MA5851 A3 Document Number 3 Hugh McMullan

Task 3-1 - TOPIC MODELLING NLP TASK

a. Brief Literature Review

In 2002 **Blei and others** describe LDA as a three-level Bayesian model, with each item modelled as a mixture of terms from a set of topics, as probabilities that those terms exist in the topics. They highlight the applicability of the method for domains including document classification. They identified LDA as a 'simple model' useful in dimensionality reduction, also able to be scaled-up and used in conjunction with other more complex models.

Snyder in 2015 discussed the [then] state of LDA modelling, noting the rapid rise in interest and applications of computational linguistics, highlighting deployments within the "… big companies – Google, Baidu, Amazon, Facebook …" [pp87]. He referenced the bag-of-words and pre-processing approaches used for LDA and other popular methods, including the application of Python.

Heck and others discuss approaches in speech applications when labelling data is unavailable. They describe an alternative Dirichlet Process Gaussian Mixture Model ['DPGMM'], utilising '… LDA in an unsupervised fashion …' [pp75] to improve that approach; LDA was superior to PCA in supporting that DPGMM model. They found that LDA estimation is '… particularly suitable …'[pp79] supporting DPGMM clustering reliably using automatically-generated labels.

Palacio-Nino and Berzal discuss techniques for evaluating the quality of clustering techniques in unsupervised machine learning. Amongst others, they identify cohesion and separation measures as internal validation methods that can be employed.

Support for the LDA techique comes from many sources. For instance **Tong and Zhang** in 2016 detail an application of LDA for analysing labelled tweets, describing the generated results as '… convincing …'[pp209]]; also **Jacobi** and others propose LDA as a '… cutting-edge technique …' for analysing new content.

Lastly, supporting the adoption of the technique by a wide-ranging community, many instructive LDA tutorials are generous;y provided online: for instance **Albalawi** and others, **Kana** and **Prabhakaran**.

b. Rationale for selection of the NLP task

The first objective of the assessment was to determine the topics of most interest (as they pertained to their interactions with CASA) to the commercial and private drone users, respectively.

A topic modelling exercise on the forums visisted and posted to by there drone users is an effective to achieve that.

Specifically, the *Latent Dirichlet Allocation* [*'LDA'*] model has been used, because of the support in the literature, and also because of the well-documented and multi-exampled materials available to support the use of that model.

Very specifically, the model applied is an unsupervised learning approach. Because posts were not labelled in any useful way for testing the model, no separation between traing and testing datasets has been applied here.

Other more complex approaches to validate the model's application might have been pursued [for example, re-gathering user identity codes, then aligning generated topic cluster with individual user ids, looking for correlation there - and then looking to see if a training/testing dataset comparison *persisted* those user-vs-topic alignment between the traing and testing datasets. This was considered to be a step too far for the prototype model development.

c. Data pre-processing of inputs, separate from WebCrawler harvesting

From the texts extracted, for the topic modelling task, the processes adopted [and detailed alongside the code in section f below] are:

- remove unwanted characters [eg the posts contained explicit and visible line break characters '\n'; any character not in the a-z or 0-9 range would be removed]
- remove superfluous white space
- remove typical stopwords [as defined in the library stopwords resource
- tokenize words, for the *LDA* model application requirements
- **stem** words - initially this was not done, but appeared as necessary after the words 'fly' and 'flying' were both prominent in the model's 'Top words' output.

The code for these actions is shown below.

d. Specification and justification of any hyperparameters

For the LDA model, as with clustering generally, the key assumption required is the *starting number of clusters*.

The approach adopted was to derive *coherence scores' across a range of choices [1-20] for that starting number of clusters, and then use the value best-indicated by a graph of those coherence scores against numbers of clusters.

The graphs generated [for both commercial and private cases] were not particularly helpful, but broadly suppoorted the chosen value used [5].

Further, later analysis using the *pyLDAvis* portal comparing output clusters, generally supported that number of 5 [for both commercial and private]. Whereas in an earlier pass trying an assumend starting number of 9 clusters, several of the generated topics overlaid each other, discouraging the view that those were good/separate clusters identified.

While other parameter [eg the *perplexity* measure] could also have been cross-experimentally tried, for the prototype model development here that would have been difficult to accomplish in a reasonable timeframe.

e. Preliminary assessment of NLP task performance

As discussed in c. above, it was felt to be impracticable to conduct training/testing dataset comparisons, to derive precision/recall/F1 metrics.

Coherence and perplexity scores are generated by the model, but there is little to use to assess the goodness or otherwise for this model's performance.

f. Code

```
dfPreppedText = dfDronePostsSince2018
```

NLP pre-processing libraries were loaded

```
# Import libraries needed for this notebook
import re

import nltk
from nltk.tokenize import RegexpTokenizer, word_tokenize
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

from sklearn.feature_extraction.text import CountVectorizer

from wordcloud import WordCloud
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Constants were established for code readability

```
# NLTK constants
PATTERN_S = re.compile("\'s")           # matches `'s` from text
PATTERN_R = re.compile("\r")            #matches `\r`
PATTERN_N = re.compile("\n")            #matches `\r` and `\n`
PATTERN_PUNC = re.compile(r"[^\w\s]")   # matches all non 0-9 A-z whitespace
STOPWORDS = set(stopwords.words('english'))
```

Functions were set up to:

(1) start with the *'target_text'* text field produced by the crawler/scraping process;

(2) remove unwanted characters;

(3) convert text to lowercase;

(4) remove stopwords; and

(5) store the resultant text in a new column *'cleaned_text'*

```
# function for text cleaning
def clean_text_using_nltk(text):
    text = text.lower()                     # Convert text to lowercase

    text = re.sub(PATTERN_S, ' ', text)     # Remove apostrophe-s
    text = re.sub(PATTERN_R, ' ', text)     # Remove escaped \r characters
    text = re.sub(PATTERN_N, ' ', text)     # Remove escaped \n characters
    text = re.sub(PATTERN_PUNC, ' ', text)  # Remove non 0-9 A-z whitespace

    return text

# Apply the text cleaning function to the target_text
dfPreppedText['cleaned_text'] = dfPreppedText['target_text'].apply(lambda s: clean_text_us


# function to remove stopwords
def remove_stopwords(text):
    no_stopword_text = [w for w in text.split() if not w in STOPWORDS]
    return ' '.join(no_stopword_text)

