Detecting Fake News

Ling Wei Hsuen U2222805J
Pushparajan Roshini U2222546A
Lim Shao Jie U2223720A
SC1015 A124 Team 9

Table Of Contents

How to spot fake news

Introduction

Practical Motivation

Increase in anti-asian hate crimes during Covid-19 pandemic

- ❏ Consequences: Slashing of asians, discrimination
- ❏ Causes: Spread of fake and dis-information
- ❏ Current solution: Corroboration
- ❏ Problem with current solution: Echo-Chambers

Problem Formulation

How can we effectively differentiate between real and fake news using features of the article?

Dataset

https://www.kaggle.com/datasets/saurabhshahane/fake-news-classification

SAURABH SHAHANE · UPDATED 3 DAYS AGO

65 New Notebook Download (97 MB)

Fake News Classification

Fake News Classification on WELFake Dataset

Data Card Code (12) Discussion (1)

About Dataset

(WELFake) is a dataset of 72,134 news articles with 35,028 real and 37,106 fake news. For this, authors merged four popular news datasets (i.e. Kaggle, McIntire, Reuters, BuzzFeed Political) to prevent over-fitting of classifiers and to provide more text data for better ML training.

Dataset contains four columns: Serial number (starting from 0); Title (about the text news heading); Text (about the news content); and Label (0 = fake and 1 = real).

There are 78098 data entries in csv file out of which only 72134 entries are accessed as per the data frame.

Published in:

IEEE Transactions on Computational Social Systems: pp. 1-13 (doi: 10.1109/TCSS.2021.3068519).

Usability
10.00

License
Attribution 4.0 International (CC ...

Update frequency
Weekly

Data Cleaning

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Somethin	always has been. Itâ€™s been categorized by twenty-first century academia as paranormal but itâ€™s something Australiaâ€™s aboriginal people a													
Duncan Ro	they have developed an awareness of individual types of visitors from what we call outer space.â€53													
The Three	an Anglo-	a ribbon (	which lie	Japan anc	Arizona in the USA.									
Located in	an area fa	Flagstaff i	the priestly tribe who are the keepers of the DinÃ©â€™s most profound secrets.											
Like a pen	Freemont Peak and Doyle Peak in the Kachina Peaks Wilderness.													
To the Ho	part of th	the remai	is the most sacred place in the Four Corners. In fact it is the most sacred place in the worldâ€¦											
The San Fr	which the	to come forth when they are called in the powerful ceremonies performed by the Hopi.												
The Kachir	the rain and the lightingâ€¦													
At 11	464 feet D	the statec	especially in ultraviolet and infrared wavelengthsâ€¦											
In 2005 â€	the heart	including	Navajo M	and both	the U.S. Naval Observatory and the United States Geological Survey Flagstaff Stationâ€¦ Rock art from Seg									
Citations														
1 â€“ Reid	Nick	and Patrick D. Nunn. â€œAncient Aboriginal Stories Preserve History of a Rise in Sea Level.â€ The Conversation. 13 Jan. 2015. Web. 25 Jul												
2 â€“ Ibid.														
3 â€“ â€œ	7 July 2016. Web. 26 July 2016. https://www.youtube.com/watch?v=J-mX1eWoj6I													
4 â€“Ibid. 29:33.														
5 â€“Ibid. 30:09.														
6â€“ MAR	JOHN. â€	Einstein. (	21 Oct. 2015. Web. 3 Aug. 2016. http://www.nytimes.com/2015/10/22/science/quantum-theory-experiment-said-to-prove-sp											
7 â€“ â€œ	5 Dec. 2012. Web. 15 Aug. 2016. https://www.youtube.com/watch?v=awFleswtH2Y													
8 â€“Ibid.	1													

Junk values

Step 1

Importing dataset, removing NaN values

```python
dataset = pd.read_csv("20k.csv",encoding='ISO-8859-1') #read csv + encoding
dataset.head() #checking table
```

Encoding required to read csv file

```python
dataset.isnull().sum(axis=0) #checking if data cleaned
dataset.rename(columns={'Unnamed: 0':'id'}, inplace = True)
```

```python
dataset['id'].replace('  ', np.nan, inplace=True) #cleaning
```

Summing to check how many data we are dropping

```python
dataset.dropna(how='any', inplace=True)
#drop all NaN (double-cleaning)
#we are dropping 2000 ish values cos its less than 5% of our
```

Dropping

Step 2

Converting dtypes, re-labeling columns for better visualisation

```
dataset.id= pd.factorize(dataset.id)[0] #convert to int
dataset['title'] = dataset.title.astype(str) #convert to str
dataset['text'] = dataset.text.astype(str) #convert to str
dataset.label= pd.factorize(dataset.label)[0] #convert to int
```

Conversion of dtypes

```
dataset.rename(columns={'Unnamed: 0':'id'}, inplace = True) #
```

Renaming labels for clarity

```
label_translated = np.where(dataset['label']==0, 'fake', 'real') #new columns
dataset.insert(loc = 4 ,column = 'label_translated',value = label_translated)
```

Step 3

Removal of symbols

```
def remove_symbols(string):
    string = re.sub('[^a-zA-Z!?\']', ' ', string) #remove symbols
    return " ".join(string.split()) #eliminate white spaces

dataset['text'] = dataset['text'].apply(remove_symbols) #applying to text and title
dataset['title'] = dataset['title'].apply(remove_symbols)
dataset.head()
```

Step 4

Removal of stopwords

```python
def add_stopword_count_column(dataset, stop_words, text_column_name, stopword_count_column_name):
    index = 0
    for text in dataset[text_column_name]:
        textlist = text.split()
        stopword_number = 0
        for word in textlist:
            if word.lower() in stop_words:
                stopword_number += 1
        dataset.at[index, stopword_count_column_name] = stopword_number
        index += 1
    dataset[stopword_count_column_name] = dataset[stopword_count_column_name].fillna(0).astype(int)
    return dataset
```

```python
# removing stopwords from title
dataset['title'] = dataset['title'].apply(lambda x: ' '.join([word for word in str(x).split() if word not in (stop_words)]))

# title's stopword count
dataset = add_stopword_count_column(dataset, stop_words, 'title', 'title_stopword_count')
```

Step 5

Lemmatizing, tokenizing data and applying to title and text

```
lemmatizer = WordNetLemmatizer()

def get_wordnet_pos(treebank_tag):
    if treebank_tag.startswith('J'):
        return wordnet.ADJ
    elif treebank_tag.startswith('V'):
        return wordnet.VERB
    elif treebank_tag.startswith('N'):
        return wordnet.NOUN
    elif treebank_tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN
```

```
# Lemmatisation of text
dataset['text_lemmatized'] = dataset['text'].apply(lambda x: " ".join([lemmatizer.lemmatize(word, get_wordnet_pos(pos_tag))
                                                    for word, pos_tag in nltk.pos_tag(x.split())]))
# Lemmatisation of title
```

Step 6

Sentiment generation - Polarity

```python
from nltk.sentiment.vader import SentimentIntensityAnalyzer as sia
```

```python
def get_polarity(text):
    #without limiting text length, code took too long to run
    text = text[:text.find(' ', 30)] if len(text) > 30 else text
    text = " ".join(text.split()[:30])

    polarity_scores = sia.polarity_scores(text)
    polarity, score = sorted(polarity_scores.items(), key=lambda x: x[1], reverse=True)[:1][0]
    if polarity == 'pos': #improving dataset visualisation
        polarity = 'positive'
    elif polarity == 'neg':
        polarity = 'negative'
    elif polarity == 'neu':
        polarity = 'neutral'
    else:
        polarity = 'compound';
    return polarity, score
```

```python
polarity_score = dataset['title'].apply(get_polarity)
dataset['title_polarity'] = polarity_score.str[0]
dataset['title_polarity_score'] = polarity_score.str[1]
```

Step 7

Emotion Generation

```
!pip install text2emotion
import text2emotion as t2e

def get_emotion(sentence):
    #important as code took too long to run without limiting
    sentence = sentence[:sentence.find(' ', 30)] if len(sentence) > 30 else sentence
    sentence = " ".join(sentence.split()[:30])

    emotion_scores = t2e.get_emotion(sentence)
    if not emotion_scores:
        return "neutral"
    emotion, score = max(emotion_scores.items(), key=lambda x: x[1])
    if score == 0.0:
        return "neutral"
    else:
        return emotion


dataset['title_emotion'] = dataset['title'].apply(get_emotion)
dataset['text_emotion'] = dataset['text'].apply(get_emotion)
```

Step 8

Final Cleaning and saving cleaned dataset

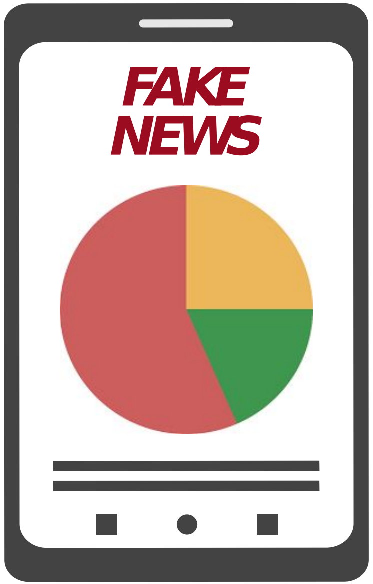
label_tran	title_char_	text_char_	title_word	text_word	title_stopv	text_stopv	text_lemm	title_lemm	title_polar	title_polar	text_polar	text_polar	title_emot	text_emotion
fake	119	3363	18	532	6	48	No comme	LAW ENFC	neutral	0.63	neutral	0.645	Surprise	Happy
fake	133	165	20	22	6	1	Now dem	UNBELIEV	positive	0.511	neutral	1	neutral	neutral
real	88	5679	12	786	0	46	A dozen p	Bobby Jinc	neutral	1	neutral	0.526	neutral	neutral
fake	78	1381	11	202	0	16	The RS Sar	SATAN Ru	neutral	0.519	neutral	1	neutral	neutral
fake	67	1131	10	154	1	10	All say one	About Tim	neutral	1	neutral	1	Surprise	neutral
fake	81	78	15	15	4	7	DR BEN CA	DR BEN CA	neutral	1	neutral	1	Fear	Sad
fake	100	986	18	156	3	15	The owne	Sports Bar	neutral	1	neutral	1	Sad	neutral
fake	66	2128	9	297	1	14	FILE In Sep	Latest Pipe	neutral	0.556	neutral	1	neutral	neutral
fake	79	3374	14	494	6	25	The punch	GOP Senat	neutral	1	neutral	1	neutral	neutral
real	58	1646	10	229	0	6	BRUSSELS	May Brexi	neutral	0.595	neutral	1	Surprise	Fear
real	63	1960	9	271	0	11	WASHING	Schumer c	neutral	1	neutral	1	Surprise	neutral
fake	82	148	13	22	2	1	After watc	WATCH HI	neutral	0.593	neutral	1	Happy	Surprise
real	82	2202	11	294	1	9	As sport fa	No Change	neutral	0.645	neutral	0.714	Happy	Surprise
real	58	1268	7	169	0	7	RIO DE JAI	Billionaire	neutral	0.508	neutral	1	Surprise	neutral
fake	75	2256	13	332	4	20	Europe lik	BRITISH W	neutral	0.569	neutral	0.69	Sad	Surprise
real	56	679	8	89	0	2	GENEVA R	U N seek h	neutral	1	neutral	0.588	neutral	neutral
fake	155	5263	21	696	7	40	The Atlant	MAJOR LIE	neutral	0.741	neutral	0.784	Angry	neutral

Exploratory Data Analysis [EDA]

Exploratory Data Analysis & Visualisation

01 Word Count Analysis

02 Corpus & Bigram Analysis



03 Stopword Count Analysis

04 Emotion & Sentiment Analysis

05 Correlation Analysis

Initial Data-Driven Insights from Dataset & Research

Fake news are **shorter in content** but uses repetitive language and less technical words

Fake news tend to **carry extreme emotions** eg. fear or surprise

Titles of fake news tend to **have fewer stopwords** for more attention-grabbing information

