FAKE NEWS DETECTION USING NLP

**TEAM MEMBER**

**810021106030:HARINI V**

PHASE 4: DEVELOPMENT PART 2

INTRODUCTION :

In the realm of Natural Language Processing (NLP), the task of fake news detection is of paramount importance in the age of information overload and misinformation. Detecting fake news relies on a multi-step process that encompasses text preprocessing, feature extraction, model training, and rigorous evaluation. This introductory guide provides an overview of these essential stages in a fake news detection project

****

**Text Preprocessing:**

Text preprocessing serves as the foundation for effective NLP tasks. In the context of fake news detection, it involves cleaning and transforming raw text data to make it suitable for machine learning models. Key preprocessing steps include:

**Data Collection:**

Gathering a dataset that includes labeled examples of real and fake news articles.

**Text Cleaning:**

Removing irrelevant information, HTML tags, special characters, and noise from the text.

**Tokenization:**

Dividing the text into individual words or tokens to facilitate analysis.

**Lowercasing**:

Ensuring uniformity by converting all text to lowercase.

**Stopword Removal:**

Eliminating common, non-informative words like “the,” “and,” and “is.”

**Stemming or Lemmatization**:

Reducing words to their root forms for normalization.

**Vectorization:**

Converting textual data into numerical form using techniques like TF-IDF or Word Embeddings.

**Feature Extraction:**

Feature extraction is the process of transforming textual data into structured representations that can be used by machine learning models. Key considerations in this phase include:

**Feature Selection:**

Choosing the most relevant features for your model, often based on metrics like chi-squared or information gain.

Word Embeddings: Leveraging pre-trained word embeddings to capture semantic information and relationships between words.

**N-grams:**

Utilizing n-grams (groups of adjacent words) to capture contextual information.

**Topic Modeling:**

Applying techniques like Latent Dirichlet Allocation (LDA) to identify latent topics within the text.

**Model Training and Evaluation:**

The core of fake news detection lies in developing, training, and assessing machine learning models. This stage includes the following components:

**Data Split:**

Dividing the dataset into training and testing subsets to ensure model evaluation.

**Selecting Algorithms:**

Choosing appropriate machine learning algorithms for text classification, including Multinomial Naïve Bayes, Logistic Regression, Random Forest, or deep learning models like LSTM or Transformers.

**Model Training:**

Training the chosen model on the training data, which involves learning the patterns in real and fake news articles.

**Model Evaluation:**

Evaluating the model’s performance using critical metrics such as accuracy, precision, recall, F1-score, and ROC AUC. It also includes creating a confusion matrix to visualize model performance.

**Hyperparameter Tuning:**

Fine-tuning the model’s hyperparameters to optimize its performance.

**Cross-validation:**

Applying cross-validation techniques to ensure the model generalizes well to unseen data.

**Interpretability:**

Exploring model interpretability techniques to understand the decision-making process of the model.

Let’s consider naive bayes algorithm

**FAKE NEWS WITH NAIVE BAYES USING NLP**

**Program:**

**In[1]:**

Import numpy as np # linear algebra

Import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

Import re

Import string

Import nltk

From nltk.corpus import stopwords

From nltk.stem import PorterStemmer

From nltk.tokenize import word\_tokenize

Import matplotlib.pyplot as plt

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

From sklearn.feature\_extraction.text import TfidfVectorizer

From keras.layers import TextVectorization

From keras.utils import pad\_sequences

From xgboost import XGBClassifier

From scipy.sparse import hstack

Import random

**DATA LOADING:**

**In[2]:**

Real\_df = pd.read\_csv(‘/kaggle/input/fake-and-real-news-dataset/True.csv’)

Fake\_df = pd.read\_csv(‘/kaggle/input/fake-and-real-news-dataset/Fake.csv’)

**In[3]:**

Real\_df.info()

**In[4]:**

Fake\_df.info()

**In[5]:**

List(real\_df.sample(5).title)

**Out[5]:**

[‘Renegade colonel surrenders in eastern Congo after clashes, seven dead’,

‘Factbox: Trump meetings include rapper Kanye West, Microsoft founder Bill Gates’,

“At under $5 each, Trump’s votes came cheap”,

“Israeli air strike hits near Syria’s Homs”,

“Trump calls storm over Russia hacking ‘political witchhunt’: NYT”]

**DATA VISUALISATION**

**IN[6]:**

fig = plt.figure(figsize=(5, 5))

labels = ‘Real’, ‘Fake’

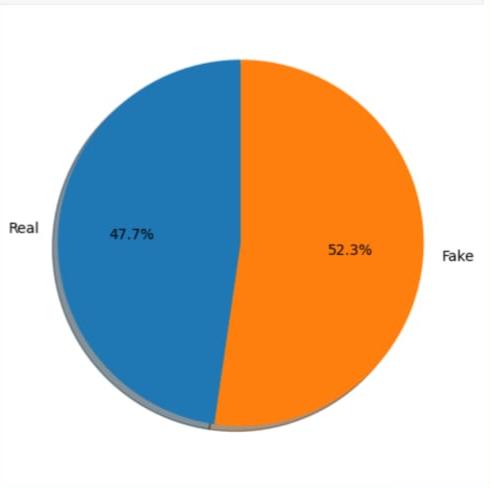
sizes = [len(real\_df), len(fake\_df)]

plt.pie(sizes, labels=labels, autopct=’%1.1f%%’,

Shadow=True, startangle=90)

plt.axis(‘equal’)

plt.show()



**TITLES**

We start by classifying the titles. First we put them in a numpy array as it is easier for us.

**In[7]:**

Positive\_titles = np.array(real\_df.title)

Negative\_titles = np.array(fake\_df.title)

**In[8]:**

Def process\_news(news):

“””Process news function.

Input:

News: a string containing a news’ text or title

Output:

Newss\_clean: a list of words containing the processed news’ text or title

“””

Stemmer = PorterStemmer()

Stopwords\_english = stopwords.words(‘english’)

# remove hyperlinks

News = re.sub(r’https?://[^\s\n\r]+’, ‘’, news)

# tokenize news

#tokenizer = word\_tokenize

News\_tokens = word\_tokenize(news)

News\_clean = []

for word in news\_tokens:

if (word not in stopwords\_english and # remove stopwords

Word not in string.punctuation): # remove punctuation

Stem\_word = stemmer.stem(word) # stemming word

News\_clean.append(stem\_word)

Return news\_clean

**In[9]:**

rand\_id = random.randint(0,len(positive\_titles))

print(positive\_titles[rand\_id],process\_news(positive\_titles[rand\_id]))

**In[10]:**

news\_title = np.concatenate((positive\_titles, negative\_titles), axis=0)

**In[11]:**

positive\_y = np.array(real\_df.Fake\_news)

Negative\_y = np.array(fake\_df.Fake\_news)

Y = np.concatenate((positive\_y, negative\_y), axis=0)

**In[12]:**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(news\_title, y, test\_size=0.20, random\_state=0)

**Building of the words frequency dictionary**

This function builds a dictionary of the different words contained in the training set (title of text) and assign the number of times it was seen in a positive news or a negative news.

**In[13]:**

Def build\_freqs(news, ys):

“””Build frequencies.

Input:

News: a list of news title or texts

Ys: an m x 1 array with the real/fake label of each title/news

(either 0 or 1)

Output:

Freqs: a dictionary mapping each (word, real/fake) pair to its

Frequency

“””

Freqs = {}

for y, new in zip(yslist, news):

for word in process\_news(new):

Pair = (word, y)

if pair in freqs:

Freqs[pair] += 1

else:

Freqs[pair] = 1

return freqs

**Training of the model**

**In[14]:**

Freqs = build\_freqs(news\_title, y)

This function train the model and return the logprior and loglikelihood parameters that will allow us to make futur predictions. For each words this will assign a value “loglikelihood[word]” that will indicate if a word is more likely to be attributed to the fake label (>0) or the real label (<0). The log prior is a regularization parameter that takes a non 0 value if the ratio of fake and real news is not equal to 1.

**In[15]:**

Def train\_naive\_bayes(freqs, train\_x, train\_y):

‘’’

Input:

Freqs: dictionary from (word, label) to how often the word appears

Train\_x: a list of news

Train\_y: a list of labels correponding to the news (0,1)

Output:

Logprior: the log prior. (equation 3 above)

Loglikelihood: the log likelihood of you Naïve bayes equation. (equation 6 above)

‘’’

Loglikelihood = {}

Logprior = 0

Vocab = set([pair[0] for pair in freqs.keys()])

V = len(vocab)

N\_pos = N\_neg = 0

for pair in freqs.keys():

if pair[1] > 0:

N\_pos+=freqs[pair]

else:

N\_neg += freqs[pair]

D = len(train\_y)

D\_pos = np.sum((train\_y == 1)

D\_neg = np.sum((train\_y == 0))

Logprior = np.log(D\_pos) – np.log(D\_neg)

for word in vocab:

Freq\_pos = freqs.get((word,1),0)

Freq\_neg = freqs.get((word,0),0)

P\_w\_pos = (freq\_pos + 1)/(N\_pos +V)

P\_w\_neg = (freq\_neg + 1)/(N\_neg +V)

Loglikelihood[word] = np.log(p\_w\_pos) – np.log(p\_w\_neg)

return logprior, loglikelihood

**In[16]:**

Logprior, loglikelihood = train\_naive\_bayes(freqs, X\_train, y\_train)

**Prediction and accuracy of our model**

**In[17]:**

Def naive\_bayes\_predict(news, logprior, loglikelihood):

‘’’

Input:

News: a string

Logprior: a number

Loglikelihood: a dictionary of words mapping to numbers

Output:

P: the sum of all the logliklihoods of each word in the news (if found in the dictionary) + logprior (a number)word\_l = process\_news(news)

# initialize probability to zero

P = 0

# add the logprior

P += logprior

for word in word\_l:

if word in loglikelihood:

P += loglikelihood[word]

return p

To test our model we look at the difference between each predicted values of the test set and each assigned value of the test set.

**In[18]:**

Def test\_naive\_bayes(test\_x, test\_y, logprior, loglikelihood, naïve\_bayes\_predict=naïve\_bayes\_predict):

“””

Input:

Test\_x: A list of news title/text

Test\_y: the corresponding labels for the list of news title/text

Logprior: the logprior

Loglikelihood: a dictionary with the loglikelihoods for each word

Output:

Accuracy: (# of news title/text classified correctly)/(total # of news title/text)

“””

Accuracy = 0 # return this properly

y\_hats = []

for new in test\_x:

if naïve\_bayes\_predict(new, logprior, loglikelihood) > 0:

y\_hat\_i = 1

else:

y\_hat\_i = 0

y\_hats.append(y\_hat\_i)

error = np.sum((y\_hats != test\_y))/len(test\_y)

accuracy = 1 – error

return accuracy

**In[19]:**

Test\_naive\_bayes(X\_test, y\_test, logprior, loglikelihood)

**Out[19]:**

0.9891982182628062

An accuracy score of almost 99% is very good for a model this simple and fast.

In this last cell you can look at the predicted value and assigned value of random news titles.

**In[20]:**

Rand\_id = random.randint(0,len(X\_test))

Value\_predict = ‘Fake’ if naïve\_bayes\_predict(X\_test[rand\_id], logprior, loglikelihood) > 0 else ‘Real’

Assigned\_value = ‘Fake’ if y\_test[rand\_id] == 1 else ‘Real’

Print(X\_test[rand\_id], ‘Predicted: ‘, Value\_predict, ‘; Assigned value:’, assigned\_value)

COMEDIAN DAVE CHAPPELLE Stuns NY Audience: SLAMS Hillary…Compares TRUMP To The Terminator:”Most Gangsta Candidate Ever” Predicted: Fake ; Assigned value: Fake

**CONCLUSION:**

In the fight against fake news, Naïve Bayes serves as a fundamental tool that can provide a reliable initial assessment of the authenticity of news articles. However, given the evolving nature of misinformation and the ever-increasing volume of data, continuous research and innovation are required to stay ahead in the battle to ensure accurate and trustworthy information in our society. When using Naïve Bayes for fake news detection, it’s essential to view it as one piece of a broader strategy that may include more complex models and ethical considerations to address the multifaceted challenges presented by fake news in the digital age.