# Apply the stopwords removal function to the corpus text
dfPreppedText['cleaned_text'] = dfPreppedText['cleaned_text'].apply(lambda s: remove_stopw
```

A function was established to tokenise the cleaned_text and store the result in a new column *tokenized_text*

```
# function to tokenize text
def apply_tokenizer(text):
    tokenizer = RegexpTokenizer('[a-zA-Z]\w+\'?\w*')    # use NLTK Regexp tokenizer
    return tokenizer.tokenize(text)

# Apply the stopwords removal function to the corpus text
dfPreppedText['tokenized_text'] = dfPreppedText['cleaned_text'].apply(lambda s: apply_toke
```

Now also STEMMING the text as well

When Topic Modelling [see Document 3] was carried, the key words included **both** 'fly' and 'flying'.

That was unsatisfactory, so STEMMING was added, saving the new field as *stemmed_text*:

```
# function to stem text
def apply_stemmer(text):
  stemmer=PorterStemmer()
  words = [stemmer.stem(word) for word in text]
  return words

# Apply the stemming function to the tokenized corpus text
dfPreppedText['stemmed_text'] = dfPreppedText['tokenized_text'].apply(lambda s: apply_stem
```

Finally the pre-processed data was stored into .csv as well

```
dfPreppedText.to_csv('/content/drive/MyDrive/DronePostsPreppedText_202112051330.csv')
```

Last, separate dataframes were selected [and saved] for **Commercial** and **Private** [drone users]:

```
# Separate COMMERCIAL corpus from dfPreppedText
dfCommercial = dfPreppedText[dfPreppedText["dronesite_group"]=="Commercial"]
dfCommercial
```

	Unnamed: 0	dronesite_group	thread_name	post_YYMM	post_YYY
5052	5953	Commercial	What is the process of ReOC	2106	202
5053	5954	Commercial	What is the process of ReOC	2106	202

```
dfCommercial.to_csv('/content/drive/MyDrive/DronePostsCommercial_202112051330.csv')
```

ReOC

```
# Separate PRIVATE corpus from dfPreppedText
dfPrivate = dfPreppedText[dfPreppedText["dronesite_group"]=="Private"]
dfPrivate
```

```
dfPrivate.to_csv('/content/drive/MyDrive/DronePostsPrivate_202112051330.csv')
```

Unnamed:

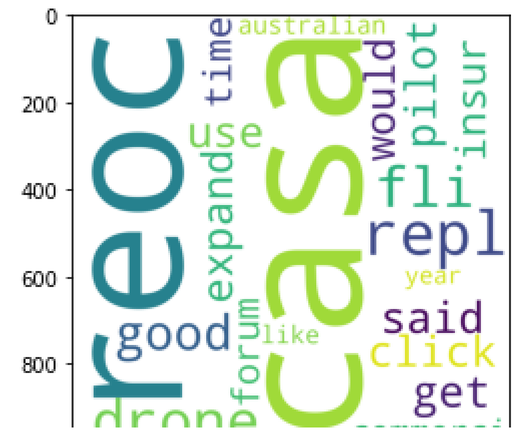
Exploratory Data Analysis on data ready for NLP

Just as simple pre-test before commencing the NLP, WordClouds were generated for the Commercial and Private datasets, showing the 'Top 20' words in each

VLOS

```
# Top 20 Words in Commercial posts
comm_top20_tuples = most_frequent_words([word for post in dfCommercial.stemmed_text for wo
comm_top20_dictionary = convert_tuples_to_dictionary(comm_top20_tuples)
print(comm_top20_dictionary)

# WordCloud for Top 20 Words in Commercial posts
comm_top20_wordcloud = WordCloud(background_color="white", width=1000, height=1000, max_wo
plt.imshow(comm_top20_wordcloud)
```

```
{'casa': 2818, 'reoc': 2465, 'repl': 2256, 'drone': 1921, 'fli':
<matplotlib.image.AxesImage at 0x7f1b0e072e90>
```



```
# Top 20 Words in Commercial posts
priv_top20_tuples = most_frequent_words([word for post in dfPrivate.stemmed_text for word
priv_top20_dictionary = convert_tuples_to_dictionary(priv_top20_tuples)
print(priv_top20_dictionary)

# WordCloud for Top 20 Words in Commercial posts
priv_top20_wordcloud = WordCloud(background_color="white", width=1000, height=1000, max_wo
plt.imshow(priv_top20_wordcloud)
```

```
{'fli': 6922, 'drone': 6589, 'said': 2831, 'casa': 2523, 'click':
<matplotlib.image.AxesImage at 0x7f1b04f06510>
```

These simple WordClouds are not significant - but still interesting.

Commercial users require **Remote Operator Certificate** (known as the **'ReOc'**). This ReOc is extremely difficult and expensive to obtain, onerous to report on and comply with, it allows businesses to provide commercial drone services - and it is the bane of commercial operators' existences. So to see that in the WordCloud for these operators is encouraging.

```
from gensim.corpora import Dictionary
from gensim.models.ldamodel import LdaModel
from gensim.models import CoherenceModel
```

TOPIC MODELLING FOR COMMERCIAL POSTS

```
# Build a dictionary where for each COMMERCIAL post, each word has its own id
comm_dictionary = Dictionary(dfCommercial.stemmed_text)

# Print the numbers of words and posts
comm_posts = len(dfCommercial)
comm_words = len(comm_dictionary)
print("COMMERCIAL:", comm_posts, "posts; ", comm_words, "words")

# build the corpus i.e. vectors with the number of occurence of each word per post
comm_corpus = [comm_dictionary.doc2bow(post) for post in dfCommercial.stemmed_text]
```

