HARAMAYA UNIVERSITY

COLLEGE OF COMPUTING AND INFORMATICS

DEPARTMENT OF SOFTWARE ENGLINEERNIG

FUNDAMENTALS OF ARTIFICIAL INTELLIGENCE COURSE

Assignment for fundamentals of Artificial Intelligence

Creating fake news detection system using machine learning

Group Members

Name	ID Number
1. Yishak Alemu	- - - 3661/14
2. Yohannes Ayenew	- - - 3676/14
3. Yoseph Dagne	- - - 3720/14
4. Yusuf Kedir	- - - 3737/14
5. Zekarias Tamiru	- - - 3747/14

Section: B

FAKE NEWS DETECTION USING MACHINE LEARNING

Fake news on different platforms are spreading widely and is a matter of serious concern, as it causes social wars and permanent breakage of the bonds established among people. A lot of research is already going on focused on the classification of fake news.

Here in this project we will try to solve this issue with the help of machine learning in python. We will build the system that helps us in identifying these fake news and separate them from real news.

We will use logistic regression model for our project. Because we are going to build a system that classifies article news as either fake or real news, so it is binary classification. Logistic regression is a statistical model used for binary classification tasks, so it is well suited for our project .

Here is Google colab link of our project:

https://colab.research.google.com/drive/1Jnj90Pu_tU6ITJdWNcU_ffwvnU5Imvbu

To make this system successfully, we will follow the following steps:

Step 1. Data collection:

Gather a labeled dataset of news articles, where each article is marked as either "fake" or "real." Sources can include platforms like Kaggle or specific news datasets.

In our case, we are using public dataset from Kaggle platform. We are using public dataset because they are often well documented and suitable for various tasks, and also to save our time.

Below is the description of each datasets we are going to use:

1. **Train.csv:** this is the primary dataset we are going to use for training our machine learning model, in our case the model is Logistic regression model.

This dataset contains labeled data, meaning each news article in this dataset has a corresponding label indicating whether it is "fake" or "real." This dataset contains five columns give below:

1. id: unique id for a news article
2. title: the title of a news article
3. author: author of the news article
4. text: the text of the article; could be incomplete
5. label: a label that marks whether the news article is real or fake:
   1: Fake news
   0: real News

2. **test.csv:** This dataset is used for evaluating the performance of our trained model. it contains the same structure as train.csv, but it does not include the labels for the articles (because it is going to be determined by our model).

After we train our model with `train.csv`, we will use `test.csv` to make predictions. The model will classify the articles in this dataset as either "fake" or "real."

3. **Submit.csv:** is used for submitting predictions after our model has been trained and evaluated. It contains an `id` column and a placeholder for predicted labels but does not include the actual article text.
   Its structure is as indicated below:
   - **`id`**: A unique identifier for each article.
   - **`label`**: (to be filled with your model's predictions).

After making predictions on the articles in `test.csv`, we will format these predictions and save them in `sumbit.csv`

Step 2: Data exploration and preparation

- **Loading the dataset:** we Pandas library to load the dataset into our Python environment.
- **Explore the dataset:** Check the first few rows and the structure of the dataset, and understand the features available (like text and label).
- **Preprocess the data:**
    - ✓ Identify and address missing values.
    - ✓ Convert all text to lowercase to ensure uniformity.
    - ✓ Clean the text by removing characters that do not contribute to the meaning (e.g., punctuation, special characters).
    - ✓ Remove stop words: Eliminate common words (like "the," "is," "and") that may not help in distinguishing between fake and real news.
    - ✓ Stemming: Reduce words to their base or root form. Stemming cuts words down to their base form.

Step 3: Feature Extraction

- ✓ **Convert Text to Features**: we use TfidfVectorizer from scikit-learn to convert text data into numerical features.

Step 4: Split the dataset

- ✓ **Train-Test Split**: Split the dataset into training and testing sets using `train_test_split`.

Step 5: Model training

- ✓ **Train the Logistic Regression Model**: Import and fit the logistic regression model.