Fake news have **longer titles**, shorter words and more capitalised words

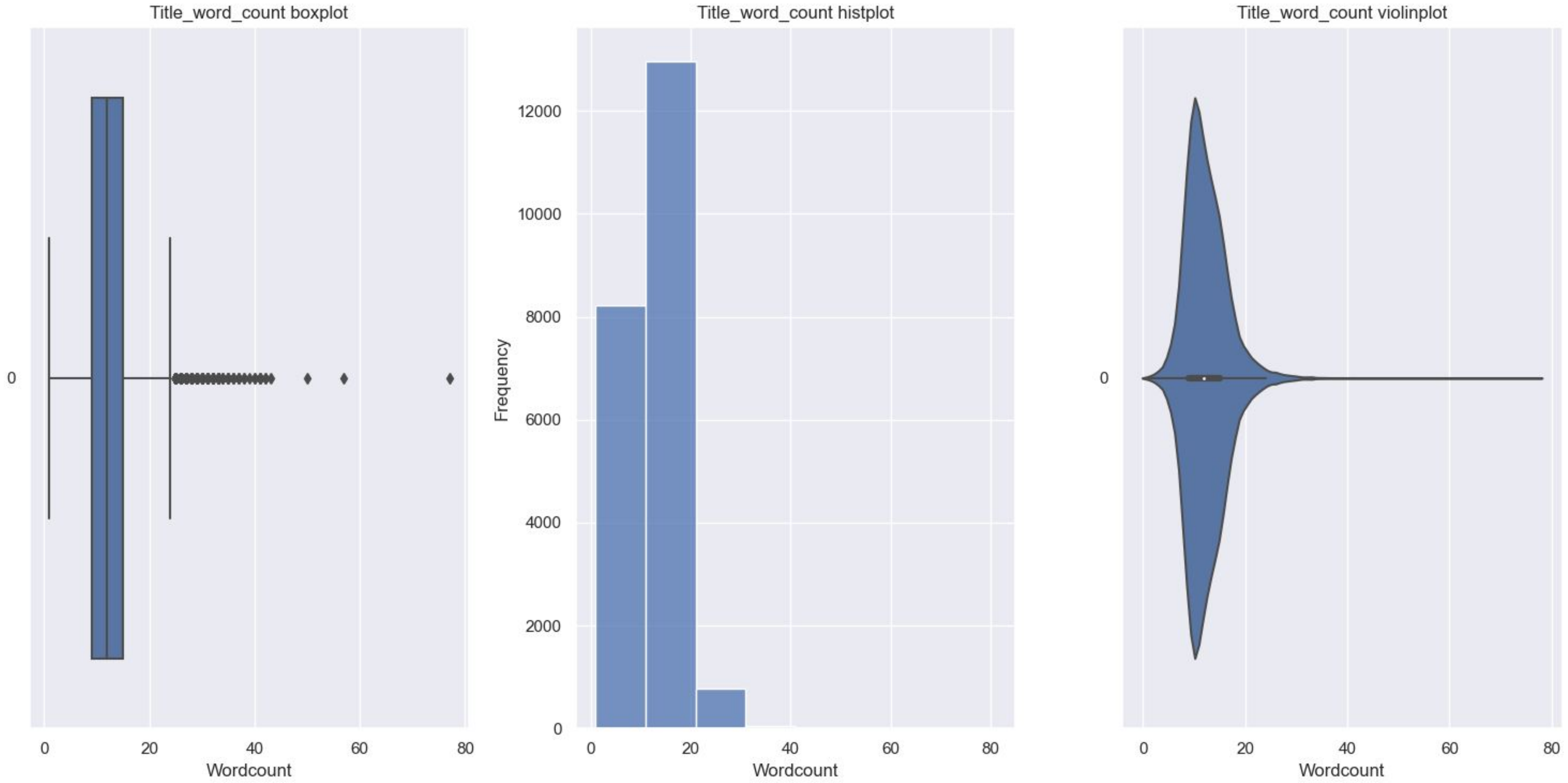
title	text	label_translated
LAW ENFORCEMENT ON HIGH ALERT Following Threats Against	No comment expected Barack Obama Memb	fake
UNBELIEVABLE! OBAMA S ATTORNEY GENERAL SAYS MOST CHA	Now demonstrators gathered last night exerc	fake
Bobby Jindal raised Hindu uses story Christian conversion woo ev	A dozen politically active pastors came privat	real
SATAN Russia unvelis image terrifying new SUPERNUKE Western	The RS Sarmat missile dubbed Satan replace S	fake
About Time! Christian Group Sues Amazon SPLC Designation Hat	All say one time someone sued Southern Pov	fake
BOOM! Danish Government Considers Seizing Migrant Valuables	Is European gravy train finally coming end?Th	fake
JOE BIDEN S SHOCKING ANNOUNCEMENT What hell man? Vide	VP Joe Biden Yeah I going run Reporter For w	fake
Trump says Brexit 'a great thing' wants quick trade deal UK	LONDON Reuters U S President elect Donald	real

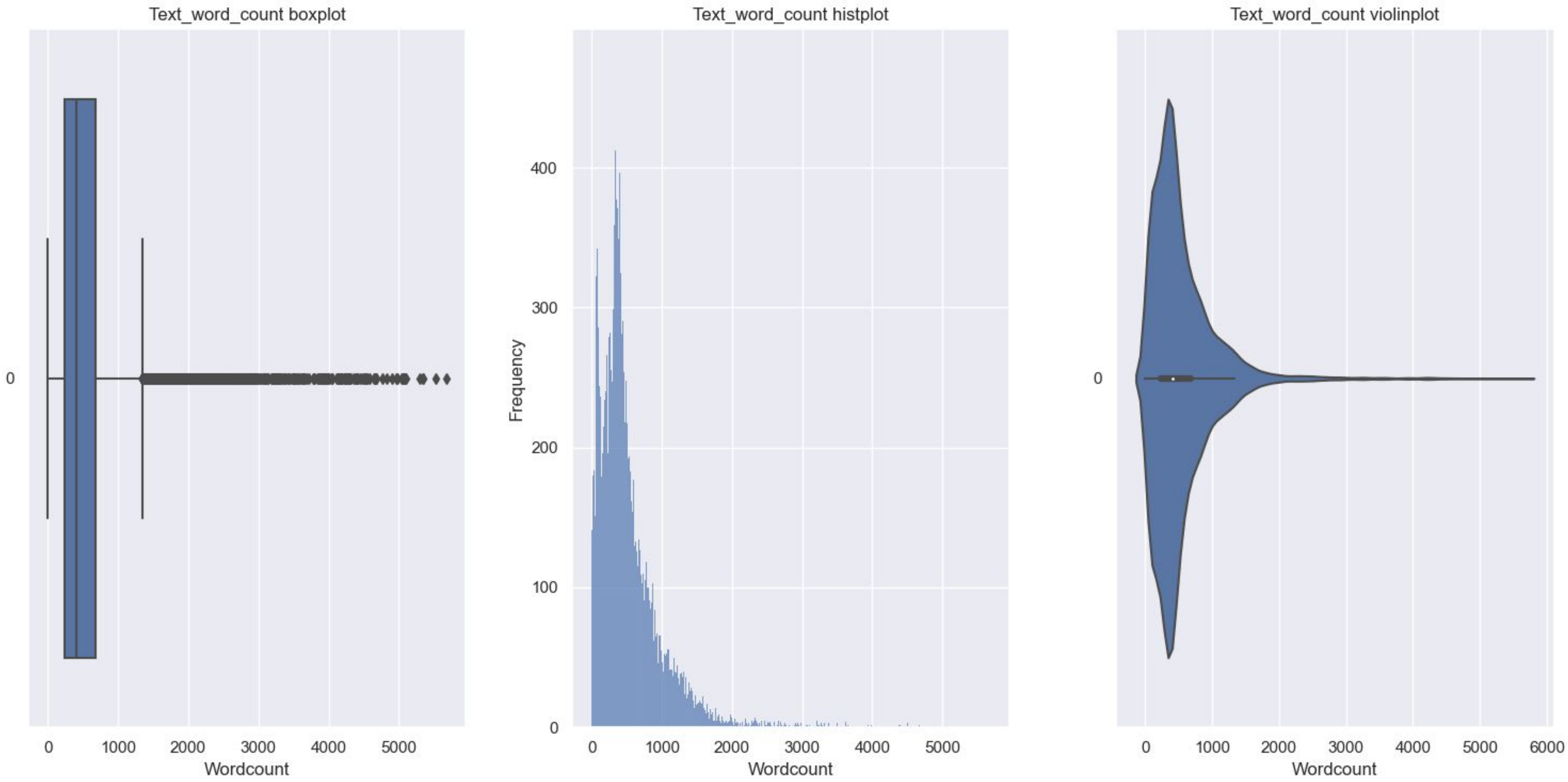
Word Count Analysis

Distribution of title and text word count

```
# Plot distribution graphs (boxplot & histogram & violinplot)
# Word count analysis (title and text)

fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(15, 15))
count = 0

for i, col in enumerate(['title_word_count', 'text_word_count']):
    sb.boxplot(data=news_data[col], orient='h', ax=axes[count, 0]).set(
        title=f"{col.capitalize()} boxplot",
        xlabel='Wordcount')
    sb.histplot(data=news_data[col],binwidth=10, ax=axes[count, 1]).set(
        title=f"{col.capitalize()} histplot",
        xlabel='Wordcount',
        ylabel ='Frequency')
    sb.violinplot(data=news_data[col],orient ="h", ax=axes[count, 2]).set(
        title=f"{col.capitalize()} violinplot",
        xlabel='Wordcount')

  #  axes[i].title.set_fontsize(14)
   # axes[i].xaxis.label.set_fontsize(14)
  #  axes[i].yaxis.label.set_fontsize(14)

    count += 1


plt.tight_layout()
plt.show()
```

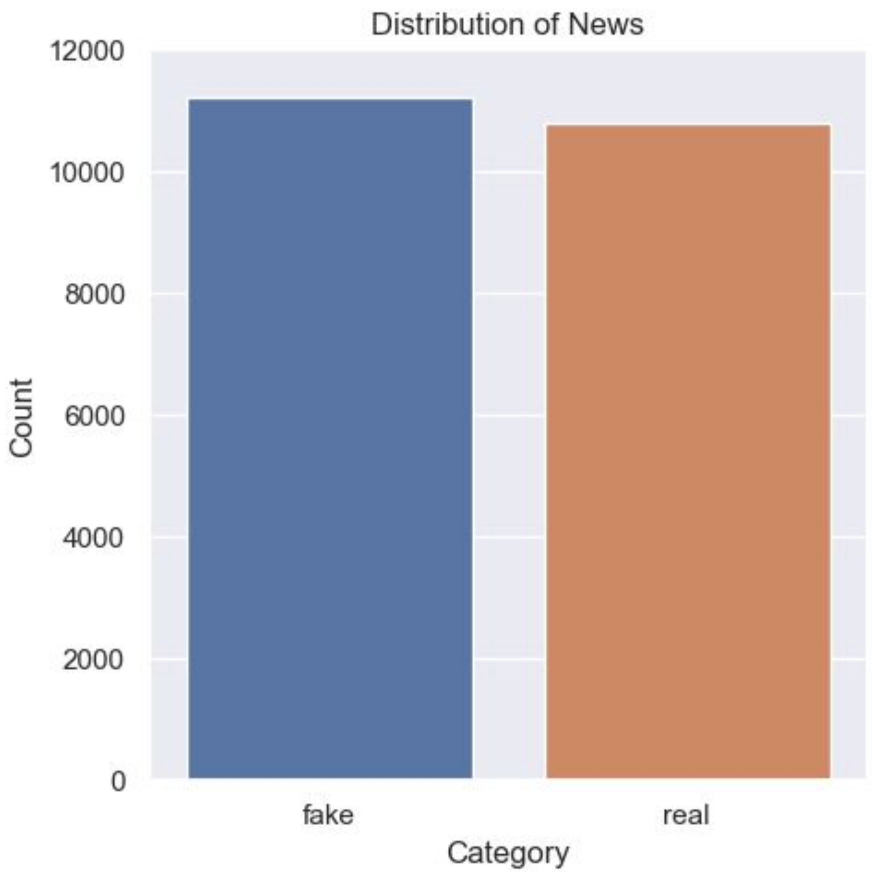
Plot boxplot, histogram and violinplot


Title_word_count boxplot
Title_word_count histplot
Title_word_count violinplot
0
0
20
40
60
80
Wordcount
Frequency
12000
10000
8000
6000
4000
2000



Text_word_count boxplot
0
0
1000
2000
3000
4000
5000
Wordcount
Text_word_count histplot
Frequency
0
100
200
300
400
0
1000
2000
3000
4000
5000
Wordcount
Text_word_count violinplot
0
0
1000
2000
3000
4000
5000
6000
Wordcount

Fake vs Real News ratio



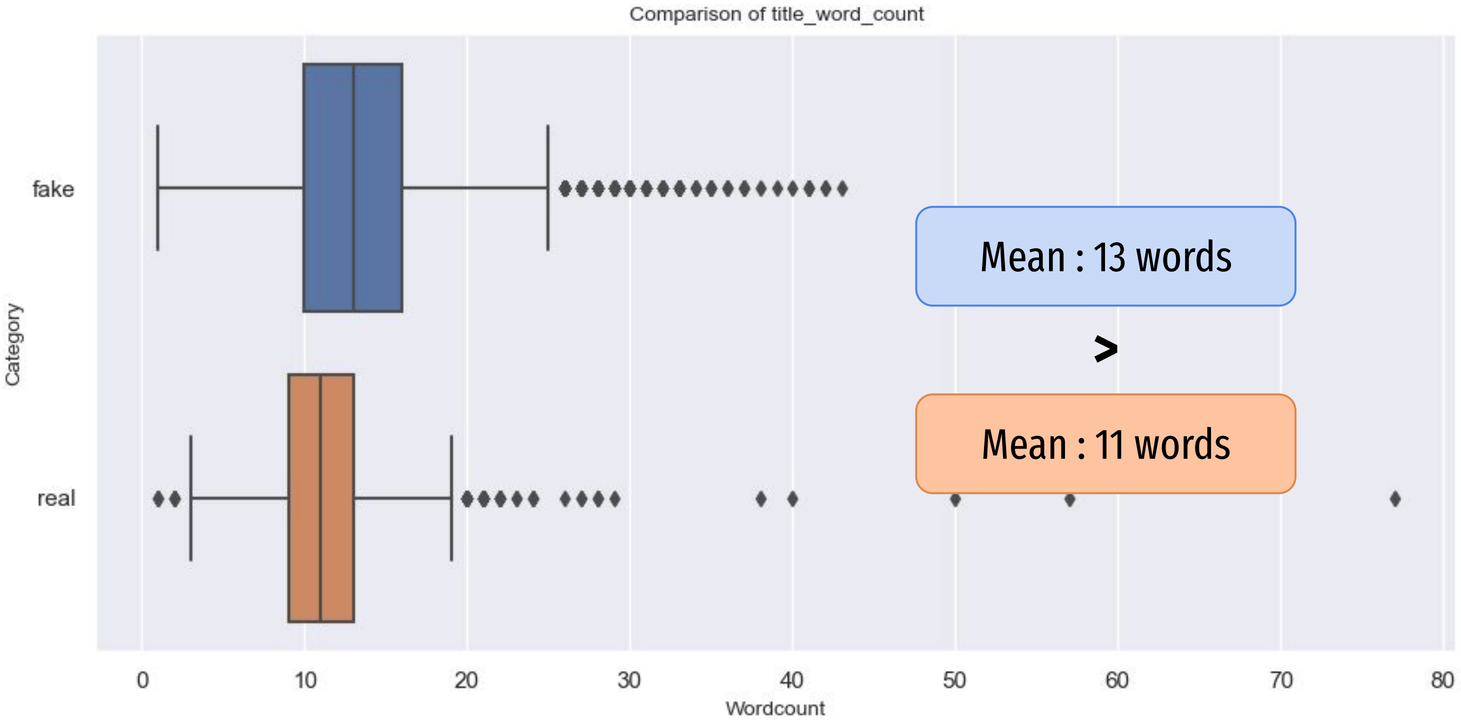
A possible concern?!

Word Count Analysis

Distribution of word count across fake and real news

Title

✔ **Assumption**

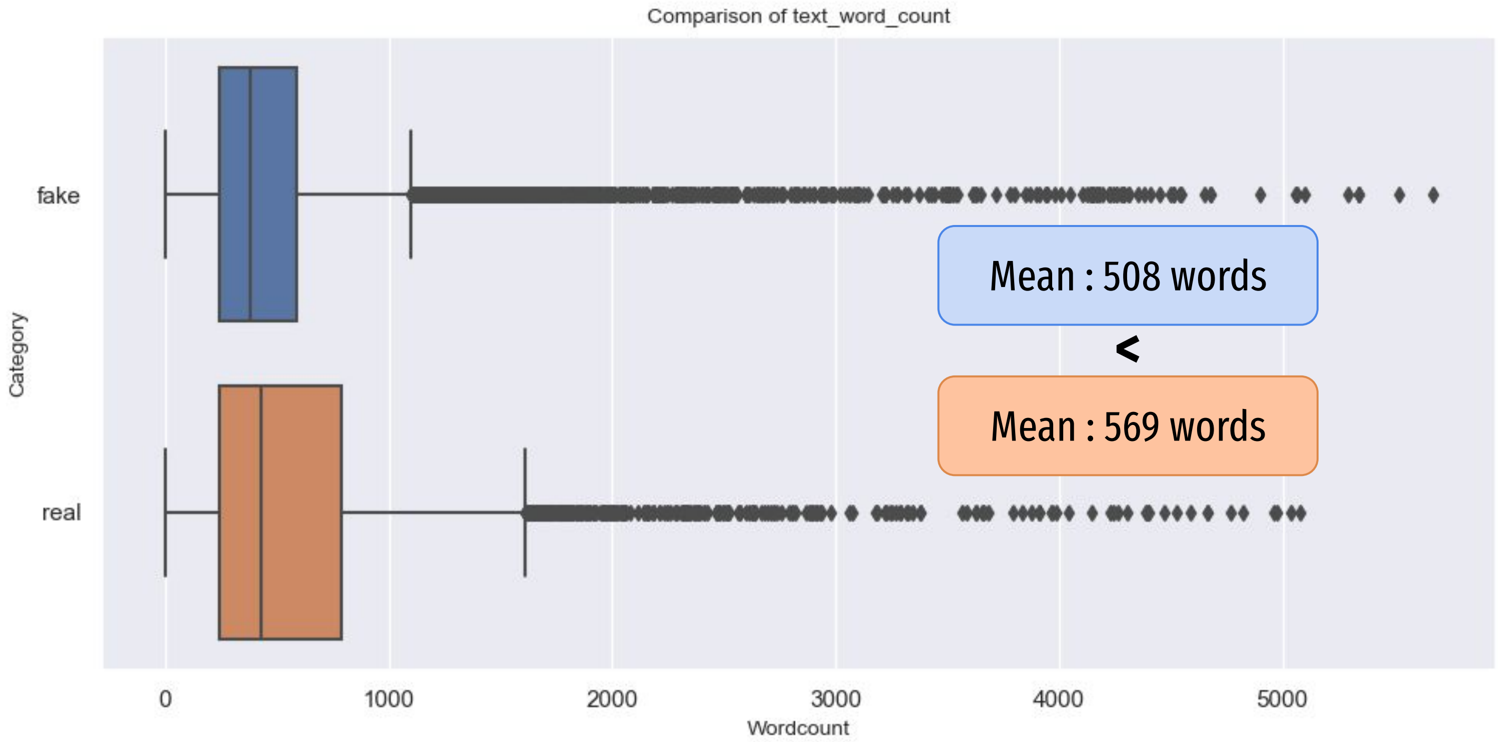


Word Count Analysis

Distribution of word count across fake and real news

Text

✔ Assumption



Corpus Analysis

Obtaining the most frequently used terms in title and text

```
#obtaining the top 15 frequently used terms in title
from collections import Counter
words = " ".join(news_data["title_lemmatized"]).split()
word_counts = Counter(words)
top_words = word_counts.most_common(15)
top_words
```

```
[('Trump', 5467),
 ('The', 4392),
 ('To', 2773),
 ('New', 2588),
 ('York', 2101),
 ('Times', 2079),
 ('S', 2061),
 ('U', 1828),
 ('VIDEO', 1726),
 ('In', 1515),
 ('For', 1509),
 ('A', 1372),
 ('Of', 1268),
 ('Is', 1141),
 ('say', 1048)]
```

Unigrams are not representative of all the words

Stopwords are not removed in lemmatised words

Corpus Analysis

Removing stopwords from lemmatized words

```
#convert all the lemmatised title and text to lower case first since stopword library is case-sensitive
news_data["text_lemmatized"]=news_data["text_lemmatized"].astype(str).str.lower()
news_data["title_lemmatized"]=news_data["title_lemmatized"].astype(str).str.lower()
news_data.head()
```

```
my_stopwords = stopwords.words('english')
print(my_stopwords)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'y
ourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those',
'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'a
n', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'b
etween', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'of
f', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both',
'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very',
's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'ar
en', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "have
n't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "should
n't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

Corpus Analysis

Added more stopwords

```
#adding some additional stopwords, including the ones which appeared in the top 15 earlier
my_stopwords.extend(["a", "about","also", "above", "after", "again", "against", "all", "am", "an", "and", "any", "are",
                     "aren't", "as", "at", "be", "because", "been", "before", "being", "below", "between", "both",
                     "but", "by", "can't","come", "cannot", "could", "couldn't", "did", "didn't", "do", "does", "doesn't",
                     "doing", "don't", "down", "during", "even","each", "few", "for", "from", "further","go","get", "had", "hadn
                     "hasn't", "have", "haven't", "having", "he", "he'd", "he'll", "he's", "her", "here", "here's", "hers",
                     "herself", "him", "himself", "his", "how", "how's", "i", "i'd", "i'll", "i'm", "i've", "if", "in", "into",
                     "is", "isn't", "it", "it's", "its", "itself","know","like","last", "let's", "me","mr", "more", "most", "mus
                     "no", "nor", "not", "of", "off", "on", "once", "only", "or", "other", "ought", "our", "ours", "ourselves",
                     "out","old", "over", "own", "same", "shan't", "she","say","she'd", "she'll", "she's", "should", "shouldn't"
                     "such", "take","than", "that", "that's", "the", "their", "theirs", "them", "themselves", "then", "there", "
                     "they", "they'd", "they'll", "they're", "they've", "this", "those", "through", "to", "too", "u","under", "u
                     "very", "was", "wasn't", "week","we", "we'd", "we'll", "we're", "we've", "were", "weren't", "what", "what's
                     "where", "watch","where's", "which", "while", "who", "who's", "whom", "why", "why's", "with", "won't", "wou
                     "you", "you'd","year", "you'll", "you're", "you've", "your", "yours", "yourself", "yourselves"
                    ])
#remove duplicates and re-assign variable
print(my_stopwords:=set(my_stopwords))
```

Corpus Analysis

Unigram << Bigram

```python
#obtaining the top 15 frequently used terms in title
from collections import Counter
words = " ".join(news_data["title_lemmatized"]).split()
word_counts = Counter(words)
top_words = word_counts.most_common(15)
top_words
```

```
[('trump', 5813),
 ('new', 2866),
 ('video', 2713),
 ('york', 2114),
 ('times', 2095),
 ('hillary', 1244),
 ('obama', 1208),
 ('clinton', 1114),
 ('house', 758),
 ('breitbart', 749),
 ('donald', 678),
 ('president', 636),
 ('white', 636),
 ('russia', 566),
 ('election', 561)]
```

```python
#obtaining the top 15 frequently used terms in text
from collections import Counter
words = " ".join(news_data["text_lemmatized"]).split()
word_counts = Counter(words)
top_words = word_counts.most_common(15)
top_words
```

```
[('trump', 65041),
 ('president', 27969),
 ('one', 27745),
 ('people', 27462),
 ('state', 25027),
 ('make', 24553),
 ('clinton', 22278),
 ('new', 21757),
 ('time', 18101),
 ('government', 16369),
 ('tell', 15211),
 ('obama', 15180),
 ('country', 14783),
 ('call', 14782),
 ('campaign', 14182)]
```

Bigram Analysis

Important and Useful Functions

```
#creating a function to obtain frequently used bigram words in title,text and across fake and true news

# define the data and column names
data = news_data
text_col = "text_lemmatized"
title_col = "title_lemmatized"

#define the number of most common bigrams to display
#n = 15

top_bigrams = {} #impt if not cannot access string through index
# loop through the columns and print the most common bigrams

def get_top_bigrams(data,col_name,n):
    # get the words as a list
    words = data[col_name].dropna().str.split().explode().tolist()

    # create bigrams from the words
    bigrams = list(ngrams(words, 2))

    # count the frequency of each bigram
    bigram_counts = Counter(bigrams)

    # get the top n most common bigrams
    top_bigrams = bigram_counts.most_common(n)

    return top_bigrams
```

```
def plot_bigrams(top_bigrams, col_name):
    #extract the words and counts from the top n bigrams
    counts = [count for _,count in top_bigrams]
    bigrams = [" ".join(bigram) for bigram, _ in top_bigrams]

    #plot the bar graph
    f = plt.figure(figsize=(20,10))
    ax = sb.barplot(y=bigrams, x=counts, orient='h')
    ax.set_title(f"Top {len(top_bigrams)} bigrams in {col_name}", fontsize=18)
    ax.set_xlabel("Frequency", fontsize=16)
    ax.set_ylabel("Bigrams", fontsize=16 )
    plt.xticks(fontsize=16)
    plt.yticks(fontsize=16)
```

Bigram Analysis

Calling the functions

Title vs Text

```
title_bigrams = get_top_bigrams(data, title_col, 15)
plot_bigrams(title_bigrams, title_col)

text_bigrams = get_top_bigrams(data, text_col, 15)
plot_bigrams(text_bigrams, text_col)
```

Fake title vs Real title

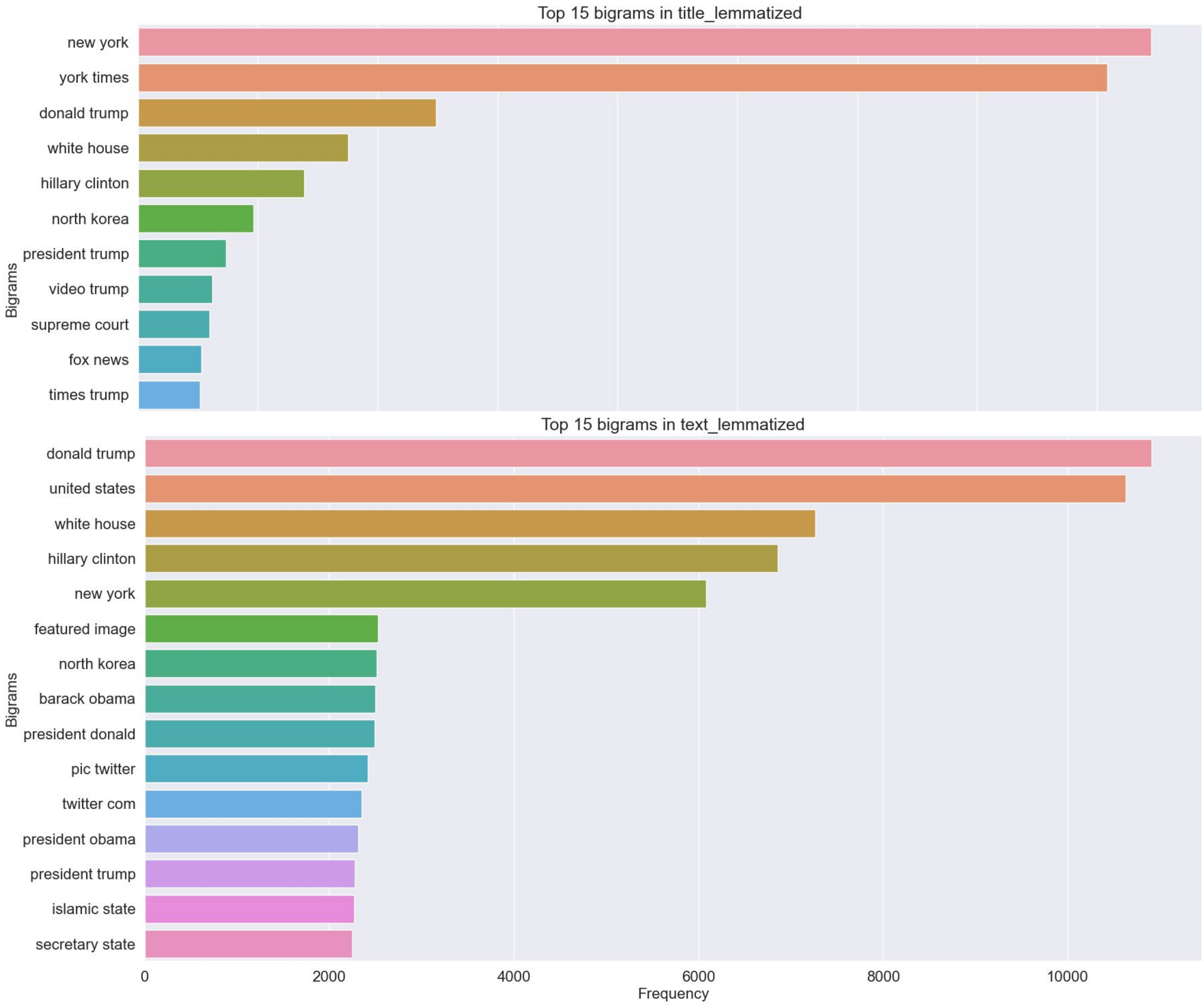
```
#bigram title classification of fake and  real

#create new columns for fake lemmatised title and real lemmatised title
news_data['title_lemmatized_fake'] = news_data[news_data['label_translated'] == 'fake']['title_lemmatized']
news_data['title_lemmatized_real'] = news_data[news_data['label_translated'] == 'real']['title_lemmatized']

fake_title_bigrams= get_top_bigrams(data, 'title_lemmatized_fake', 15)
plot_bigrams(fake_title_bigrams,"Fake News Title")

real_title_bigrams = get_top_bigrams(data,'title_lemmatized_real', 15)
plot_bigrams(real_title_bigrams, "Real News Title")
```

Fake text vs Real text

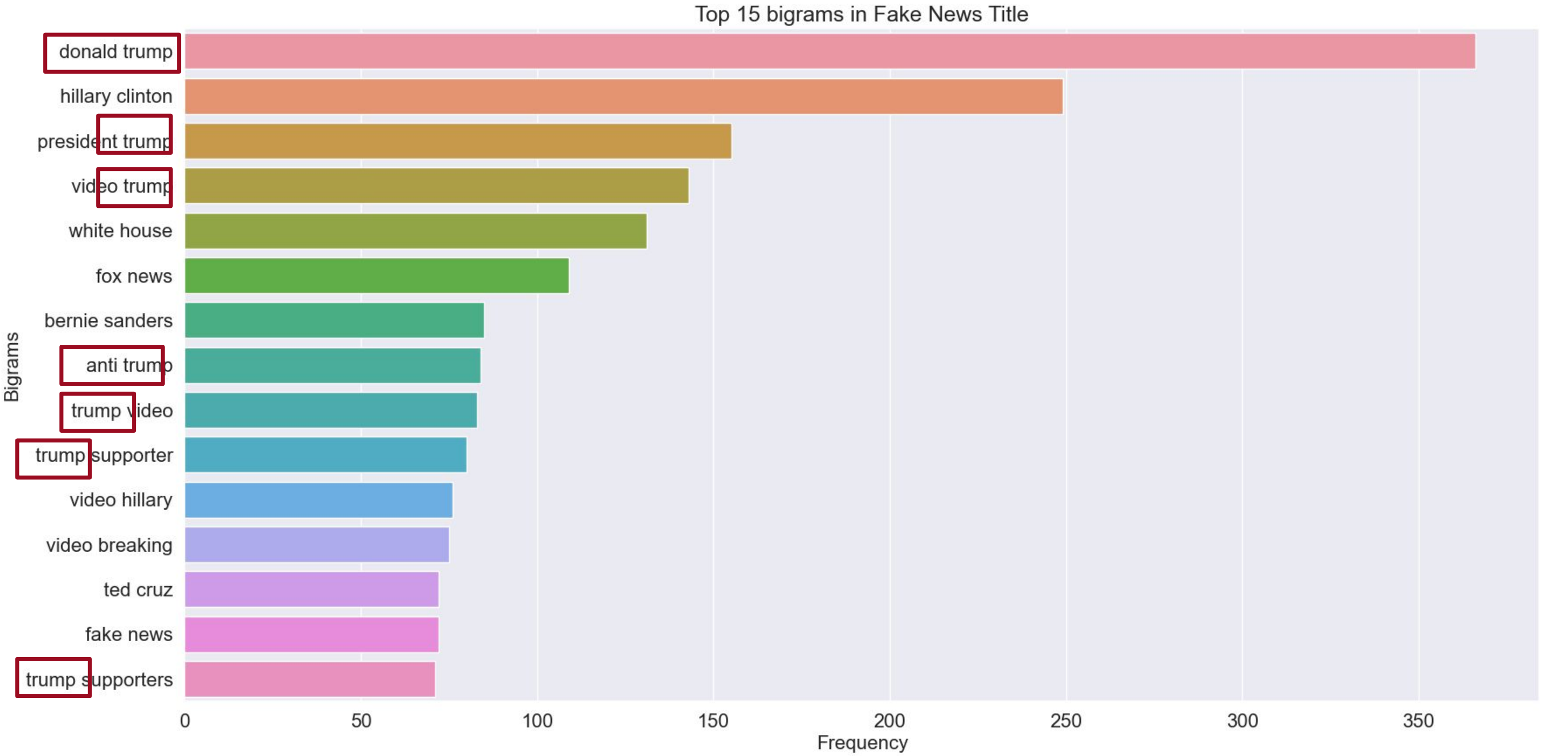
```
#bigram text classification of fake and  real

news_data['text_lemmatized_fake'] = news_data[news_data['label_translated'] == 'fake']['text_lemmatized']
news_data['text_lemmatized_real'] = news_data[news_data['label_translated'] == 'real']['text_lemmatized']

fake_text_bigrams= get_top_bigrams(data, 'text_lemmatized_fake', 15)
plot_bigrams(fake_text_bigrams,"Fake News Text")

real_text_bigrams = get_top_bigrams(data,'text_lemmatized_real', 15)
plot_bigrams(real_text_bigrams, "Real News Text")
```

Bigram Analysis

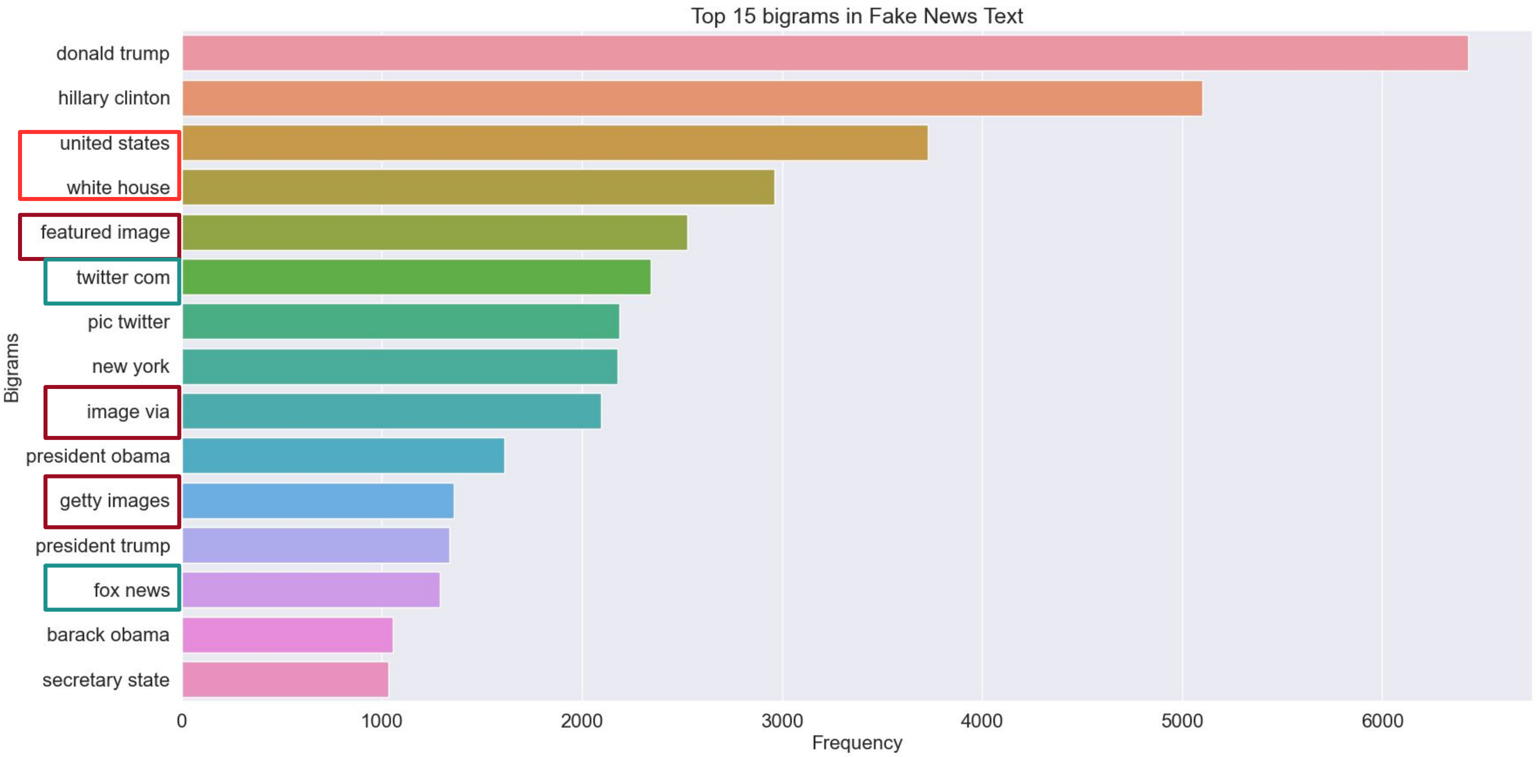
Plotting of bigram graphs of title,text



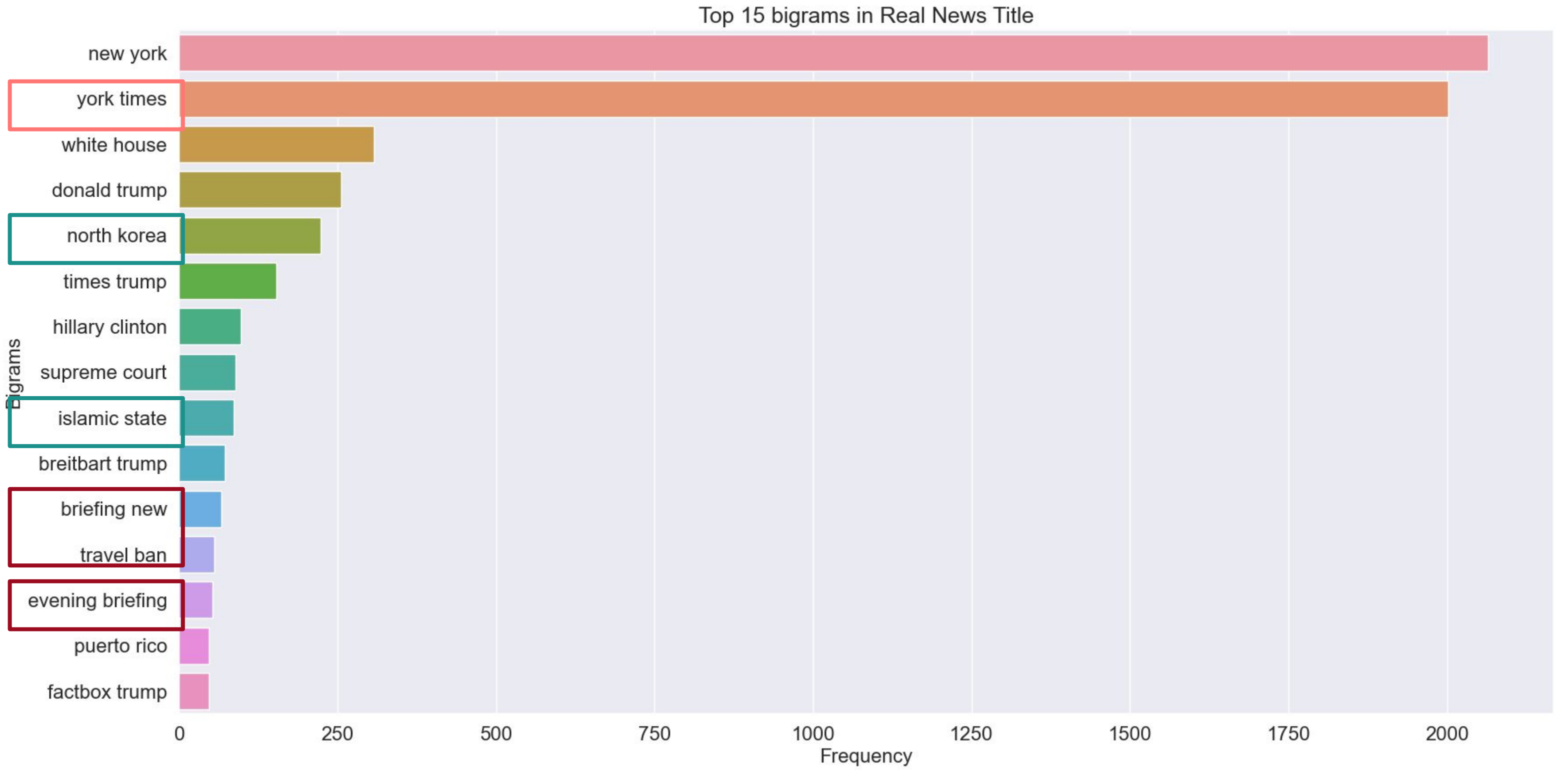
Fake news' title



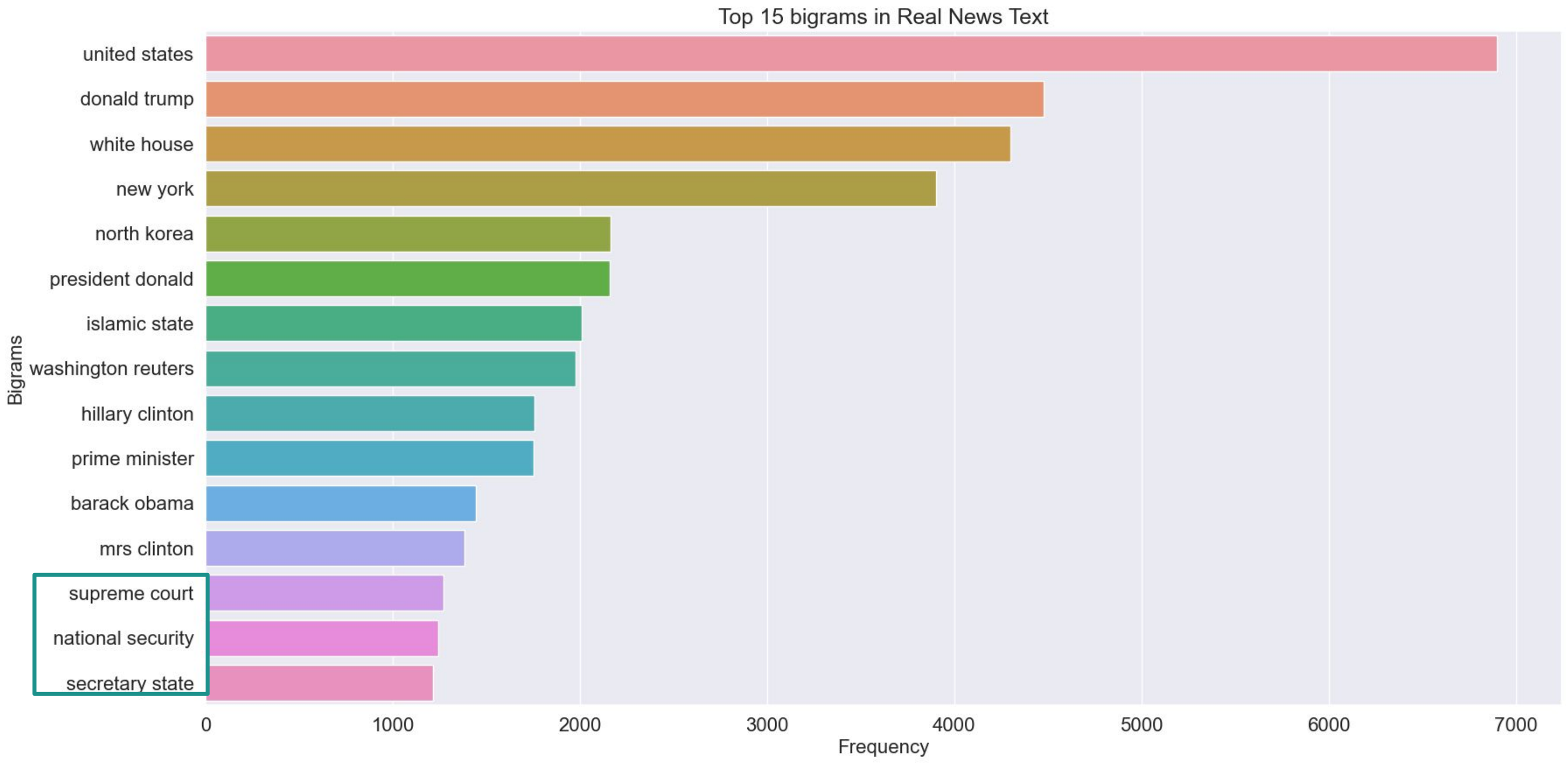
Fake news' text



Real news' title



Real news' text

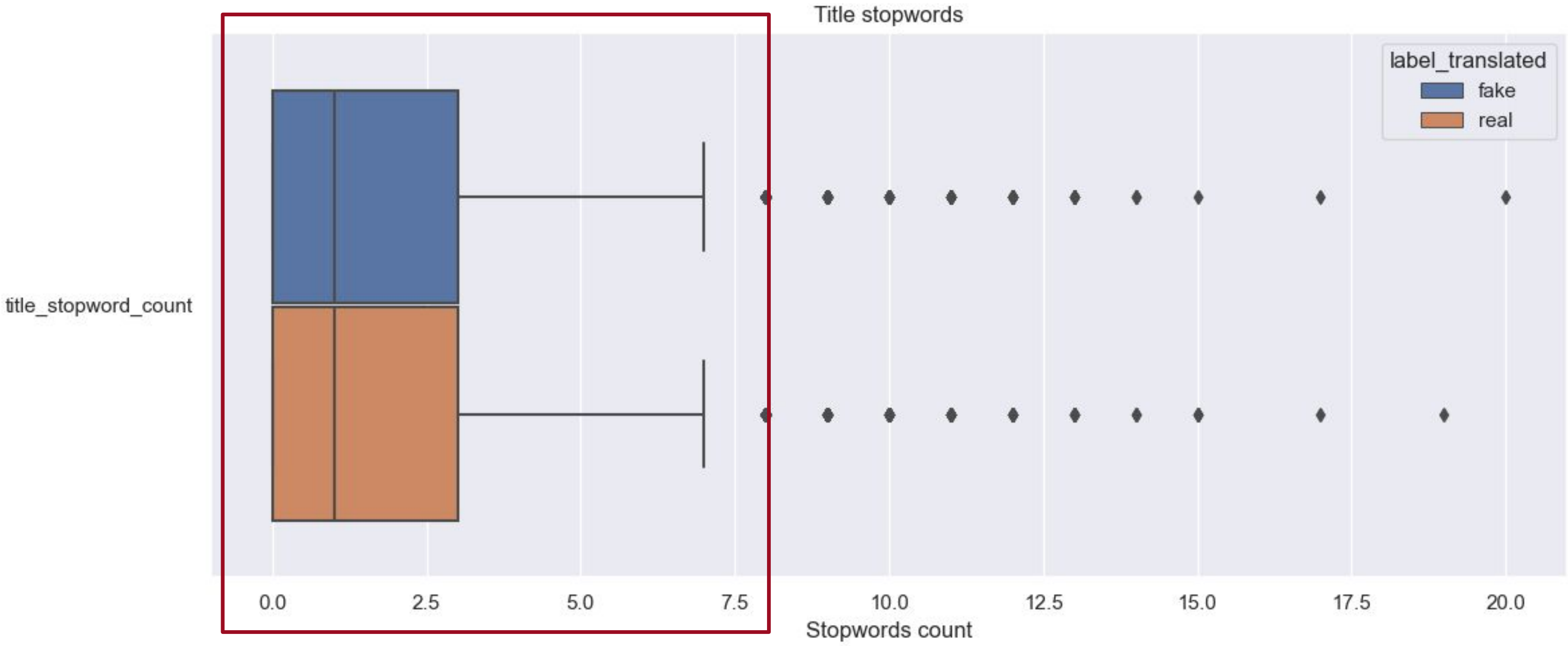


Stopword Count Analysis

```
cols = ["title_stopword_count", "text_stopword_count"]
titles = ["Title stopwords", "Text stopwords"]
fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(12, 10))

for i, col in enumerate(cols):
    pos_df = pd.concat([news_data[col], news_data["label_translated"]], axis=1)
    pos_df = pd.melt(pos_df, id_vars=['label_translated'], var_name=['Stopword'])
    sb.boxplot(x='value', y="Stopword", hue="label_translated", data=pos_df, orient="h", showfliers=True, ax=axes[i]).set(
        title=titles[i], xlabel='Stopwords count', ylabel=None)

plt.tight_layout()
plt.show()
```

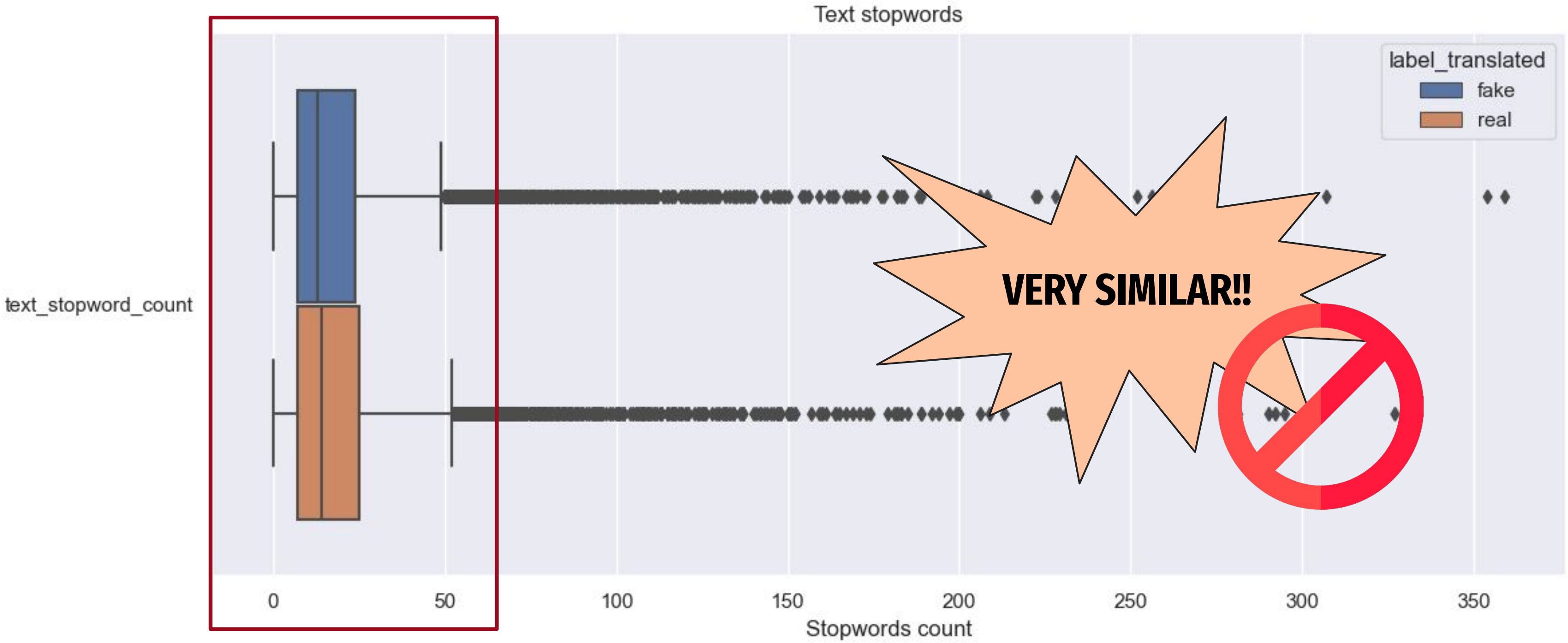
Stopword Count Analysis

❌ Assumption



Stopword Count Analysis

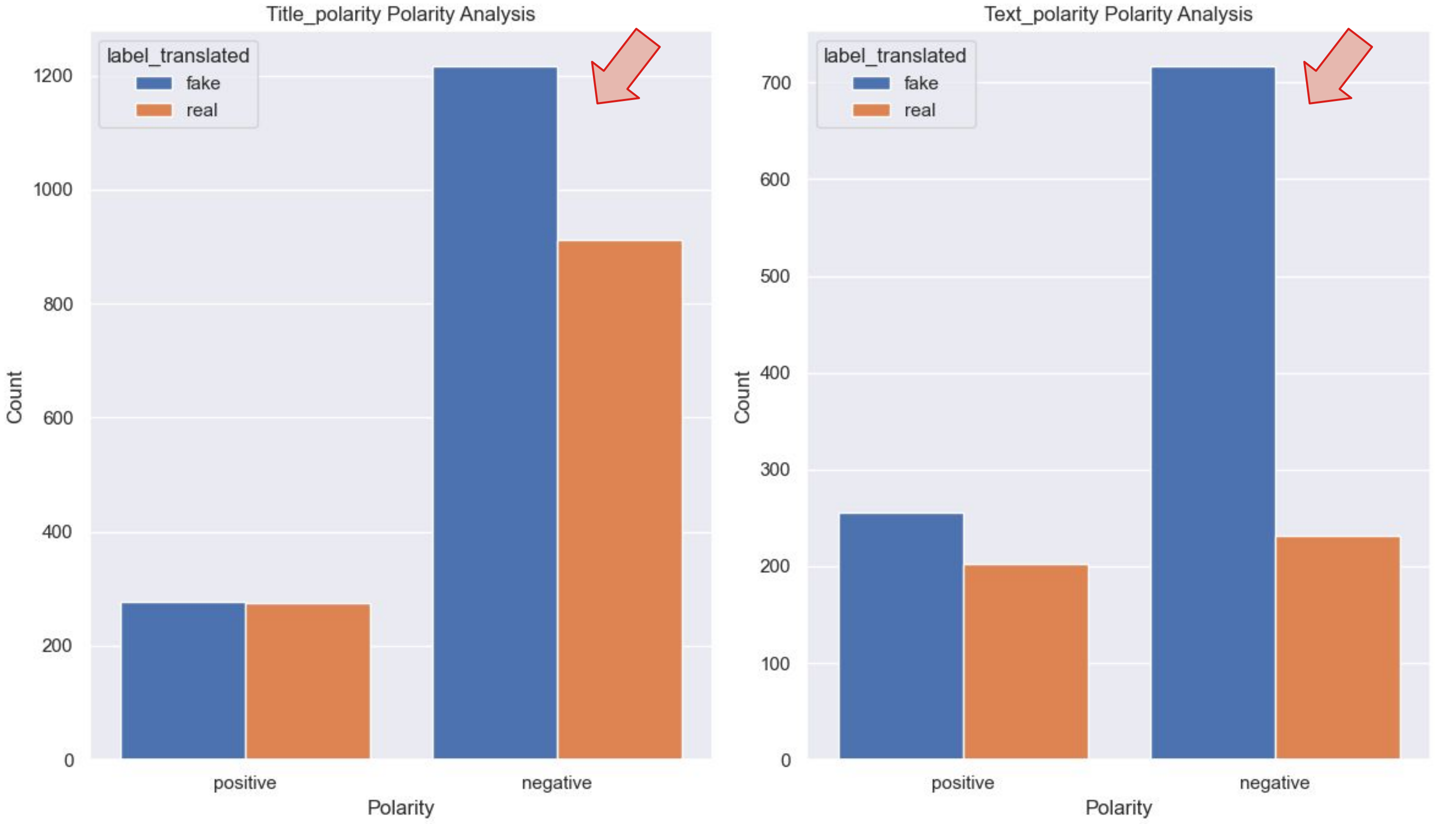
❌ Assumption



Sentiment Analysis

```
order = ['neutral', 'positive', 'negative']

fig, axs = plt.subplots(1, 2, figsize=(12, 7))

for i, col in enumerate(['title_polarity', 'text_polarity']):
    sb.countplot(x=col, hue='label_translated', data=news_data,
                 ax=axs[i], saturation=1, order=order)
    axs[i].set(title=f"{col.capitalize()} Polarity Analysis",
               xlabel='Polarity', ylabel='Count')

plt.tight_layout()
plt.show()
```

Sentiment Analysis

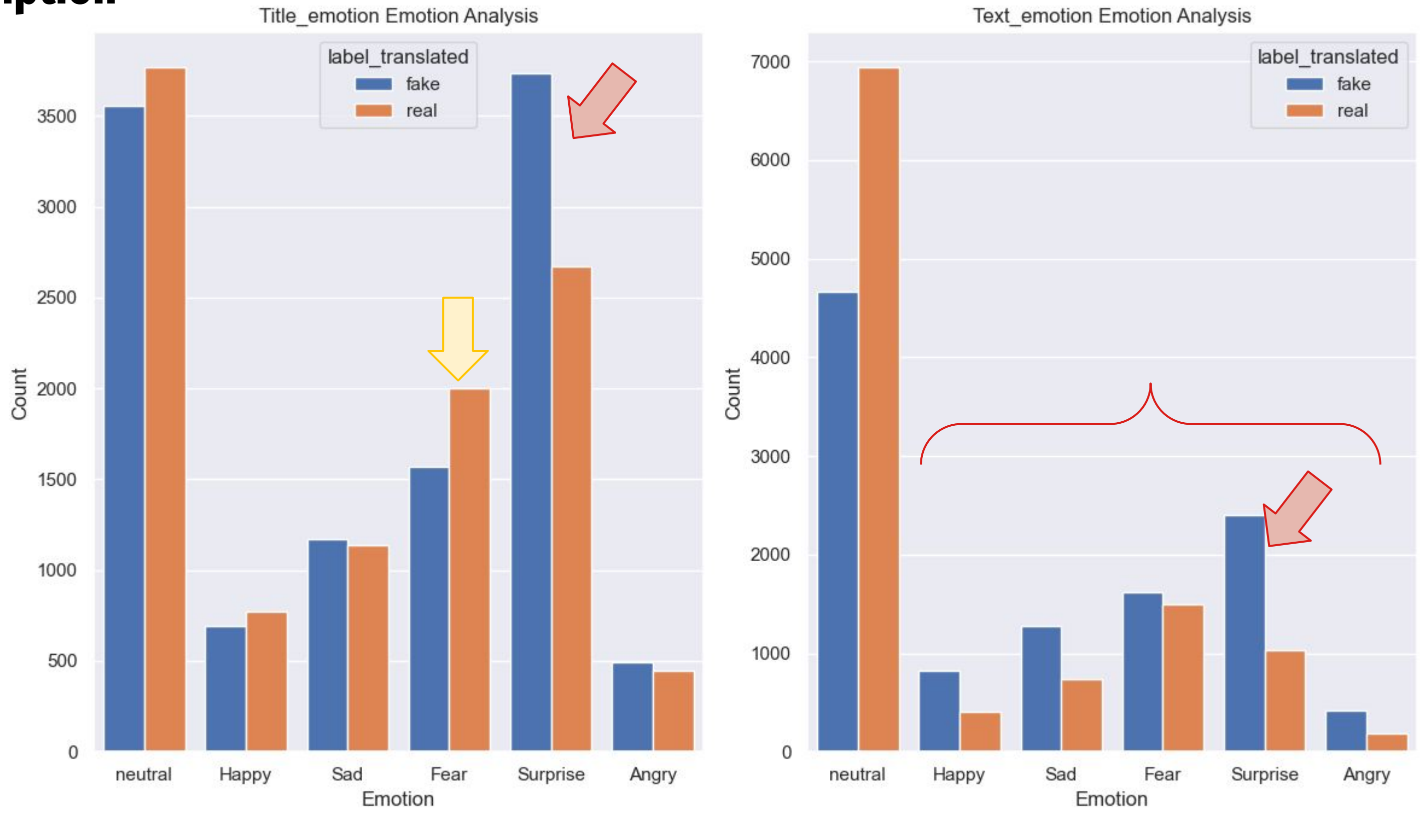


Emotion Analysis

```
order = ['neutral', 'Happy', 'Sad', 'Fear','Surprise','Angry']

fig, axs = plt.subplots(1, 2, figsize=(12, 7))

for i, col in enumerate(['title_emotion', 'text_emotion']):
    sb.countplot(x=col, hue='label_translated', data=news_data,
                 ax=axs[i], saturation=1, order=order)
    axs[i].set(title=f"{col.capitalize()} Emotion Analysis",
               xlabel='Emotion', ylabel='Count')

plt.tight_layout()
plt.show()
```

Emotion Analysis

Assumption



Correlation

Converting categorical data into numeric using Label Encoding

```
#for corelation
from sklearn.feature_extraction.text import CountVectorizer
```

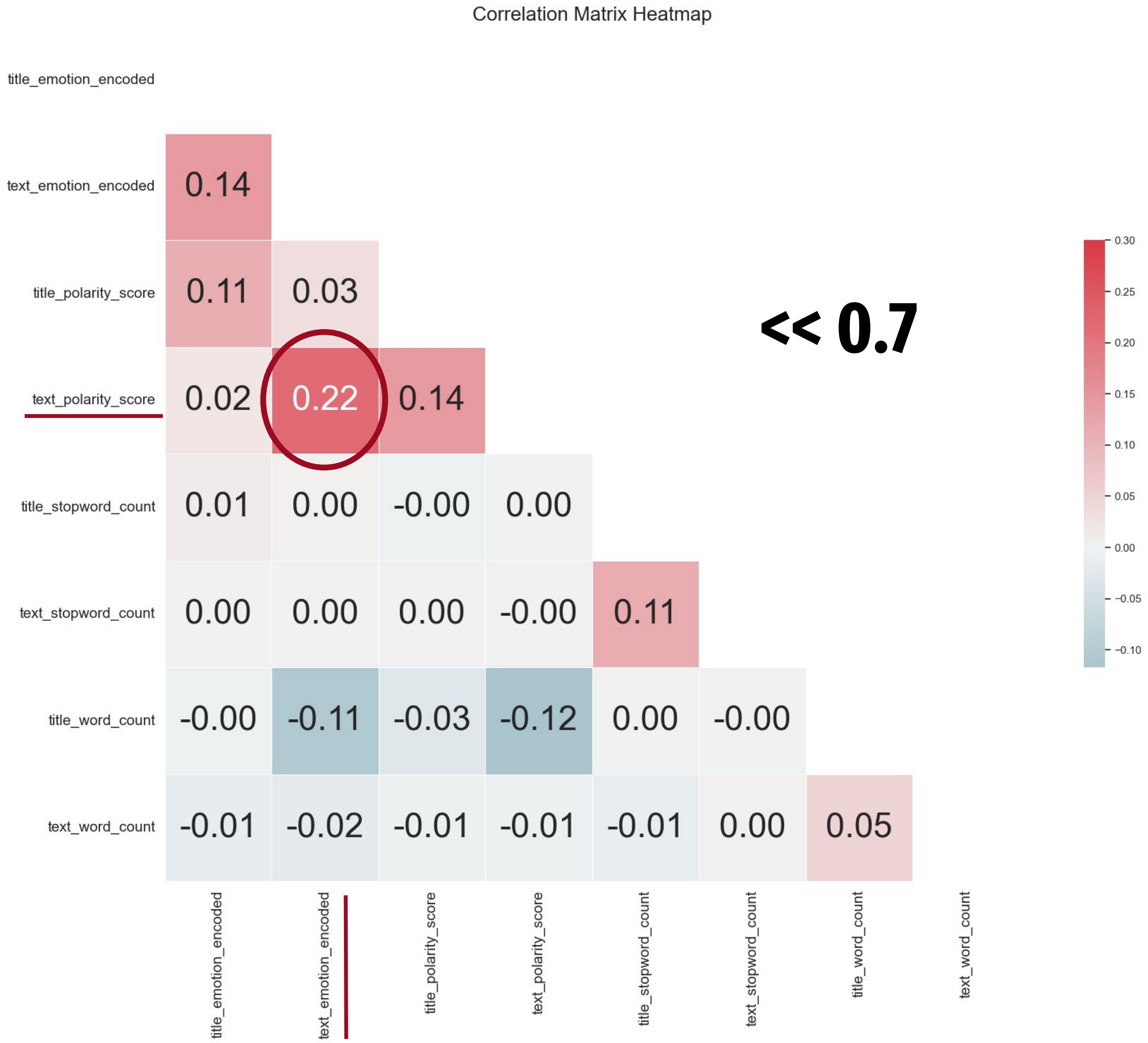
```
le = LabelEncoder()
#changing categorical to numeric values
news_data['title_emotion_encoded'] = le.fit_transform(news_data['title_emotion'])
news_data['text_emotion_encoded'] = le.fit_transform(news_data['text_emotion'])
```

title_emotion	text_emotion
Surprise	Happy
neutral	neutral
neutral	neutral
neutral	neutral
Surprise	neutral

title_emotion_encoded	text_emotion_encoded
4	2
5	5
5	5
5	5
4	5

Correlation

```
# define the columns to include in the correlation matrix
columns = ["title_emotion_encoded", "text_emotion_encoded", "title_polarity_score", "text_polarity_score",
           "title_stopword_count", "text_stopword_count", "title_word_count", "text_word_count"]

# create the correlation matrix
corr = news_data[columns].corr()

# plot heatmap
sb.set(style='white')
mask = np.triu(np.ones_like(corr, dtype=bool))
f, ax = plt.subplots(figsize=(20, 15))
cmap = sb.diverging_palette(220, 10, as_cmap=True)
sb.heatmap(corr, annot=True, fmt='.2f', mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5},annot_kws={"size": 35})
plt.title('Correlation Matrix Heatmap',fontsize = 20)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.tight_layout()
plt.show()
```


Correlation Matrix Heatmap
title_emotion_encoded
text_emotion_encoded
title_polarity_score
text_polarity_score
title_stopword_count
text_stopword_count
title_word_count
text_word_count
0.14
0.11
0.03
0.02
0.22
0.14
0.01
0.00
-0.00
0.00
0.00
0.00
0.00
-0.00
0.11
-0.00
-0.11
-0.03
-0.12
0.00
-0.00
-0.01
-0.02
-0.01
-0.01
-0.01
0.00
0.05
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10

<< 0.7

Machine Learning

Models used

Logistic Regression
Classification and prediction tasks

Decision Tree
A predictive model that can be used for classification or regression analysis

Random Forest
Improve the accuracy and robustness of decision tree models

Support Vector Machine Classifier
Classify data by finding the best possible boundary that separates different classes in the dataset.

Ensemble
Improve the accuracy and generalization performance of machine learning models

Top Three Predictors

Title Word Count

Text Word Count

Text Emotion

Title Emotion

Title Polarity

Text Polarity

Title Stopword Count

Text Stopword Count

```
Predictor: title_word_count, score: 0.66
Predictor: text_emotion, score: 0.62
Predictor: text_word_count, score: 0.61
Predictor: title_emotion, score: 0.54
Predictor: text_polarity, score: 0.53
Predictor: title_polarity, score: 0.52
Predictor: title_stopword_count, score: 0.51
Predictor: text_stopword_count, score: 0.51
```

1. Title Word Count
2. Text Emotion
3. Text Word Count

Training and Test Dataset

```
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.2)
```

80% Train Data

20% Test Data

```
print(f"Train: {x_train.shape[0]} & {y_train.shape[0]}")
```

```
print(f"Test: {x_test.shape[0]} & {y_test.shape[0]}")
```

```
Train: 17609 & 17609
Test: 4403 & 4403
```

Predictors Used

Logistic Regression	Decision Tree, Random Forest, SVM
Title	Title Word Count Text Word Count Text Emotion

Model 1 : Logistic Regression

Step 1 : TF-IDF Analysis

```
tfidf = TfidfVectorizer(stop_words='english',
                        strip_accents=None,
                        lowercase=True,
                        preprocessor=None,
                        use_idf=True,
                        norm='l2',
                        smooth_idf=False, max_features=15000)
```

TF-IDF Extraction of top n features

```
def tfidf_top_n_features(tfidf_data, features, n):
    tfidf_df = pd.DataFrame(tfidf_data.toarray(), columns = features)
    tfidf_df = tfidf_df.transpose()
    tfidf_means = np.mean(tfidf_df, axis=1)
    tfidf_df_summed = pd.DataFrame({'feature':tfidf_means.index, 'avg_tfidf':tfidf_means.values})
    tfidf_df_summed = tfidf_df_summed.sort_values(by='avg_tfidf', ascending=False)[:n]
    return tfidf_df_summed
```

	feature	avg_tfidf
13492	trump	0.033033
8225	new	0.020477
14178	video	0.017489
14916	york	0.016650
13134	times	0.016525
10787	says	0.011719
6237	hillary	0.011178
2105	clinton	0.011041
8345	obama	0.010767
6350	house	0.007964
3924	donald	0.007495
1513	breitbart	0.007244
9149	president	0.006867
10602	russia	0.006748
14595	white	0.006563

Table of features extracted with TF-IDF

Model 1 : Logistic Regression

Step 2 : Model Training

```
x_train, x_test, y_train, y_test = train_test_split(df['title'], df['label_translated'], test_size=0.2)
```

```
logreg = LogisticRegression(penalty='l2', C=1.0)
```

```
logreg.fit(tfidf_train, y_train)
```

```
Score: 0.91993
Score: 0.93242
Score: 0.92561
Score: 0.92419
Score: 0.93269
Mean score: 0.92697
```

Step 3 : K-Fold Cross Validation

```
sf = StratifiedKFold(n_splits=5, shuffle=True, random_state=2)
```

```
cv_results = cross_val_score(logreg, tfidf_train, y_train, cv=sf)
```

Great!

Model 1 : Logistic Regression Results

```
# Predict test dataset
y_train_pred = logreg.predict(tfidf_train)

# Print the Classification Accuracy
print("Train Data")
print("Accuracy  :\t", logreg.score(tfidf_train, y_train))
print()

# Print the Accuracy Measures from the Confusion Matrix
cmTrain = confusion_matrix(y_train, y_train_pred)
tpTrain = cmTrain[1][1] # True Positives : Good (1) predicted Good (1)
fpTrain = cmTrain[0][1] # False Positives : Bad (0) predicted Good (1)
tnTrain = cmTrain[0][0] # True Negatives : Bad (0) predicted Bad (0)
fnTrain = cmTrain[1][0] # False Negatives : Good (1) predicted Bad (0)

print("TPR Train :\t", (tpTrain/(tpTrain + fnTrain)))
print("TNR Train :\t", (tnTrain/(tnTrain + fpTrain)))
print()

print("FPR Train :\t", (fpTrain/(tnTrain + fpTrain)))
print("FNR Train :\t", (fnTrain/(tpTrain + fnTrain)))
```

Output

Train Data
Accuracy : 0.961042648645579

TPR Train : 0.9598048327137546
TNR Train : 0.9622264192867459

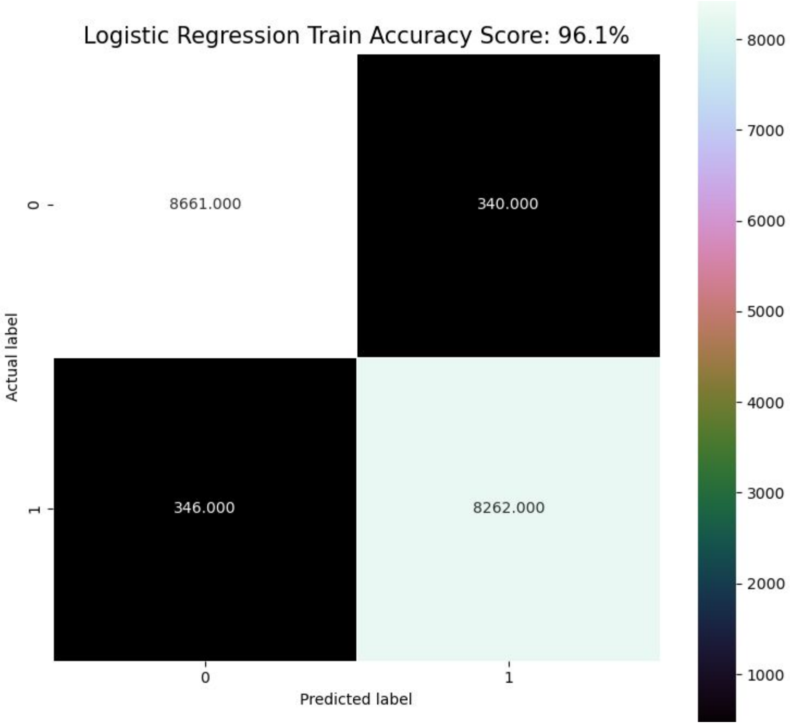
FPR Train : 0.03777358071325408
FNR Train : 0.04019516728624535

Test Data
Accuracy : 0.929366341131047

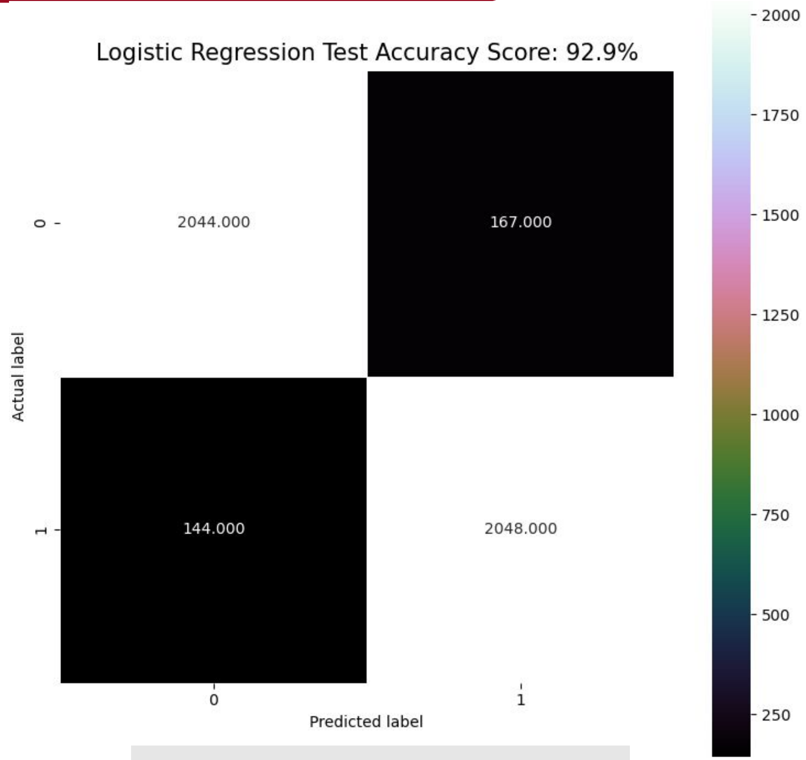
TPR Test : 0.9343065693430657
TNR Test : 0.924468566259611

FPR Test : 0.07553143374038897
FNR Test : 0.06569343065693431

Model 1 : Logistic Regression Result



TPR Train :	**0.9598048327137546**
TNR Train :	**0.9622264192867459**
FPR Train :	**0.03777358071325408**
FNR Train :	**0.04019516728624535**



TPR Test :	**0.9343065693430657**
TNR Test :	**0.924468566259611**
FPR Test :	**0.07553143374038897**
FNR Test :	**0.06569343065693431**

Model 2 : Decision Tree

```
dectree = DecisionTreeClassifier(max_depth = 4)
```

Response and Predictors

```
y = pd.DataFrame(news_final['label_translated'])
x = pd.DataFrame(news_final.drop('label_translated', axis = 1))
```

label_translated

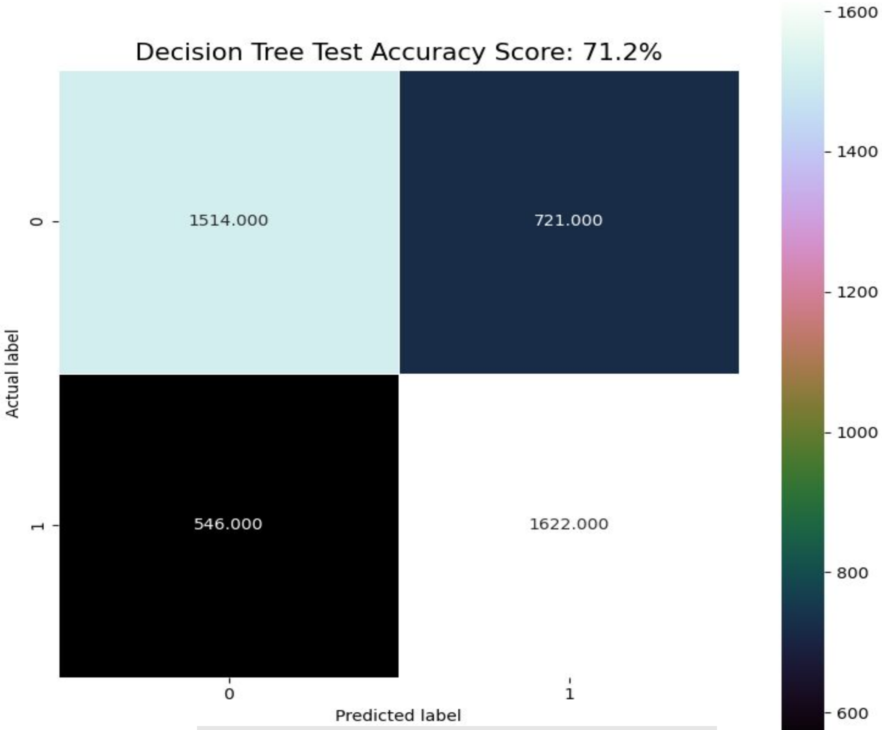
Title Word Count

Text Word Count

Text Emotion

Model 2 : Decision Tree Result

TPR Train :	**0.7594995366079703**
TNR Train :	**0.6804054806728306**
FPR Train :	**0.3195945193271694**
FNR Train :	**0.24050046339202966**



TPR Test :	**0.7481549815498155**
TNR Test :	**0.6774049217002237**
FPR Test :	**0.3225950782997763**
FNR Test :	**0.2518450184501845**

Model 2 : Decision Tree Result Analysis

Train Data

Accuracy :	**0.7191776932250553**
TPR Train :	**0.7594995366079703**
TNR Train :	**0.6804054806728306**
FPR Train :	**0.3195945193271694**
FNR Train :	**0.24050046339202966**

Test Data

Accuracy :	**0.712241653418124**
TPR Test :	**0.7481549815498155**
TNR Test :	**0.6774049217002237**
FPR Test :	**0.3225950782997763**
FNR Test :	**0.2518450184501845**

What have we done?

Tuning of hyperparameters

dectree = DecisionTreeClassifier(max_depth = 2)	70.1%
dectree = DecisionTreeClassifier(max_depth = 3)	71.1%
dectree = DecisionTreeClassifier(max_depth = 4)	71.9%

Data Pre-processing

- Data Cleaning
- Data Transformation
- Feature Selection
- Feature Scaling
- Handling missing values
- Data Splitting

What else can we do?

Model 3 : Random Forest

```
rforest = RandomForestClassifier(n_estimators = 100, max_depth = 4)
```

Response and Predictors

```
y = pd.DataFrame(news_final['label_translated'])
x = pd.DataFrame(news_final.drop('label_translated', axis = 1))
```

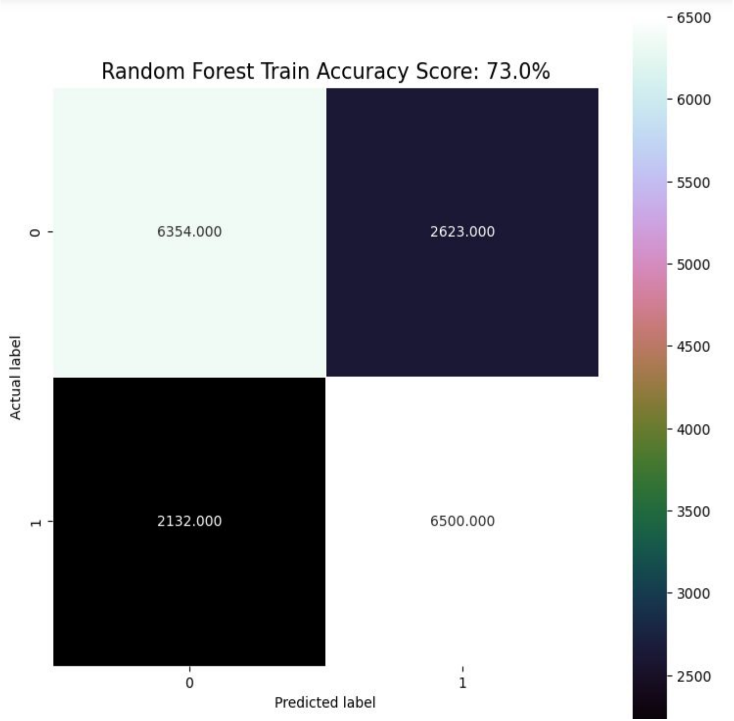
label_translated

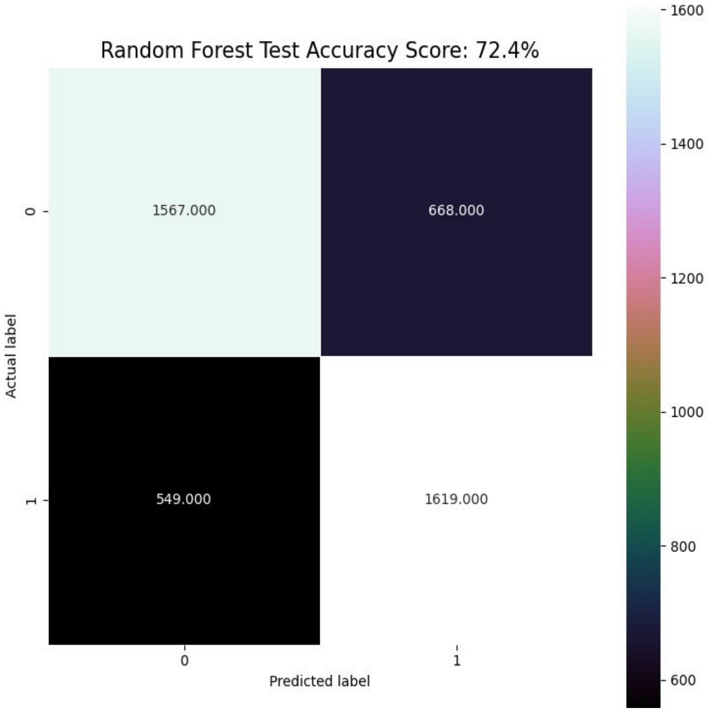
Title Word Count

Text Word Count

Text Emotion

Model 3 : Random Forest Result





TPR Train :	**0.7530120481927711**
TNR Train :	**0.7078088448256656**
FPR Train :	**0.2921911551743344**
FNR Train :	**0.2469879518072289**

TPR Test :	**0.7467712177121771**
TNR Test :	**0.7011185682326622**
FPR Test :	**0.29888143176733784**
FNR Test :	**0.25322878228782286**

Model 3 : Random Forest

Tuning of hyperparameters using Cross-Validation

```
param_grid = {'n_estimators': np.arange(100,1001,100),
              'max_depth': np.arange(2, 11)}

hpGrid = GridSearchCV(RandomForestClassifier(),
                      param_grid,
                      cv = 5,
                      scoring = 'accuracy')

hpGrid.fit(x_train, y_train.label_translated.ravel())
```

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'max_depth': array([ 2,  3,  4,  5,  6,  7,  8,  9, 10]),
                         'n_estimators': array([ 100,  200,  300,  400,  500,  600,  700,  800,  900, 1000])},
             scoring='accuracy')
```

```
print(hpGrid.best_estimator_)
print(np.abs(hpGrid.best_score_))
```

```
RandomForestClassifier(max_depth=9, n_estimators=700)
0.7404164451112744
```

The best accuracy we can obtain from this model is 74% with max_depth of 9 and 700 estimators.

Model 4 : Support Vector Machine Classifier

```
svmclf = svm.SVC(kernel='linear')
```

Response and Predictors

```
y = pd.DataFrame(news_final['label_translated'])
x = pd.DataFrame(news_final.drop('label_translated', axis = 1))
```

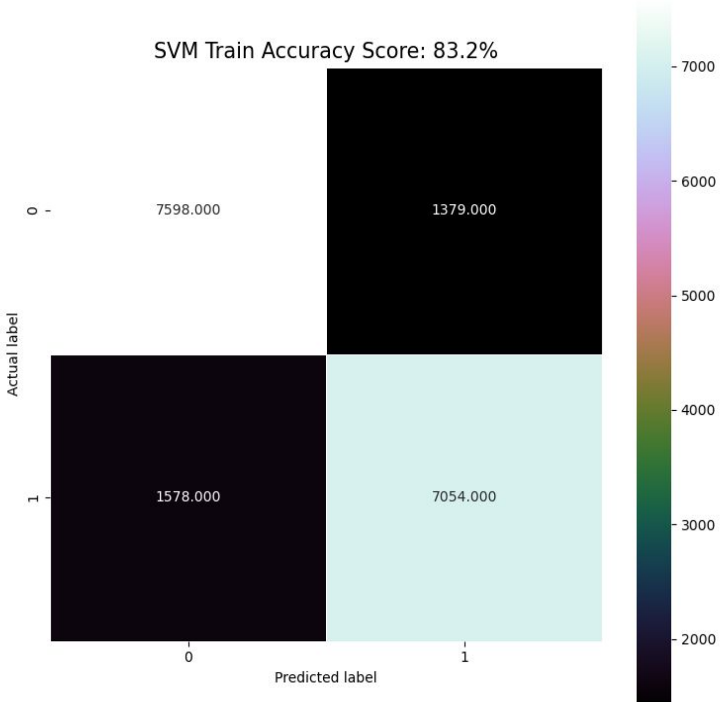
label_translated

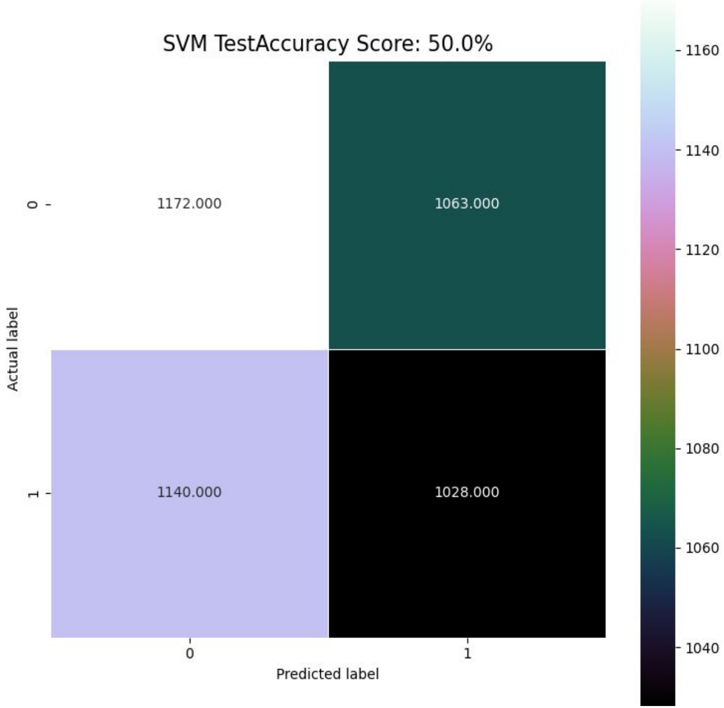
Title Word Count

Text Word Count

Text Emotion

Model 4 : Support Vector Machine Classifier Result





TPR Train : 0.817191844300278
TNR Train : 0.846385206639189

FPR Train : 0.15361479336081096
FNR Train : 0.18280815569972197

TPR Test : 0.474169741697417
TNR Test : 0.5243847874720358

FPR Test : 0.4756152125279642
FNR Test : 0.525830258302583

Model 4 : Support Vector Machine Classifier Result Analysis

What have we done?

Testing Kernel Function

Linear SVM

```
svmclf = svm.SVC(kernel='linear')
```

Train: 83%, Test :50%

Non-Linear SVM

```
svmclf = svm.SVC(kernel='rbf')
```

Train: 97%, Test :49%

```
svmclf = svm.SVC(kernel='poly')
```

Train: 98%, Test :49%

```
svmclf = SVC(kernel='sigmoid')
```

Train: 73%, Test :49%

Tuning hyperparameters

Parameters : gamma, coef0, C

```
svmclf = SVC(kernel='sigmoid', gamma=1, coef0=1)
```

Train: 61%, Test :49%

```
svmclf = svm.SVC(kernel='rbf', gamma=0.5, coef0=1)
```

Train: 88%, Test :49%

```
svmclf = svm.SVC(kernel='rbf', C=1)
```

Train: 97%, Test :49%

Model 4 : Support Vector Machine Classifier Result Analysis

Train Data
Accuracy : 0.8320745073541939

TPR Train : 0.817191844300278
TNR Train : 0.846385206639189

FPR Train : 0.15361479336081096
FNR Train : 0.18280815569972197

Big Gap!

Test Data
Accuracy : 0.49965932318873496

TPR Test : 0.474169741697417
TNR Test : 0.5243847874720358

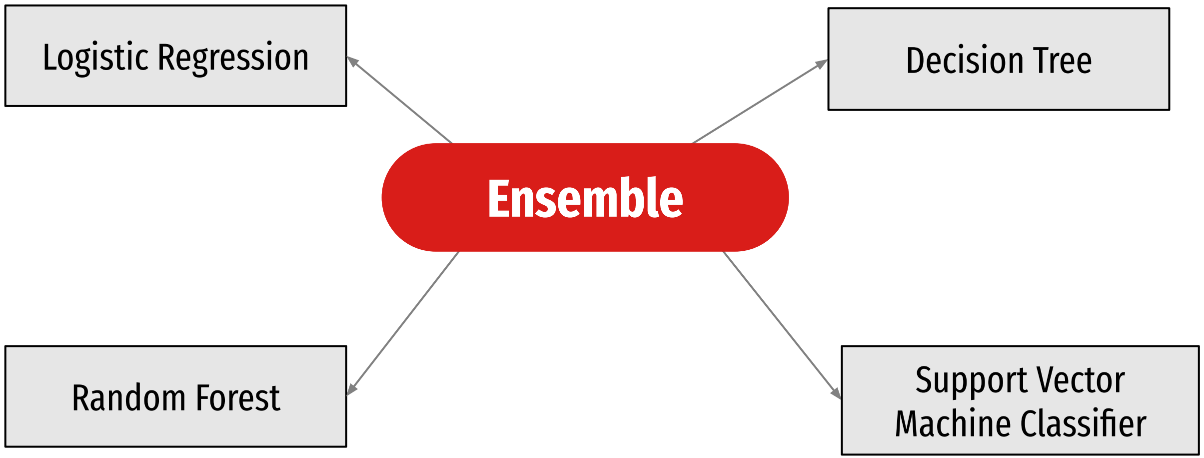
FPR Test : 0.4756152125279642
FNR Test : 0.525830258302583

While SVM has a high accuracy of 83.2% for the training dataset, the test dataset however plummets to 50%.

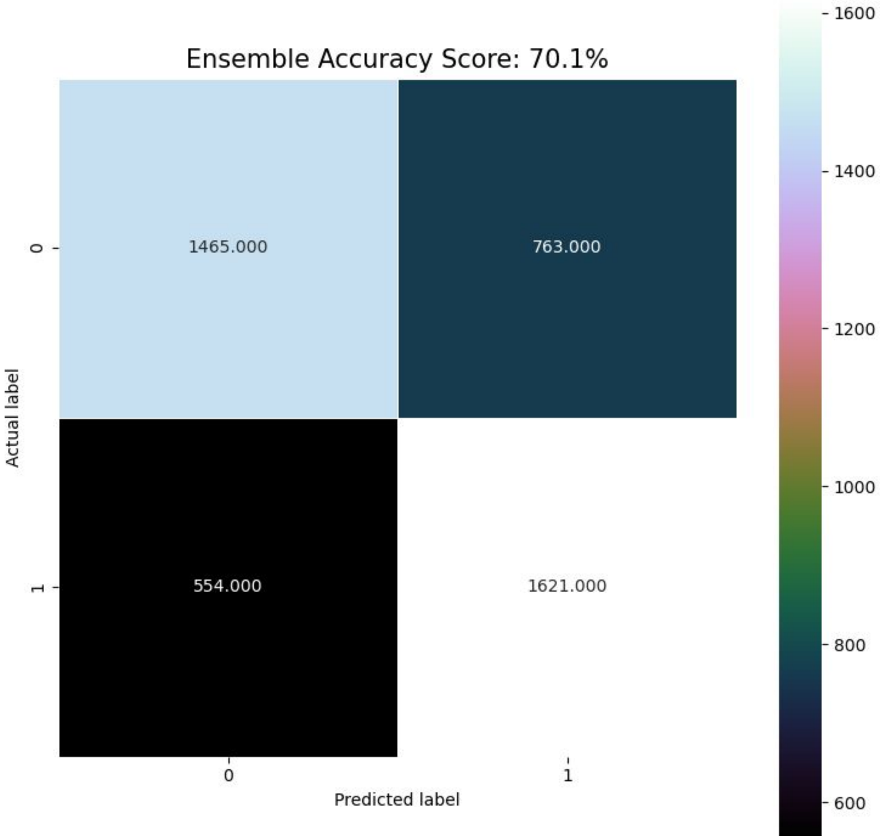
While the high accuracy is a good sign, the big gap of accuracy makes this model not an ideal one.

Model 5 : Ensemble

```
from sklearn.ensemble import VotingClassifier
estimators=[('logreg',logreg), ('clf', clf), ('rforest', rforest), ('svmclf', svmclf)]
ensemble = VotingClassifier(estimators, voting='hard')
```



Model 5: Ensemble Result



While ensemble models commonly helps to improve accuracy by combining multiple models, the model did not work for our dataset with only an accuracy of <u>70%</u>.

Model Accuracy Table

```
accuracy_df = pd.DataFrame(columns = ['Model', 'Accuracy', ' True Pos', 'False Pos', 'True Neg', 'False Neg'])
def update_accuracy(cm, acc, model):
    df = [1,2,3,4,5,6]
    FP = float(cm[0][1])
    TP = float(cm[1][1])
    FN = float(cm[1][0])
    TN = float(cm[0][0])
    TPR = (TP/(TP+FP))*100
    FPR = 100 - TPR
    TNR = (TN/(TN+FN))*100
    FNR = 100 - TNR
    df[0] = model
    df[1] = round(float(acc*100),1)
    df[2] = round(float(TPR),1)
    df[3] = round(float(FPR),1)
    df[4] = round(float(TNR),1)
    df[5] = round(float(FNR),1)
    return df
```

BEST

INACCURATE

WORST

	Model	Accuracy	True Pos	False Pos	True Neg	False Neg
0	Logistic-Regression Train	96.1	96.0	4.0	96.2	3.8
1	Logistic-Regression Test	93.3	92.6	7.4	94.0	6.0
2	Decision Tree Train	71.6	67.6	32.4	77.3	22.7
3	Decision Tree Test	70.9	67.5	32.5	75.6	24.4
4	Random Forest Train	73.4	71.5	28.5	75.4	24.6
5	Random Forest Test	72.7	71.7	28.3	73.9	26.1
6	SVM Train	97.4	97.9	2.1	96.9	3.1
7	SVM Test	50.0	49.3	50.7	50.6	49.4
8	Ensemble	70.1	68.0	32.0	72.6	27.4

Conclusion

Outcome

1 — **Logistic Regression performed the best**

Interpretability, robustness to noise and ability to capture non-linear relationships

2 — **Title_word_count is the best indicator**

3 — **Gap seen in SVM is probably caused of overfitting**

Problem requires **high accuracy on new unseen data**

1. Insufficient data
2. Skewed and improper data preprocessing

4 — **No model is guaranteed to always perform well**

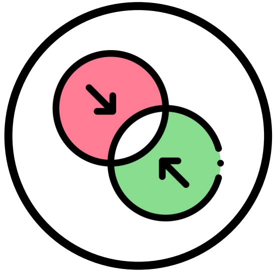
Effectiveness depends on many factors

Key takeaways we learnt

1

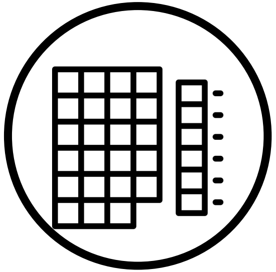
Data Cleaning and Preparation are crucial

2



Possible to <u>combine</u> other machine learning models = new model

3



Correlation can involve categorical data using Label Encoding

4

Loops and functions save time and reduce errors

Data-Driven Insights and Recommendations

Other relevant predators — Source of news, credibility of author, date of publishcation — 1

Sampling techniques — Fake news : Real news proportion is high — 2

Socio-cultural context of dataset — Current dataset - very western based, may not suit asian — 3

Conclusion

Fake news is constantly evolving

Continually monitor and update models to ensure effectiveness in fake news prediction

Be careful of what you believe and share!