```
COMMERCIAL: 2615 posts;  3464 words
```

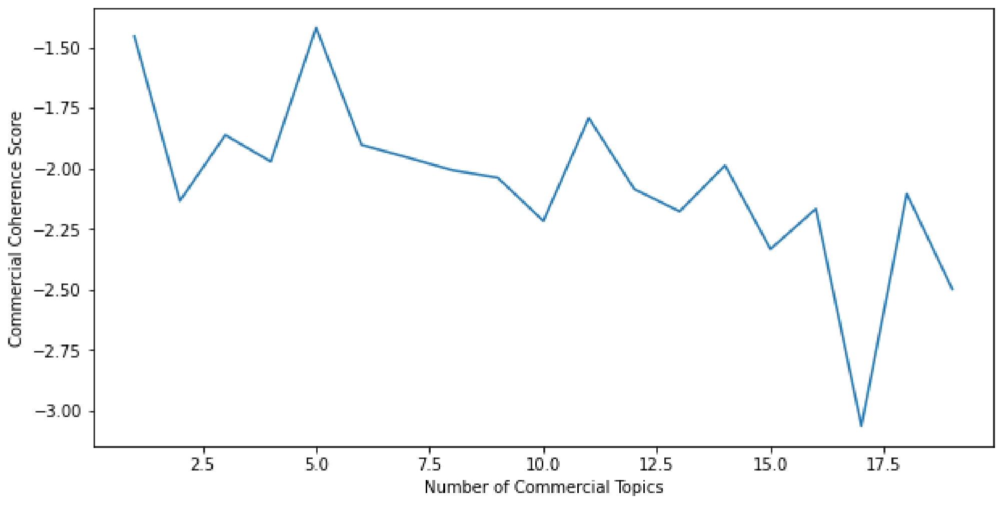
LDA Model development approach as described by *Michel Kana*

The code here is adapted from a model presented by Michel Kana in a *'towards data science'* 2020 tutorial [Michel Kana].

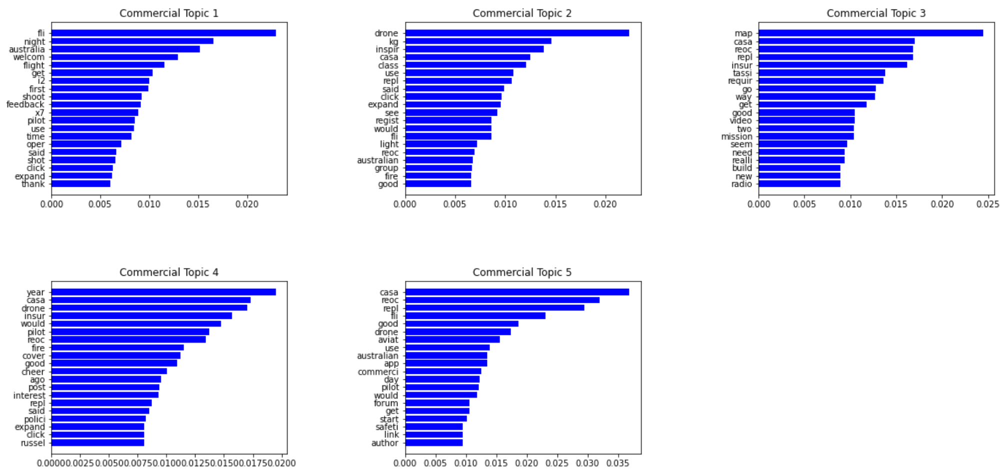
According to Kana, '*... Topic Coherence is usually preferred to Perplexity techniques*', and the '*... the number of topics for which the average score plateaus, is the sweet spot we are looking for ...*' [for the selection of an assumed number of topics to base the model training on.

```
# Compute coherence scores for a big number of tested topic numbers
comm_coherence = []
for nb_topics in range(1,20):
    comm_lda = LdaModel(comm_corpus, num_topics = nb_topics, id2word = comm_dictionary, pa
    comm_cohm = CoherenceModel(model=comm_lda, corpus=comm_corpus, dictionary=comm_diction
    comm_coh = comm_cohm.get_coherence()
    comm_coherence.append(comm_coh)
```

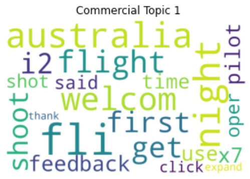
```
# visualize coherence
plt.figure(figsize=(10,5))
plt.plot(range(1,20),comm_coherence)
plt.xlabel("Number of Commercial Topics Tested")
plt.ylabel("Commercial Coherence Score");
```

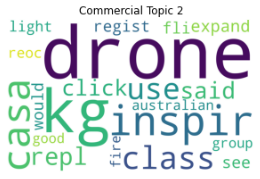


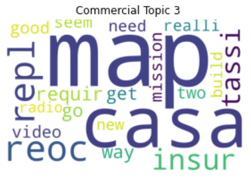
```
# Generate Top 5 topics summaries for COMMERCIAL posts
posts_group = "Commercial"
comm_no_of_topics = 5
top_n_words = 20
comm_lda = LdaModel(comm_corpus, num_topics = comm_no_of_topics, id2word = comm_dictionary

# Generate lists of words and frequencies
comm_top_words = [[word for word,_ in comm_lda.show_topic(topic_id, topn=50)] for topic_id
comm_top_betas = [[beta for _,beta in comm_lda.show_topic(topic_id, topn=50)] for topic_id


def plot_top_words(top_words, top_betas, nb_topics, nb_words, posts_group):
    gs  = gridspec.GridSpec(round(math.sqrt(nb_topics))+1,round(math.sqrt(nb_topics))+1)
    gs.update(wspace=0.5, hspace=0.5)
    plt.figure(figsize=(20, 15))
    for i in range(nb_topics):
      topic = i + 1
      ax = plt.subplot(gs[i])
      plt.barh(range(nb_words), top_betas[i][:nb_words], align='center',color='blue', ecol
      ax.invert_yaxis()
      ax.set_yticks(range(nb_words))
      ax.set_yticklabels(top_words[i][:nb_words])
      plt.title(posts_group + " Topic " + str(topic))


plot_top_words(comm_top_words, comm_top_betas, comm_no_of_topics, top_n_words, posts_group
```

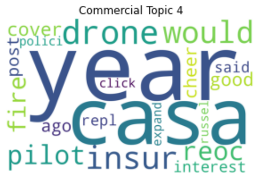


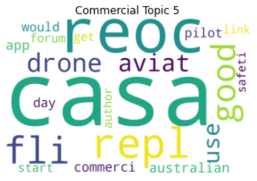
```
def wordcloud_top_words(top_words, top_betas, nb_topics, nb_words, posts_group):
    gs  = gridspec.GridSpec(round(math.sqrt(nb_topics))+1,round(math.sqrt(nb_topics))+1)
    gs.update(wspace=0.5, hspace=0.5)
    plt.figure(figsize=(20,15))
    for i in range(nb_topics):
      words = top_words[i]
      betas = top_betas[i]
      zip_iterator = zip(words, betas)
      top_words_dict = dict(zip_iterator)
      wc = WordCloud(background_color="white", width=1600, height=1000, max_words=nb_words
                     relative_scaling=0.5, normalize_plurals=False).generate_from_frequenc

      topic = i + 1
      ax = plt.subplot(gs[i])
      ax.set_title(posts_group + " Topic " + str(topic))
      ax.axis('off')
      plt.imshow(wc)


wordcloud_top_words(comm_top_words, comm_top_betas, comm_no_of_topics, top_n_words, posts_
```


Commercial Topic 1


Commercial Topic 2


Commercial Topic 3


Commercial Topic 4

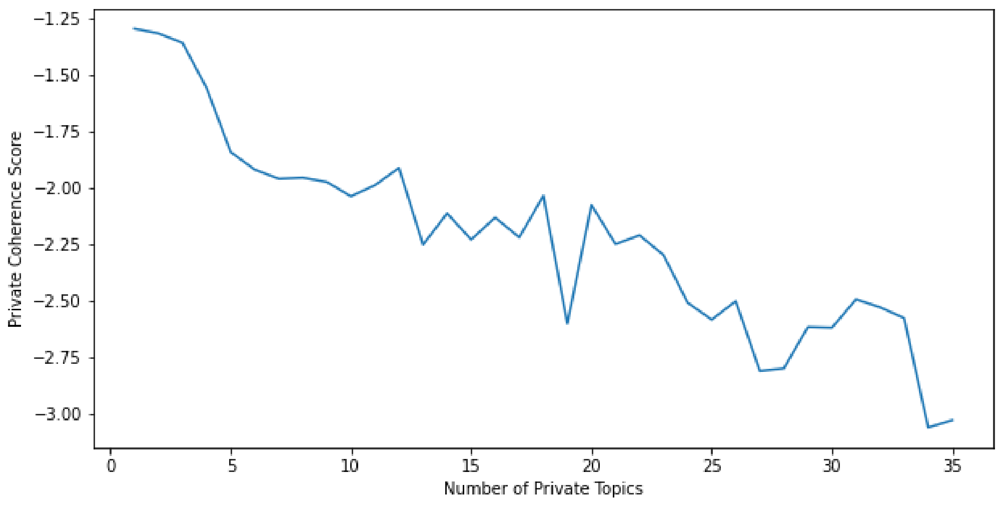

Commercial Topic 5

TOPIC MODELLING FOR PRIVATE POSTS

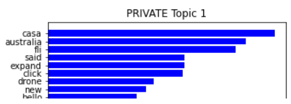
```
# Build a dictionary where for each PRIVATE post, each word has its own id
priv_dictionary = Dictionary(dfPrivate.stemmed_text)

# Print the numbers of words and posts
priv_posts = len(dfPrivate)
priv_words = len(priv_dictionary)
print("PRIVATE:", priv_posts, "posts; ", priv_words, "words")

# build the corpus i.e. vectors with the number of occurence of each word per post
priv_corpus = [priv_dictionary.doc2bow(post) for post in dfPrivate.stemmed_text]
```

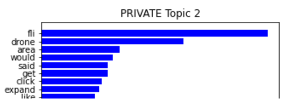
```
     PRIVATE: 5052 posts;  6814 words
```

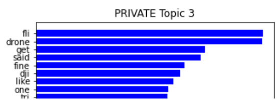
```
# Compute coherence scores for a big number of tested topic numbers
priv_coherence = []
for nb_topics in range(1,20):
    priv_lda = LdaModel(priv_corpus, num_topics = nb_topics, id2word = priv_dictionary, pa
    priv_cohm = CoherenceModel(model=priv_lda, corpus=priv_corpus, dictionary=priv_diction
    priv_coh = priv_cohm.get_coherence()
    priv_coherence.append(priv_coh)

# visualize coherence
plt.figure(figsize=(10,5))
plt.plot(range(1,20),priv_coherence)
plt.xlabel("Number of Private Topics")
plt.ylabel("Private Coherence Score");
```



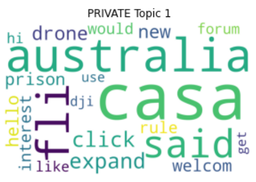
```
# Generate Top 5 topics summaries for PRIVATE posts
posts_group = "PRIVATE"
priv_no_of_topics = 5
top_n_words = 20
priv_lda = LdaModel(priv_corpus, num_topics = priv_no_of_topics, id2word = priv_dictionary

# Generate lists of words and frequencies
priv_top_words = [[word for word,_ in priv_lda.show_topic(topic_id, topn=50)] for topic_id
priv_top_betas = [[beta for _,beta in priv_lda.show_topic(topic_id, topn=50)] for topic_id


