Fake News Detection Using Machine Learning

Table of Contents:

Task Formulation

Fake news's simple meaning is to incorporate information that leads people to the wrong path. Nowadays fake news spreading like water and people share this information without verifying it. This is often done to further or impose certain ideas and is often achieved with political agendas.

For media outlets, the ability to attract viewers to their websites is necessary to generate online advertising revenue. So it is necessary to detect fake news.

And in this project we'll be doing that using some **Machine Learning** and **Natural Language Processing (NLP)** libraries like **NLTK**, **re** (Regular Expression), **Scikit Learn**. Let's first import all the neccessary libraries.

In [1]:
```
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
import pattern
from pattern.en import lemma
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
```

Natural Language Processing

Machine learning model only works with numerical features so we have to convert text data into numerical columns. This kind of text preprocessing is called natural language processing.

In-text preprocess we are cleaning our text by steaming, lemmatization, remove stopwords, remove special symbols and numbers, etc. After cleaning the data we have to feed this text data into a vectorizer which will convert this text data into numerical features.

Data Overview

I downloaded two datasets from Kaggle. One dataset includes **fake news** and the other one is for **true news**.

In [2]:
```
true = pd.read_csv('data/fake.csv')
fake = pd.read_csv('data/true.csv')
```

In [3]:
```
true.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23481 entries, 0 to 23480
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   title    23481 non-null  object
 1   text     23481 non-null  object
 2   subject  23481 non-null  object
 3   date     23481 non-null  object
dtypes: object(4)
memory usage: 733.9+ KB
```

In [4]:
```
fake.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21417 entries, 0 to 21416
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   title    21417 non-null  object
 1   text     21417 non-null  object
 2   subject  21417 non-null  object
 3   date     21417 non-null  object
dtypes: object(4)
memory usage: 669.4+ KB
```

In [5]:
```
fake.isnull().sum().sum()
```

Out[5]:
```
0
```

In [6]:
```
true.isnull().sum().sum()
```

Out[6]:
```
0
```

There are **23481** entries for **true news** dataset and **21417** - in **fake news** dataset. Both datasets include the following information - **title, subject, date** and the news **text** itself. There are **no missing values** in both sets, so no need to **impute** any value.

Firstly, we need to create in each dataset a column labeled with 1 and 0 indicating **fake** (1) and **true** (0) news and then **combine** both datasets in one final set.

Our **final dataset** would be **balanced** because both categories approximate same number of entries.

In [7]:
```
fake.shape
```

Out[7]:
```
(21417, 4)
```

In [8]:
```
true.shape
```

Out[8]:
```
(23481, 4)
```

In [9]:
```
fake['label'] = 1
```

```
true['label'] = 0
df = pd.merge(fake[['text', 'label']], true[['text', 'label']], how='outer')

# shuffle
df = df.sample(frac=1)

df.head(10)
```

Out[9]:

	text	label
22958	Trump and his team made a great deal of the ca...	0
13148	CAIRO (Reuters) - The Arab League on Tuesday c...	1
23066	A student at Transylvania University was injur...	0
41358	Not a bad imitation of a black pastor from a g...	0
12592	(Reuters) - Reuters photographers witnessed th...	1
4163	SAN ANTONIO, Texas (Reuters) - A special feder...	1
28874	With only three clowns left in GOP clown car, ...	0
33074	Mark Steyn was on fire last night! If you were...	0
20960	WASHINGTON (Reuters) - U.S. Ambassador to the ...	1
42792	The Hoke County School school system defended ...	0

In [10]:
```
df.label.value_counts()
```

Out[10]:
```
0    23481
1    21417
Name: label, dtype: int64
```

Data Preprocessing

Cleaning Data

If we use text data directly without cleaning then it is very hard for the Machine Learning algorithm to detect patterns in that text. Moreover, it can generate errors. So, we need first to clean text data - remove some unusable words, special symbols etc.

I will create a separate function to clean the data and then run it on our data. It will **lemmatize** the text - convert each word in its **base form**, remove **stopwords**, **numbers** and **special symbols**.

In [11]:
```
stop_words = set(stopwords.words('english'))

def clean_text(text):
    # convert text to lower case
    text = text.lower()

    # replace numbers and special symbols with a space
    text = re.sub('[^a-zA-Z]', ' ', text)

    # make tokens from splitted data
    tokens = text.split()

    # remove stopwords and lemmatization
    cleaned_text = [lemma(word) for word in tokens if word not in stop_words]

    # join token with space
    cleaned_text = ' '.join(cleaned_text)
```

```
    # return final clean text
    return cleaned_text
```

In [13]:
```
df['text'] = df['text'].map(clean_text)
```

In [14]:
```
df.head()
```

Out[14]:

	text	label
22958	trump team make great deal campaign hillary cl...	0
13148	cairo reuter arab league tuesday condemn kill ...	1
23066	student transylvania university injure attack ...	0
41358	bad imitation black pastor guy quit go church ...	0
12592	reuter reuter photographer witness biggest sto...	1

Split Data

This is an important step. We are going to train the machine learning model on the training dataset and then test the model on the testing set.

Our target `y` will be a **label** column, and the **text** column is the feature in our dataset - `X`.

Then we split our main dataset into `x_train`, `y_train`, `x_test` and `y_test` using `train_test_split` function from the **Scikit learn** library.

We split our 80% data for the training set and the remaining 20% data for the testing set.

In [15]:
```
X = df['text']
y = df['label']
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=.2, random_state=1)
```

Tfidf Vectorizer

Tfidf-Vectorizer: *Term Frequency* Inverse Document Frequency*

Term Frequency: The number of times word appear in the text divided by the total number of words there:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{n,j}}$$

Inverse Document Frequency: a measure of how much information the word provides, i.e., if it is common or rare across all documents. It is the logarithmically scaled inverse fraction of the documents that contain the word (obtained by taking the log of the total number of documents divided by the number of documents containing the term):

$$idf(w) = \log(\frac{N}{df_t})$$

So, Tfidf vectorizer is:

$$w_{i,j} = tf_{i,j} \times \log(\frac{N}{df_i})$$

Tfidf Vectorizer is already in **Scikit Learn** library, so we can take it, fit on our training dataset and transform its values on both train and test datasets with respect to the vectorizer.

In [16]:
```
vectorizer = TfidfVectorizer(max_features=50000, lowercase=False, ngram_range=(1,2))

vec_train_data = vectorizer.fit_transform(x_train)
vec_test_data = vectorizer.transform(x_test)
```

After vectorizing the data it will return the sparse matrix so that for machine learning algorithms we have to convert it into arrays.

In [17]:
```
vec_train_data = vec_train_data.toarray()
vec_test_data = vec_test_data.toarray()
```

Classification Model

Multinomial Naive Bayes Classifier

Naive Bayes: the Naive Bayes Classifier is based on the Bayesian theorem and is particularly suited with high dimensional data.

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

where:

- A, B - events
- $P(A|B)$ - probability of A given B is true
- $P(B|A)$ - probability of B given A is true
- $P(A), P(B)$ - independent probabilities of A and B

Multinomial Naive Bayes is used for classification of discrete data. It is very useful in text processing. The text will be converted to a vector of word count.

This technique is predefined in **Scikit Learn** Library. So we can import that class and create an object of Multinomial Naive Bayes Class.

1. Fit the classifier on our vectorized train data.
2. Use the predict method to predict the result on the test set.

In [18]:
```
vec_train_data.shape , vec_test_data.shape
```

Out[18]:
```
((35918, 50000), (8980, 50000))
```

In [19]:
```
training_data = pd.DataFrame(vec_train_data , columns=vectorizer.get_feature_names_out()
testing_data = pd.DataFrame(vec_test_data , columns= vectorizer.get_feature_names_out())
```

In [20]:
```
clf = MultinomialNB()
clf.fit(training_data, y_train)
y_pred  = clf.predict(testing_data)
```

In [21]:
```
y_pred
```

Out[21]:
```
array([0, 0, 1, ..., 1, 0, 0], dtype=int64)
```

Classification Metrics

To check how good is our model we use some metrics to find the accuracy of our model on both train and test sets. There are many types of classification metrics available in Scikit learn, we'll use some of them:

- Precision
- Recall
- F1-Score
- Accuracy Score

In [22]:
```
print(classification_report(y_test , y_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.96      0.95      4641
           1       0.95      0.94      0.95      4339

    accuracy                           0.95      8980
   macro avg       0.95      0.95      0.95      8980
weighted avg       0.95      0.95      0.95      8980
```

In [23]:
```
y_pred_train = clf.predict(training_data)
print(classification_report(y_train , y_pred_train))
```

```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96     18840
           1       0.96      0.95      0.95     17078

    accuracy                           0.96     35918
   macro avg       0.96      0.96      0.96     35918
weighted avg       0.96      0.96      0.96     35918
```

In [24]:
```
# accuracy on train data
accuracy_score(y_train , y_pred_train)
```

Out[24]:
```
0.9572080850826884
```

In [25]:
```
# accuracy on test data
accuracy_score(y_test , y_pred)
```

Out[25]:
```
0.948663697104677
```

Scores on both train and test datasets are very high. Thus, our model performs well.

Running the Model

We have a **line of text** that I took from the **news website**. Let's check if it is **fake** or **true**. As it was done before with the **train data**, I'm going to **clean** this text line, **lemmatize**, **vectorize** and then put this line into the **model**.

In [26]:
```
line = "Ukraine's most nationalist region was once a hotbed of pro-Russian sentiment - h
news = clean_text(line)
vec_news = vectorizer.transform([news]).toarray()
vec_news_data = pd.DataFrame(vec_news, columns= vectorizer.get_feature_names_out())
```

In [27]:
```
single_prediction = clf.predict(vec_news_data)
```

```
print('Fake News!' if single_prediction[0] else 'True News')
```

```
Fake News!
```

Recall that **1** is fake and **0** is true. So, the news line I found is **Fake**.

Save the Model

Now we can save the model to have a possibility to use it again without double training.

In [28]:

```
import joblib
joblib.dump(clf, 'data/model.pkl')
model = joblib.load('data/model.pkl')
```

Summary

With this ML model we can detect **fake news**. We took **Fake** and **True News** datasets, implemented a **Text cleaning** function, **TfidfVectorizer**, initialized **Multinomial Naive Bayes Classifier**, and fit our model. We ended up obtaining an **accuracy of 94.87%**.

In [ ]:

```

```