When we start making a system to find fake news, it's really important to look carefully at the data we have. The data usually has news stories, and each one is marked as real or fake. We start by putting this data into a computer program like Python with a special library called pandas. Then we look closely at how the data is organized. We want to understand things like what information is there (like the news text), what labels are given (real or fake), and how everything is arranged.

After we understand the basic organization of the data, we need to prepare the text part. This means cleaning it up to make it easier for the computer to understand. We get rid of unnecessary stuff like punctuation and weird symbols. We also make everything lowercase so the computer doesn't treat the same word with different cases as different words. Then we break the text into individual words or smaller parts, which helps us study it better. We also remove common words like "the" and "is" because they don't help much in telling if a news story is real or fake. Sometimes, we also change words to their simpler forms to make it easier for the computer to understand.

Once the text is ready, we start looking deeper into the data. We want to see how often different words appear in both real and fake news stories. We use graphs like histograms and word clouds to see if there are any patterns. We pay special attention to how many real and fake news stories we have to make sure we're not biased towards one type.

Then comes the part where we decide which information is most important for telling real and fake news apart. We use fancy techniques to turn the text into numbers that the computer can understand better. These techniques help us find patterns in the text.

The main goal during all this is to really understand our data well. This helps us build a good system that can tell if news is real or fake. Once we've thoroughly explored and preprocessed our dataset, the next step is to build a machine learning model for our fake news detection system. The model will learn from the features extracted from the news articles and their labels indicating whether they are real or fake.

First, we need to split our dataset into training and testing sets. The training set will be used to teach the model to recognize patterns and associations between the features and the labels. The testing set, on the other hand, will be used to evaluate how well the model performs on new, unseen data.

There are several machine learning algorithms we can consider for this task, each with its strengths and weaknesses. Commonly used algorithms include logistic regression, naive Bayes, support vector machines (SVM), random forests, and gradient boosting machines (GBM).

Once we've selected an algorithm, we train the model using the training data. During training, the model adjusts its parameters to minimize the difference between its predictions and the actual labels in the training set. This process involves feeding the features and labels into the algorithm and iteratively updating the model's parameters until it converges to the best possible solution.

After training, we evaluate the model's performance using the testing set. We compare the model's predictions against the actual labels in the testing set to assess its accuracy, precision, recall, F1-score, and other performance metrics. This evaluation helps us understand how well the model generalizes to new, unseen data.

Depending on the results of the evaluation, we may need to fine-tune the model's hyperparameters to improve its performance further. Hyperparameters are settings that control the behavior of the algorithm, such as the learning rate or the number of trees in a random forest.

Once we're satisfied with the model's performance, we can deploy it into production. This involves integrating the model into our fake news detection system, where it can automatically classify news articles as real or fake in real-time.

It's important to note that building a machine learning model is an iterative process. We may need to revisit earlier steps, such as data preprocessing or feature engineering, based on the performance of the model. Additionally, we should continuously monitor the model's performance in production and update it as needed to ensure its effectiveness over time.

Loading the data

import pandas as pd

import numpy as np

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import PassiveAggressiveClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

import itertools

import seaborn as sns

import matplotlib.pyplot as plt

Reading and exploring the dataset

news\_data= pd.read\_csv("news.csv")

news\_data.head(10)

Identifying all fields in the files in the order they appear

news\_data.info()

Returning a tuple containing the number of rows and columns present in the dataset

news\_data.shape

Counting the occurrences of each unique value of Real and Fake news

news\_data["label"].value\_counts()

Fetching the starting 10 entries from the dataset to understand the pattern for the further development of the model

labels= news\_data.label

labels.head(10)

Building the Model

Splitting the dataset into train and Test samples

x\_train, x\_test, y\_train, y\_test= train\_test\_split(news\_data["text"], labels, test\_size= 0.4, random\_state= 7)

Using the Tfid vectorizer with english Stop Words

vectorizer=TfidfVectorizer(stop\_words='english', max\_df=0.7)

tfidf\_train=vectorizer.fit\_transform(x\_train)

tfidf\_test=vectorizer.transform(x\_test)

Creating a passiveaggressive Classifier

passive=PassiveAggressiveClassifier(max\_iter=50)

passive.fit(tfidf\_train,y\_train)

y\_pred=passive.predict(tfidf\_test)

Evaluating the Model Accuracy using the confusion matrix

Creating a Confusion matrix

matrix= confusion\_matrix(y\_test,y\_pred, labels=['FAKE','REAL'])

matrix

O/p

array([[1188, 82], [ 89, 1175]], dtype=int64)

Visualising the Confusion Matrix

sns.heatmap(matrix, annot=True)

plt.show()

Calculating the Model’s Accuracy

Accuracy=accuracy\_score(y\_test,y\_pred)

Accuracy\*100

O/p

93.25177584846092

Printing the Report of the model

Report= classification\_report(y\_test, y\_pred)

print(Report)