plot_top_words(priv_top_words, priv_top_betas, priv_no_of_topics, top_n_words, posts_group
```

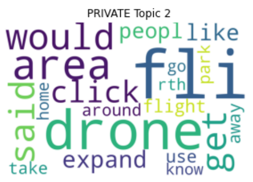


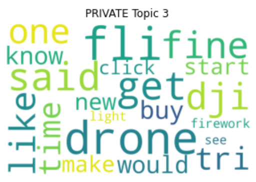


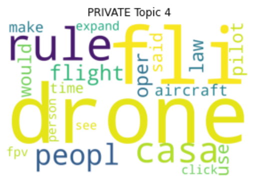


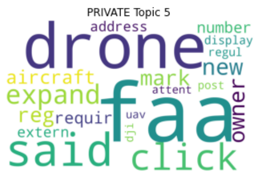
```
wordcloud_top_words(priv_top_words, priv_top_betas, priv_no_of_topics, top_n_words, posts_
```







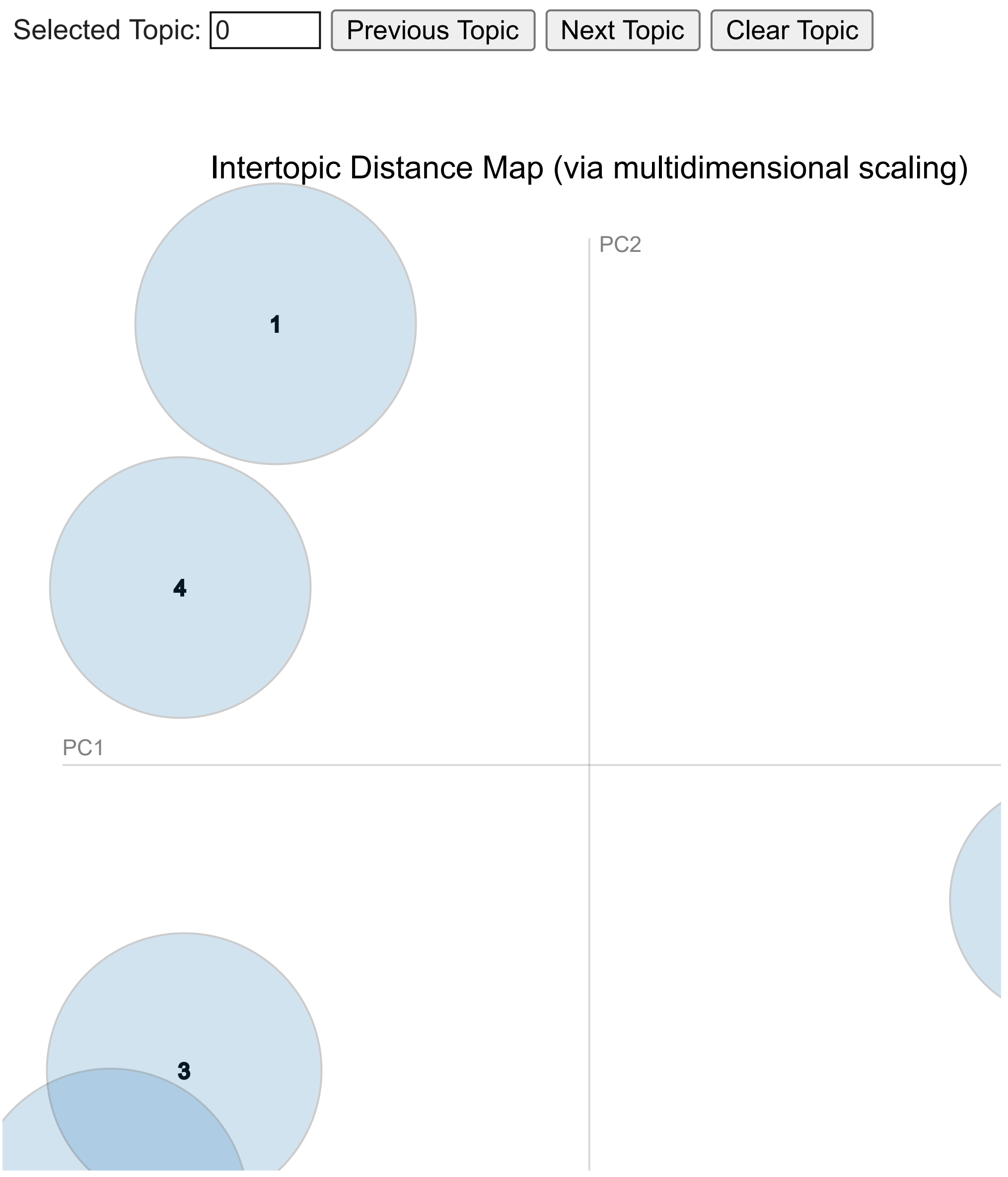




Implement pyLDAvis workbooks

```
# feed the LDA model into the pyLDAvis instance
comm_lda_viz = gensimvis.prepare(comm_lda, comm_corpus, comm_dictionary)
pyLDAvis.display(comm_lda_viz)
```

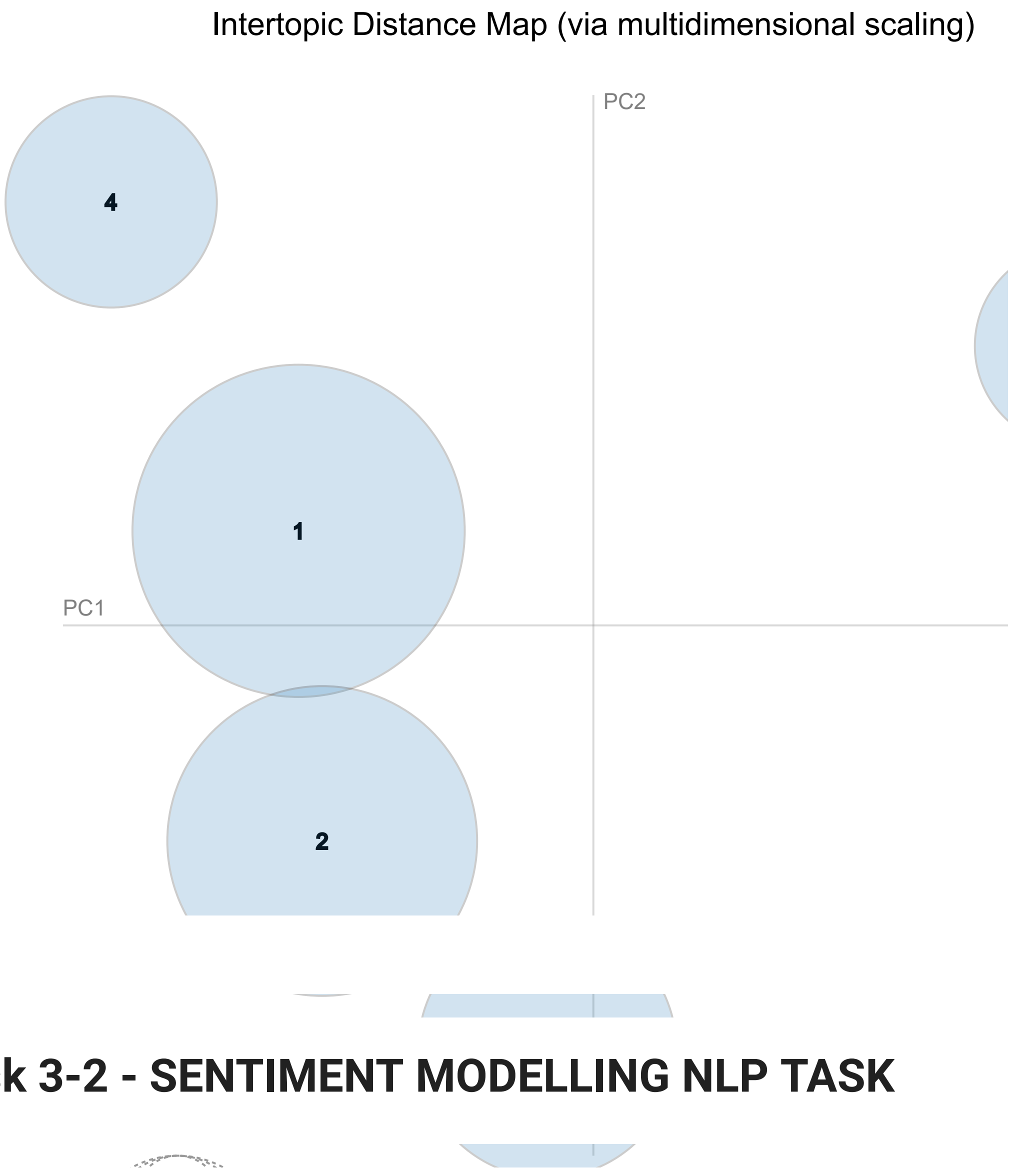
```
/usr/local/lib/python3.7/dist-packages/pyLDAvis/_prepare.py:247: I
  by='saliency', ascending=False).head(R).drop('saliency', 1)
```



```
# feed the PRIVATE LDA model into the pyLDAvis instance
priv_lda_viz = gensimvis.prepare(priv_lda, priv_corpus, priv_dictionary)
pyLDAvis.display(priv_lda_viz)
```

```
/usr/local/lib/python3.7/dist-packages/pyLDAvis/_prepare.py:247: I
  by='saliency', ascending=False).head(R).drop('saliency', 1)
```

Selected Topic: 0 Previous Topic Next Topic Clear Topic



Task 3-2 - SENTIMENT MODELLING NLP TASK

a. Brief Literature Review

Hutto and Gilbert introduce their *VADER* [‘Valence Aware Dictionary for sEntiment Reasoning’]rule-based model for general sentiment analysis, on blog-like applications (particularly *twitter*). They find that their model compares fvourably to a wide range of alternative ML approaches. Alongside their model they devop and use a *’gold-standard lexicon’* ‘… especially attuned to micro-blog-type contexts’ [pp216]. Amongst other things, their paper sets illustrates a strong framework ‘… methods and process approach overview’ for such applications. [pp219]

Sarlis and Maglogiannis evaluate a wide range of ML algorithms for labelled sentiment analysis applications, using the IMDb dataset and twitter datasets for test cases. On the IMDb data, neural network-based approaches provide the best results; on the twitter data, moderate results only are achieved.

Elbagir and Yang extend previous applications of *VADER* alongside *NLTK* in a multi-classification system for analysing tweets. They referred to previous studies (e.g. Wagh, 2018 where training sentiment analysis models with data from one domain had not generated good results when applied to other domains. Their multi-classification model generated effective results.

As with the *LDA* community support mentioned above, there are many community-contributed learning resources for the *VADER*-based sentiment analysis modelling e.g.Parul Pandey, *Analytics Vidhya*.

b. Rationale for selection of the NLP task

The second objective of the assessment exercise here was to form a view on how well CASA relates to the commercial and private drone users subject to its regulation.

To that extent, sentiment analysis is a useful tool, especially when viewed over time.

The *VADER* library and associated resources was chosed [like the LDA choice for the topic modelling] for its wide community acceptance and respect, and also for the extensive community support with papers and tutorials to assist an inexperienced and inexpert user [me].

c. Data pre-processing of inputs, separate from WebCrawler harvesting

The *VADER* library, and the *sentiment analysis* task itself, required different pre-processing from that used for the topic modelling task.

Specifically:

(1) the NLP task needed to be applied to **sentences**, not just word tokens;

(2) and punctuation could impact sentiment, so leave that puctuation in place;

(3) the *VADER* library removes stopwords itself, so that did not need to be done explicitly here;

(4) capitals matter for sentiment analyses, so lowercasing text is not appropriate;

(5) so in the end, the only required action was to remove the "\n" characters.

A function was set up to re-apply just that to the original 'target_text' extracted by the crawler/scraper process, and store the result in a separate 'vader_ready_text' column. This was applied directly to the *dfCommercial* and *dfPrivate* dataframes.

```
# function for text cleaning
def vader_clean_text_using_nltk(text):
    text = re.sub(PATTERN_N, ' ', text)     # Remove escaped \n characters
    return text
```

```
# Apply the text cleaning function to the target_text in dfCommercial and dfPrivate
dfCommercial['vader_ready_text'] = dfCommercial['target_text'].apply(lambda s: vader_clean
dfPrivate['vader_ready_text'] = dfPrivate['target_text'].apply(lambda s: vader_clean_text_
```

d. Specification and justification of any hyperparameters

No hyperparameters are required for the VADER sentimet analysis modelling.

e. Preliminary assessment of NLP task performance

As described in the literature mentioned above, evaluating performance is not straightforward for the unsupervised clustering task used here.

As with the LDA task above, one could attempt a complex exercise, cross-relating other domains (e.g.users identified against posts, matched in a training set against derived sentiment analyses) - then testing the sentiment analysis model by comparing new sentiment scores for the same users - but in a different time period (say), as a testing dataset. This would be extremely complex to effect, beyond the scope of a prototype system development, and might not establish validity in any event.

f. Code

```
import pandas as pd
import numpy as np
import re

from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

import nltk
from nltk.tokenize import sent_tokenize
nltk.download('punkt')

# Install the VADER sentiment analysis library
!pip install vaderSentiment
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')


# The data files are located on the Google Drive, so do a drive.mount
from google.colab import drive
drive.mount('/content/drive')


# Read in the COMMERCIAL and PRIVATE posts data, created in the Part 2 notebook
```

```
dfCommercial = pd.read_csv('/content/drive/MyDrive/DronePostsCommercial_202112051330.csv')
dfPrivate = pd.read_csv('/content/drive/MyDrive/DronePostsPrivate_202112051330.csv')
```

Different pre-processing for VADER library requirements

(1) using sentences, not tokens;

(2) so leave punctuation in place;

(3) VADER removes stopwords itself, so do not need to do that explicitly;

(4) don't lowercase text, because capitals matter for sentiment analyses;

(5) just remove "\n" characters.

```
# NLTK constants
PATTERN_N = re.compile("\n")            #matches `\r` and `\n`


# function for text cleaning
def vader_clean_text_using_nltk(text):
    text = re.sub(PATTERN_N, ' ', text)     # Remove escaped \n characters
    return text

# Apply the text cleaning function to the target_text in dfCommercial and dfPrivate
dfCommercial['vader_ready_text'] = dfCommercial['target_text'].apply(lambda s: vader_clean
dfPrivate['vader_ready_text'] = dfPrivate['target_text'].apply(lambda s: vader_clean_text_
```

Apply VADER functionality to add Sentiment score to eeach row in dfCommercial and dfPrivate

The process deploys the *VADER* library *SentimentIntensityAnalyzer* object, containing the methods to generate polarity scores on blocks of words [as sentences in this case].

