FAKE NEWS DETECTION USING NNLP

BY BOOBALAN.S-411421205008

(B.TECH/Information Technology,3rd year)

Domain name: Artificial Intelligence

Project: To design a fake news detection using NLP(Datasets& training model):

What is "Fake News"? “Fake news” is a term that has come to mean different things to different people. At its core, we are defining “fake news” as those news stories that are false: the story itself is fabricated, with no verifiable facts, sources or quotes. Sometimes these stories may be propaganda that is intentionally designed to mislead the reader, or may be designed as “clickbait” written for economic incentives (the writer profits on the number of people who click on the story). In recent years, fake news stories have proliferated via social media, in part because they are so easily and quickly shared online.

- FAKE news detection using NLP problem statement

Problem Statement: Fake News Detection using NLP:

Background:

The spread of fake news and misinformation has become a significant challenge in today's digital age. Fake news can have serious consequences, including influencing public opinion, causing panic, and even inciting violence. Natural Language Processing (NLP) techniques can be employed to develop systems that automatically detect and classify news articles as either real or fake.

Problem:

Design and implement a fake news detection system using NLP techniques. The system should take a news article or text as input and classify it as either "Real" or "Fake."

Key Components:

- Data Collection: Gather a diverse dataset of news articles labeled as real or fake. The dataset should cover a range of topics and sources.
- Preprocessing: Clean and preprocess the text data. This may include tokenization, stop word removal, stemming or lemmatization, and handling of special characters and numbers.
- Feature Extraction: Use NLP techniques to extract meaningful features from the text, such as TF-IDF (Term Frequency-Inverse Document Frequency) vectors, word embeddings, or other relevant representations.
- Model Selection: Choose an appropriate machine learning or deep learning model for classification. Common models include Logistic Regression, Naive Bayes, Random Forest, or LSTM (Long Short-Term Memory) for deep learning.
- Training: Train the selected model on the labeled dataset. Implement techniques to handle class imbalance if necessary.
- Evaluation: Evaluate the model's performance using metrics like accuracy, precision, recall, F1-score, and confusion matrices.
- Deployment: Develop an application or platform where users can input news articles, and the model can classify them as real or fake.
- Continuous Learning: Implement mechanisms for updating the model over time with new data to adapt to evolving fake news patterns.

Challenges:

Dealing with an imbalanced dataset where fake news might be significantly less prevalent.Handling different types of fake news, such as misinformation, satire, and manipulated content.Detecting fake news in multiple languages.Real-time processing and scalability for a large number of news articles.

Success Criteria:

The system should achieve a high accuracy and other relevant evaluation metrics in classifying news articles as real or fake. It should also be user-friendly and easily accessible to the public or news organizations to assist in identifying and countering fake news.

Future Work:

In the future, the system can be extended to include more advanced NLP techniques, such as sentiment analysis, credibility scoring, and source verification, to provide a more comprehensive fake news detection solution.

Remember to adapt the problem statement based on the specific goals and resources available for your project.

Design thinking about fake news detection using NLP

Design Thinking for Fake News Detection using NLP involves a structured approach to solving the problem creatively and innovatively. Here's a step-by-step design thinking process for this project:

1. Empathize:

Understand the end-users and stakeholders' perspectives, including journalists, fact-checkers, and the general public.

Conduct interviews, surveys, or workshops to gather insights into the challenges they face in identifying fake news.

2. Define:

Clearly define the problem statement, its scope, and objectives.Prioritize key challenges, such as real-time detection, multilingual support, and handling various types of fake news.

3. Ideate:

Brainstorm solutions and features that could address the identified challenges.Encourage creative thinking, and consider a wide range of ideas for fake news detection using NLP.

4. Prototype:

Create a low-fidelity prototype of the fake news detection system.Develop a basic user interface and the NLP components for text processing and classification.Test the prototype with a small group of users and gather feedback.

5. Test:

Gather user feedback on the prototype's usability and functionality.Iterate on the design based on user input and refine the system's features and NLP algorithms.Test the system with a diverse dataset of news articles.

6. Develop:

Build the full-scale fake news detection system based on the refined prototype.

Implement data collection, preprocessing, feature extraction, and model training components.

Integrate NLP libraries and tools for text analysis and classification.

7. Test Again:

Conduct extensive testing, including unit testing, integration testing, and user acceptance testing.

Evaluate the system's performance, identifying areas that require improvement.

8. Deploy:

Deploy the system on a reliable and scalable platform, ensuring it can handle real-time news article inputs.

Implement a user-friendly interface for easy access by end-users.

9. Continuous Improvement:

Establish mechanisms for continuous learning and model updates to adapt to evolving fake news tactics.

Encourage user feedback and actively seek ways to improve the system based on real-world usage.

10. Monitoring and Evaluation:

Implement monitoring to track the system's performance and detect anomalies.

Evaluate the system's effectiveness using metrics like accuracy, precision, recall, and F1-score.

11. Education and Awareness:

Educate users and the general public on how to use the system effectively to identify fake news.

Raise awareness about the importance of critical thinking and source verification.

12. Collaboration:

Collaborate with fact-checking organizations, news outlets, and other stakeholders to improve the system's accuracy and reach.

13. Feedback Loop:

Establish a feedback loop with users and stakeholders to gather ongoing input and ensure the system continues to meet their needs.

Remember that design thinking is an iterative process. It's important to continuously revisit and refine each step as you learn more about the users and the evolving landscape of fake news. Your ultimate goal should be to create an effective, user-centric, and adaptable NLP-based fake news detection system.

2 / 2

Data preprocessing steps :

Data preprocessing is a critical step in fake news detection using NLP. It involves cleaning and transforming raw text data into a format suitable for machine learning models. Here are the key data preprocessing steps:

Text Cleaning:

Remove any HTML tags or special characters that might not be relevant for analysis.

Convert text to lowercase to ensure consistency in text processing.

Tokenization:

Break the text into individual words or tokens. Tokenization is essential for further NLP processing.

Stop Word Removal:

Remove common stop words (e.g., "the," "and," "is") that don't carry significant meaning and can be found in both real and fake news articles.

Stemming or Lemmatization:

Reduce words to their base or root form to normalize variations (e.g., "running" becomes "run").

You can choose either stemming (more aggressive) or lemmatization (preserves the word's meaning).

Removing Numbers and Symbols:

Remove numeric values and punctuation marks as they might not be informative for fake news detection.

Handling Missing Data:

Address missing or null values in the dataset. Depending on the severity, you might choose to fill them or remove the corresponding data points.

Handling Imbalanced Data:

If your dataset has a significant class imbalance (more real news than fake or vice versa), consider oversampling, undersampling, or using techniques like Synthetic Minority Over-sampling Technique (SMOTE).

Feature Extraction:

Convert the preprocessed text into numerical features. Common techniques include:

TF-IDF (Term Frequency-Inverse Document Frequency) vectorization: Measures the importance of a word in a document relative to the entire corpus.

Word Embeddings (e.g., Word2Vec, GloVe): Represent words as dense vectors capturing semantic relationships.

Vector Normalization:

Normalize the numerical feature vectors to have a consistent scale. Common methods include L2 normalization (Euclidean normalization).

Data Splitting:

Divide the dataset into training, validation, and test sets to evaluate your fake news detection model effectively.

Encoding Labels:

Convert the labels (real or fake) into a numerical format (e.g., 0 for real, 1 for fake) for model training.

Data Balancing (Optional):

Depending on the outcome of handling imbalanced data, ensure that the training dataset is balanced.

Dimensionality Reduction (Optional):

In the case of high-dimensional data, consider techniques like Principal Component Analysis (PCA) to reduce the dimensionality while preserving important information.

Save Preprocessed Data:

Save the preprocessed data in a format suitable for model training. This might be a CSV file, a database, or any other preferred data storage method.

These preprocessing steps are essential for preparing the text data for training machine learning or deep learning models for fake news detection. The choice of techniques and the order of operations may vary depending on your specific dataset and goals, so it's important to adapt them to your project's needs.

- **Feature extraction technique**

Feature extraction is a crucial step in fake news detection using NLP. It involves transforming the preprocessed text data into numerical features that machine

learning models can understand. Here are some common feature extraction techniques for fake news detection:

TF-IDF (Term Frequency-Inverse Document Frequency):

TF-IDF is a widely used technique that measures the importance of a word in a document relative to the entire corpus. It assigns a weight to each term based on its frequency in the document and its rarity in the corpus. High TF-IDF values indicate the importance of a term in a specific document.

Word Embeddings:

Word embeddings are dense vector representations of words in a continuous vector space. Popular word embedding techniques include Word2Vec, GloVe, and FastText. These embeddings capture semantic relationships between words and can be used as features.

Bag of Words (BoW):

BoW represents a document as an unordered collection of words, ignoring grammar and word order. It counts the frequency of each word in the document, creating a sparse vector of word counts.

N-grams:

N-grams are contiguous sequences of N words from a given document. Bigrams (2-grams) and trigrams (3-grams) can capture local word patterns and are often used as features.

Word Frequency:

Create features based on the frequency of specific words or phrases. For example, you can count the occurrences of certain words associated with fake news, like "hoax," "conspiracy," or "unverified."

Part-of-Speech (POS) Tagging:

Extract features based on the distribution of parts of speech in a document. This can help identify grammatical patterns that distinguish real and fake news.

Sentiment Analysis:

Analyze the sentiment of the text using sentiment lexicons or machine learning models. The sentiment of the text can be a valuable feature in fake news detection.

Named Entity Recognition (NER):

Recognize named entities in the text, such as names of people, organizations, and locations. The presence of certain named entities might indicate the credibility of a news source.

Readability Metrics:

Calculate readability metrics like Flesch-Kincaid or Gunning Fog Index to measure the complexity of the text. Fake news may exhibit distinct readability patterns.

Topic Modeling:

Apply topic modeling techniques like Latent Dirichlet Allocation (LDA) to identify topics in the text. The distribution of topics can serve as features.

Syntax-based Features:

Extract syntactic features, such as the use of passive voice, sentence length, or punctuation patterns. Fake news might exhibit specific syntactic characteristics.

Graph-based Features:

Analyze text as a graph and extract features based on graph properties, such as centrality or connectivity.

The choice of feature extraction technique depends on the specific characteristics of your dataset and the goals of your fake news detection model. It's common to experiment with multiple techniques and assess their effectiveness in improving model accuracy, precision, and recall. Feature selection methods can also be employed to choose the most relevant features for the task.

Machine learning algorithm for fake news detection using NLP

Fake news detection using NLP involves classifying text data as real or fake. Several machine learning algorithms can be used for this task, depending on the complexity of the problem and the size of your dataset. Here are some commonly used machine learning algorithms for fake news detection:

Logistic Regression:

Logistic regression is a simple and interpretable algorithm. It's a good starting point for binary classification tasks like fake news detection. It works well with text-based features, especially when combined with TF-IDF or word embeddings.

Naive Bayes:

Naive Bayes classifiers, such as Multinomial Naive Bayes, are known for their effectiveness in text classification tasks. They assume that features are conditionally independent, which is a simplification, but they can work well for fake news detection.

Random Forest:

Random Forest is an ensemble learning method that combines multiple decision trees. It can handle high-dimensional data and capture complex patterns in text. It's robust and less prone to overfitting.

Support Vector Machines (SVM):

SVM is suitable for text classification, especially when combined with kernel functions like the linear kernel or the radial basis function (RBF) kernel. SVM aims to find a hyperplane that best separates real and fake news.

Gradient Boosting Algorithms:

Gradient boosting techniques like XGBoost, LightGBM, and CatBoost have been successful in text classification tasks. They can handle imbalanced datasets and provide high predictive accuracy.

Neural Networks:

Deep learning models, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), can be used for fake news detection. RNNs, especially Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), are suitable for sequential data like text.

BERT (Bidirectional Encoder Representations from Transformers):

BERT-based models, such as RoBERTa and DistilBERT, have achieved state-of-the-art results in NLP tasks, including fake news detection. Fine-tuning a pre-trained BERT model on your dataset can be highly effective.

Ensemble Models:

Combining multiple models through techniques like stacking or voting can improve fake news detection accuracy. For example, you can combine the predictions of logistic regression, random forest, and an LSTM-based model.

Multimodal Models:

If your dataset includes not only text but also images or videos, you can explore multimodal models that integrate both text and visual features for a more comprehensive approach to fake news detection.

The choice of machine learning algorithm depends on various factors, including the size of your dataset, the quality of your features, and the computational resources available. It's common to experiment with different algorithms and fine-tune hyperparameters to achieve the best performance. Additionally, consider using evaluation metrics like accuracy, precision, recall, F1-score, and ROC AUC to assess the effectiveness of the chosen algorithm.

Training module

Creating a training module for fake news detection using NLP involves several steps to prepare, train, and evaluate your machine learning model. Here's a structured approach to building a training module for fake news detection:

1. Data Preparation:

Data Collection: Gather a diverse and labeled dataset of news articles, with labels indicating whether each article is real or fake.

Data Preprocessing: Preprocess the text data, as described earlier, including text cleaning, tokenization, stop word removal, and feature extraction using techniques like TF-IDF or word embeddings.

Data Splitting: Divide the dataset into training, validation, and test sets. Common ratios are 70-15-15 or 80-10-10, depending on the size of your dataset.

