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

In [2]:
```
# dowload dataset
train_df= pd.read_csv('liar_dataset/train.tsv', delimiter='\t', heade
test_df= pd.read_csv('liar_dataset/test.tsv', delimiter='\t', header
```

In [3]:
```
train_df.drop([0],axis=1,inplace=True)
train_df.drop([3],axis=1,inplace=True)
train_df.drop([4],axis=1,inplace=True)
train_df.drop([5],axis=1,inplace=True)
train_df.drop([6],axis=1,inplace=True)
train_df.drop([7],axis=1,inplace=True)
train_df.drop([8],axis=1,inplace=True)
train_df.drop([9],axis=1,inplace=True)
train_df.drop([10],axis=1,inplace=True)
train_df.drop([11],axis=1,inplace=True)
train_df.drop([12],axis=1,inplace=True)
train_df.drop([13],axis=1,inplace=True)
train_df.head()
```

Out[3]:

	1	2
0	false	Says the Annies List political group supports ...
1	half-true	When did the decline of coal start? It started...
2	mostly-true	Hillary Clinton agrees with John McCain "by vo...
3	false	Health care reform legislation is likely to ma...
4	half-true	The economic turnaround started at the end of ...

In [4]:
```
test_df.drop([0],axis=1,inplace=True)
test_df.drop([3],axis=1,inplace=True)
test_df.drop([4],axis=1,inplace=True)
test_df.drop([5],axis=1,inplace=True)
test_df.drop([6],axis=1,inplace=True)
test_df.drop([7],axis=1,inplace=True)
test_df.drop([8],axis=1,inplace=True)
test_df.drop([9],axis=1,inplace=True)
test_df.drop([10],axis=1,inplace=True)
test_df.drop([11],axis=1,inplace=True)
test_df.drop([12],axis=1,inplace=True)
test_df.drop([13],axis=1,inplace=True)
test_df.head()
```

Out[4]:

	1	2
0	true	Building a wall on the U.S.-Mexico border will...
1	false	Wisconsin is on pace to double the number of l...
2	false	Says John McCain has done nothing to help the ...
3	half-true	Suzanne Bonamici supports a plan that will cut...
4	pants-fire	When asked by a reporter whether hes at the ce...

In [5]:
```
def map_f(x):
    if x=='mostly-true' or x == 'true':
        return 0
    else:
        return 1

train = pd.DataFrame()
valid = pd.DataFrame()
test = pd.DataFrame()

train['text'] = train_df[2]
train['label'] = train_df[1].apply(map_f)



test['text'] = test_df[2]
test['label'] = test_df[1].apply(map_f)
```

In [6]:
```
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[6]: True

In [7]:
```
data=train
data
```

Out[7]:

	text	label
0	Says the Annies List political group supports ...	1
1	When did the decline of coal start? It started...	1
2	Hillary Clinton agrees with John McCain "by vo...	0
3	Health care reform legislation is likely to ma...	1
4	The economic turnaround started at the end of ...	1
...	...	...
10235	There are a larger number of shark attacks in ...	0
10236	Democrats have now become the party of the [At...	0
10237	Says an alternative to Social Security that op...	1
10238	On lifting the U.S. Cuban embargo and allowing...	1
10239	The Department of Veterans Affairs has a manua...	1

10240 rows × 2 columns

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

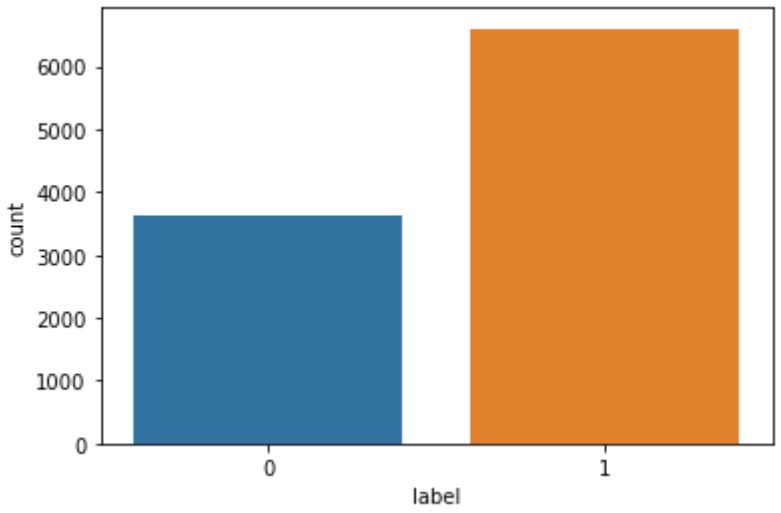
Out[8]:
```
count    10240.000000
mean        18.010059
std          9.658572
min          2.000000
25%         12.000000
50%         17.000000
75%         22.000000
max        467.000000
Name: text, dtype: float64
```

In [9]:
```
print("Shape of dataset ", data.shape)
print("Columns ", data.columns)
```

```
Shape of dataset  (10240, 2)
Columns  Index(['text', 'label'], dtype='object')
```

In [10]:
```
# Class Distribution
# 1: Unreliable
# 2: Reliable
sns.countplot(x='label', data= data)
```

Out[10]:
```
<AxesSubplot:xlabel='label', ylabel='count'>
```



In [11]:
```
print(data.label.value_counts())
print()
print(round(data.label.value_counts(normalize=True),2)*100)
```

```
1    6602
0    3638
Name: label, dtype: int64

1    64.0
0    36.0
Name: label, dtype: float64
```

In [12]:
```
data.isnull().sum()
```

Out[12]:

```
text     0
```

In [13]:
```
categorical_features = []
target_col = ['label']
text_f = [ 'text']
```

In [14]:
```
import tensorflow as tf
with tf.device('/GPU:1'):
    # 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 u
        text = str(text).replace(r'[^\.\w\s]', ' ')  # remove everyth
        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
        return  text
```

```
2022-05-16 14:24:21.331239: I tensorflow/core/platform/cpu_feature_
guard.cc:151] This TensorFlow binary is optimized with oneAPI Deep
Neural Network Library (oneDNN) to use the following CPU instructio
ns in performance-critical operations:  AVX2 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the app
ropriate compiler flags.
2022-05-16 14:24:22.265097: I tensorflow/core/common_runtime/gpu/gp
u_device.cc:1532] Created device /job:localhost/replica:0/task:0/de
vice:GPU:0 with 5196 MB memory:  -> device: 0, name: Tesla V100-PCI
```

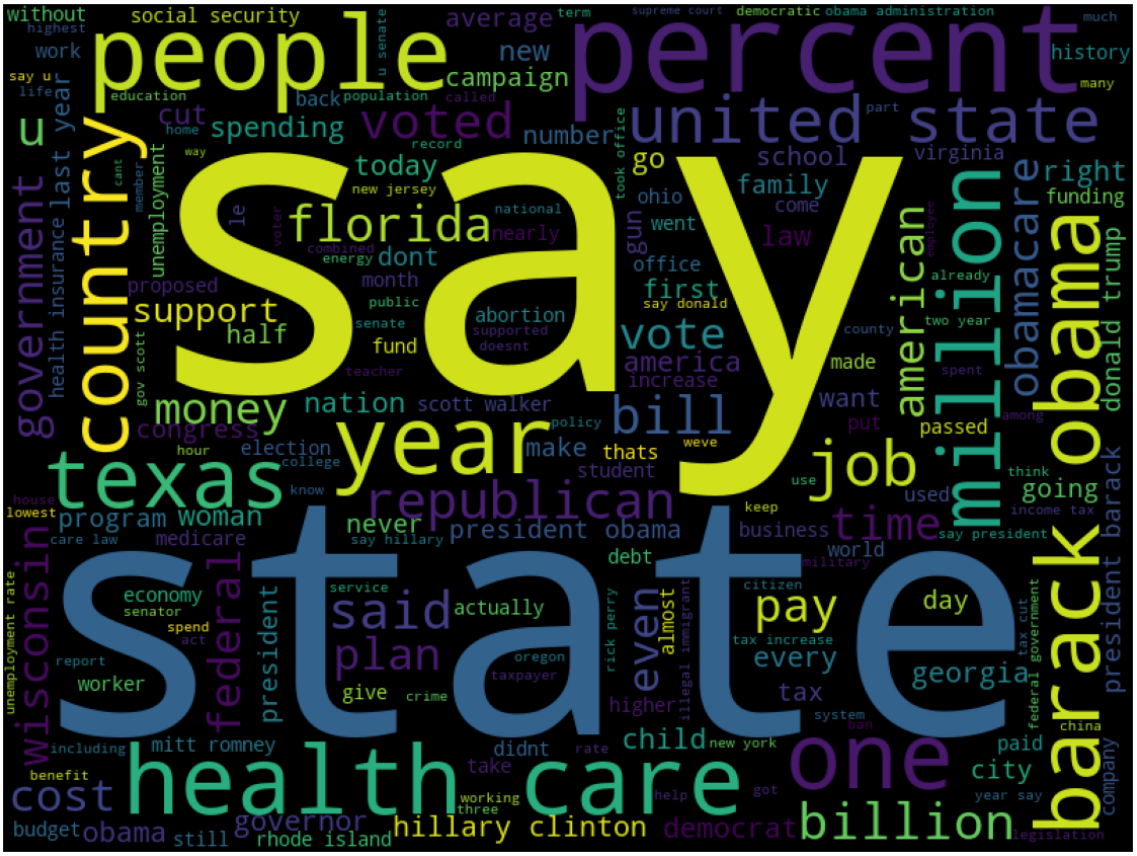
In [15]:
```
with tf.device('GPU:1'):
    df = clean_dataset(data)
    df['text'] = df.text.apply(nltk_preprocess)
```

In [16]:
```
df.head()
```

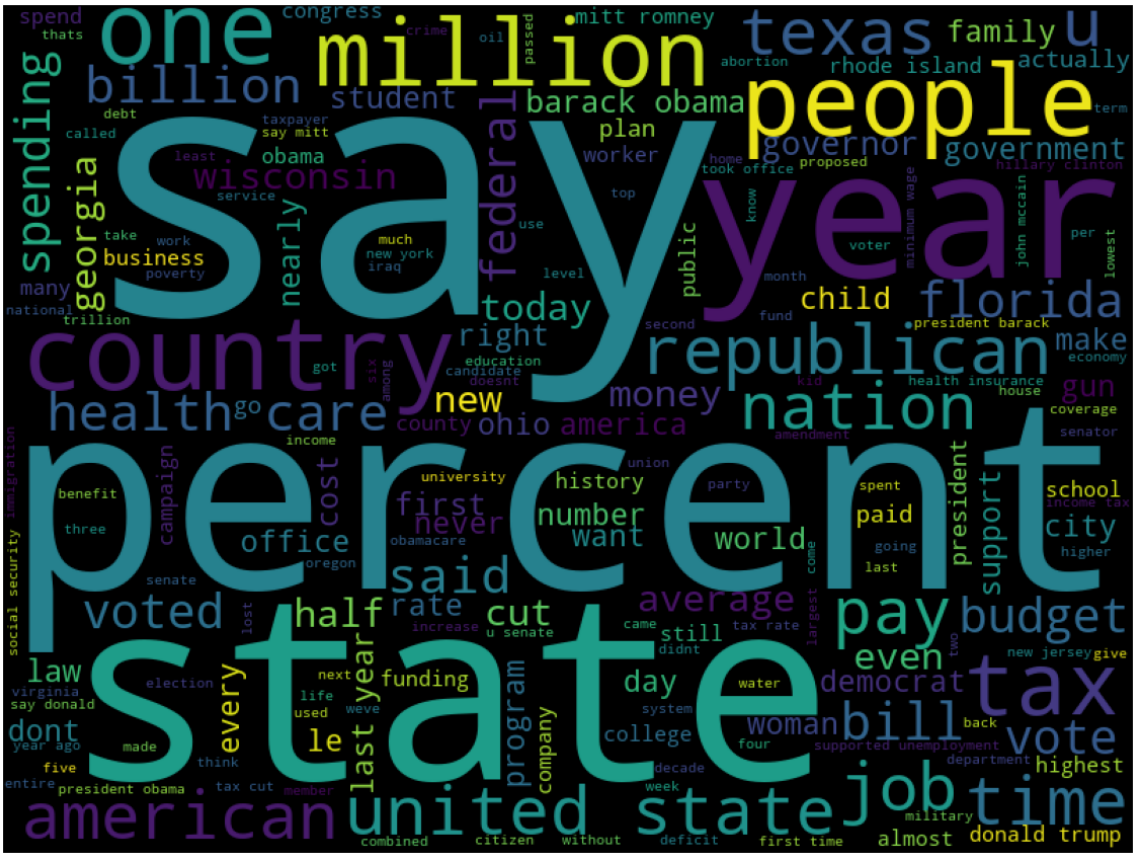
Out[16]:

	text	label
0	say annies list political group support thirdt...	1
1	decline coal start started natural gas took st...	1
2	hillary clinton agrees john mccain voting give...	0
3	health care reform legislation likely mandate ...	1
4	economic turnaround started end term	1

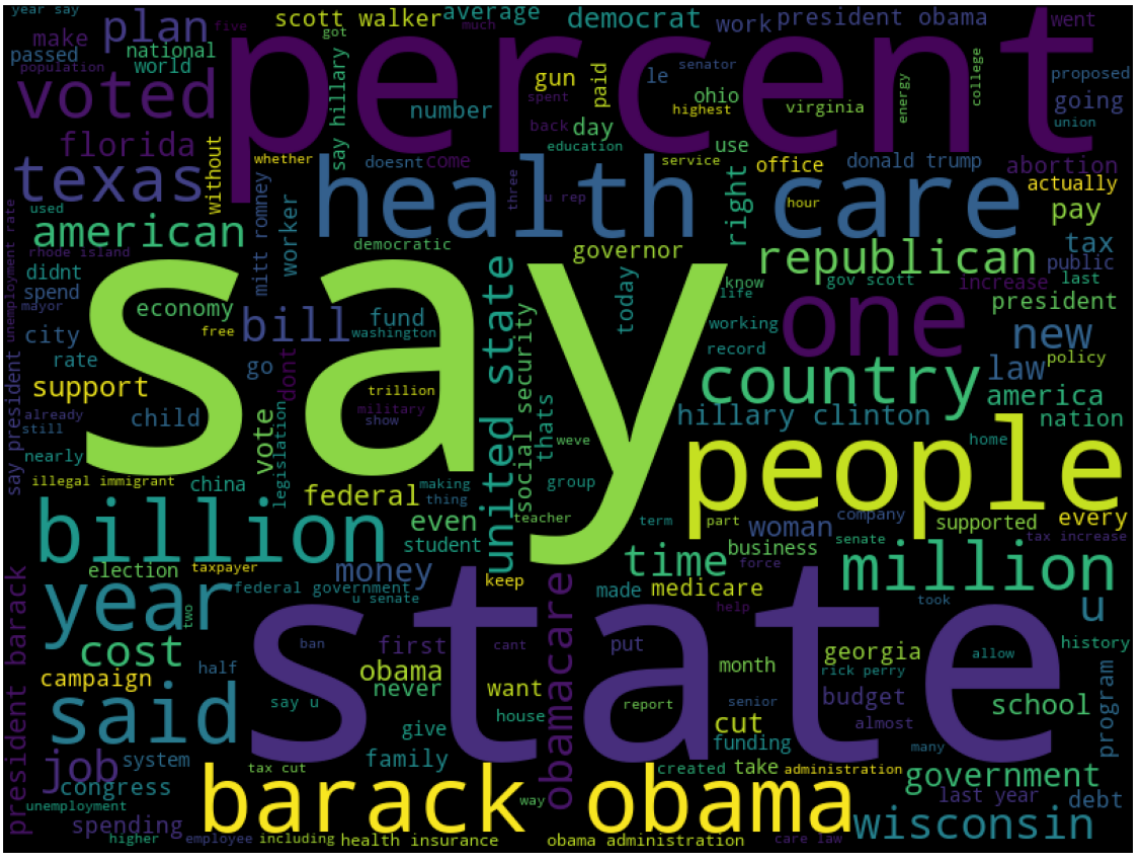
In [17]:
```
with tf.device('GPU:1'):
    from wordcloud import WordCloud, STOPWORDS

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



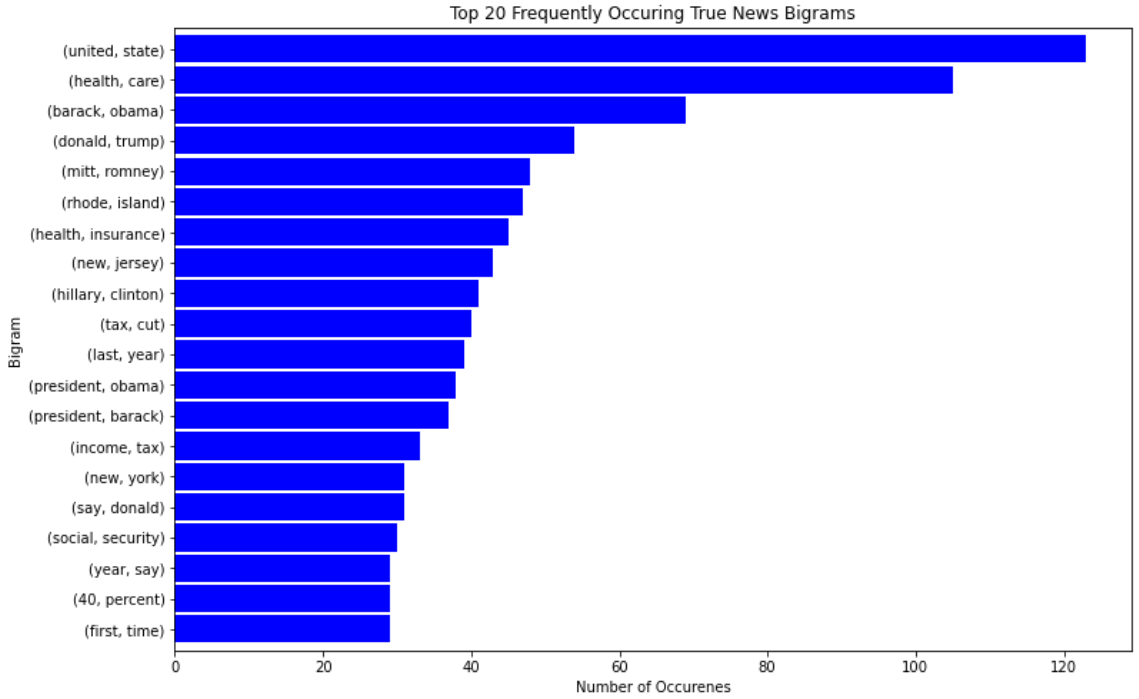
In [18]:
```
with tf.device('GPU:1'):
    # reliable news (0)
    reliable_news = " ".join(df[df['label']==0]['text'])
    wc = wordcloud.generate(reliable_news)
    plt.figure(figsize=(20,30))
    plt.imshow(wc)
    plt.axis('off')
    plt.show()
```



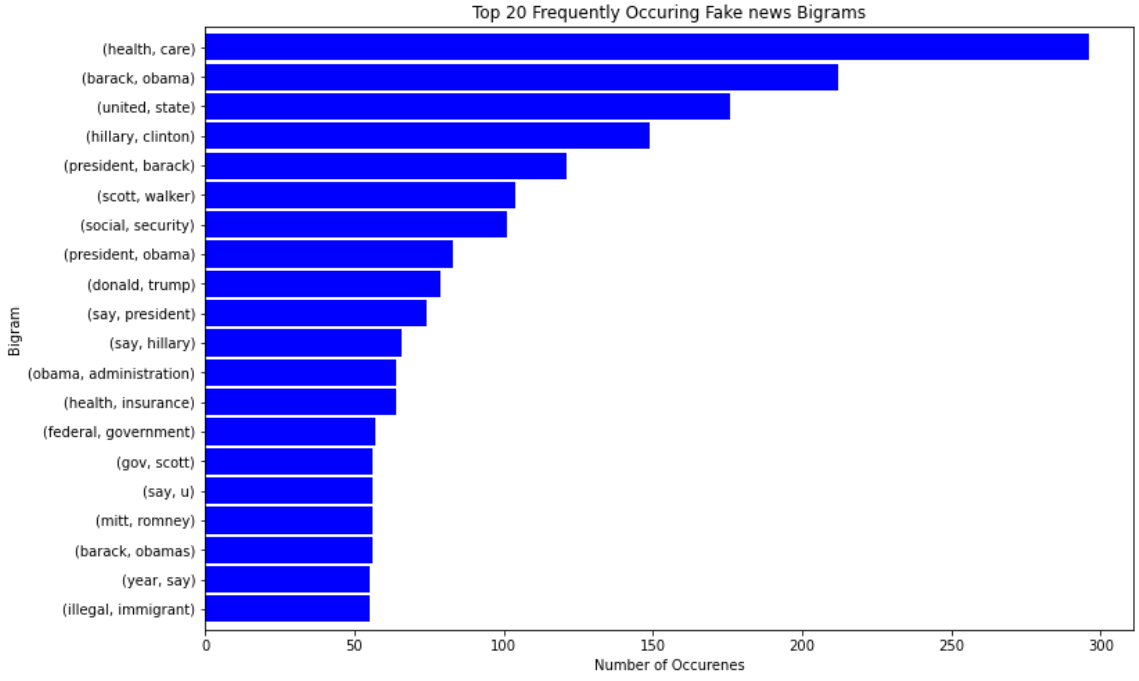
In [19]:
```
with tf.device('GPU:1'):

# 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()
```



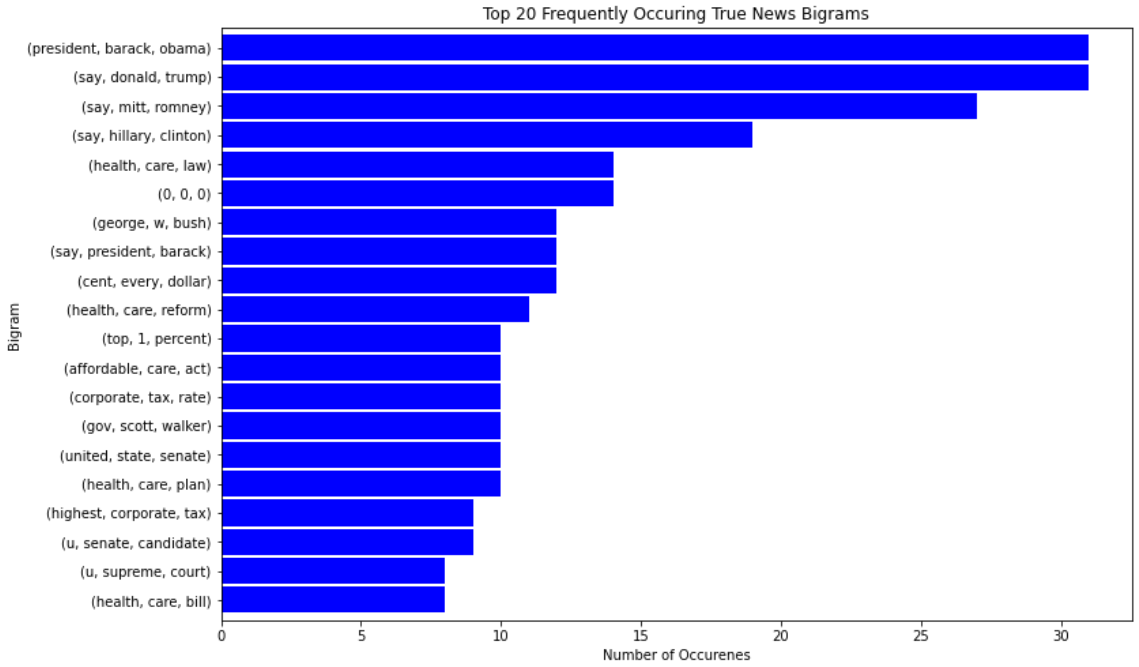
In [20]:
```
with tf.device('GPU:1'):
# Bigram

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


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



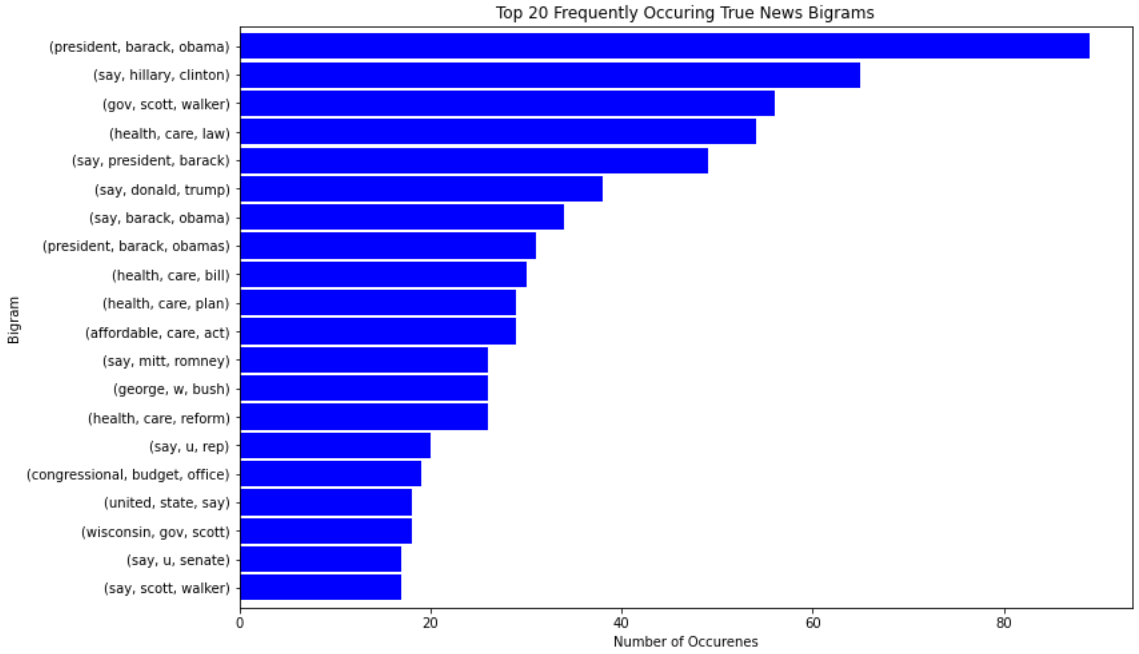
In [21]:
```
with tf.device('GPU:1'):
    plot_top_ngrams(unreliable_news, 'Top 20 Frequently Occuring Fake
```



In [22]:
```
 with tf.device('GPU:1'):
# Trigram
    plot_top_ngrams(reliable_news, "Top 20 Frequently Occuring True N
```



In [23]:
```
with tf.device('GPU:1'):
    plot_top_ngrams(unreliable_news, "Top 20 Frequently Occuring True
```



In [24]:
```
!pip install transformers
```

```
Requirement already satisfied: transformers in /home/administrator/
anaconda3/lib/python3.9/site-packages (4.18.0)
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: packaging>=20.0 in /home/administrat
or/anaconda3/lib/python3.9/site-packages (from transformers) (21.0)
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: filelock in /home/administrator/anac
onda3/lib/python3.9/site-packages (from transformers) (3.3.1)
Requirement already satisfied: requests in /home/administrator/anac
onda3/lib/python3.9/site-packages (from transformers) (2.26.0)
Requirement already satisfied: sacremoses in /home/administrator/an
aconda3/lib/python3.9/site-packages (from transformers) (0.0.49)
Requirement already satisfied: pyyaml>=5.1 in /home/administrator/a
naconda3/lib/python3.9/site-packages (from transformers) (6.0)
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: tqdm>=4.27 in /home/administrator/an
aconda3/lib/python3.9/site-packages (from transformers) (4.62.3)
Requirement already satisfied: numpy>=1.17 in /home/administrator/a
naconda3/lib/python3.9/site-packages (from transformers) (1.20.3)
Requirement already satisfied: 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: idna<4,>=2.5 in /home/administrator/
anaconda3/lib/python3.9/site-packages (from requests->transformers)
(3.2)
Requirement already satisfied: charset-normalizer~=2.0.0 in /home/a
dministrator/anaconda3/lib/python3.9/site-packages (from requests->
transformers) (2.0.4)
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 [25]:
```
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 [26]:
```
with tf.device('GPU:1'):
    def set_seed(seed: int):
        """
        Helper function for reproducible behavior to set the seed in
        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 avai
        if is_tf_available():
            import tensorflow as tf

            tf.random.set_seed(seed)

    set_seed(123)
```

In [27]:
```
with tf.device('GPU:1'):
    model_name = "bert-base-uncased"
    max_length= 512
```

In [28]:
```
with tf.device('GPU:1'):
    tokenizer = BertTokenizerFast.from_pretrained(model_name, do_lowe
```

In [29]:
```
data.head()
```

Out[29]:

	text	label
0	say annies list political group support thirdt...	1
1	decline coal start started natural gas took st...	1
2	hillary clinton agrees john mccain voting give...	0
3	health care reform legislation likely mandate ...	1
4	economic turnaround started end term	1

In [30]:
```
with tf.device('GPU:1'):
    ## Data Preparation
    data = data[data['text'].notna()]
```

In [31]:
```
with tf.device('GPU:1'):
    def prepare_data(df, test_size=0.2, include_title=True, include_a
        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_da
```

In [32]:
```
print(len(train_texts), len(train_labels))
print(len(valid_texts), len(valid_labels))
```

```
8192 8192
2048 2048
```

In [33]:
```
with tf.device('GPU:1'):
    # tokenizing the dataset
    train_encodings = tokenizer(train_texts, truncation=True, padding
    valid_encodings = tokenizer(valid_texts, truncation=True, padding
```

In [34]:
```
with tf.device('GPU:1'):
    # 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.encoding
            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 [35]:
```
with tf.device('GPU:1'):
    model = BertForSequenceClassification.from_pretrained(model_name,
```

```
Some weights of the model checkpoint at bert-base-uncased were not
used when initializing BertForSequenceClassification: ['cls.seq_rel
ationship.weight', 'cls.seq_relationship.bias', 'cls.predictions.bi
as', 'cls.predictions.transform.LayerNorm.weight', 'cls.prediction
s.transform.LayerNorm.bias', 'cls.predictions.transform.dense.bias
', 'cls.predictions.decoder.weight', 'cls.predictions.transform.den
se.weight']
- 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 [36]:
```
with tf.device('GPU:1'):
    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(la
        acc = accuracy_score(labels, preds)
```

```
        return {
            'accuracy': acc,
            'f1': f1,
            'precision': precision,
            'recall': recall
        }
```

In [37]:
```
with tf.device('GPU:1'):
    training_args = TrainingArguments(
        output_dir='./results',          # output directory
        num_train_epochs=1,              # total number of training e
        per_device_train_batch_size=5,  # batch size per device durir
        per_device_eval_batch_size=10,   # batch size for evaluation
        warmup_steps=100,                # number of warmup steps for
        logging_dir='./logs',            # directory for storing logs
        load_best_model_at_end=True,     # load the best model when t
        # but you can specify `metric_for_best_model` argument to cha
        logging_steps=200,               # log & save weights each lo
        save_steps=200,
        evaluation_strategy="steps",     # evaluate each `logging_ste
    )
```

In [38]:
```
with tf.device('GPU:1'):
    trainer = Trainer(
        model = model,
        args = training_args,
        train_dataset=train_dataset,
        eval_dataset=valid_dataset,
        compute_metrics=computer_metrics,
    )
```

In [39]:
```
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 = 8192
  Num Epochs = 1
  Instantaneous batch size per device = 5
  Total train batch size (w. parallel, distributed & accumulation)
= 10
  Gradient Accumulation steps = 1
  Total optimization steps = 820
/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 [40]:
```
# evaluate the current model after training
trainer.evaluate()
```

```
***** Running Evaluation *****
  Num examples = 2048
  Batch size = 10
```

[103/103 00:05]

```
Attempted to log scalar metric eval_loss:
0.6355127692222595
Attempted to log scalar metric eval_accuracy:
0.640625
Attempted to log scalar metric eval_f1:
0.7397454031117398
Attempted to log scalar metric eval_precision:
0.6908850726552179
Attempted to log scalar metric eval_recall:
0.7960426179604262
Attempted to log scalar metric eval_runtime:
5.2521
Attempted to log scalar metric eval_samples_per_second:
389.937
Attempted to log scalar metric eval_steps_per_second:
19.611
Attempted to log scalar metric epoch:
1.0
```

Out[40]:
```
{'eval_loss': 0.6355127692222595,
 'eval_accuracy': 0.640625,
 'eval_f1': 0.7397454031117398,
 'eval_precision': 0.6908850726552179,
 'eval_recall': 0.7960426179604262,
 'eval_runtime': 5.2521,
 'eval_samples_per_second': 389.937,
 'eval_steps_per_second': 19.611,
 'epoch': 1.0}
```

In [41]:
```
# 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[41]:
```
('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 [42]:
```
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 [43]:
```
real_news = """
Says the Annies List political group supports third-trimester abortic
"""

get_prediction(real_news, convert_to_label=True)
```

Out[43]: 'fake'

In [44]:
```
# read the test set
test_df = test
# make a copy of the testing set
new_df = test_df.copy()
# add a new column that contains the author, title and article conter
new_df["new_text"] = new_df["text"].astype(str)
# get the prediction of all the test set
new_df["label"] = new_df["new_text"].apply(get_prediction)
# make the submission file
final_df = new_df[["text", "label"]]
final_df.to_csv("submit_final-LIAR.csv", index=False)
```

In [ ]: