FAKE NEWS DETECTION USING NLP

INTRODUCTION:

Fake news detection is a research area that uses machine learning algorithms, especially geometric deep learning and natural language processing, to identify and categorize online content according to its features and sources. AI models such as Grover and Fabula AI can generate and detect fake news by mimicking the language of specific publications and using social spread to spot misinformation. AI can also analyse the headline, subject, geolocation , and main body text of a web story to compare it with other sites and mainstream.

Necessary step to follow:

1. Loading the dataset:

- First, you need a dataset containing labeled examples of real and fake news articles.
- This dataset could be in CSV, JSON, or any other suitable format.You can use libraries like pandas in Python to load data from CSV or Excel files.

Program:

```
import pandas as pd

# Load data from CSV file

data = pd.read_csv('fake_news_dataset.csv')
```

2. Exploring the dataset:

- Understand the structure of your dataset: the columns, data types, and the distribution of real vs. fake news labels.
- Use functions like `head()`, `info()`, and `describe()` in pandas to explore the dataset.

3. Text preprocessing:

- Text cleaning: Remove special characters, links, and irrelevant symbols from the text.
- Tokenization: Split the text into words or tokens.
- Lowercasing: Convert all text to lowercase to ensure consistency.
- Stopword removal: Remove common words like "and," "the," etc., as they don't carry significant meaning.
- Lemmatization or stemming: Reduce words to their base or root form to capture core meaning.

Program:

```
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

def clean_text(text):
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    text = re.sub(r'\s+', ' ', text)
    text = re.sub('[^A-Za-z]', ' ', text)
    text = text.lower()
    words = word_tokenize(text)
    words = [word for word in words if word.isalpha()]
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if word not in stop_words]
    lemmatizer = WordNetLemmatizer()
    words = [lemmatizer.lemmatize(word) for word in words]
    return ' '.join(words)

# Apply the cleaning function to the 'text' column
data['cleaned_text'] = data['text'].apply(clean_text)
```

4. Vectorization:

Convert text data into numerical format that machine learning algorithms can understand. Common techniques include TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings like Word2Vec or GloVe.

Program:

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(max_features=5000)  # You can adjust the number of features based on your dataset size

tfidf_matrix = tfidf_vectorizer.fit_transform(data['cleaned_text'])
```

5. Splitting the dataset:

Split the preprocessed data into training and testing sets to evaluate the model's performance.

Program:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(tfidf_matrix, data['label'], test_size=0.2, random_state=42)
```

Importance of loading and preprocessing dataset:

Importance of loading and processing dataset: Loading and preprocessing the dataset is an important first step in building any machine learning model. However, it is especially important for house price prediction models, as house price datasets are often complex and noisy. By loading and preprocessing the dataset, we can ensure that the machine learning algorithm is able to learn from the data effectively and accurately.

HOW TO OVERCOME THE CHALLENGES OF LOADING AND PREPROCESSING A FAKE NEWS DETECTION USION NLP:

1.Lowercase text & URL removal:

Before we start any of the pre-processing heavy lifting, we want to convert our text to lowercase and remove any URLs in our text. A simple regex expression can handle this for us.

Program:

```
Import re
```

```
text = "http://www.google.com hello world"

text = re.sub(r'http\S+', '', text.lower())

print(text)
```

OUTPUT:

hello world

2.Split contractions:

Similar to URLs, contractions can produce unintended results if left alone. The aptly named contractions python library to the rescue! It looks for contractions and splits them into root words.

Program:

```
Importcontractions
                    def remove_contractions(text):
                      return ' '.join([contractions.fix(word) for word in
                    text.split()])

                    text = """can't won't shouldn't there's mustn't"""
                    print(remove_contractions(text))
                    OUTPUT:
                    can not will not should not there is must not
```

3. Tokenization:

Tokenization is breaking the raw text into small chunks. Tokenization breaks the raw text into words, sentences called tokens. These tokens help in understanding the context or developing the model for the NLP. The tokenization helps in interpreting the meaning of the text by analyzing the sequence of the words.

There are a multitude of ways to implement tokenization and their approaches varied. For our project we utilized RegexpTokenizer within the

NLTK library. Using regular expressions, RegexpTokenizer will match either tokens or separators (i.e. include or exclude regex matches).

Program:

```
import nltk
  from nltk.tokenize import RegexpTokenizer
  # Create tokens out of alphanumeric characters
  tokenizer = RegexpTokenizer(r'\w+')
  tokens = tokenizer.tokenize("I think pineapple pizza is gross and not worth $15!")
  print(tokens)
```

OUTPUT:

```
['I', 'think', 'pineapple', 'pizza', 'is', 'gross', 'and', 'not', 'worth', '15']
```

4. Stemming:

Text normalization is the process of simplifying multiple variations or tenses of the same word. Stemming and lemmatization are two methods of text normalization, the former being the simpler of the two. To stem a word, we simply remove the suffix of a word to reduce it to its root.

Program:

```
#using porter stemmer implementation in nltk
    from nltk.stem import PorterStemmer
    stemmer = PorterStemmer()
   def stem(tokens):
      return [stemmer.stem(token) for token in tokens]
   tokens = ['jumped', 'jumps', 'jumped']
```

```
print(stem(tokens))
```

OUTPUT:

```
['jump', 'jump', 'jump']
```

As an example, “jumping”, “jumps”, and “jumped” all are stemmed to “jump.”

Stemming is not without its faults, however. We can run into the issue of *overstemming*. Overstemming is when words with different meanings are stemmed to the same root — a false positive.

Program:

```
token=[‘universal’, ‘university’, ‘universe’]
        print(stem(tokens))
```

OUTPUT:

```
['univers', 'univers', 'univers']
```

Understemming is also a concern. See how words that should stem to the same root do not — a false negative.

Let’s take a look at a more nuanced approach to text normalization, lemmatization.

Program:

```
token=[‘alumnus’,’alumni’,’alumnae’]
    print(stem(tokens))
```

OUTPUT:

```
['alumnu', 'alumni', 'alumna']
```

5.Lemmatization:

Lemmatization is the process of converting a word to its base form. The difference between stemming and lemmatization is, lemmatization considers the context and converts the word to its meaningful base form, whereas stemming just removes the last few characters, often leading to incorrect meanings and spelling errors.

Lemmatization differs from stemming in that it determines a words part of speech by looking at surrounding words for context. For this example we use nltk.pos_tag to assign parts of speech to tokens. We then pass the token and its assigned tag into WordNetLemmatizer, which decides how to lemmatize the token.

Program:

```
import nltk
from nltk.corpus import wordnet
lemmatizer = WordNetLemmatizer()
# Convert the nltk pos tags to tags that wordnet can recognize
def nltkToWordnet(nltk_tag):
 if nltk_tag.startswith('J'):
  return wordnet.ADJ
 elif nltk_tag.startswith('V'):
  return wordnet.VERB
 elif nltk_tag.startswith('N'):
  return wordnet.NOUN
 elif nltk_tag.startswith('R'):
  return wordnet.ADV
 else:
  return None

# Lemmatize a list of words/tokens
def lemmatize(tokens):
 pos_tags = nltk.pos_tag(tokens)
 res_words = []
 for word, tag in pos_tags:
  tag = nltkToWordnet(tag)
  if tag is None:
   res_words.append(word)
  else:
```

```
        res_words.append(lemmatizer.lemmatize(word, tag))
    return res_words
```

Using the following text we can compare the results of our approaches to stemming and lemmatization.

Program:

text=”it takes a great deal of bravery to stand up to our enemies, but just as much to stand up to our friends”

```
# STEMMING RESULTS

print(stem(tokens))
```

OUTPUT:

['it', 'take', 'a', 'great', 'deal', 'of', 'braveri', 'to', 'stand', 'up', 'to', 'our', 'enemi', 'but', 'just', 'as', 'much', 'to', 'stand', 'up', 'to', 'our', 'friend']

```
# LEMMATIZING RESULTS

print(lemmatize(tokens))
```

OUTPUT:

['it', 'take', 'a', 'great', 'deal', 'of', 'bravery', 'to', 'stand', 'up', 'to', 'our', 'enemy', 'but', 'just', 'as', 'much', 'to', 'stand', 'up', 'to', 'our', 'friend']

Notice that ‘enemies’ was stemmed to ‘enemi’ but lemmatized to ‘enemy’. Interestingly, ‘bravery’ was stemmed to ‘braveri’ but the lemmatizer did not change the original word. In general, lemmatization is more precise, but at the expense of complexity.

6.Stop Word Removal:

Stop words are words in the text which do not add any meaning to the sentence and their removal will not affect the processing of text for the defined purpose. They are removed from the vocabulary to reduce noise and to reduce the dimension of the feature set.

Program:

```
Import nltk

    nltk.download('words') #download list of english words

    nltk.download('stopwords') #download list of stopwords

    from nltk.corpus import stopwords

    stopWords = stopwords.words('english')

    englishWords = set(nltk.corpus.words.words())

    def remove_stopWords(tokens):

      return [w for w in tokens if (w in englishWords and w not in
    stopWords)]


    tokens =
    ['the','quick','brown','fox','jumped','over','the','lazy','dog']

    print(remove_stopWords(tokens))

    OUTPUT:

    ['quick', 'brown', 'fox', 'lazy', 'dog']
```

CONCLUSION:

Fake news detection techniques can be divided into those based on style and those based on content, or fact-checking. Too often it is assumed that bad style (bad spelling, bad punctuation, limited vocabulary, using terms of abuse, ungrammaticality, etc.) is a safe indicator of fake news.More than ever, this is a case where the machine's opinion must be backed up by clear and fully verifiable indications for the

basis of its decision, in terms of the facts checked and the authority by which the truth of each fact was determined.