

Notes

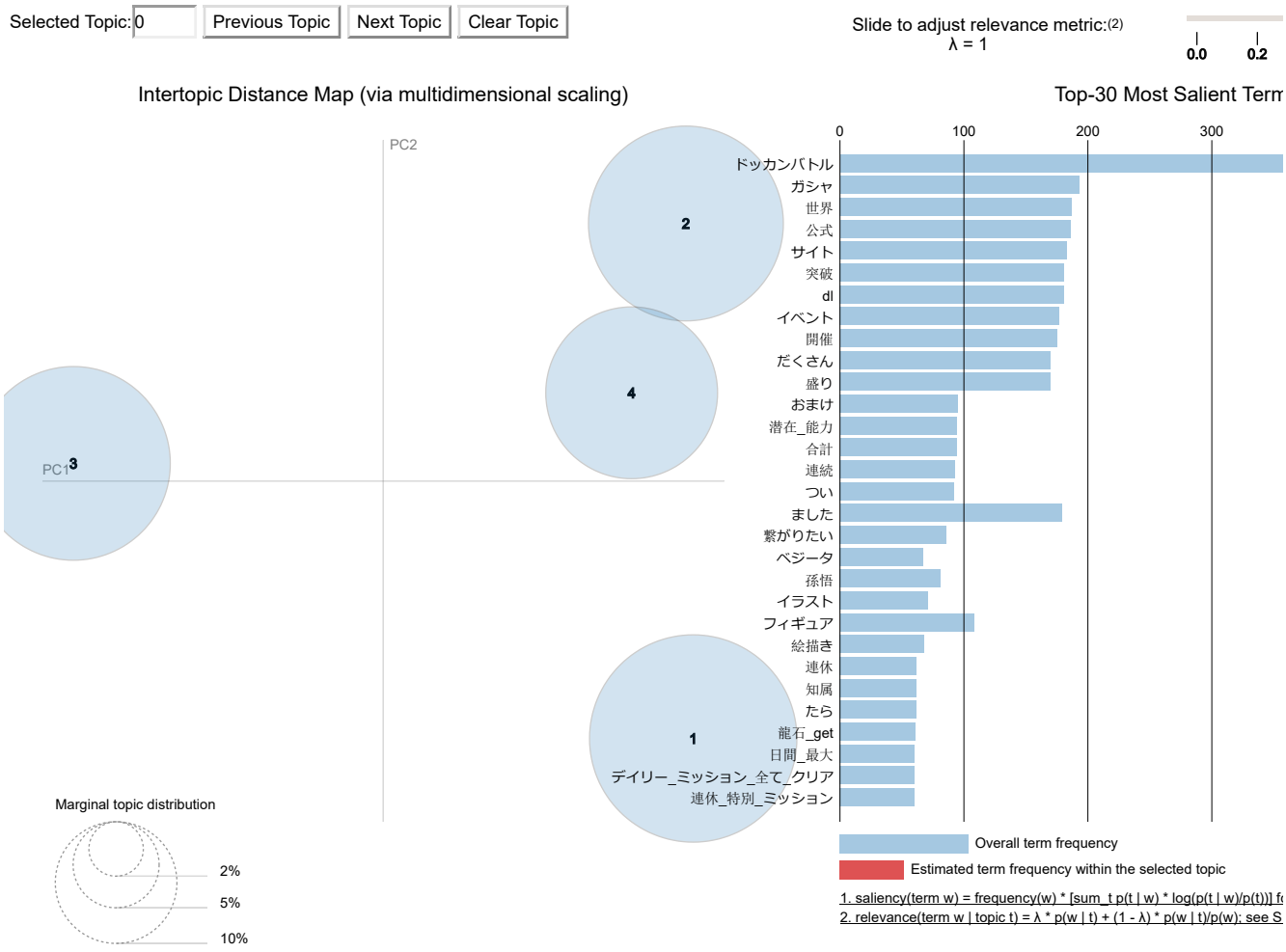
- Tweets identified by their dominant topic have been grouped well.
- There might be another topic identified by the isolated clustering in the right hand side.

16. pyLDAVis

In [33]:
```
import pyLDAvis.gensim
pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim.prepare(lda_model_ord, corpus, dictionary=lda_model_ord.id2word)
vis
```

```
C:\Users\CD250050\AppData\Local\Continuum\anaconda3\lib\site-packages\pyLDAvis\_prepare.py:257: FutureWarning: Sorting because nor
e version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  return pd.concat([default_term_info] + list(topic_dfs))
```

Out[33]:



Notes

- Each bubble on the left-hand side plot represents a topic. The larger the bubble, the more prevalent is that topic.
- A good topic model has fairly big, non-overlapping bubbles scattered throughout the chart instead of being clustered in one quadrant.
- A model with too many topics, will typically have many overlaps, small sized bubbles clustered in one region of the chart. (Possibly suggests another topic)