In [1]:
```
import numpy as np # linear algebra
import pandas as pd
```

In [2]:
```
# dowload dataset
data1= pd.read_csv('twitter/shorttextpreprocessedtest.csv')
data2= pd.read_csv('twitter/shorttextpreprocessedtrain.csv')
data3= pd.read_csv('twitter/newdatasetwithcoviddata.csv')
```

In [3]:
```
data=pd.concat([data1,data2,data3]).reset_index(drop=True)
```

In [4]:
```
data.shape
```

Out[4]:
```
(193604, 2)
```

In [5]:
```
data.head()
```

Out[5]:

	text	label
0	torrance named europe s fifth ryder cup vice c...	0
1	i have never asked for a single earmark pork b...	0
2	hitting the media center to recap strong debat...	0
3	creflo dollar needed a million gulfstream g to...	0
4	NaN	0

In [6]:
```
## Data Preparation
data = data[data['text'].notna()]
```

In [7]:
```
import matplotlib.pyplot as plt
import seaborn as sns

import nltk
nltk.download('stopwords')
nltk.download('wordnet')
```

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

Out[7]:
```
True
```

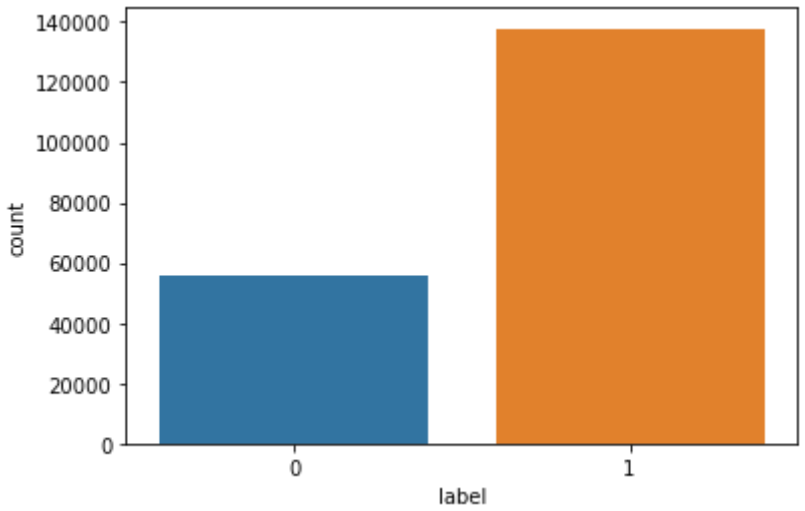
In [8]:
```
# Let's do some statistics of the text columns
txt_len = data.text.str.split().str.len()
txt_len.describe()
```

Out[8]:

```
count    193445.000000
mean         14.685156
std           5.930034
```

In [9]:
```
# Class Distribution
# 1: Unreliable
# 2: Reliable
sns.countplot(x='label', data= data)
```

Out[9]:
```
<AxesSubplot:xlabel='label', ylabel='count'>
```



In [10]:
```
print(data.label.value_counts())
print()
print(round(data.label.value_counts(normalize=True),2)*100)
```

```
1    137796
0     55649
Name: label, dtype: int64

1    71.0
0    29.0
Name: label, dtype: float64
```

In [11]:
```
data.isnull().sum()
```

Out[11]:
```
text     0
label    0
dtype: int64
```

In [12]:
```
column_n = [ 'text', 'label']

categorical_features = []
target_col = ['label']
text_f = ['text']
```

In [13]:
```
# cleaning
import nltk
from nltk.corpus import stopwords
import re
from nltk.stem.porter import PorterStemmer
from collections import Counter
```

```
ps = PorterStemmer()
wnl = nltk.stem.WordNetLemmatizer()

stop_words = stopwords.words('english')
stopwords_dict = Counter(stop_words)


# impute null values with none
def null_process(feature_df):
    for col in text_f:
        feature_df.loc[feature_df[col].isnull(),col] = "None"
    return feature_df

# clean_data
def clean_dataset(df):
    #impute null value
    df = null_process(df)

    return df

# Cleaning text from unused characters
def clean_text(text):
    text = str(text).replace(r'http[\w:/\.]+', ' ')  # removing urls
    text = str(text).replace(r'[^\.\w\s]', ' ')  # remove everything
    text = str(text).replace('[^a-zA-Z]', ' ')
    text = str(text).replace(r'\s\s+', ' ')
    text = text.lower().strip()
    #text = ' '.join(text)
    return text

## Nltk Preprocessing include:
# Stop words, Stemming and Lemmetization
# For our project we use only Stop word removal
def nltk_preprocess(text):
    text = clean_text(text)
    wordlist = re.sub(r'[^\w\s]', '', text).split()
    text = ' '.join([wnl.lemmatize(word) for word in wordlist if word
    return  text
```

In [14]:
```
df = clean_dataset(data)
df['text'] = df.text.apply(nltk_preprocess)
```

In [15]:
```
df.head()
```

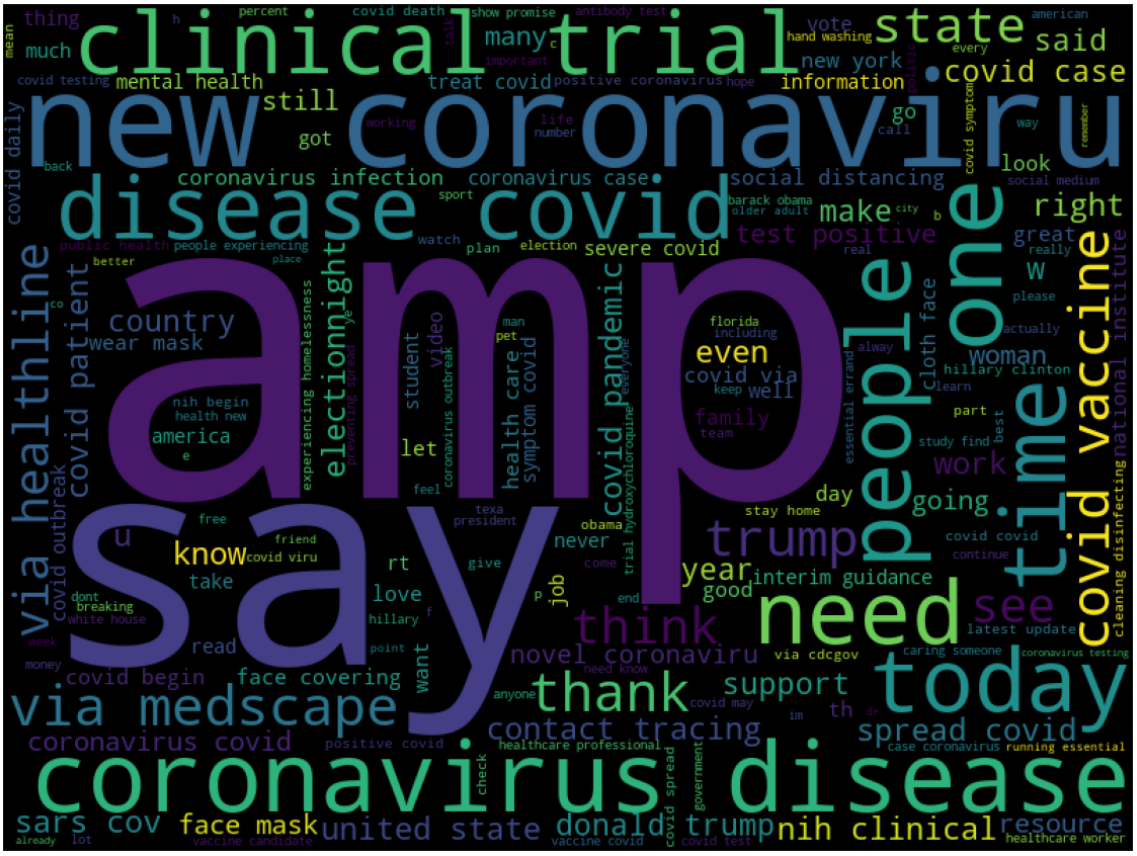
Out[15]:

	text	label
0	torrance named europe fifth ryder cup vice cap...	0
1	never asked single earmark pork barrel project...	0
2	hitting medium center recap strong debate perf...	0
3	creflo dollar needed million gulfstream g carr...	0
5	wednesday morning meal trump win electionday	0

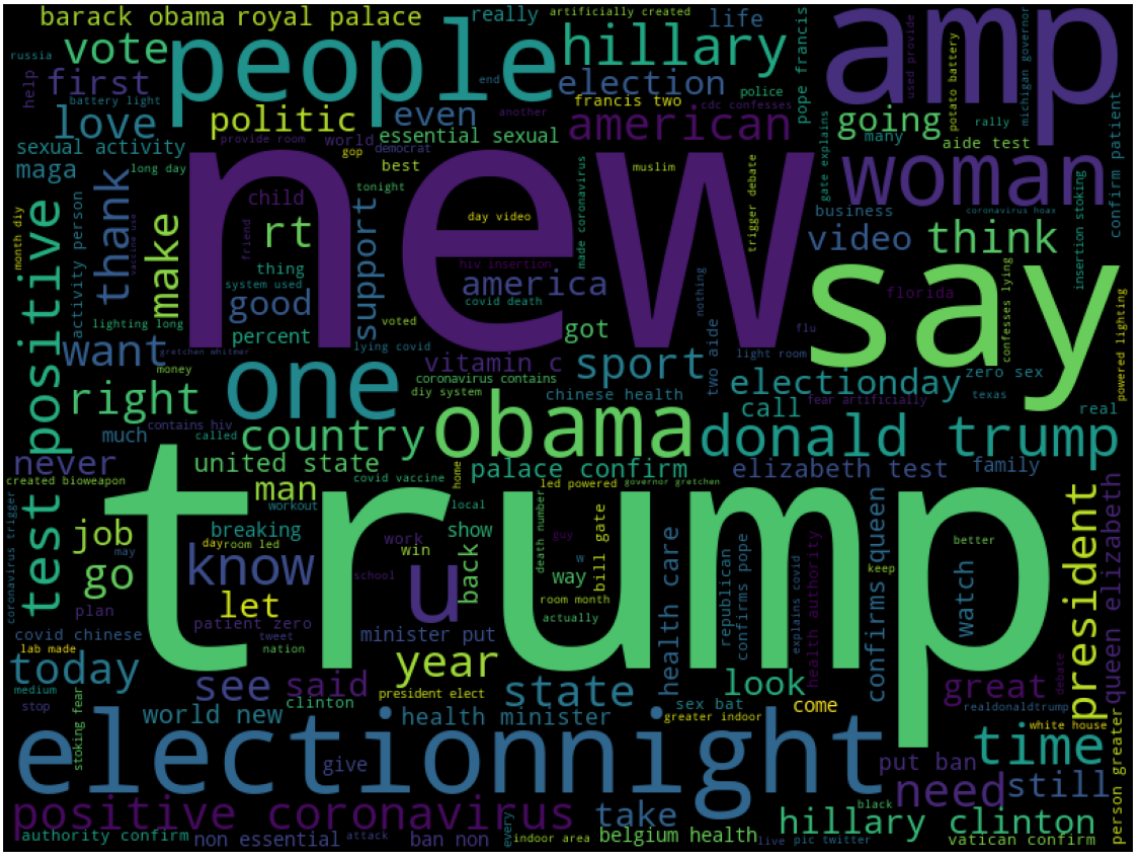
In [16]:
```
from wordcloud import WordCloud, STOPWORDS

# initialize the word cloud
wordcloud = WordCloud(background_color='black', width=800, height=60
```

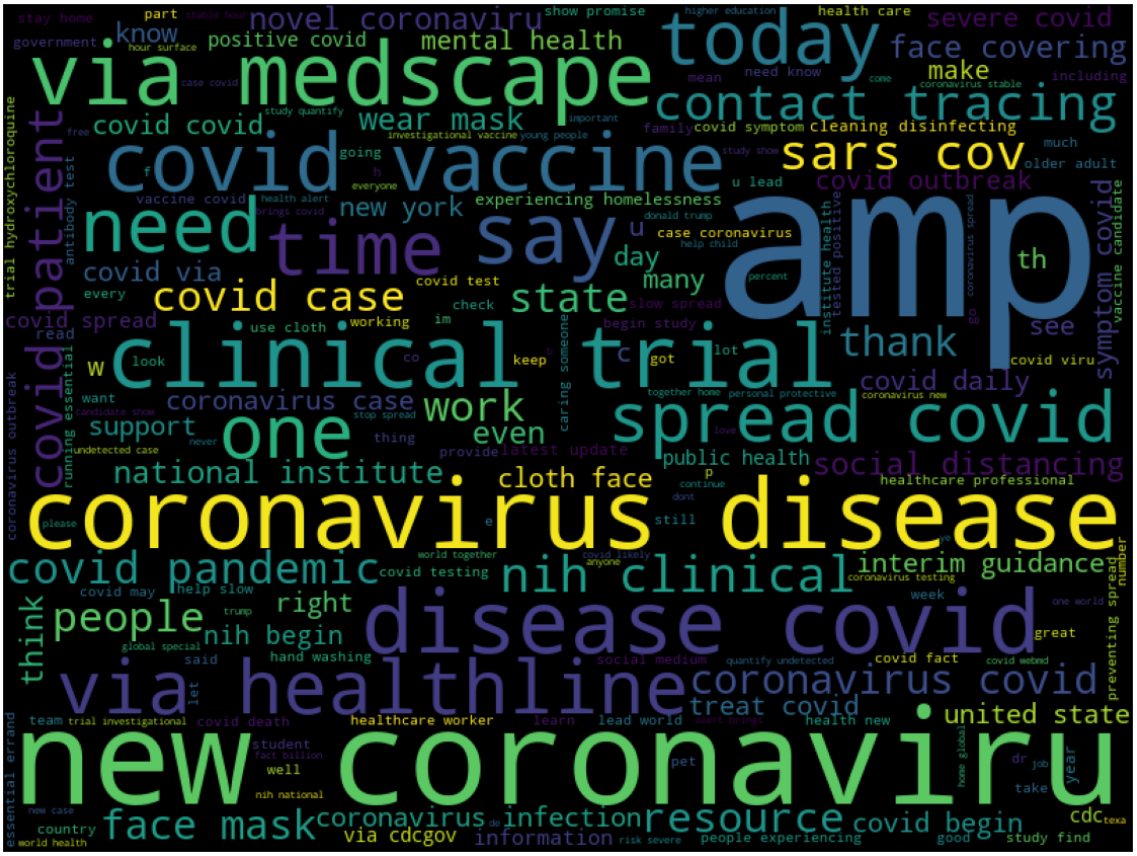
```
# generate the word cloud
text_cloud = wordcloud.generate(" ".join(df['text']))
# plotting the word cloud
plt.figure(figsize=(20,30))
plt.imshow(text_cloud)
plt.axis('off')
plt.show()
```



In [17]:
```
# reliable news (0)
reliable_news = " ".join(df[df['label']==0]['text'])
wc = wordcloud.generate(reliable_news)
plt.figure(figsize=(20,30))
plt.imshow(wc)
plt.axis('off')
plt.show()
```



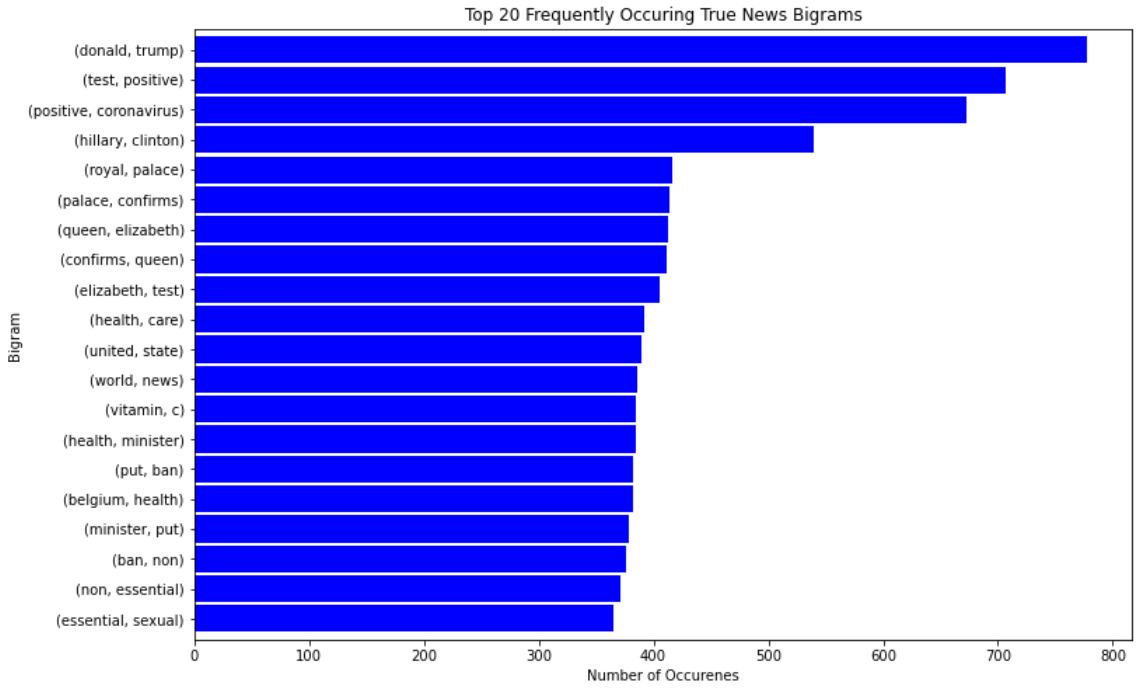
In [18]:
```
# unreliable news (1)
unreliable_news  = ' '.join(df[df['label']==1]['text'])
wc= wordcloud.generate(unreliable_news)
plt.figure(figsize=(20,30))
plt.imshow(wc)
plt.axis('off')
plt.show()
```



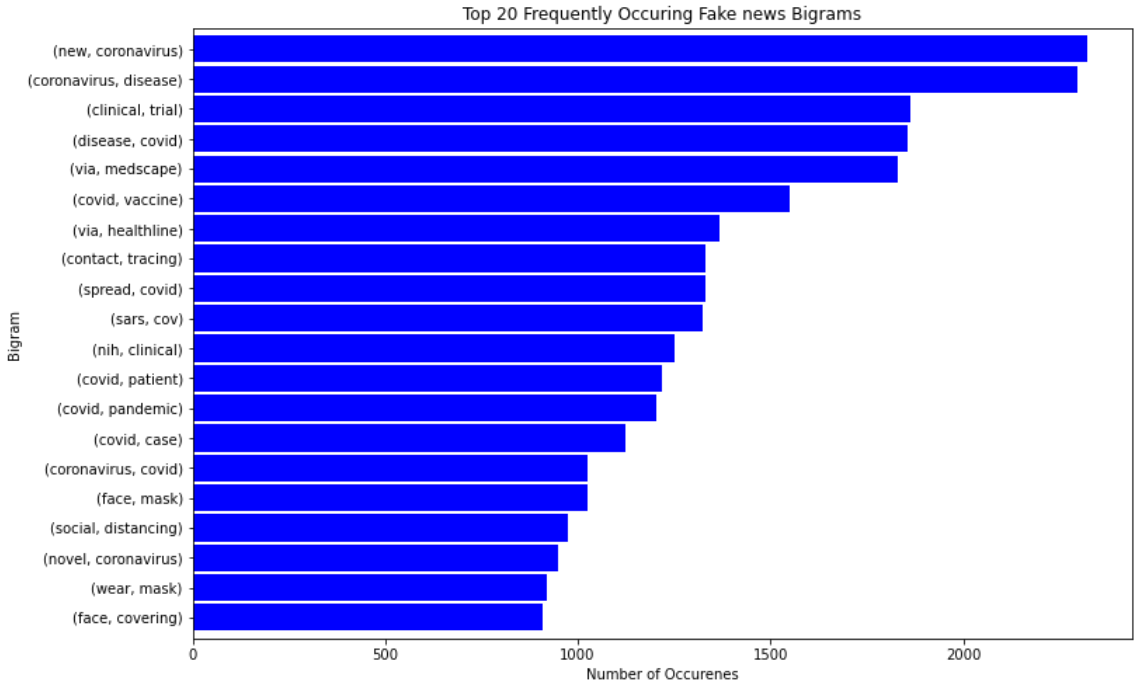
In [19]:
```
# Bigram

def plot_top_ngrams(corpus, title, ylabel, xlabel="Number of Occurene
    true_b = (pd.Series(nltk.ngrams(corpus.split(), n)).value_counts(
    true_b.sort_values().plot.barh(color='blue', width=.9, figsize=(1
    plt.title(title)
    plt.ylabel(ylabel)
    plt.xlabel(xlabel)
    plt.show()


plot_top_ngrams(reliable_news, "Top 20 Frequently Occuring True News
```

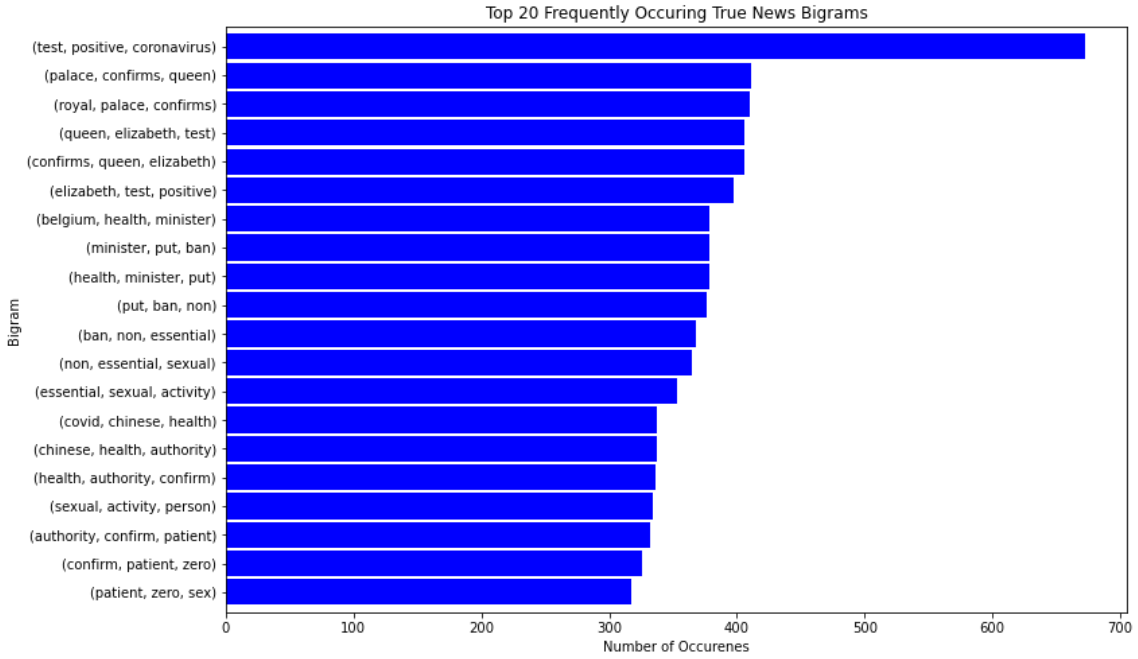


In [20]:
```
plot_top_ngrams(unreliable_news, 'Top 20 Frequently Occuring Fake new
```

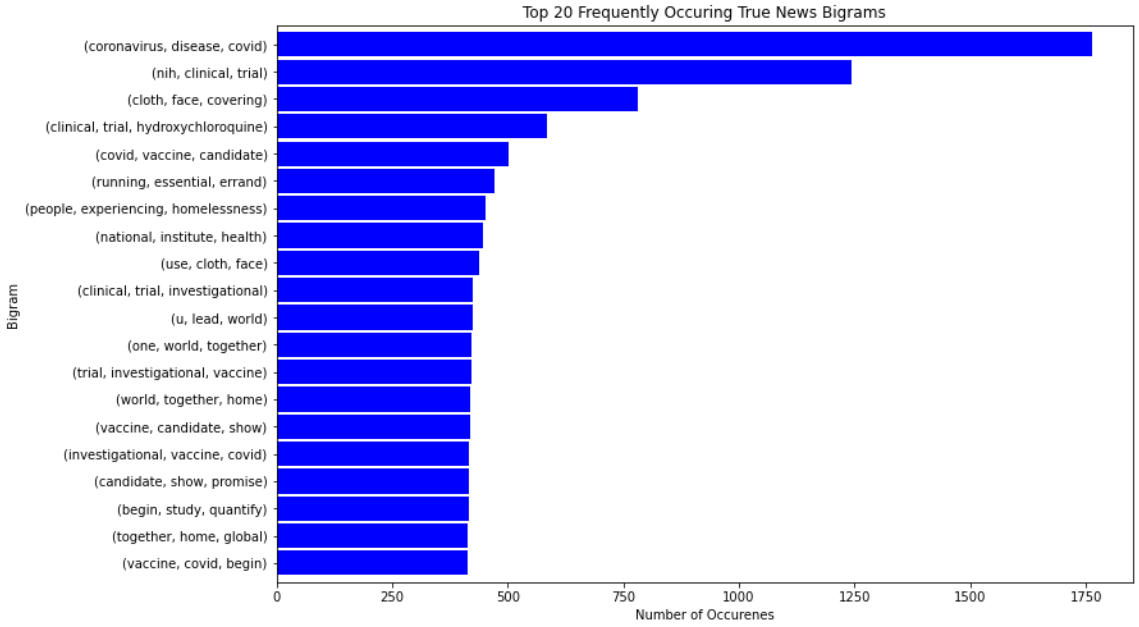


In [21]:
```
# Trigram
```

```
plot_top_ngrams(reliable_news, "Top 20 Frequently Occuring True News
```



In [22]:
```
plot_top_ngrams(unreliable_news, "Top 20 Frequently Occuring True Nev
```



In [23]:
```
!pip install transformers
```

```
Requirement already satisfied: transformers in /home/administrator/
anaconda3/lib/python3.9/site-packages (4.18.0)
Requirement already satisfied: regex!=2019.12.17 in /home/administr
ator/anaconda3/lib/python3.9/site-packages (from transformers) (202
1.8.3)
Requirement already satisfied: numpy>=1.17 in /home/administrator/a
naconda3/lib/python3.9/site-packages (from transformers) (1.20.3)
Requirement already satisfied: requests in /home/administrator/anac
onda3/lib/python3.9/site-packages (from transformers) (2.26.0)
Requirement already satisfied: pyyaml>=5.1 in /home/administrator/a
naconda3/lib/python3.9/site-packages (from transformers) (6.0)
Requirement already satisfied: sacremoses in /home/administrator/an
aconda3/lib/python3.9/site-packages (from transformers) (0.0.49)
Requirement already satisfied: filelock in /home/administrator/anac
onda3/lib/python3.9/site-packages (from transformers) (3.3.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.1.0 in /home
/administrator/anaconda3/lib/python3.9/site-packages (from transfor
mers) (0.5.1)
Requirement already satisfied: tqdm>=4.27 in /home/administrator/an
aconda3/lib/python3.9/site-packages (from transformers) (4.62.3)
Requirement already satisfied: packaging>=20.0 in /home/administrat
or/anaconda3/lib/python3.9/site-packages (from transformers) (21.0)
Requirement already satisfied: tokenizers!=0.11.3,<0.13,>=0.11.1 in
/home/administrator/anaconda3/lib/python3.9/site-packages (from tra
nsformers) (0.12.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /home/
administrator/anaconda3/lib/python3.9/site-packages (from huggingfa
ce-hub<1.0,>=0.1.0->transformers) (3.10.0.2)
Requirement already satisfied: pyparsing>=2.0.2 in /home/administra
tor/anaconda3/lib/python3.9/site-packages (from packaging>=20.0->tr
ansformers) (3.0.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /home/admin
istrator/anaconda3/lib/python3.9/site-packages (from requests->tran
sformers) (1.26.7)
Requirement already satisfied: certifi>=2017.4.17 in /home/administ
rator/anaconda3/lib/python3.9/site-packages (from requests->transfo
rmers) (2021.10.8)
```

In [24]:
```
import torch
from transformers.file_utils import is_tf_available, is_torch_availab
from transformers import BertTokenizerFast, BertForSequenceClassifica
from transformers import Trainer, TrainingArguments
from sklearn.model_selection import train_test_split
import random
```

In [25]:
```
def set_seed(seed: int):
    """
    Helper function for reproducible behavior to set the seed in ``ra
    installed).

    Args:
        seed (:obj:`int`): The seed to set.
    """
    random.seed(seed)
    np.random.seed(seed)
    if is_torch_available():
        torch.manual_seed(seed)
        torch.cuda.manual_seed_all(seed)
        # ^^ safe to call this function even if cuda is not available
```

```
    if is_tf_available():
        import tensorflow as tf

        tf.random.set_seed(seed)

set_seed(123)
```

In [26]:
```
model_name = "bert-base-uncased"
max_length= 512
```

In [27]:
```
tokenizer = BertTokenizerFast.from_pretrained(model_name, do_lower_ca
```

In [28]:
```
data.head()
```

Out[28]:

	text	label
0	torrance named europe fifth ryder cup vice cap...	0
1	never asked single earmark pork barrel project...	0
2	hitting medium center recap strong debate perf...	0
3	creflo dollar needed million gulfstream g carr...	0
5	wednesday morning meal trump win electionday	0

In [29]:
```
## Data Preparation
data = data[data['text'].notna()]
```

In [30]:
```
def prepare_data(df, test_size=0.2, include_title=True, include_autho
    texts = []
    labels = []

    for i in range(len(df)):
        text = df['text'].iloc[i]
        label = df['label'].iloc[i]

        if text and label in [0,1]:
            texts.append(text)
            labels.append(label)

    return train_test_split(texts, labels, test_size=test_size)

train_texts, valid_texts, train_labels, valid_labels = prepare_data(d
```

In [31]:
```
print(len(train_texts), len(train_labels))
print(len(valid_texts), len(valid_labels))
```

```
154673 154673
38669 38669
```

In [32]:
```
# tokenizing the dataset
train_encodings = tokenizer(train_texts, truncation=True, padding=Tru
valid_encodings = tokenizer(valid_texts, truncation=True, padding=Tru
```

In [33]:
```
# converting the encoding into a PyTorch datset
class NewsGroupsDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
```

```
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {k: torch.tensor(v[idx]) for k, v in self.encodings.it
        item['labels'] = torch.tensor([self.labels[idx]])
        return item

    def __len__(self):
        return len(self.labels)

# convert tokenize data into torch dataset
train_dataset = NewsGroupsDataset(train_encodings, train_labels)
valid_dataset = NewsGroupsDataset(valid_encodings, valid_labels)
```

In [34]:
```
model = BertForSequenceClassification.from_pretrained(model_name, nun
```

```
Some weights of the model checkpoint at bert-base-uncased were not
used when initializing BertForSequenceClassification: ['cls.predict
ions.transform.dense.weight', 'cls.predictions.transform.LayerNorm.
weight', 'cls.seq_relationship.bias', 'cls.seq_relationship.weight
', 'cls.predictions.decoder.weight', 'cls.predictions.bias', 'cls.p
redictions.transform.dense.bias', 'cls.predictions.transform.LayerN
orm.bias']
- This IS expected if you are initializing BertForSequenceClassific
ation from the checkpoint of a model trained on another task or wit
h another architecture (e.g. initializing a BertForSequenceClassifi
cation model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForSequenceClass
ification from the checkpoint of a model that you expect to be exac
tly identical (initializing a BertForSequenceClassification model f
rom a BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized
from the model checkpoint at bert-base-uncased and are newly initia
lized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be ab
le to use it for predictions and inference.
```

In [42]:
```
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score
def computer_metrics(pred):
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)
    precision, recall, f1, _ = precision_recall_fscore_support(labels
    acc = accuracy_score(labels, preds)
    return {
        'accuracy': acc,
        'f1': f1,
        'precision': precision,
        'recall': recall
    }
```

In [43]:
```
training_args = TrainingArguments(
    output_dir='./results',          # output directory
    num_train_epochs=1,              # total number of training epoch
    per_device_train_batch_size=10,  # batch size per device during t
    per_device_eval_batch_size=20,   # batch size for evaluation
    warmup_steps=100,                # number of warmup steps for lea
```

```
    logging_dir='./logs',            # directory for storing logs
    load_best_model_at_end=True,     # load the best model when finis
    # but you can specify `metric_for_best_model` argument to change
    logging_steps=200,               # log & save weights each loggi
    save_steps=200,
    evaluation_strategy="steps",     # evaluate each `logging_steps`
)
```

```
using `logging_steps` to initialize `eval_steps` to 200
PyTorch: setting up devices
The default value for the training argument `--report_to` will chan
ge in v5 (from all installed integrations to none). In v5, you will
need to use `--report_to all` to get the same behavior as now. You
should start updating your code and make this info disappear :-).
```

In [44]:
```
trainer = Trainer(
    model = model,
    args = training_args,
    train_dataset=train_dataset,
    eval_dataset=valid_dataset,
    compute_metrics=computer_metrics,
)
```

In [45]:
```
trainer.train()
```

```
/home/administrator/anaconda3/lib/python3.9/site-packages/transform
ers/optimization.py:306: FutureWarning: This implementation of Adam
W is deprecated and will be removed in a future version. Use the Py
Torch implementation torch.optim.AdamW instead, or set `no_deprecat
ion_warning=True` to disable this warning
  warnings.warn(
***** Running training *****
  Num examples = 154673
  Num Epochs = 1
  Instantaneous batch size per device = 10
  Total train batch size (w. parallel, distributed & accumulation)
= 20
  Gradient Accumulation steps = 1
  Total optimization steps = 7734
/home/administrator/anaconda3/lib/python3.9/site-packages/torch/nn/
parallel/_functions.py:68: UserWarning: Was asked to gather along d
imension 0, but all input tensors were scalars; will instead unsque
eze and return a vector.
  warnings.warn('Was asked to gather along dimension 0, but all '
```

In [48]:
```
# evaluate the current model after training
trainer.evaluate()
```

```
***** Running Evaluation *****
  Num examples = 38669
  Batch size = 20
```

[967/967 01:38]

```
Attempted to log scalar metric eval_loss:
0.13251514732837677
Attempted to log scalar metric eval_accuracy:
0.9485893092658202
Attempted to log scalar metric eval_f1:
0.9639672297542231
Attempted to log scalar metric eval_precision:
0.9595842956120092
Attempted to log scalar metric eval_recall:
0.9683903860160233
Attempted to log scalar metric eval_runtime:
98.2727
```

Out[48]:
```
{'eval_loss': 0.13251514732837677,
 'eval_accuracy': 0.9485893092658202,
 'eval_f1': 0.9639672297542231,
 'eval_precision': 0.9595842956120092,
 'eval_recall': 0.9683903860160233,
 'eval_runtime': 98.2727,
 'eval_samples_per_second': 393.487,
 'eval_steps_per_second': 9.84,
 'epoch': 1.0}
```

In [61]:
```
# saving the fine tuned model & tokenizer
model_path = "fake-news-bert-base-uncased"
model.save_pretrained(model_path)
tokenizer.save_pretrained(model_path)
```

```
Configuration saved in fake-news-bert-base-uncased/config.json
Model weights saved in fake-news-bert-base-uncased/pytorch_model.bi
n
tokenizer config file saved in fake-news-bert-base-uncased/tokenize
r_config.json
Special tokens file saved in fake-news-bert-base-uncased/special_to
kens_map.json
```

Out[61]:
```
('fake-news-bert-base-uncased/tokenizer_config.json',
 'fake-news-bert-base-uncased/special_tokens_map.json',
 'fake-news-bert-base-uncased/vocab.txt',
 'fake-news-bert-base-uncased/added_tokens.json',
 'fake-news-bert-base-uncased/tokenizer.json')
```

In [62]:
```
def get_prediction(text, convert_to_label=False):
    # prepare our text into tokenized sequence
    inputs = tokenizer(text, padding=True, truncation=True, max_lengt
    # perform inference to our model
    outputs = model(**inputs)
    # get output probabilities by doing softmax
    probs = outputs[0].softmax(1)
    # executing argmax function to get the candidate label
    d = {
        0: "reliable",
        1: "fake"
    }
    if convert_to_label:
        return d[int(probs.argmax())]
    else:
        return int(probs.argmax())
```

In [63]:
```
real_news = """
dude seriously the fuck is this you will go down as the most childish
"""

get_prediction(real_news, convert_to_label=True)
```

Out[63]: 'reliable'

In [ ]:

In [ ]: