Question 1

For (most of) the questions below, please use the fake news dataset uploaded on BlackBoard (called ‘corona_fake.csv’). You can find the file under ‘Data’ tab. Please include your code also in your .pdf file (in code blocks).

Data Pre-Processing (40 points)

[20 points] Using the pandas package for Python, import the corona_fake.csv dataset, and do the following:

```
a) [5 points] Import the nltk package. Check the documentation:
https://www.nltk.org/

b) [15 points] Take a look at the text column in the dataset, and do the following:

    i. [3 points] Using nltk.word_tokenize(), tokenize the text.

    ii. [3 points] Using the POS-tagging feature (nltk.pos_tag), POS-tag the
    tokenized words.

    iii. [3 points] Using WordNetLemmatizer (from nltk.stem import
    WordNetLemmatizer) lemmatize the pos-tagged words you obtained
    above. (Hint: If there is no available tag, append the token as is; else, use the
    tag to lemmatize the token)

    iv. [3 points] Using the list of stop words that can be imported (nltk.corpus
    import stopwords), remove the stopwords in lemmatized text [Note: the
    language needs to be set as ‘english’.].

    v. [3 points] Finally, also remove numbers, words that are shorter than 2
    characters, punctuation, links and emojis. Finally, convert the obtained list of
    tokenized+tagged+lemmatized+cleaned list of words back into a joined string
    (joined by space ‘ ‘ ) and add the result as text_clean column to your dataset.
```

In [1]:

```
# import the required libraries


# nltk.download()
```

```
import pandas as pd
import numpy as np
import nltk
from nltk import pos_tag
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize

nltk.download('punkt')
```

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

Out[1]: True

In [2]:

```
# import the dataset

df_coronaFake = pd.read_csv("/Users/haileythanki/Downloads/corona_fake.csv")
df_coronaFake
```

Out[2]:

	title	text	source	label
0	Due to the recent outbreak for the Coronavirus...	You just need to add water, and the drugs and ...	coronavirusmedicalkit.com	fake
1	NaN	Hydroxychloroquine has been shown to have a 10...	RudyGiuliani	fake
2	NaN	Fact: Hydroxychloroquine has been shown to hav...	CharlieKirk	fake
3	NaN	The Corona virus is a man made virus created i...	JoanneWrightForCongress	fake
4	NaN	Doesn't @BillGates finance research at the Wuh...	JoanneWrightForCongress	fake
...	...	...	...	...
1154	Could the Power of the Sun Slow the Coronavirus?	A study suggests that ultraviolet rays could s...	https://www.nytimes.com/	true
1155	Key evidence for coronavirus spread is flawed ...	Last week, a medical journal reported that a b...	https://www.nytimes.com/	true
1156	Summer Heat May Not Diminish Coronavirus Strength	A new report, sent to the White House science ...	https://www.nytimes.com/	true
1157	How Long Will a Vaccine Really Take?	A vaccine would be the ultimate weapon against...	https://www.nytimes.com/	true
1158	Why Funding the Covid-19 Response Could Be the...	Developing and delivering coronavirus vaccines...	https://www.nytimes.com/	true

1159 rows × 4 columns

In [3]:

```
# function definition to identify empty strings

def isNaN(string):
    return string != string


# tokenize the strings in the text column and store them in an array

text_tokenized_arr = []

for i in range(0, 1159):
    if (isNaN(df_coronaFake['text'][i])):
        text_tokenized_arr.append(np.nan)
    else:
        text_tokenized = nltk.word_tokenize(df_coronaFake['text'][i])
        text_tokenized_arr.append(text_tokenized)

# print the first tokenized string

text_tokenized_arr[0]
```