Step 6: Model Evaluation

- ✓ **Make Predictions**: Predict the labels for the test set.
- ✓ **Evaluate the Model**: Calculate the accuracy score to evaluate the performance.

Now, let’s begin building our system by following the above steps:

First we need to import the dependencies, the dependencies are the labels and the functions we need in this code.

Here is the code to import all the dependencies we need:

```
import numpy as np
import pandas as pd
import re
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

The table below summarizes the libraries we imported above for building a fake news prediction system in machine learning, along with their purposes:

Library	**Purpose**
NumPy (`numpy`)	A fundamental library for numerical computing in Python. It provides support for arrays, matrices, and a wide range of mathematical functions. Used for efficient data manipulation and numerical operations.
Pandas (`pandas`)	A powerful data manipulation and analysis library. It provides data structures like DataFrames for handling structured data, making it easy to read, write, and manipulate datasets.
Regular Expressions (`re`)	A module for performing string matching and manipulation using

	regular expressions. Useful for cleaning text data, such as removing special characters and formatting.
NLTK Stopwords (`nltk.corpus.stopwords`)	Part of the Natural Language Toolkit (NLTK), this module provides a list of common stop words in various languages. These words are often removed from text as they carry little meaning and can be considered noise in natural language processing tasks.
NLTK Stemmer (`nltk.stem.porter.PorterStemmer`)	Provides stemming functionality, which reduces words to their root form (e.g., "running" to "run"). This helps in normalizing words for better text analysis and feature extraction.
TF-IDF Vectorizer (`sklearn.feature_extraction.text.TfidfVectorizer`)	A feature extraction tool that converts text documents into numerical feature vectors based on the Term Frequency-Inverse Document Frequency (TF-IDF) method. This representation helps in quantifying the importance of words in the context of the dataset.
Train-Test Split (`sklearn.model_selection.train_test_split`)	A utility function to split a dataset into training and testing subsets. This is crucial for evaluating the performance of machine learning models on unseen data.
Logistic Regression (`sklearn.linear_model.LogisticRegression`)	A classification algorithm used for binary classification tasks. In the context of fake news detection, it predicts whether a news article is "fake" or "real" based on the

	features extracted from the text.
Accuracy Score (`sklearn.metrics.accuracy_score`)	A function used to calculate the accuracy of the model's predictions compared to the true labels. It measures the proportion of correctly classified instances out of the total instances, providing a straightforward performance metric for classification models.

In the below, we are importing “stopwords” from nltk library, stopwords are words that does not add much value to paragraph. These can be the words such as “the”, “a”, or words like “where”, “what” etc. so we need to remove those words from our dataset because it does not add much value to the context of the data.

```
import nltk
nltk.download('stopwords')
```

output:

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

we can print these stopwords in English as below:

```
# printing the stopwords in English
print(stopwords.words('english'))
```

output:

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
"you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself',
'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers',
'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',
'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that',
"that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be',
'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did',
'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as',
'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against',
'between', 'into', 'through', 'during', 'before', 'after', 'above',
'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over',
'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when',
'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most',
'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so',
'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't",
'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain',
'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn',
"doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn',
"isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn',
"needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't",
'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

The next step is collecting and pre-processing the data (here, our dataset we are referring to is train.csv, this dataset is a csv file, csv means comma separated value file), we load this dataset into Pandas DataFrame, Pandas data Frame loads the data set into a more structured table.

```
# loading the dataset to a pandas DataFrame
news_dataset = pd.read_csv('/content/train.csv')
```

lets check the number of rows and columns in the dataset:

```
news_dataset.shape
```

output: (20800, 5)

This indicates that there are 20800 rows, and 5 columns.

We can print the sample of our dataset:

```
# print the first 5 rows of the dataframe
news_dataset.head()
```

output:

	id	title	author	text	label
0	0	House Dem Aide: We Didn’t Even See Comey’s Let...	Darrell Lucus	House Dem Aide: We Didn’t Even See Comey’s Let...	1
1	1	FLYNN: Hillary Clinton, Big Woman on Campus - ...	Daniel J. Flynn	Ever get the feeling your life circles the rou...	0
2	2	Why the Truth Might Get You Fired	Consortiumnews.com	Why the Truth Might Get You Fired October 29, ...	1
3	3	15 Civilians Killed In Single US Airstrike Hav...	Jessica Purkiss	Videos 15 Civilians Killed In Single US Airstr...	1
4	4	Iranian woman jailed for fictional unpublished...	Howard Portnoy	Print \nAn Iranian woman has been sentenced to...	1

Now we need to check whether some values are missing in our dataset, we can do this by using the function isnull().

```
# counting the number of missing values in the dataset
news_dataset.isnull().sum()
```

output:

	0
id	0
title	558
author	1957
text	39
label	0

```
dtype: int64
```

This indicates that there are no missing values in id, 558 news are missing their title values, 1957 are missing their author name values, and so on.

Why these values are missing in our datasetis that while they are being prepared, title of the news, or author of the news, or the text of a particular news may not be found as it is very large dataset.

In our case we have a very large dataset, it is about 20,000 news articles. These missing values are not that much important, so we can drop these missing values or we can replace it with null string. As we have large dataset we can replace missing values with null string (empty string).

```
# replacing the missing values with empty string
news_dataset = news_dataset.fillna('')
```

for our prediction we are going to include title and author, so what we will do is we will combine title and author together, and then we use this data for our prediction. We are not going to use the text because the text is so large that it takes a lot of times for processing. So, we are going to use the title and author column together for our processing or to predict whether the news is real or fake. Using this title and author together yields very good accuracy and performance.

```
# merging the author name and news title
news_dataset['content'] = news_dataset['author']+' '+news_dataset['title']
```

let us print the column that is resulted from merging author and title columns, we named this column as ‘content’ in above.

```
print(news_dataset['content'])
```

output:

```
0        Darrell Lucus House Dem Aide: We Didn’t Even S...
1        Daniel J. Flynn FLYNN: Hillary Clinton, Big Wo...
2        Consortiumnews.com Why the Truth Might Get You...
3        Jessica Purkiss 15 Civilians Killed In Single ...
```

```
4        Howard Portnoy Iranian woman jailed for fictio...
                                ...
20795    Jerome Hudson Rapper T.I.: Trump a ’Poster Chi...
20796    Benjamin Hoffman N.F.L. Playoffs: Schedule, Ma...
20797    Michael J. de la Merced and Rachel Abrams Macy...
20798    Alex Ansary NATO, Russia To Hold Parallel Exer...
20799              David Swanson What Keeps the F-35 Alive
Name: content, Length: 20800, dtype: object
```

This shows that the new column ‘content’ is formed by merging author and title columns together.

Next we will separate the data and label into different content.

```
# separating the data & label
X = news_dataset.drop(columns='label', axis=1)
Y = news_dataset['label']
```

We can print them separately:

```
print(X)
```

output:

```
          id  ...                                            content
0          0  ...  Darrell Lucus House Dem Aide: We Didn’t Even S...
1          1  ...  Daniel J. Flynn FLYNN: Hillary Clinton, Big Wo...
2          2  ...  Consortiumnews.com Why the Truth Might Get You...
3          3  ...  Jessica Purkiss 15 Civilians Killed In Single ...
4          4  ...  Howard Portnoy Iranian woman jailed for fictio...
...      ...  ...                                                ...
20795  20795  ...  Jerome Hudson Rapper T.I.: Trump a ’Poster Chi...
20796  20796  ...  Benjamin Hoffman N.F.L. Playoffs: Schedule, Ma...
20797  20797  ...  Michael J. de la Merced and Rachel Abrams Macy...
20798  20798  ...  Alex Ansary NATO, Russia To Hold Parallel Exer...
20799  20799  ...            David Swanson What Keeps the F-35 Alive

[20800 rows x 5 columns]
```

```
print(Y)
```

output:

```
0        1
1        0
2        1
3        1
4        1
        ..
20795    0
20796    0
20797    0
20798    1
20799    1
Name: label, Length: 20800, dtype: int64
```

Now we will do the stemming procedure, stemming is the process of reducing a word to its Root word, Example: actor, actress, acting --> act, this is very important step because we need to reduce the words as much as possible to have better performance on our model.

Once we do this procedure of reducing the words into its root word, we will perform the function of vectorizing, in vectorizing step we will convert these words to their respective feature vectors, feature vectors are numerical data.

Once we convert the text data to a numerical data, we can feed it to our machine learning model.

PorterStemmer is the function we are going to use for this purpose. We create the variable called port_stem , PorterStemmer function will be loaded to this variable.

```
port_stem = PorterStemmer()
```

In below we will define the function for stemming our text data, removing unnecessary punctuations and special characters from our data, making all letters in our data lowercase, removing stop words. this function takes Input and processes it, and then returns stemmed content.

```
def stemming(content):
    stemmed_content = re.sub('[^a-zA-Z]',' ',content)
    stemmed_content = stemmed_content.lower()
    stemmed_content = stemmed_content.split()
    stemmed_content = [port_stem.stem(word) for word in stemmed_content if
not word in stopwords.words('english')]
    stemmed_content = ' '.join(stemmed_content)
    return stemmed_content
```

```
news_dataset['content'] = news_dataset['content'].apply(stemming)
```

let's print our data after performing these operations on it.

```
print(news_dataset['content'])
```

output:

```
0        darrel lucu hous dem aid even see comey letter...
1        daniel j flynn flynn hillari clinton big woman...
2                   consortiumnew com truth might get fire
3        jessica purkiss civilian kill singl us airstri...
4        howard portnoy iranian woman jail fiction unpu...
                               ...
20795    jerom hudson rapper trump poster child white s...
20796    benjamin hoffman n f l playoff schedul matchup...
20797    michael j de la merc rachel abram maci said re...
20798    alex ansari nato russia hold parallel exercis ...
20799                            david swanson keep f aliv
Name: content, Length: 20800, dtype: object
```

From above output we can see that the text is in lowercase, and punctuations and special characters, stop words are removed.

Since it is rejoined during stemming, again we need to separate our data from label.

```
#separating the data and label
X = news_dataset['content'].values
Y = news_dataset['label'].values
```

In X we have only the content column (this is what we are going to feed our machine learning model) and we can print it using:

```
print(X)
```

output:

```
['darrel lucu hous dem aid even see comey letter jason chaffetz tweet'
 'daniel j flynn flynn hillari clinton big woman campu breitbart'
 'consortiumnew com truth might get fire' ...
 'michael j de la merc rachel abram maci said receiv takeov approach
hudson bay new york time'
 'alex ansari nato russia hold parallel exercis balkan'
 'david swanson keep f aliv']
```

we can print the labels stored in Y:

```
print(Y)
```

output:

```
[1 0 1 ... 0 1 1]
```

we can check the shape of Y:

```
Y.shape
```

Output:

```
(20800,)
```

Still our dataset is in textual form. Computers cannot understand text, so we need to convert it into meaningful numbers that the computers can understand.

For this purpose we will use the vectorizer function TfidfVectorizer. We convert the content that is in X to their respective numbers. Labels in Y are already numbers so we don’t need to convert it (it contains 0 and 1, 0 representing real news and 1 representing fake news).

```
# converting the textual data to numerical data
vectorizer = TfidfVectorizer()
vectorizer.fit(X)

X = vectorizer.transform(X)
```

```
print(X)
```

output:

```
(0, 267)  0.2701012497770876
(0, 2483) 0.36765196867972083
(0, 2959) 0.24684501285337127
(0, 3600) 0.3598939188262558
(0, 3792) 0.27053324808454915
(0, 4973) 0.23331696690935097
(0, 7005) 0.2187416908935914
(0, 7692) 0.24785219520671598
(0, 8630) 0.2921251408704368
(0, 8909) 0.36359638063260746
(0, 13473)      0.2565896679337956
(0, 15686)      0.2848506356272864
(1, 1497) 0.2939891562094648
(1, 1894) 0.15521974226349364
(1, 2223) 0.3827320386859759
(1, 2813) 0.19094574062359204
(1, 3568) 0.26373768806048464
(1, 5503) 0.7143299355715573
(1, 6816) 0.1904660198296849
(1, 16799)      0.30071745655510157
(2, 2943) 0.3179886800654691
(2, 3103) 0.46097489583229645
(2, 5389) 0.3866530551182615
(2, 5968) 0.3474613386728292
(2, 9620) 0.49351492943649944
:   :
(20797, 3643)   0.2115550061362374
(20797, 7042)   0.21799048897828685
(20797, 8364)   0.22322585870464115
(20797, 8988)   0.36160868928090795
(20797, 9518)   0.29542040034203126
(20797, 9588)   0.17455348025522197
(20797, 10306)  0.08038079000566466
(20797, 12138)  0.24778257724396505
(20797, 12344)  0.27263457663336677
(20797, 13122)  0.24825263521976057
(20797, 14967)  0.3115945315488075
(20797, 15295)  0.08159261204402356
(20797, 16996)  0.08315655906109998
(20798, 350)    0.2844693781907258
(20798, 588)    0.3112141524638974
(20798, 1125)   0.4460515589182237
(20798, 5032)   0.40837014502395297
(20798, 6889)   0.3249628569429943
(20798, 10177)  0.31924963701870285
(20798, 11052)  0.4460515589182237
(20798, 13046)  0.2236326748827061
(20799, 377)    0.5677577267055112
(20799, 3623)   0.37927626273066584
(20799, 8036)   0.45983893273780013
```

```
(20799, 14852) 0.5677577267055112
```

From its output, we can see that out text data is converted to numerical format.

We will feed this converted data to our machine learning model, so that it can understand and do the prediction.

first, let’s split the dataset to training and test data:

```
# Splitting the dataset to training & test data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2,
stratify=Y, random_state=2)
```

We have splitted the dataset into train and test data, now we will train our logistic regression model with train data. We have already imported the logistic regression function, so we will use it.

In below, we will initialize logistic regression model, which will be used to classify news articles in your fake news detection system. This model will learn from the training data and later be used to make predictions on new, unseen data.

`model` is a variable that stores an instance of the `LogisticRegression` class from the `sklearn.linear_model` module. By calling LogisticRegression(), we are initializing a new logistic regression model object.

```
model = LogisticRegression()
```

After this, we will train the model using our training dataset by calling the `fit()` method on the `model` object. Here, X_train contains the features (processed text data), and y_train contains the corresponding labels (fake or real).

```
model.fit(X_train, Y_train)
```

output:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
warm_start=False)
```

The output we see after running the line `model.fit(X_train, y_train)` indicates the properties and configuration of the logistic regression model that has been fitted to our training data.

Here's a breakdown of what each part of the output means:

Breakdown of the Output

1. **LogisticRegression**:

   - This indicates that we are using the `LogisticRegression` class from the `sklearn.linear_model` module.

2. **Parameters**:

   - The output lists various parameters of the logistic regression model, each followed by its value. Here's what these parameters mean:

Parameter	**Description**
C	Inverse of regularization strength; smaller values specify stronger regularization (default is 1.0).
Class_weight	Weights associated with classes. If set to `None`, all classes are supposed to have weight one.
Dual	A boolean that specifies whether to solve the dual or primal optimization problem. The default is `False`.
fit_intercept	Indicates whether to include an intercept term in the model. Default is `True`.
intercept_scaling	Scaling of the intercept. Default is 1. Only relevant when `fit_intercept` is `True`.
l1_ratio	The elastic net mixing parameter (used if the penalty is 'elasticnet'). Default is `None`.
max_iter	Maximum number of iterations for the solver to converge. Default is 100.
multi_class	Indicates the strategy to use for multi-class classification. Default is 'auto'.
n_jobs	Number of CPU cores to use for computation. Default is `None`, meaning it uses 1 core.

Penalty	The type of regularization to apply. Default is 'l2', which applies L2 regularization.
random_state	Controls the randomness of the model. Default is `None`, which means results may vary between runs.
solver	The algorithm used for optimization. Default is 'lbfgs', a robust algorithm for small datasets.
tol	Tolerance for stopping criteria. Default is 0.0001.
verbose	Controls the verbosity of the output. Default is 0 (no output).
warm_start	If `True`, reuse the solution of the previous call to fit as initialization for the next call. Default is `False`.

Interpretation

- **Model Configuration**: This output provides a summary of how our logistic regression model is configured. For instance, it shows that the model is using L2 regularization (`penalty='l2'`) and the 'lbfgs' solver for optimization.
- **Regularization**: The value of `C` (1.0) indicates that the model is using a standard level of regularization. If we were to change this value (e.g., to 0.1), it would increase the strength of regularization, potentially reducing overfitting.
- **Multi-class Handling**: The `multi_class='auto'` setting indicates that the model will automatically choose the best approach for handling multi-class classification, which is relevant if we extend our model to classify more than two categories.
- **Max Iterations**: The `max_iter=100` means the optimization algorithm will run for a maximum of 100 iterations to find the best parameters.

In summary, the output we see is a detailed configuration of the logistic regression model we have just trained. It shows the parameter settings that govern how the model behaves during training and prediction.

Evaluation:

The evaluation process after training our model involves making predictions on the test dataset, comparing these predictions to the actual labels. We have already imported the function called accuracy_score, so we will use it to calculate the accuracy score of our model to understand how effective our model is performing.

What happens here is that the model will be asked to predict label values and the model's prediction will be compared to the original label values.

First let's do the prediction on training data.

Accuracy score:

```
# accuracy score on the training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction,
Y_train)
```

Now, we can print the accuracy score:

```
print('Accuracy score of the training data : ',
training_data_accuracy)
```

the output is:

```
Accuracy score of the training data :  0.9863581730769231
```

This shows that the accuracy score is about 98%, this is actually very good. We used very large dataset which contains about 20,000 news articles and we get a very good accuracy score. For binary classification problems logistic regression the best model that's why we get a very good accuracy score.

However, the accuracy score on training data is not that much important, instead it is important on the test data. Because our model is trained on training data, but it has not seen the test data. So the accuracy score on test data will tell us how good our model is performing.

For example: if we take student studying for an exam, this student can practice many questions and train himself for the exam. But how well this student can perform is not determined by the questions he answer on training exam that he practices, but instead it is determined by the exam he has been preparing for.

This is why we have splitted our dataset into training data and testing data.

Now let's find the accuracy on test data:

```
# accuracy score on the test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

Now, we can print the accuracy score for test data:

```
print('Accuracy score of the test data : ', test_data_accuracy)

the output is:
Accuracy score of the test data :  0.9790865384615385
```

We can see that it is almost equal to the accuracy score of the training data, this means our model is performing very well.

We have successfully trained and evaluated our model, and we have got very good accuracy score.

Now we are in final step. What we will do in this step is we are going to build a predictive system. when we give news data to this system it should predict whether the news is real or fake news.

Making a Predictive System:

```
X_new = X_test[3]

prediction = model.predict(X_new)
print(prediction)

if (prediction[0]==0):
  print('The news is Real')
```

```
else:
  print('The news is Fake')
```

```
when we run this it displays the output: The news is Real

let’s check whether this prediction is correct or not, we do
this by printing the actual label of the respective column we
used which is X_test[3]:

# Y_test[3] is the label correspond to the content we used,
which is X_test[3]
print(Y_test[3])

this prints the output: 0
```

so, our model has predicted correctly. We can use different values (example, X_test[0], X_test[1],etc) and check whether the model is predicted correctly or not.

This is how we can make a predictive sytem. when we give a new value to our model it can predict its label and check whether it is real or fake.

So we have successfully built Fake news prediction system.