2. Model Selection and Architecture:

Choose an appropriate machine learning or deep learning model for fake news detection. Options include logistic regression, Naive Bayes, random forest, SVM, LSTM, or BERT-based models.

Design the architecture of the chosen model, including the input layer, hidden layers, and output layer. For deep learning models, consider the network's depth and complexity.

3. Model Training:

Train the selected model on the training data using suitable training algorithms and loss functions. Be mindful of overfitting, and use techniques like dropout or early stopping to mitigate it.

Fine-tune hyperparameters, including learning rate, batch size, and regularization strength, to optimize the model's performance.

4. Evaluation:

Assess the model's performance using a range of evaluation metrics, including accuracy, precision, recall, F1-score, and ROC AUC. These metrics provide a comprehensive view of your model's effectiveness.

Use confusion matrices to visualize the model's classification results, highlighting true positives, true negatives, false positives, and false negatives.

5. Model Optimization:

If the initial model performance is not satisfactory, consider model optimization techniques. You can explore different architectures, fine-tune hyperparameters, or apply techniques like class weighting to handle imbalanced datasets.

6. Cross-Validation (Optional):

Implement k-fold cross-validation to assess the model's performance more robustly. This is particularly useful when dealing with smaller datasets.

7. Model Interpretability (Optional):

Investigate techniques for model interpretability to understand which features and patterns the model uses to make predictions. This can help in identifying why certain articles are classified as fake or real.

8. Save and Deploy the Model:

Once you're satisfied with the model's performance, save the trained model to a file for future use. Deploy the model in your fake news detection system.

9. Continuous Learning:

Implement mechanisms for model updates to adapt to evolving fake news tactics. Periodically retrain the model with new data to maintain its accuracy.

10. Documentation:

Document the entire training process, including data sources, preprocessing steps, model architecture, hyperparameters, and evaluation results. Clear documentation is essential for reproducibility.

Remember that fake news detection is an ongoing challenge, and your model may require periodic updates and retraining to remain effective. Additionally, consider collaborating with domain experts, fact-checking organizations, and stakeholders to enhance the model's accuracy and relevance.

Evaluation steps

Evaluating a fake news detection system using NLP is essential to assess its performance and ensure its effectiveness. Here are the key evaluation steps for fake news detection:

1. Data Splitting:

Split your dataset into three parts: a training set, a validation set, and a test set. Common ratios are 70-15-15 or 80-10-10, depending on the size of your dataset.

2. Model Training and Validation:

Train your fake news detection model on the training set. Use the validation set to fine-tune hyperparameters and prevent overfitting. Common metrics to monitor during this phase include loss and accuracy on the validation set.

3. Evaluation Metrics:

Use a combination of evaluation metrics to assess the model's performance comprehensively. Common metrics include:

Accuracy: The proportion of correctly classified news articles.

Precision: The ratio of true positives to the total predicted positives. It measures the ability to correctly identify fake news.

Recall: The ratio of true positives to the total actual positives. It measures the ability to find all instances of fake news.

F1-Score: The harmonic mean of precision and recall, providing a balance between the two metrics.

ROC AUC (Receiver Operating Characteristic Area Under the Curve): It measures the model's ability to distinguish between real and fake news across various thresholds.

Confusion Matrix: Visualize the model's performance with true positives, true negatives, false positives, and false negatives.

4. Threshold Selection:

Depending on the application and the trade-offs between precision and recall, choose an appropriate classification threshold. Adjusting the threshold can impact the balance between false positives and false negatives.

5. Cross-Validation (Optional):

Implement k-fold cross-validation to assess the model's performance more robustly, especially if you have a limited dataset. Cross-validation helps ensure that the results are not biased by a specific data split.

<u>6. Baseline Models:</u>Compare the performance of your NLP-based model to baseline models, such as random guessing or a simple rule-based classifier. This provides context for understanding the quality of your model.

<u>7. Evaluation on Test Set:</u>

Once you're satisfied with the model's performance on the validation set, evaluate it on the independent test set. This provides a final assessment of how well your model generalizes to new, unseen data.

<u>8. Class-Imbalanced Data</u>:

If your dataset is imbalanced (i.e., significantly more real news than fake news), consider using appropriate metrics and techniques for imbalanced datasets. These include precision-recall curves, class weighting, or resampling strategies.

<u>9. Model Interpretability:</u>

Consider using techniques for model interpretability to understand which features or words are influencing the model's decisions. Interpretability is essential for identifying why specific articles are classified as fake or real.

<u>10. Reporting and Documentation:</u>

Clearly report the results of your evaluation, including the evaluation metrics, the chosen threshold, and any insights gained from model interpretation.

<u>11. Continuous Learning:</u>

Regularly reevaluate your model's performance and update it as necessary to adapt to evolving fake news patterns. Keep documentation of model updates.

Effective evaluation ensures that your fake news detection system using NLP is accurate and reliable in classifying news articles, helping to combat the spread of misinformation.

<u>DATASET:</u>

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set_style('darkgrid')

import nltk
from sklearn.preprocessing import LabelBinarizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from wordcloud import STOPWORDS,WordCloud
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize,sent_tokenize
from bs4 import BeautifulSoup
import re,string,unicodedata


from keras.preprocessing import text,sequence
from nltk.tokenize.toktok import TokTokTokenizer
from sklearn.metrics import
classification_report,confusion_matrix,accuracy_score
from sklearn.model_selection import train_test_split
from string import punctuation
from nltk import pos_tag
from nltk.corpus import wordnet
import keras
from keras.models import Sequential
from keras.layers import LSTM,Dense,Dropout,Embedding
from keras.callbacks import ReduceLROnPlateau
import tensorflow as tf
```

Loading the data
Having imported all the necessary libraries , now we will go ahead and load our data.

```
real_news=pd.read_csv('../input/fake-and-real-news-dataset/True.csv')
fake_news=pd.read_csv('../input/fake-and-real-news-dataset/Fake.csv')
```

Let's take a sneak peak at our data !

```
real_news.head()
title	text	subject	date
0	As U.S. budget fight looms, Republicans flip t...	WASHINGTON (Reuters) - The head of a conservat... politicsNewsDecember 31, 2017
1	U.S. military to accept transgender recruits o...	WASHINGTON (Reuters) - Transgender people will... politicsNewsDecember 29, 2017
2	Senior U.S. Republican senator: 'Let Mr. Muell...	WASHINGTON (Reuters) - The special counsel inv...	politicsNewsDecember 31, 2017
3	FBI Russia probe helped by Australian diplomat...	WASHINGTON (Reuters) - Trump campaign adviser ...politicsNewsDecember 30, 2017
4	Trump wants Postal Service to charge 'much mor...	SEATTLE/WASHINGTON (Reuters) - President Donal...	politicsNews	December 29, 2017
fake_news.head()
title	text	subject	date
0	Donald Trump Sends Out Embarrassing New Year'...	Donald Trump just couldn t wish all Americans ...	News	December 31, 2017
1	Drunk Bragging Trump Staffer Started Russian ...	House Intelligence Committee Chairman Devin Nu...	News	December 31, 2017
2	Sheriff David Clarke Becomes An Internet Joke...	On Friday, it was revealed that former Milwauk...	News	December 30, 2017
3	Trump Is So Obsessed He Even Has Obama's Name...	On Christmas day, Donald Trump announced that ... News	December 29, 2017
4	Pope Francis Just Called Out Donald Trump Dur...	Pope Francis used his annual Christmas Day mes...	News	December 25, 2017
```

We will now combine both of these data and add a column of 'Isfake' so that we can use all the data as once and the 'Isfake' column will also be our target column , which will determine if the news is fake or not.

```
real_news*'Isfake'+=0
```

```
fake_news*'Isfake'+=1
```
Using conactenate function of pandas :

```
df=pd.concat(*real_news,fake_news+)
```
So how does our data look now ?

```
df.sample(5)
title    text    subject         date   Isfake
19919COMMUNIST George Soros Says Trump Will Win Pop...     George Soros: Here I have to confess to a lit...  left-news       Sep 26, 2016          1
17566BREAKING NEWS: Leftist Media Publishes Major F...          How many times in one week can ABC News publis...      left-news      Dec 5, 2017 1
12093Brexit will not be derailed, says May ahead of...       LONDON (Reuters) - Prime Minister Theresa May ...  worldnews   December 17, 2017          0
15561Catalonia's ex-leader granted freedom to campa. .. BRUSSELS/MADRID (Reuters) - Catalonia s former... worldnews November 6, 2017 0
11132LIBERAL MEDIA IGNORES MELANIA’S Visit To Home ...       First Lady Melania Trump visits HomeSafe, phot...        politics         Apr 15, 2017          1
```
Are there any null values?

```
df.isnull().sum()
title      0
text       0
subject    0
date       0
Isfake     0
dtype: int64
```
As there are no null values , we are saved from the hassle of making up for the missing values. Now we will visualize the data.

Visualizing the data

How many of the given news are fake and how many of them are real?

```
sns.countplot(df.Isfake)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1bce38fc90>
```

The number of fake and real news are almost equal.
Now let us see how many unqiue titles are there. Are any of the titles repeated?

```
df.title.count()
44898
```

How many subjects are there ? We can see that using value_counts()

```
df.subject.value_counts()
politicsNews      11272
worldnews         10145
News               9050
politics           6841
left-news          4459
Government News    1570
US_News             783
Middle-east         778
Name: subject, dtype: int64
```

Let's see how much of the news in different subject are fake !

```
plt.figure(figsize=(10,10))
chart=sns.countplot(x='subject',hue='Isfake',data=df,palette='muted')
chart.set_xticklabels(chart.get_xticklabels(),rotation=90,fontsize=10)
*Text(0, 0, 'politicsNews'),
 Text(0, 0, 'worldnews'),
 Text(0, 0, 'News'),
 Text(0, 0, 'politics'),
 Text(0, 0, 'Government News'),
 Text(0, 0, 'left-news'),
 Text(0, 0, 'US_News'),
 Text(0, 0, 'Middle-east')+
```

Now we will place all of the required columns in one and delete all the not-so-required columns.

```
df*'text'+= df*'text'++ " " + df*'title'+
del df*'title'+
del df*'subject'+
del df*'date'+
```

We are done with this now , we shall head towards cleaning our data!

Cleaning the data

Our data may consist URLs , HTML tags which might make it difficult for our model to predict properly. To prevent that from happening we will clean our data so as to make our model more efficient.

We will be removing punctuation , stopwords,URLS, html tags from our text data.
For this we shall use beautifulsoup and re library which we imported earlier.

```
stop_words=set(stopwords.words('english'))
punctuation=list(string.punctuation)
stop_words.update(punctuation)
def string_html(text):
   soup=BeautifulSoup(text,"html.parser")
   return soup.get_text()

def remove_square_brackets(text):
   return re.sub('\**^++*\+','',text)

def remove_URL(text):
   return re.sub(r'http\S+','',text)

def remove_stopwords(text):
   final_text=*+
   for i in text.split():
       if i.strip().lower() not in stop_words:
           final_text.append(i.strip())
   return " ".join(final_text)

def clean_text_data(text):
   text=string_html(text)
   text=remove_square_brackets(text)
   text=remove_stopwords(text)
   text=remove_URL(text)
   return text
```

Now that we have defined the cleaning functions , let us use em' on our text data.

```
df*'text'+=df*'text'+.apply(clean_text_data)
```

We are all done with cleaning and have with us cleaned text data now.Next up are some awesome wordclouds.

Frequent Words

I wonder what words were the most used in fake news and real news and i guess you do too!

So let's see what these frequent words are , and for that we will use wordcloud.

Let's see the fake news texts first !

```
plt.figure(figsize=(20,20))
wordcloud=WordCloud(stopwords=STOPWORDS,height=600,width=1200).generate(" ".join(df*df.Isfake==1+.text))
plt.imshow(wordcloud,interpolation='bilinear')
<matplotlib.image.AxesImage at 0x7f1bcc7dd810>
```

Now what about the real news?

```
plt.figure(figsize=(20,20))
wordcloud=WordCloud(stopwords=STOPWORDS,height=600,width=1200).gene
rate(" ".join(df*df.Isfake==0+.text))
plt.imshow(wordcloud,interpolation='bilinear')
<matplotlib.image.AxesImage at 0x7f1bcc7cde10>
```

Those were some nice wordclouds , and clearly Donald Trump , United States , etc were very frequent.

Tokenization

We shall now tokenize our data ,i.e convert the text data into vectors.

```
X_train,X_test,y_train,y_test=train_test_split(df.text,df.Isfake,random_state=0)
max_features=10000
max_len=300
```

To tokinize our data , I am using tokenizer here. There are other ways to tokenize data , you can also try them out.

Here is what is happening in the next code tab:

First we initialized the tokenizer with it's size being 10k.
Then we fit the training data on this tokenizer.
Then we convert the text to sequences and save it in X_train variable.
Lastly we add a padding layer around our sequence.
Here is a example of what tokenizer does

```
tokenizer=text.Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(X_train)
tokenizer_train=tokenizer.texts_to_sequences(X_train)
X_train=sequence.pad_sequences(tokenizer_train,maxlen=max_len)
tokenizer_test=tokenizer.texts_to_sequences(X_test)
X_test=sequence.pad_sequences(tokenizer_test,maxlen=max_len)
```

Now we will import our GLOVE file , I am using the 100d version here.

```
glove_file='../input/glove-twitter/glove.twitter.27B.100d.txt'
```

Now we will get the coefficients from the glove file and save it in embedding index variable.

```
def get_coefs(word, *arr):
   return word, np.asarray(arr,dtype='float32')
embeddings_index=dict(get_coefs(*o.rstrip().rsplit(' ')) for o in
open(glove_file,encoding="utf8"))
```

What's happening in the next code tab:

We first take all the values of the embeddings and store it in all_embs.
Then we take the mean and standard deviation of all the embeddings.
We now take the word indices using .word_index function of tokenizer.
Then we will see what the length of each vector would be and save it in nb_words.
We make an embedding matrix.

```
all_embs=np.stack(embeddings_index.values())
emb_mean,emb_std=all_embs.mean(),all_embs.std()
emb_size=all_embs.shape*1+

word_index=tokenizer.word_index
nb_words=min(max_features,len(word_index))

embedding_matrix =
np.random.normal(emb_mean,emb_std,(nb_words,emb_size))

for word,i in word_index.items():
   if i>=max_features: continue
   embedding_vector=embeddings_index.get(word)
   if embedding_vector is not None: embedding_matrix*i+=embedding_vector
/opt/conda/lib/python3.7/site-
packages/IPython/core/interactiveshell.py:3254: FutureWarning: arrays to
stack must be passed as a "sequence" type such as list or tuple. Support for
non-sequence iterables such as generators is deprecated as of NumPy 1.16 and
will raise an error in the future.
```

```
  if (await self.run_code(code, result,  async_=asy)):
```

Building our model
We have succesfully done the tokenization part , let's build our model now!
Here are the parameters I'm taking.

```
batch_size=256
epochs=10
emb_size=100
```

Let's initialize our callback.

```
leaning_rate_reduction=ReduceLROnPlateau(monitor='val_accuracy',patience=2,verbose=10,factor=0.5,min_lr=0.00001)
```

Let's build our model.Here are the layers I'm using:

Starting with an embedding layer
Then 3 LSTM layers
Then 2 Dense layers
I am using Adam optimizer for our model.

```
model=Sequential()
model.add(Embedding(max_features,output_dim=emb_size,weights=*embedding_matrix+,input_length=max_len,trainable=False))
model.add(LSTM(units=256,return_sequences=True,recurrent_dropout=0.25,dropout=0.25))
model.add(LSTM(units=128,return_sequences=True,recurrent_dropout=0.25,dropout=0.25))
model.add(LSTM(units=64,recurrent_dropout=0.1,dropout=0.1))
model.add(Dense(units=32,activation='relu'))
model.add(Dense(1,'sigmoid'))
model.compile(optimizer=keras.optimizers.Adam(lr=0.01),loss='binary_crossentropy',metrics=*'accuracy'+)
model.summary()
Model: "sequential"
_________________________________________________________________
```

```
_
Layer (type)              Output Shape            Param #
=============================================================
=
embedding (Embedding)      (None, 300, 100)        1000000
_____________________________________________________________
_
lstm (LSTM)                (None, 300, 256)        365568
_____________________________________________________________
_
lstm_1 (LSTM)              (None, 300, 128)        197120
_____________________________________________________________
_
lstm_2 (LSTM)              (None, 64)              49408
_____________________________________________________________
_
dense (Dense)              (None, 32)              2080
_____________________________________________________________
_
dense_1 (Dense)            (None, 1)               33
=============================================================
=
Total params: 1,614,209
Trainable params: 614,209
Non-trainable params: 1,000,000
_____________________________________________________________
_
```

Let's train our model now !

```
history=model.fit(X_train,y_train,batch_size=batch_size,validation_data=(X_tes
t,y_test),epochs=epochs,callbacks=*leaning_rate_reduction+)
Epoch 1/10
132/132 *==============================+ - 452s 3s/step - loss: 0.3564 -
accuracy: 0.8259 - val_loss: 0.0776 - val_accuracy: 0.9769 - lr: 0.0100
Epoch 2/10
```

```
132/132 *==============================+ - 451s 3s/step - loss: 0.0360 -
accuracy: 0.9894 - val_loss: 0.0184 - val_accuracy: 0.9933 - lr: 0.0100
Epoch 3/10
132/132 *==============================+ - 458s 3s/step - loss: 0.0237 -
accuracy: 0.9929 - val_loss: 0.0155 - val_accuracy: 0.9952 - lr: 0.0100
Epoch 4/10
132/132 *==============================+ - 457s 3s/step - loss: 0.0127 -
accuracy: 0.9955 - val_loss: 0.0174 - val_accuracy: 0.9941 - lr: 0.0100
Epoch 5/10
132/132 *==============================+ - 453s 3s/step - loss: 0.0138 -
accuracy: 0.9958 - val_loss: 0.0144 - val_accuracy: 0.9962 - lr: 0.0100
Epoch 6/10
132/132 *==============================+ - 460s 3s/step - loss: 0.0079 -
accuracy: 0.9975 - val_loss: 0.0117 - val_accuracy: 0.9972 - lr: 0.0100
Epoch 7/10
132/132 *==============================+ - 461s 3s/step - loss: 0.0081 -
accuracy: 0.9976 - val_loss: 0.0147 - val_accuracy: 0.9955 - lr: 0.0100
Epoch 8/10
132/132 *==============================+ - ETA: 0s - loss: 0.0051 -
accuracy: 0.9983
Epoch 00008: ReduceLROnPlateau reducing learning rate to
0.004999999888241291.
132/132 *==============================+ - 461s 3s/step - loss: 0.0051 -
accuracy: 0.9983 - val_loss: 0.0103 - val_accuracy: 0.9972 - lr: 0.0100
Epoch 9/10
132/132 *==============================+ - 454s 3s/step - loss: 0.0035 -
accuracy: 0.9989 - val_loss: 0.0088 - val_accuracy: 0.9979 - lr: 0.0050
Epoch 10/10
132/132 *==============================+ - 455s 3s/step - loss: 0.0035 -
accuracy: 0.9989 - val_loss: 0.0075 - val_accuracy: 0.9985 - lr: 0.0050
```

Let's see our model in action ! ;)

```
pred = model.predict_classes(X_test)
pred*5:10+
array(**0+,
       *0+,
       *1+,
       *0+,
       *1++, dtype=int32)
```

Analyzing our model
Let's see how the accuracy and loss graphs of our model look now !

```
epochs = *i for i in range(10)+
fig , ax = plt.subplots(1,2)
train_acc = history.history*'accuracy'+
train_loss = history.history*'loss'+
val_acc = history.history*'val_accuracy'+
val_loss = history.history*'val_loss'+
fig.set_size_inches(20,10)

ax*0+.plot(epochs,train_acc,'go-',label='Training    Accuracy')
ax*0+.plot(epochs,val_acc,'ro-',label='Validation    Accuracy')
ax*0+.set_xlabel('Epochs')
ax*0+.set_ylabel('Accuracy')
ax*0+.legend()

ax*1+.plot(epochs,train_loss,'go-',label='Training    Loss')
ax*1+.plot(epochs,val_loss,'ro-',label='Validation    Loss')
ax*1+.set_xlabel('Loss')
ax*1+.set_ylabel('Accuracy')
ax*1+.legend()

plt.show()
```

We will now see how many of the samples were wrongly predicted using the confusion matrix.

```
cm=confusion_matrix(y_test,pred)
cm=pd.DataFrame(cm,index=*'Fake','Not  Fake'+,columns=*'Fake','Not  Fake'+)
```

```
cm
Fake   Not Fake
Fake   5353 14
Not Fake       3         5855
plt.figure(figsize=(10,10))
sns.heatmap(cm,cmap="Blues",linecolor='black',linewidth=1,annot=True,fmt=''
,xticklabels=*'Fake','Not  Fake'+,yticklabels=*'Fake','Not  Fake'+)
plt.xlabel('Actual')
plt.ylabel('Predicted')
Text(69.0, 0.5, 'Predicted')
```

Now what is our accuracy on Test and Train set?

```
print(f'Accuracy of the model on Training Data is - ,
model.evaluate(X_train,y_train)*1+*100:.2f-')
print(f'Accuracy of the model on Testing Data is -
,model.evaluate(X_test,y_test)*1+*100:.2f-')
1053/1053 *==============================+ - 219s 208ms/step - loss:
4.7703e 04 - accuracy: 0.9999
Accuracy of the model on Training Data is - 99.99
351/351 *==============================+ - 72s 206ms/step - loss: 0.0075
- accuracy: 0.9985
Accuracy of the model on Testing Data is -  99.85
```

Some last words:
Thank you for reading! I'm still a beginner and want to improve myself in every way I can. So if you have any ideas to feedback please let me know in the comments section!