Out[3]:

```
['You',
 'just',
 'need',
 'to',
 'add',
 'water',
 ',',
 'and',
 'the',
 'drugs',
 'and',
 'vaccines',
 'are',
 'ready',
 'to',
 'be',
 'administered',
 '.',
 'There',
 'are',
 'two',
 'parts',
 'to',
 'the',
 'kit',
 ':',
 'one',
 'holds',
 'pellets',
 'containing',
 'the',
 'chemical',
 'machinery',
 'that',
 'synthesises',
 'the',
 'end',
 'product',
 ',',
 'and',
 'the',
 'other',
 'holds',
 'pellets',
 'containing',
 'instructions',
 'that',
 'telll',
 'the',
 'drug',
 'which',
 'compound',
 'to',
 'create',
 '.',
 'Mix',
 'two',
 'parts',
 'together',
 'in',
 'a',
 'chosen',
 'combination',
 ',',
 'add',
 'water',
 ',',
 'and',
 'the',
 'treatment',
 'is',
 'ready',
 '.']
```

In [4]:

```
# POS tag the tokenized strings and store them in an array

text_pos_tagged_arr = []

for i in range(0, 1159):
    if (isNaN(df_coronaFake['text'][i])):
        text_pos_tagged_arr.append(np.nan)

    else:
        text = df_coronaFake['text'][i]
        text_tokenized = nltk.word_tokenize(text)
        text_tokenized = [word for word in text_tokenized]
        text_pos_tagged = nltk.pos_tag(text_tokenized)
        text_pos_tagged_arr.append(text_pos_tagged)

# print the first tokenized and POS tagged string

text_pos_tagged_arr[0]
```

Out[4]:

```
[('You', 'PRP'),
 ('just', 'RB'),
 ('need', 'VB'),
 ('to', 'TO'),
 ('add', 'VB'),
 ('water', 'NN'),
 (',', ','),
 ('and', 'CC'),
 ('the', 'DT'),
 ('drugs', 'NNS'),
 ('and', 'CC'),
 ('vaccines', 'NNS'),
 ('are', 'VBP'),
 ('ready', 'JJ'),
 ('to', 'TO'),
 ('be', 'VB'),
 ('administered', 'VBN'),
 ('.', '.'),
 ('There', 'EX'),
 ('are', 'VBP'),
 ('two', 'CD'),
 ('parts', 'NNS'),
 ('to', 'TO'),
 ('the', 'DT'),
 ('kit', 'NN'),
 (':', ':'),
 ('one', 'CD'),
 ('holds', 'VBZ'),
 ('pellets', 'NNS'),
 ('containing', 'VBG'),
 ('the', 'DT'),
 ('chemical', 'NN'),
 ('machinery', 'NN'),
 ('that', 'WDT'),
 ('synthesises', 'VBZ'),
 ('the', 'DT'),
 ('end', 'NN'),
 ('product', 'NN'),
 (',', ','),
 ('and', 'CC'),
 ('the', 'DT'),
 ('other', 'JJ'),
 ('holds', 'VBZ'),
 ('pellets', 'NNS'),
 ('containing', 'VBG'),
 ('instructions', 'NNS'),
 ('that', 'WDT'),
 ('telll', 'VBP'),
 ('the', 'DT'),
 ('drug', 'NN'),
 ('which', 'WDT'),
 ('compound', 'NN'),
 ('to', 'TO'),
 ('create', 'VB'),
 ('.', '.'),
 ('Mix', 'NNP'),
 ('two', 'CD'),
 ('parts', 'NNS'),
 ('together', 'RB'),
 ('in', 'IN'),
 ('a', 'DT'),
 ('chosen', 'NN'),
 ('combination', 'NN'),
 (',', ','),
 ('add', 'JJ'),
 ('water', 'NN'),
 (',', ','),
 ('and', 'CC'),
 ('the', 'DT'),
 ('treatment', 'NN'),
 ('is', 'VBZ'),
 ('ready', 'JJ'),
 ('.', '.')]
```

In [5]:

```
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet

lemmatizer = WordNetLemmatizer()

# function definition for converting the POS tagged strings to nltk-friendly POS tags

def nltk_pos_tagger(nltk_tag):
    if nltk_tag.startswith('J'):
        return wordnet.ADJ
    elif nltk_tag.startswith('V'):
        return wordnet.VERB
    elif nltk_tag.startswith('N'):
        return wordnet.NOUN
    elif nltk_tag.startswith('R'):
        return wordnet.ADV
    else:
        return None

# function definition for lemmatizing POS tagged strings

def lemmatize_sentence(text):

    nltk_tagged = nltk.pos_tag(nltk.word_tokenize(text))
    text_wordnet_tagged = map(lambda x: (x[0], nltk_pos_tagger(x[1])), nltk_tagged)
    text_lemmatized = []

    for word, tag in text_wordnet_tagged:
        if tag is None:
            text_lemmatized.append(word)
        else:
            text_lemmatized.append(lemmatizer.lemmatize(word, tag))
    return " ".join(text_lemmatized)

text_lemmatized_arr = []

# calling the lemmatizer function to convert all strings in the text column to lemmatized strings

for i in range(0, 1159):
    if (isNaN(df_coronaFake['text'][i])):
        text_lemmatized_arr.append(np.nan)
    else:
        text = df_coronaFake['text'][i]
        text_lemmatized = lemmatize_sentence(text)
        text_lemmatized_arr.append(text_lemmatized)

# printing the first lemmatized string

text_lemmatized_arr[0]
```

Out[5]:

```
'You just need to add water , and the drug and vaccine be ready to be administer . There be two part to the kit
: one hold pellet contain the chemical machinery that synthesise the end product , and the other hold pellet co
ntain instruction that telll the drug which compound to create . Mix two part together in a chosen combination
, add water , and the treatment be ready .'
```

Reference: https://www.holisticseo.digital/python-seo/nltk/lemmatize

In [6]:

```
lemmatizer = WordNetLemmatizer()

stop_words = set(stopwords.words('english'))

text_stopwords_rem_arr = []

# function definition for lemmatizing sentences and removing stop words at the same time

def lemmatize_sentence_rem_stop_words(title):

    nltk_tagged = nltk.pos_tag(nltk.word_tokenize(text))
    text_wordnet_tagged = map(lambda x: (x[0], nltk_pos_tagger(x[1])), nltk_tagged)
    text_lemmatized = []

    for word, tag in text_wordnet_tagged:
            if tag is None and word not in stop_words :
                text_lemmatized.append(word)
            elif word not in stop_words:
                text_lemmatized.append(lemmatizer.lemmatize(word, tag))
    return " ".join(text_lemmatized)

text_lemmatized_arr = []

# calling the function for lemmatization and removing stop words

for i in range(0, 1159):
    if (isNaN(df_coronaFake['text'][i])):
        text_stopwords_rem_arr.append(np.nan)
    else:
        text = df_coronaFake['text'][i]
        text_stopwords_rem = lemmatize_sentence_rem_stop_words(text)
        text_stopwords_rem_arr.append(text_stopwords_rem)

text_stopwords_rem_arr[0]
```

Out[6]:

```
'You need add water , drug vaccine ready administer . There two part kit : one hold pellet contain chemical mac
hinery synthesise end product , hold pellet contain instruction telll drug compound create . Mix two part toget
her chosen combination , add water , treatment ready .'
```

In [7]:

```
# remove numbers, words that are shorter than 2 characters, punctuation, links and emojis

import re

text_clean_arr = []

emoji_pattern = re.compile("["u"\U0001F600-\U0001F64F]+")

# remove punctuation, numbers and words that are shorter than 2 characters
for text in text_stopwords_rem_arr:
    if (isNaN(text)):
        text_clean_arr.append(np.nan)
    else:

        text_split = text.split(" ")
        text_clean = []
        for word in text_split:
            if (len(word)>=2 and not(word.replace('.','',1).isnumeric()) and not(word.replace(',','',1).isnumer
                text_clean.append(word)
        text_clean = (" ".join(text_clean))
        text_clean = emoji_pattern.sub(r'', text_clean) # remove emojis
        text_clean = re.sub(r'^https?:\/\/.*[\r\n]*', '', text_clean, flags=re.MULTILINE) # remove links
        text_clean_arr.append(text_clean)

text_clean_arr[0]
```

Out[7]:

```
'You need add water drug vaccine ready administer There two part kit one hold pellet contain chemical machinery
synthesise end product hold pellet contain instruction telll drug compound create Mix two part together chosen
combination add water treatment ready'
```

In [8]:

```
# dataframe with the text_clean column

df_coronaFake["text_clean"] = text_clean_arr
df_coronaFake
```

Out[8]:

	title	text	source	label	text_clean
0	Due to the recent outbreak for the Coronavirus...	You just need to add water, and the drugs and ...	coronavirusmedicalkit.com	fake	You need add water drug vaccine ready administ...
1	NaN	Hydroxychloroquine has been shown to have a 10...	RudyGiuliani	fake	Hydroxychloroquine show effective rate treat C...
2	NaN	Fact: Hydroxychloroquine has been shown to hav...	CharlieKirk	fake	Fact Hydroxychloroquine show effective rate tr...
3	NaN	The Corona virus is a man made virus created i...	JoanneWrightForCongress	fake	The Corona virus man make virus create Wuhan l...
4	NaN	Doesn’t @BillGates finance research at the Wuh...	JoanneWrightForCongress	fake	Doesn BillGates finance research Wuhan lab Cor...
...	...	...	...	...	...
1154	Could the Power of the Sun Slow the Coronavirus?	A study suggests that ultraviolet rays could s...	https://www.nytimes.com/	true	study suggest ultraviolet ray could slow virus...
1155	Key evidence for coronavirus spread is flawed ...	Last week, a medical journal reported that a b...	https://www.nytimes.com/	true	Last week medical journal report business trav...
1156	Summer Heat May Not Diminish Coronavirus Strength	A new report, sent to the White House science ...	https://www.nytimes.com/	true	new report send White House science adviser sa...
1157	How Long Will a Vaccine Really Take?	A vaccine would be the ultimate weapon against...	https://www.nytimes.com/	true	vaccine would ultimate weapon coronavirus best...
1158	Why Funding the Covid-19 Response Could Be the...	Developing and delivering coronavirus vaccines...	https://www.nytimes.com/	true	Developing deliver coronavirus vaccine test tr...

1159 rows × 5 columns

Question 2

[20 points] Let’s vectorize the data we produced above by using two approaches: Bag of Words (BOW) and TF-IDF; and, at the end, we will make a prediction:

```
a. [5 points] Read the following page: https://en.wikipedia.org/wiki/N-gram. Explain
what an ‘n-gram’ is and why it is helpful in max. 200 words.

b. [5 points] Import CountVectorizer and TfidfVectorizer:
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer

c. [5 points] Using CountVectorizer, create three vectorized representations of
text_clean [set lowercase=True]:

    i. One vectorized representation where ngram_range = (1,1)
    ii. One vectorized representation where ngram_range = (1,2)
    iii. One vectorized representation where ngram_range = (1,3)

d. [5 points] Using TfidfVectorizer, create three vectorized representations of
text_clean [set lowercase=True]:

    i. One vectorized representation where ngram_range = (1,1)
    ii. One vectorized representation where ngram_range = (1,2)
    iii. One vectorized representation where ngram_range = (1,3)
```

N-gram models are based on Markov models. An N-gram is a sequence of N words where the probability of a word depends on the previous words of the sequence.

A simple way to look at it is as follows:

$P(w|H)$: probability of word ‘w’, given some history ‘H’

History refers to the sequence of words before the word for which we are trying to calculate the probability.

N-gram models can be used to approximate the probability of a word given all the previous words by using conditional probability of all the preceding words.