```
# Instantiate the VADER SentimentIntensityAnalyzer
vsia = SentimentIntensityAnalyzer()
```

Tests of the SentimentIntensityAnalyzer are applied here to ensure my understanding of the library function's treatment of text is correct.

```
# function to score a sentence
def sentiment_analyzer_sentence_score(sentence):
    score = vsia.polarity_scores(sentence)
    print("{:-<40} {}".format(sentence, str(score)))

text = "THe Swans are the best team ever"
sentiment_analyzer_sentence_score(text)

text = "The Swans are the best team ever!"
sentiment_analyzer_sentence_score(text)
```

```
text = "The Swans are the BEST team ever!"
sentiment_analyzer_sentence_score(text)

text = "The Swans, they are the BEST team ever!"
sentiment_analyzer_sentence_score(text)

text = "The Swans, they are absolutely the BEST team ever!"
sentiment_analyzer_sentence_score(text)

text = "The Swans, they are hardly the BEST team ever!"
sentiment_analyzer_sentence_score(text)

text = "wtf, The Swans, they are hardly the BEST team ever!"
sentiment_analyzer_sentence_score(text)

text = "wtf, those Swans. They are hardly the BEST team ever!"
sentiment_analyzer_sentence_score(text)
```

```
THe Swans are the best team ever-------- {'neg': 0.0, 'neu': 0.588, 'pos': 0.412, 'co
The Swans are the best team ever!------- {'neg': 0.0, 'neu': 0.572, 'pos': 0.428, 'co
The Swans are the BEST team ever!------- {'neg': 0.0, 'neu': 0.535, 'pos': 0.465, 'co
The Swans, they are the BEST team ever!- {'neg': 0.0, 'neu': 0.573, 'pos': 0.427, 'co
The Swans, they are absolutely the BEST team ever! {'neg': 0.0, 'neu': 0.592, 'pos':
The Swans, they are hardly the BEST team ever! {'neg': 0.0, 'neu': 0.618, 'pos': 0.38
wtf, The Swans, they are hardly the BEST team ever! {'neg': 0.227, 'neu': 0.478, 'pos
wtf, those Swans. They are hardly the BEST team ever! {'neg': 0.227, 'neu': 0.478, 'p
```

Each post can contain multiple sentences.

The approach chosen here is to:

(1) extract each separate sentence from each post;

(2) obtain a polarity score for each separate sentence;

(3) for each post, take the **average sentence polarity score across senetences within each post**.

A function to split posts into sentences:

```
def split_text_into_sentences(texts):
  sentences = nltk.sent_tokenize(texts)
  return sentences

# text = "Wtf, those Swans. They are definitely the BEST team ever! Just ask them."
# sentences = split_text_into_sentences(text)
# sentences
```

A function to generate the averaged-over-sentences-within-posts score for each post:

```
# function for scoring multiple sentences in a single post
def sentiment_analyzer_multiple_sentences_in_post_score(post):
  sentences = split_text_into_sentences(post)
  scores = []
  for sentence in sentences:
    score = vsia.polarity_scores(sentence)
    scores.append(score['compound'])
    # print(sentence, str(score))
  average_score = sum(scores) / (len(scores) + 0.0001)
  return average_score

# text = "Those Swans. They are definitely the BEST team ever! Just ask them."
# sentiment_analyzer_multiple_sentences_in_post_score(text)
```

Apply the scoring function to each row in the dataframes, generating a new column '*sentiment_score*':

```
dfCommercial['sentiment_score'] = dfCommercial['vader_ready_text'].apply(lambda s: sentime
dfPrivate['sentiment_score'] = dfPrivate['vader_ready_text'].apply(lambda s: sentiment_ana
# dfCommercial
```

And add an additional column '*sentiment_polarity*', showing as 'Positive' or 'Negative' or 'Neutral', based on the sentiment_score value:

```
# function for assigning sentiment polarity based on compound score
def sentiment_analyzer_polarity_from_score(score):
  polarity = "Neutral"
  if score > 0:
    polarity = "Positive"
  if score < 0:
    polarity = "Negative"
  return polarity

# text = "Those Swans. They are not the BEST team ever! Just ask them."
# score = sentiment_analyzer_multiple_sentences_in_post_score(text)
# polarity = sentiment_analyzer_polarity_from_score(score)
# print(score, polarity)
```

```
dfCommercial['sentiment_polarity'] = dfCommercial['sentiment_score'].apply(lambda f: senti
dfPrivate['sentiment_polarity'] = dfPrivate['sentiment_score'].apply(lambda f: sentiment_a
dfCommercial
```

	Unnamed: 0	Unnamed: 0.1	dronesite_group	thread_name	post_YYMM
0	5052	5953	Commercial	What is the process of ReOC	2106
1	5053	5954	Commercial	What is the process of ReOC	2106
2	5054	5955	Commercial	What is the process of ReOC	2106
3	5055	5956	Commercial	What is the process of ReOC	2106

Review the generated sentiment by month

Count numbers of 'Positive' or 'Negative' or 'Neutral' by month, for each of the commercial and private drone users datasets:

DATA ISSUE From the initial EDA after the dfCommercial dataframe had been established, it became clear that a large pile of all-Positive posts had been loaded into the commercial-oriented web sites [likely just a migration from another discontinued site].

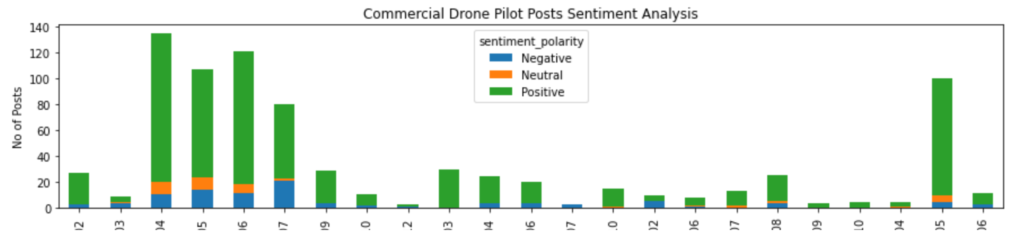
Even though dropping those reduces the numbers of posts for reporting, the patterns for later periods are still useful.

Accordingly, the '1801' month (for the commercial group) has been excluded **just for this reporting here**.

2612	7665	12457	Commercial	Mountains	1804

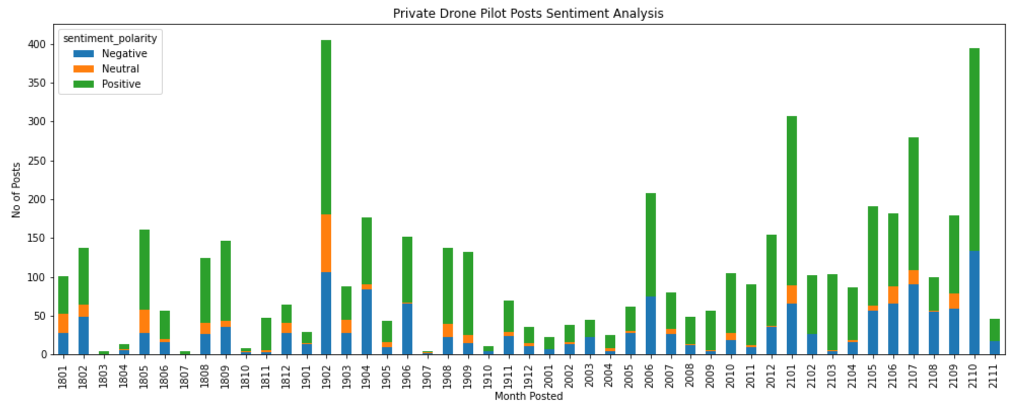
```
dfCommExcl1801 = dfCommercial[dfCommercial["post_YYMM"]>1801]
dfSIComm = dfCommExcl1801.groupby('post_YYMM')['sentiment_polarity'].value_counts().unstac
dfSIComm.plot.bar(stacked=True, title='Commercial Drone Pilot Posts Sentiment Analysis', y
         xlabel='Month Posted', figsize=(15, 3))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f71737167d0>
```



```
dfSIPriv = dfPrivate.groupby('post_YYMM')['sentiment_polarity'].value_counts().unstack(lev
dfSIPriv.plot.bar(stacked=True, title='Private Drone Pilot Posts Sentiment Analysis', ylab
          xlabel='Month Posted', figsize=(16.8, 6))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7173b1ec90>
```



References

Albalawi, R. a. (2020). Using Topic Modeling Methods for Short-Text Data: A Comparative Analysis. Frontiers in Artificial Intelligence. Retrieved from https://www.frontiersin.org/articles/10.3389/frai.2020.00042/full

David Blei, A. N. (2002). Latent Dirichlet Allocation. Journal of Machine Learning Research. Retrieved from https://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf

González-Castaño, M. F.-G.-L.-M.-M. (2016). Unsupervised method for sentiment analysis in online texts. Expert Systems with Applications. doi:https://doi.org/10.1016/j.eswa.2016.03.031

Hutto, C. a. (2014). VADER: A Parsimonious Rule-based Model for. Eighth International AAAI Conference on Weblogs and Social Media. Retrieved from http://comp.social.gatech.edu/papers/icwsm14.vader.hutto.pdf

Kana, M. (2020). Topic Modeling Tutorial with Latent Dirichlet Allocation (LDA). Retrieved from towards data science: https://towardsdatascience.com/topic-modeling-with-latent-dirichlet-allocation-by-example-3b22cd10c835

Michael Heck, S. S. (2016). Unsupervised Linear Discriminant Analysis for Supporting DPGMM Clustering in the Zero Resource Scenario. Procedia Computer Science. Retrieved from https://www.sciencedirect.com/science/article/pii/S1877050916300461

Palacio-Nino, J.-O. a. (2019). Evaluation Metrics for Unsupervised Learning. Retrieved from https://arxiv.org/pdf/1905.05667.pdf

Pandey, P. (2018). Simplifying Sentiment Analysis using VADER in Python (on Social Media Text). Retrieved from Analytics Vidhya: https://medium.com/analytics-vidhya/simplifying-social-media-sentiment-analysis-using-vader-in-python-f9e6ec6fc52f

Prabhakaran, S. (2018). Topic Modeling with Gensim (Python). machine learning +. Retrieved from https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/#12buildingthetopicmodel

Rana, R. M. (2019). Topic subject creation using unsupervised learning for topic modeling. Retrieved from https://arxiv.org/abs/1912.08868

Sarlis, S. a. (2020). On the Reusability of Sentiment Analysis Datasets in Applications with Dissimilar Contexts. IFIP International Conference on Artificial Intelligence Applications and Innovations. Retrieved from https://link.springer.com/chapter/10.1007/978-3-030-49161-1_34

Snyder, R. (2015). An Introduction to Topic Modeling as an Unsupervised Machine. 2015 ASCUE Proceedings. Retrieved from https://files.eric.ed.gov/fulltext/ED571275.pdf

Topic Modeling with Gensim (Python). (2018). Retrieved from machine learning +: https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/#12buildingthetopicmodel

Wagh, B. a. (2018). A Twitter Sentiment Analysis Using NLTK and Machine Learning Techniques. Int. J. Emerg. Res Manag. Technol. Retrieved from https://www.researchgate.net/publication/326054514_A_Twitter_Sentiment_Analysis_Using_NLTK_and_Machine_Learning_Techniques

Welbers, C. J. (2016). Quantitative analysis of large amounts of journalistic texts using topic modelling. Digital Journalism. Retrieved from https://www.tandfonline.com/doi/abs/10.1080/21670811.2015.1093271

Yang, S. E. (2019). Twitter Sentiment Analysis Using Natural. International MultiConference of Engineers and Computer Scientists. Hong Kong. Retrieved from http://www.iaeng.org/publication/IMECS2019/IMECS2019_pp12-16.pdf

Zhang, Z. T. (2016). A TEXT MINING RESEARCH BASED ON. Sixth International Conference on Computer Science Engineering and Information Technology. Vienna: Acadia Universit. Retrieved from https://www.researchgate.net/profile/Solomia-

Fedushko/publication/331276764_Proceedings_of_the_Sixth_International_Conference_on_Computer_Science_Engineering_and_Information_Technology_CCSEIT_2016_Vienna_Austria_May_2122_2016/links/5c6fcd63299bf1268d1bc2b