```
Text classification - classify if a news is fake or not (Use ML)
```

In [1]:

```
import pandas as pd
import nltk
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')

import matplotlib.pyplot as plt
from nltk.tokenize import word_tokenize
from nltk.corpus import  stopwords

from wordcloud import WordCloud

from nltk.stem import WordNetLemmatizer

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

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\swapn\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\swapn\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\swapn\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [2]:

```
!pip3 install wordcloud
```

```
Requirement already satisfied: wordcloud in c:\users\swapn\anaconda3\lib\sit
e-packages (1.8.1)
Requirement already satisfied: matplotlib in c:\users\swapn\anaconda3\lib\si
te-packages (from wordcloud) (3.2.2)
Requirement already satisfied: numpy>=1.6.1 in c:\users\swapn\anaconda3\lib
\site-packages (from wordcloud) (1.19.5)
Requirement already satisfied: pillow in c:\users\swapn\anaconda3\lib\site-p
ackages (from wordcloud) (7.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\swapn\anaconda3\lib
\site-packages (from matplotlib->wordcloud) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\swapn\anacon
da3\lib\site-packages (from matplotlib->wordcloud) (2.8.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
c:\users\swapn\anaconda3\lib\site-packages (from matplotlib->wordcloud) (2.
4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\swapn\anaconda3
\lib\site-packages (from matplotlib->wordcloud) (1.2.0)
Requirement already satisfied: six in c:\users\swapn\anaconda3\lib\site-pack
ages (from cycler>=0.10->matplotlib->wordcloud) (1.15.0)
```

In [3]:

```
df = pd.read_csv("news.csv")
```

In [4]:

```
df
```

Out[4]:

	text	subject	fake
0	Donald Trump just couldn t wish all Americans ...	News	1
1	House Intelligence Committee Chairman Devin Nu...	News	1
2	On Friday, it was revealed that former Milwauk...	News	1
3	On Christmas day, Donald Trump announced that ...	News	1
4	Pope Francis used his annual Christmas Day mes...	News	1
...	...	...	...
403	Tune in to the Alternate Current Radio Network...	US_News	0
404	Shawn Helton 21st Century WireWhen looking at...	US_News	0
405	Antifa (Photo: Twitter)Diana Johnstone 21st C...	US_News	0
406	TWO PROTAGONISTS: Jesus Campos, and alleged sh...	US_News	0
407	This latest move by America s notorious Transp...	US_News	0

408 rows × 3 columns

In [5]:

```
# cleaning text content
# 0. Convert data to lower/upper case
# 1. creation of tokens - tokenization
# 2. Remove Stopwords, punctuation marks, numbers
# 3. Stemming or Lemmitization
stop = stopwords.words("english")
```

In [6]:

```
def clean_text(text):
  # FREE, Free, free
  # Lower and tokenization
  tokens = word_tokenize(text.lower())
  # Filter only alphabets
  word_tokens = [t for t in tokens if t.isalpha()]
  # Remove stop words
  clean_tokens = [t for t in word_tokens if t not in stop]
  # Lemmitization
  lemma = WordNetLemmatizer()
  lemma_tokens = [lemma.lemmatize(t) for t in clean_tokens]
  return " ".join(lemma_tokens)
```

In [7]:

```
text = "hello ABC #$%^#@@ 123 and is to words"
```

In [8]:

```
clean_text(text)
```

Out[8]:

```
'hello abc word'
```

In [9]:

```
df['text'] = df['text'].apply(clean_text)
```

In [10]:

```
x = df['text']
y = df['fake']
```

In [11]:

```
y
```

Out[11]:

```
0      1
1      1
2      1
3      1
4      1
      ..
403    0
404    0
405    0
406    0
407    0
Name: fake, Length: 408, dtype: int64
```

In [12]:

```
from sklearn.model_selection import train_test_split
```

In [13]:

```
y.value_counts()
```

Out[13]:

```
1    204
0    204
Name: fake, dtype: int64
```

In [14]:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)
```

In [15]:

```
y_train.value_counts()
```

Out[15]:

```
0    145
1    140
Name: fake, dtype: int64
```

In [16]:

```
y_test.value_counts()
```

Out[16]:

```
1    64
0    59
Name: fake, dtype: int64
```

In [17]:

```
# Count Vectorization
cv = CountVectorizer()
x_train_cv = cv.fit_transform(x_train)
x_test_cv = cv.transform(x_test)
```

In [18]:

```
x_train_cv.toarray()
```

Out[18]:

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 1, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

In [19]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
```

In [20]:

```
dt = DecisionTreeClassifier()
dt.fit(x_train_cv, y_train)
y_pred = dt.predict(x_test_cv)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.19      0.24      0.21        59
           1       0.06      0.05      0.05        64

    accuracy                           0.14       123
   macro avg       0.12      0.14      0.13       123
weighted avg       0.12      0.14      0.13       123
```

In [21]:

```
tf = TfidfVectorizer()
x_train_tf = tf.fit_transform(x_train)
x_test_tf = tf.transform(x_test)
```

In [22]:

```
x_train_tf.toarray()
```

Out[22]:

```
array([[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.06063358,
        0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ]])
```

In [23]:

```
dt = DecisionTreeClassifier()
dt.fit(x_train_tf, y_train)
y_pred = dt.predict(x_test_tf)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.16      0.19      0.17        59
           1       0.11      0.09      0.10        64

    accuracy                           0.14       123
   macro avg       0.14      0.14      0.14       123
weighted avg       0.13      0.14      0.14       123
```

In [24]:

```
# ANN
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

In [25]:

```
# In Ann you need to convert the x_train_cv and x_test_cv, x_train_tf and x_test_tf
# into an array
x_train_cv.shape[1]
```

Out[25]:

```
9507
```

In [26]:

```
x_train_cv = x_train_cv.toarray()
x_test_cv = x_test_cv.toarray()
```

In [27]:

```
x_train_cv.shape[1]
```

Out[27]:

```
9507
```

In [28]:

```
model = Sequential()
model.add(Dense(10, activation="relu", input_shape=(x_train_cv.shape[1], )))
model.add(Dense(6, activation="relu"))
model.add(Dense(1, activation="sigmoid"))

model.compile(loss="binary_crossentropy", optimizer="adam")
```

In [29]:

```
model.fit(x_train_cv, y_train, batch_size=70, epochs=50)
```

```
Epoch 1/50
5/5 [==============================] - 1s 6ms/step - loss: 0.7221
Epoch 2/50
5/5 [==============================] - 0s 7ms/step - loss: 0.6747
Epoch 3/50
5/5 [==============================] - 0s 6ms/step - loss: 0.6638
Epoch 4/50
5/5 [==============================] - 0s 6ms/step - loss: 0.6428
Epoch 5/50
5/5 [==============================] - 0s 7ms/step - loss: 0.6398
Epoch 6/50
5/5 [==============================] - 0s 7ms/step - loss: 0.6199
Epoch 7/50
5/5 [==============================] - 0s 7ms/step - loss: 0.6174
Epoch 8/50
5/5 [==============================] - 0s 7ms/step - loss: 0.6219
Epoch 9/50
5/5 [==============================] - 0s 8ms/step - loss: 0.6067
Epoch 10/50
5/5 [==============================] - 0s 7ms/step - loss: 0.5987
Epoch 11/50
5/5 [==============================] - 0s 10ms/step - loss: 0.5839
Epoch 12/50
5/5 [==============================] - 0s 8ms/step - loss: 0.5767
Epoch 13/50
5/5 [==============================] - 0s 9ms/step - loss: 0.5664
Epoch 14/50
5/5 [==============================] - 0s 8ms/step - loss: 0.5543
Epoch 15/50
5/5 [==============================] - 0s 7ms/step - loss: 0.5567
Epoch 16/50
5/5 [==============================] - 0s 9ms/step - loss: 0.5427
Epoch 17/50
5/5 [==============================] - 0s 7ms/step - loss: 0.5440
Epoch 18/50
5/5 [==============================] - 0s 6ms/step - loss: 0.5403
Epoch 19/50
5/5 [==============================] - 0s 6ms/step - loss: 0.5275
Epoch 20/50
5/5 [==============================] - 0s 7ms/step - loss: 0.5457
Epoch 21/50
5/5 [==============================] - 0s 9ms/step - loss: 0.5363
Epoch 22/50
5/5 [==============================] - 0s 8ms/step - loss: 0.5374
Epoch 23/50
5/5 [==============================] - 0s 8ms/step - loss: 0.5292
Epoch 24/50
5/5 [==============================] - 0s 9ms/step - loss: 0.5172
Epoch 25/50
5/5 [==============================] - 0s 8ms/step - loss: 0.5201
Epoch 26/50
5/5 [==============================] - 0s 8ms/step - loss: 0.5163
Epoch 27/50
5/5 [==============================] - 0s 8ms/step - loss: 0.5109
Epoch 28/50
5/5 [==============================] - 0s 8ms/step - loss: 0.5342
```

```
Epoch 29/50
5/5 [==============================] - 0s 8ms/step - loss: 0.5544
Epoch 30/50
5/5 [==============================] - 0s 8ms/step - loss: 0.5438
Epoch 31/50
5/5 [==============================] - 0s 8ms/step - loss: 0.5155
Epoch 32/50
5/5 [==============================] - 0s 8ms/step - loss: 0.5092
Epoch 33/50
5/5 [==============================] - 0s 8ms/step - loss: 0.4998
Epoch 34/50
5/5 [==============================] - 0s 7ms/step - loss: 0.5145
Epoch 35/50
5/5 [==============================] - 0s 9ms/step - loss: 0.5133
Epoch 36/50
5/5 [==============================] - 0s 10ms/step - loss: 0.5204
Epoch 37/50
5/5 [==============================] - 0s 9ms/step - loss: 0.5140
Epoch 38/50
5/5 [==============================] - 0s 9ms/step - loss: 0.5164
Epoch 39/50
5/5 [==============================] - 0s 9ms/step - loss: 0.5119
Epoch 40/50
5/5 [==============================] - 0s 8ms/step - loss: 0.5301
Epoch 41/50
5/5 [==============================] - 0s 7ms/step - loss: 0.5205
Epoch 42/50
5/5 [==============================] - 0s 7ms/step - loss: 0.5009
Epoch 43/50
5/5 [==============================] - 0s 7ms/step - loss: 0.5222
Epoch 44/50
5/5 [==============================] - 0s 8ms/step - loss: 0.5139
Epoch 45/50
5/5 [==============================] - 0s 7ms/step - loss: 0.5157
Epoch 46/50
5/5 [==============================] - ETA: 0s - loss: 0.502 - 0s 8ms/step -
loss: 0.5009
Epoch 47/50
5/5 [==============================] - 0s 7ms/step - loss: 0.4927
Epoch 48/50
5/5 [==============================] - 0s 8ms/step - loss: 0.4983
Epoch 49/50
5/5 [==============================] - 0s 7ms/step - loss: 0.4912
Epoch 50/50
5/5 [==============================] - 0s 8ms/step - loss: 0.5080
```

Out[29]:

```
<tensorflow.python.keras.callbacks.History at 0x240391abe80>
```

In [30]:

```
y_hat = model.predict(x_test_cv)
```

In [31]:

```
y_hat
```

Out[31]:

```
array([[3.79017979e-01],
       [1.49106979e-02],
       [3.79017979e-01],
       [3.36206257e-02],
       [2.30003595e-02],
       [6.72600031e-01],
       [7.26979971e-03],
       [9.90286946e-01],
       [1.07938856e-01],
       [9.67122257e-01],
       [4.51464593e-01],
       [9.71855581e-01],
       [5.39871573e-01],
       [9.79553342e-01],
       [4.18351889e-01],
       [1.78017318e-02],
       [9.98740971e-01],
       [2.50308484e-01],
```

In [32]:

```
import numpy as np
```

In [33]:

```
y_hat = np.where(y_hat >= 0.5, 1, 0)
print(classification_report(y_test, y_hat))
```

```
              precision    recall  f1-score   support

           0       0.13      0.14      0.13        59
           1       0.15      0.14      0.15        64

    accuracy                           0.14       123
   macro avg       0.14      0.14      0.14       123
weighted avg       0.14      0.14      0.14       123
```

In [34]:

```
x_train_tf = x_train_tf.toarray()
x_test_tf = x_test_tf.toarray()
```

In [35]:

```
model1 = Sequential()
model1.add(Dense(10, activation="relu", input_shape=(x_train_cv.shape[1], )))
model1.add(Dense(6, activation="relu"))
model1.add(Dense(1, activation="sigmoid"))

model1.compile(loss="binary_crossentropy", optimizer="adam")

model1.fit(x_train_tf, y_train, batch_size=60, epochs=50)
```

```
Epoch 1/50
5/5 [==============================] - 1s 4ms/step - loss: 0.6940
Epoch 2/50
5/5 [==============================] - 0s 6ms/step - loss: 0.6918
Epoch 3/50
5/5 [==============================] - 0s 6ms/step - loss: 0.6902
Epoch 4/50
5/5 [==============================] - 0s 6ms/step - loss: 0.6884
Epoch 5/50
5/5 [==============================] - 0s 7ms/step - loss: 0.6850
Epoch 6/50
5/5 [==============================] - 0s 6ms/step - loss: 0.6811
Epoch 7/50
5/5 [==============================] - 0s 6ms/step - loss: 0.6780
Epoch 8/50
5/5 [==============================] - 0s 7ms/step - loss: 0.6739
Epoch 9/50
5/5 [==============================] - 0s 6ms/step - loss: 0.6697
Epoch 10/50
5/5 [==============================] - 0s 7ms/step - loss: 0.6650
Epoch 11/50
5/5 [==============================] - 0s 7ms/step - loss: 0.6549
Epoch 12/50
5/5 [==============================] - 0s 7ms/step - loss: 0.6583
Epoch 13/50
5/5 [==============================] - 0s 6ms/step - loss: 0.6488
Epoch 14/50
5/5 [==============================] - 0s 6ms/step - loss: 0.6393
Epoch 15/50
5/5 [==============================] - 0s 7ms/step - loss: 0.6387
Epoch 16/50
5/5 [==============================] - 0s 6ms/step - loss: 0.6319
Epoch 17/50
5/5 [==============================] - 0s 7ms/step - loss: 0.6275
Epoch 18/50
5/5 [==============================] - 0s 7ms/step - loss: 0.6201
Epoch 19/50
5/5 [==============================] - 0s 8ms/step - loss: 0.6169
Epoch 20/50
5/5 [==============================] - 0s 7ms/step - loss: 0.5972
Epoch 21/50
5/5 [==============================] - 0s 6ms/step - loss: 0.5977
Epoch 22/50
5/5 [==============================] - 0s 8ms/step - loss: 0.5877
Epoch 23/50
5/5 [==============================] - 0s 8ms/step - loss: 0.5889
Epoch 24/50
5/5 [==============================] - 0s 6ms/step - loss: 0.5877
Epoch 25/50
```

```
5/5 [==============================] - 0s 7ms/step - loss: 0.5808
Epoch 26/50
5/5 [==============================] - 0s 7ms/step - loss: 0.5898
Epoch 27/50
5/5 [==============================] - 0s 7ms/step - loss: 0.5730
Epoch 28/50
5/5 [==============================] - 0s 7ms/step - loss: 0.5575
Epoch 29/50
5/5 [==============================] - 0s 7ms/step - loss: 0.5510
Epoch 30/50
5/5 [==============================] - 0s 7ms/step - loss: 0.5579
Epoch 31/50
5/5 [==============================] - 0s 6ms/step - loss: 0.5440
Epoch 32/50
5/5 [==============================] - 0s 6ms/step - loss: 0.5516
Epoch 33/50
5/5 [==============================] - 0s 6ms/step - loss: 0.5326
Epoch 34/50
5/5 [==============================] - 0s 6ms/step - loss: 0.5511
Epoch 35/50
5/5 [==============================] - 0s 8ms/step - loss: 0.5293
Epoch 36/50
5/5 [==============================] - 0s 7ms/step - loss: 0.5381
Epoch 37/50
5/5 [==============================] - 0s 7ms/step - loss: 0.5481
Epoch 38/50
5/5 [==============================] - 0s 6ms/step - loss: 0.5294
Epoch 39/50
5/5 [==============================] - 0s 7ms/step - loss: 0.5300
Epoch 40/50
5/5 [==============================] - 0s 6ms/step - loss: 0.5225
Epoch 41/50
5/5 [==============================] - 0s 6ms/step - loss: 0.5145
Epoch 42/50
5/5 [==============================] - 0s 6ms/step - loss: 0.5318
Epoch 43/50
5/5 [==============================] - 0s 6ms/step - loss: 0.5433
Epoch 44/50
5/5 [==============================] - 0s 8ms/step - loss: 0.5147
Epoch 45/50
5/5 [==============================] - 0s 6ms/step - loss: 0.5009
Epoch 46/50
5/5 [==============================] - 0s 8ms/step - loss: 0.5002
Epoch 47/50
5/5 [==============================] - 0s 7ms/step - loss: 0.5215
Epoch 48/50
5/5 [==============================] - 0s 8ms/step - loss: 0.4932
Epoch 49/50
5/5 [==============================] - 0s 7ms/step - loss: 0.5025
Epoch 50/50
5/5 [==============================] - 0s 6ms/step - loss: 0.5123
```

Out[35]:

```
<tensorflow.python.keras.callbacks.History at 0x24042e27eb0>
```

In [36]:

```
y_hat = model.predict(x_test_tf)
y_hat = np.where(y_hat >= 0.5, 1, 0)
print(classification_report(y_test, y_hat))
```

```
              precision    recall  f1-score   support

           0       0.13      0.14      0.13        59
           1       0.15      0.14      0.15        64

    accuracy                           0.14       123
   macro avg       0.14      0.14      0.14       123
weighted avg       0.14      0.14      0.14       123
```

In [ ]:

```

```