They are especially useful for speech recognition, machine translation and predictive text input.

In [9]:
```
# drop rows with na values in the test_clean column

df_coronaFake = df_coronaFake[df_coronaFake['text_clean'].notna()]
df_coronaFake
```

Out[9]:

	title	text	source	label	text_clean
0	Due to the recent outbreak for the Coronavirus...	You just need to add water, and the drugs and ...	coronavirusmedicalkit.com	fake	You need add water drug vaccine ready administ...
1	NaN	Hydroxychloroquine has been shown to have a 10...	RudyGiuliani	fake	Hydroxychloroquine show effective rate treat C...
2	NaN	Fact: Hydroxychloroquine has been shown to hav...	CharlieKirk	fake	Fact Hydroxychloroquine show effective rate tr...
3	NaN	The Corona virus is a man made virus created i...	JoanneWrightForCongress	fake	The Corona virus man make virus create Wuhan l...
4	NaN	Doesn't @BillGates finance research at the Wuh...	JoanneWrightForCongress	fake	Doesn BillGates finance research Wuhan lab Cor...
...	...	...	...	...	...
1154	Could the Power of the Sun Slow the Coronavirus?	A study suggests that ultraviolet rays could s...	https://www.nytimes.com/	true	study suggest ultraviolet ray could slow virus...
1155	Key evidence for coronavirus spread is flawed ...	Last week, a medical journal reported that a b...	https://www.nytimes.com/	true	Last week medical journal report business trav...
1156	Summer Heat May Not Diminish Coronavirus Strength	A new report, sent to the White House science ...	https://www.nytimes.com/	true	new report send White House science adviser sa...
1157	How Long Will a Vaccine Really Take?	A vaccine would be the ultimate weapon against...	https://www.nytimes.com/	true	vaccine would ultimate weapon coronavirus best...
1158	Why Funding the Covid-19 Response Could Be the...	Developing and delivering coronavirus vaccines...	https://www.nytimes.com/	true	Developing deliver coronavirus vaccine test tr...

1151 rows × 5 columns

In [10]:

```python
df_coronaFake[df_coronaFake['text_clean'].isna()]
```

Out[10]:

title	text	source	label	text_clean

In [11]:

```python
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
```

In [15]:

```
# transform data using CountVectorizer with ngram_range = (1,1)

text_clean = df_coronaFake["text_clean"]
cv_11 = CountVectorizer(ngram_range=(1, 1), lowercase=True)
df_vec_cv_11 = cv_11.fit_transform(text_clean)
cv_11.get_feature_names()[0:5]
```

Out[15]:

```
['00', '000', '000km', '000mg', '000th']
```

In [17]:

```
df_vec_cv_11.A
```

Out[17]:

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])
```

In [16]:

```
# transform data using CountVectorizer with ngram_range = (1,2)

text_clean = df_coronaFake["text_clean"]
cv_12 = CountVectorizer(ngram_range=(1, 2), lowercase=True)
df_vec_cv_12 = cv_12.fit_transform(text_clean)
cv_12.get_feature_names()[0:5]
```

Out[16]:

```
['00', '00 am', '00 minute', '00 pay', '00 pm']
```

In [18]:

```
df_vec_cv_12.A
```

Out[18]:

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])
```

In [19]:

```
# transform data using CountVectorizer with ngram_range = (1,3)
```

```
text_clean = df_coronaFake["text_clean"]
cv_13 = CountVectorizer(ngram_range=(1, 3), lowercase=True)
df_vec_cv_13 = cv_13.fit_transform(text_clean)
cv_13.get_feature_names()[0:5]
```

Out[19]:

```
['00', '00 am', '00 am every', '00 minute', '00 minute mark']
```

In [20]:

```
df_vec_cv_13.A
```

Out[20]:

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])
```

In [21]:

```
# transform data using TfidfVectorizer with ngram_range = (1,1)

text_clean = df_coronaFake["text_clean"]
tfid_11 = TfidfVectorizer(ngram_range=(1, 1), lowercase=True)
df_vec_tfid_11 = tfid_11.fit_transform(text_clean)
tfid_11.get_feature_names()[0:5]
```

Out[21]:

```
['00', '000', '000km', '000mg', '000th']
```

In [22]:

```
df_vec_tfid_11.A
```

Out[22]:

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [23]:

```
# transform data using TfidfVectorizer with ngram_range = (1,2)

text_clean = df_coronaFake["text_clean"]
tfid_12 = TfidfVectorizer(ngram_range=(1, 2), lowercase=True)
```

```
df_vec_tfid_12 = tfid_12.fit_transform(text_clean)
tfid_12.get_feature_names()[0:5]
```

```
Out[23]: ['00', '00 am', '00 minute', '00 pay', '00 pm']
```

```
In [24]: df_vec_tfid_12.A
```

```
Out[24]: array([[0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                ...,
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.]])
```

```
In [25]: # transform data using TfidfVectorizer with ngram_range = (1,3)

         text_clean = df_coronaFake["text_clean"]
         tfid_13 = TfidfVectorizer(ngram_range=(1, 3), lowercase=True)
         df_vec_tfid_13 = tfid_13.fit_transform(text_clean)
         tfid_13.get_feature_names()[0:5]
```

```
Out[25]: ['00', '00 am', '00 am every', '00 minute', '00 minute mark']
```

```
In [26]: df_vec_tfid_13.A
```

```
Out[26]: array([[0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                ...,
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.]])
```

Question 3

[20 points] Now, let's use sklearn.linear_model.LogisticRegressionCV to do some predictions.

Set cv = 5, random_state = 265, and max_iter = 1000, and n_jobs = -1 (other parameters should be left as default)

[Note: training size is 70%, test size is 30%, split by random_state = 265].

```
a. [10 points] By using the three (3) different versions of the CountVectorizer
dataset you created above, run logistic regression to predict class labels (fake,
true). Report three (3) accuracy values associated with each of the regressions.

b. [10 points] By using the three (3) different versions of the TfidfVectorizer
dataset you created above, run logistic regression to predict class labels (fake,
true). Report three (3) accuracy values associated with each of the regressions.

c. Combine and report all accuracy values in a table (6 values in total).
```

In [27]:

```python
from sklearn.linear_model import LogisticRegressionCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

In [28]:

```python
# define and create LogisticRegressionCV model

X = df_coronaFake["text_clean"]
y = df_coronaFake["label"]
logreg_cv = LogisticRegressionCV(cv = 5, random_state = 265, max_iter = 1000, n_jobs = -1)
```

In [29]:

```python
# split data into training and testing sets, predict using data transformed with CountVectorizer with ngram_ran

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 265)
X_train = cv_11.fit_transform(X_train)
X_test = cv_11.transform(X_test)
logreg_cv.fit(X_train, y_train)
print(accuracy_score(y_test, logreg_cv.predict(X_test)))
```

```
0.9075144508670521
```

In [30]:

```python
# split data into training and testing sets, predict using data transformed with CountVectorizer with ngram_ran

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 265)
X_train = cv_12.fit_transform(X_train)
X_test = cv_12.transform(X_test)
logreg_cv.fit(X_train, y_train)
print(accuracy_score(y_test, logreg_cv.predict(X_test)))
```

```
0.9132947976878613
```

In [31]:

```
# split data into training and testing sets, predict using data transformed with CountVectorizer with ngram_ran

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 265)
X_train = cv_13.fit_transform(X_train)
X_test = cv_13.transform(X_test)
logreg_cv.fit(X_train, y_train)
print(accuracy_score(y_test, logreg_cv.predict(X_test)))
```

```
0.8988439306358381
```

In [32]:

```
# split data into training and testing sets, predict using data transformed with TFIDVectorizer with ngram_rang

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 265)
X_train = tfid_11.fit_transform(X_train)
X_test = tfid_11.transform(X_test)
logreg_cv.fit(X_train, y_train)
print(accuracy_score(y_test, logreg_cv.predict(X_test)))
```

```
0.9248554913294798
```

In [33]:

```
# split data into training and testing sets, predict using data transformed with TFIDVectorizer with ngram_rang

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 265)
X_train = tfid_12.fit_transform(X_train)
X_test = tfid_12.transform(X_test)
logreg_cv.fit(X_train, y_train)
print(accuracy_score(y_test, logreg_cv.predict(X_test)))
```

```
0.9190751445086706
```

In [34]:

```
# split data into training and testing sets, predict using data transformed with TFIDVectorizer with ngram_rang

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 265)
X_train = tfid_13.fit_transform(X_train)
X_test = tfid_13.transform(X_test)
logreg_cv.fit(X_train, y_train)
print(accuracy_score(y_test, logreg_cv.predict(X_test)))
```

```
0.9161849710982659
```

In [ ]:

```
# accuracies in tabular format
```

Vectorizer		Accuracy
CountVectorizer	ngram_range = (1,1)	0.9075144508670521
	ngram_range = (1,2)	0.9132947976878613
	ngram_range = (1,3)	0.8988439306358381
TfidfVectorizer	ngram_range = (1,1)	0.9248554913294798
	ngram_range = (1,2)	0.9190751445086706
	ngram_range = (1,3)	0.9161849710982659

Question 4

[40 points] Check the optimizer (solver) functions used by sklearn.linear_model.LogisticRegressionCV. For each function, explain in around 100 words what they mean; specifically:

```
   a. [8 points] What does newton-cg mean?
   b. [8 points] What does lbfgs mean?
   c. [8 points] What does liblinear mean?
   d. [8 points] What does sag mean?
   e. [8 points] What does saga mean?
```

Note: For this question you might need to do some online research. It is your job to find out how they work. You are also welcome to use formulas / matrices in your description.

newton-cg: CG stands for conjugate gradient. It is one of the algorithms that can be used in optimization problems. Newton CG does quadratic approximations using iterative methods for solving nonlinear optimization problems. It calculates Hessian explicitly which can be computationally expensive in high dimensions. The Hessian is a square matrix of second-order partial derivatives of order n X n.

lbfgs: lbfgs stands for Limited-memory Broyden–Fletcher–Goldfarb–Shanno Algorithm. It is an analogue of the Newton's Method. Here the Hessian matrix is approximated using updates specified by the gradient evaluations i.e. approximate gradient evaluations. It uses an estimation to the inverse Hessian matrix. It stores only a few vectors that represent the approximation implicitly hence

"Limited-memory". It performs well when the dataset is small compared to other algorithms. However, if not used properly, it may not converge to anything.

liblinear: It is a "library for large linear classifications". The solver uses Coordinate Descent algorithm. Coordinate Descent algorith solves optimization problems by successively performing approximate minimization along coordinate directions/coordinate hyperplanes. There are a few drawbacks of this solver, it may get stuck at a non-stationary point if the level curves of a function are not smooth, it cannot run in parallel and it cannot learn a true multinomial/multiclass model (the optimization problem is decomposed in a “one-vs-rest” fashion, so separate binary classifiers are trained for all classes)

sag: It stands for Stochastic Average Gradient. This method optimizes the sum of a limited number of smooth convex functions. The iteration cost is independent of the number of terms in the sum, similar to stochastic gradient descent. However, by incorporating a memory of previous gradient values the SAG method achieves a faster convergence rate compared to stochastic gradient descent.

saga: It is a variant of SAG. It supports the non-smooth penalty L1 option i.e. L1 Regularization. It is commonly used for sparse multinomial logistic regression. It is also great for dealing with larger datasets.

In [ ]: