Fake news detection using NLP

Introduction

In today's fast-paced digital world, spreading fake news has become a significant concern. With the increasing ease of access to social media platforms and other online sources of information, it has become more challenging to distinguish between real and fake news. In this project-based article, we will learn how to build a machine-learning model to detect fake news accurately.

Learning objectives:

1. Understand the basics of natural language processing (NLP) and how it can be used to preprocess textual data for machine learning models.
2. Learn how to use the CountVectorizer class from the scikit-learn library to convert text data into numerical feature vectors.
3. Build a fake news detection system using machine learning algorithms such as logistic regression and evaluate its performance.

This article was published as a part of the Data Science Blogathon.

Table of Contents

1. Project Description
2. Problem Statement
3. Prerequisites
4. Dataset Description
5. Data Collection and Exploration

6. Text Preprocessing
7. Lowercasing the Text
8. Removing Punctuation and Digits
9. Removing Stop Words
10. Stemming or Lemmatizing the Text
11. Model Training
12. Model Evaluation
13. Improving the Model
14. Model Deployment

Project Description

The spread of fake news has become a major concern in today's society, and it is important to be able to identify news articles that are not based on facts or are intentionally misleading. In this project, we will use machine learning to classify news articles as either real or fake based on their content. By identifying fake news articles, we can prevent the spread of misinformation and help people make more informed decisions.

This project is relevant to the media industry, news outlets, and social media platforms that are responsible for sharing news articles. Classifying news articles as real or fake can help these organizations improve their content moderation and reduce the spread of fake news.

Problem Statement

This project aims to classify news articles as real or fake based on their content. Specifically, we will use machine learning to build a model to predict whether a given news article is real or fake based on its text.

Prerequisites

To complete this project, you should understand Python programming, data manipulation, visualization libraries such as Pandas and Matplotlib, and machine learning libraries such as Scikit-Learn. Additionally, some background knowledge of natural language processing (NLP) techniques and text classification methods would be helpful.

Dataset Description

The dataset used in this project is the "Fake and real news dataset" available on Kaggle, which contains 50,000 news articles labeled as either real or fake. The dataset was collected from various news websites and has been preprocessed to remove extraneous content such as HTML tags, advertisements, and boilerplate text. The dataset provides features such as each news article's title, text, subject, and publication date. The dataset can be downloaded from the following link: https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset.

The steps we will follow in this project are:

1. Data collection and exploration
2. Text preprocessing

3. Feature extraction
4. Model training and evaluation
5. Deployment

1. Data Collection and Exploration

For this project, we will use the Fake and Real News Dataset available on Kaggle. The dataset contains two CSV files: one with real news articles and another with fake news articles. You can download the dataset from this link: https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset

Once you have downloaded the dataset, you can load it into a Pandas DataFrame.
The *'real_news'* DataFrame contains real news articles and their labels, and the *'fake_news'* DataFrame contains fake news articles and their labels. Let's take a look at the first few rows of each DataFrame to get an idea of what the data looks like::

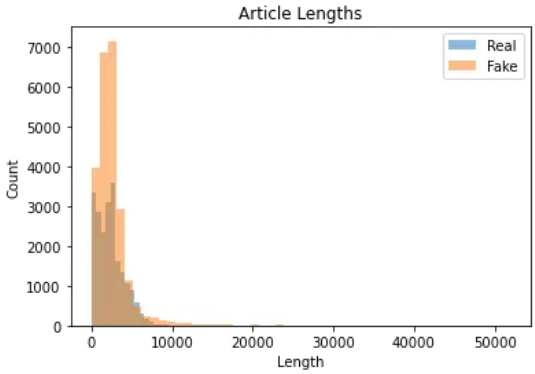
Python Code:

As we can see, the data contains several columns: the title of the article, the text of the article, the subject of the article, and the date it was published. We will be using the title and text columns to train our model.

Before we can start training our model, we need to do some exploratory data analysis to get a sense of the data. For example, we can plot the distribution of article lengths in each dataset using the following code:

```
import matplotlib.pyplot as plt real_lengths = real_news['text'].apply(len) fake_lengths = fake_news['text'].apply(len) plt.hist(real_lengths, bins=50, alpha=0.5, label='Real') plt.hist(fake_lengths, bins=50, alpha=0.5, label='Fake') plt.title('Article Lengths') plt.xlabel('Length') plt.ylabel('Count') plt.legend() plt.show()
```

The output should look something like this:



As we can see, the length of the articles is highly variable, with some articles being very short (less than 1000 characters) and others being quite long (more than 40,000 characters). We will need to take this into account when preprocessing the text.

We can also look at the most common words in each dataset using the following code:

```
from collections import Counter import nltk #downloading stopwords and punkt nltk.download('stopwords') nltk.download('punkt') def get_most_common_words(texts, num_words=10): all_words = [] for text in texts: all_words.extend(nltk.word_tokenize(text.lower())) stop_words = set(nltk.corpus.stopwords.words('english')) words = [word for word in all_words if word.isalpha() and word not in stop_words] word_counts = Counter(words) return word_counts.most_common(num_words) real_words = get_most_common_words(real_news['text']) fake_words = get_most_common_words(fake_news['text']) print('Real News:', real_words) print('Fake News:', fake_words)
```

The output should look something like this:

```
Real News: [('trump', 32505), ('said', 15757), ('us', 15247), ('president', 12788), ('would', 12337), ('people', 10749), ('one', 10681), ('also', 9927), ('new', 9825), ('state', 9820)] Fake News: [('trump', 10382), ('said', 7161), ('hillary', 3890), ('clinton', 3588), ('one', 3466), ('people', 3305), ('would', 3257), ('us', 3073), ('like', 3056), ('also', 3005)]
```

As we can see, some of the most common words in both datasets are related to politics and the current US president, Donald Trump. However, there are some differences between the two datasets, with the fake news dataset containing more references to Hillary Clinton and a greater use of words like “like”.

Model Performance without removing stopwords(used logistic regression)

```
Accuracy: 0.9953 Precision: 0.9940 Recall: 0.9963 F1 Score: 0.9951
```

2. Text Preprocessing

Before we can start training our model, we need to preprocess the text data. The preprocessing steps we will perform are:

1. Lowercasing the text
2. Removing punctuation and digits
3. Removing stop words
4. Stemming or lemmatizing the text

Lowercasing the Text

Lowercasing the text refers to converting all the letters in a piece of text to lowercase. This is a common text preprocessing step that can be useful for improving the accuracy of text classification models. For example, “Hello” and “hello” would be considered two different words by a model that does not account for case, whereas if the text is converted to lowercase, they would be treated as the same word.

Removing Punctuation and Digits

Removing punctuation and digits refers to removing non-alphabetic characters from a text. This can be useful for reducing the complexity of the text and making it easier for a model to analyze. For example, the words “Hello,” and “Hello!” would be considered different words by a text analysis model if it doesn’t account for the punctuation.

Removing Stop Words

Stop words are words that are very common in a language and do not carry much meaning, such as “the”, “and”, “in”, etc. Removing stop words from a piece of text can help reduce the dimensionality of the data and focus on the most important words in the text. This can also help improve the accuracy of a text classification model by reducing noise in the data.

Stemming or Lemmatizing the Text

Stemming and lemmatizing are common techniques for reducing words to their base form. Stemming involves removing the suffixes of words to produce a stem or root word. For example, the word “jumping” would be stemmed to “jump.” This technique can be useful for reducing the dimensionality of the data, but it can sometimes result in stems that are not actual words.

Conversely, Lemmatizing involves reducing words to their base form using a dictionary or morphological analysis. For example, the word “jumping” would be lemmatized to “jump”, which is an actual word. This technique can be more accurate than stemming but also more computationally expensive.

Both stemming and lemmatizing can reduce the dimensionality of text data and make it easier for a model to analyze. However, it is important to note that they can sometimes result in loss of information, so it is important to experiment with both techniques and determine which works best for a particular text classification problem.

We will perform these steps using the NLTK library, which provides various text-processing tools.

```
from nltk.corpus import stopwords from nltk.tokenize import word_tokenize from nltk.stem import PorterStemmer, WordNetLemmatizer import string nltk.download('wordnet') stop_words = set(stopwords.words('english')) stemmer = PorterStemmer() lemmatizer = WordNetLemmatizer() def preprocess_text(text): # Lowercase the text text = text.lower() # Remove punctuation and digits text = text.translate(str.maketrans('', '', string.punctuation + string.digits)) # Tokenize the text words = word_tokenize(text) # Remove stop words words = [word for word in words if word not in stop_words] # Stem or lemmatize the words words = [stemmer.stem(word) for word in words] # Join the words back into a string text = ' '.join(words) return text
```

We can now apply this preprocessing function to each article in our datasets:

```
real_news['text'] = real_news['text'].apply(preprocess_text) fake_news['text'] = fake_news['text'].apply(preprocess_text)
```

3. Model Training

We can train our model now that we have preprocessed our text data. We will use a simple bag-of-words approach, representing each article as a vector of word frequencies. We will use the *CountVectorizer* class from the *sklearn* library to convert the preprocessed text into feature vectors.

CountVectorizer is a commonly used text preprocessing technique in natural language processing. It transforms a collection of text documents into a matrix of word counts. Each row in the matrix represents a document, and each column represents a word in the document collection.

The CountVectorizer converts a collection of text documents into a matrix of token counts. It works by first tokenizing the text into words and then counting the frequency of each word in each document. The resulting matrix can be used as input to machine learning algorithms for tasks such as text classification.

The CountVectorizer has several parameters that can be adjusted to customize the text preprocessing. For example, the “stop_words” parameter can be used to specify a list of words that should be removed from the text before counting. The “max_df” parameter can specify the maximum document frequency for a word, beyond which the word is considered a stop word and removed from the text.

One advantage of CountVectorizer is that it is simple to use and works well for many types of text classification problems. It is also very efficient regarding memory usage, as it only stores the frequency counts of each word in each document. Another advantage is that it is easy to interpret, as the resulting matrix can be directly inspected to understand the importance of different words in the classification process.

Other methods for converting textual data into numerical features include TF-IDF (term frequency-inverse document frequency), Word2Vec, Doc2Vec, and GloVe (Global Vectors for Word Representation).

TF-IDF is similar to CountVectorizer, but instead of just counting the frequency of each word, it considers how often the word appears in the entire corpus and assigns a weight to each word based on how important it is in the document.

Word2Vec and Doc2Vec are methods for learning low-dimensional vector representations of words and documents that capture the underlying semantic relationships between them.

GloVe is another method for learning vector representations of words that combines the advantages of TF-IDF and Word2Vec.

Each method has its advantages and disadvantages, and the choice of method depends on the problem and dataset at hand. For this dataset, we are using CountVectorizer as follows:

```
from sklearn.feature_extraction.text import CountVectorizer import scipy.sparse as sp import numpy as np vectorizer = CountVectorizer() X_real = vectorizer.fit_transform(real_news['text']) X_fake = vectorizer.transform(fake_news['text']) X = sp.vstack([X_real, X_fake]) y = np.concatenate([np.ones(X_real.shape[0]), np.zeros(X_fake.shape[0])])
```

Here, we first create a *CountVectorizer* object and *fit* it to the preprocessed text in the real news dataset. We then use the same vectorizer to transform the preprocessed text in the fake news dataset. We then stack the feature matrices for both datasets vertically and create a corresponding label vector, y.

Now that we have our feature and label vectors, we can split the data into training and testing sets:

```
from sklearn.model_selection import train_test_split X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

We can now train our model using a logistic regression classifier:

```
from sklearn.linear_model import LogisticRegression clf = LogisticRegression(random_state=42) clf.fit(X_train, y_train)
```

4. Model Evaluation

Now that we have trained our model, we can evaluate its performance on the test set. We will use our evaluation metrics for accuracy, precision, recall, and F1 score.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score y_pred = clf.predict(X_test) accuracy = accuracy_score(y_test, y_pred) precision = precision_score(y_test, y_pred) recall = recall_score(y_test, y_pred) f1 = f1_score(y_test, y_pred) print('Accuracy:', accuracy) print('Precision:', precision) print('Recall:', recall) print('F1 Score:', f1)
```

The output should look something like this:

```
Accuracy: 0.992522617676591 Precision: 0.9918478260869565 Recall: 0.9932118684430505 F1 Score: 0.9925293344993434
```

As we can see, our model performs very well, with an accuracy of over 99%.

Our dataset achieved a test accuracy of over 99%, indicating that the model can accurately classify news articles as real or fake.

Improving the Model

While our logistic regression model achieved high accuracy on the test set, there are several ways we could potentially improve its performance:

- Feature engineering: Instead of using a bag-of-words approach, we could use more advanced text representations, such as word embeddings or topic models, which may capture more nuanced relationships between words.
- Hyperparameter tuning: We could tune the hyperparameters of the logistic regression model using methods such as grid search or randomized search to find the optimal set of parameters for our dataset.

```
from sklearn.naive_bayes import MultinomialNB from sklearn.linear_model import LogisticRegression from sklearn.svm import SVC # Define a function to train and evaluate a model def train_and_evaluate_model(model, X_train, y_train, X_test, y_test): # Train the model on the training data model.fit(X_train, y_train) # Predict the labels for the testing data y_pred = model.predict(X_test) # Evaluate the model accuracy = accuracy_score(y_test, y_pred) precision = precision_score(y_test, y_pred, average='weighted') recall = recall_score(y_test, y_pred, average='weighted') f1 = f1_score(y_test, y_pred, average='weighted') # Print the evaluation metrics print(f"Accuracy: {accuracy:.4f}") print(f"Precision: {precision:.4f}") print(f"Recall: {recall:.4f}") print(f"F1-score: {f1:.4f}") # Train and evaluate a Multinomial Naive Bayes model print("Training and evaluating Multinomial Naive Bayes model...") nb = MultinomialNB() train_and_evaluate_model(nb, X_train, y_train, X_test, y_test) print() # Train and evaluate a Support Vector
```

```
Machine model print("Training and evaluating Support Vector Machine model...") svm = SVC() train_and_evaluate_model(svm, X_train, y_train, X_test, y_test)
```

And the results are:

I have added a code snippet to tune hyperparameters using GridSearchCV. You also use RandomSearchCV or BayesSearchCV to tune the hyperparameters.

```
from sklearn.model_selection import GridSearchCV # Define a list of hyperparameters to search over hyperparameters = { 'penalty': ['l1', 'l2'], 'C': [0.1, 1, 10, 100], 'solver': ['liblinear', 'saga'] } # Perform grid search to find the best hyperparameters grid_search = GridSearchCV(LogisticRegression(), hyperparameters, cv=5) grid_search.fit(X_train, y_train) # Print the best hyperparameters and test accuracy print('Best hyperparameters:', grid_search.best_params_) print('Test accuracy:', grid_search.score(X_test, y_test))
```

Experimenting with these methods may improve our model's accuracy even further.

Saving our model:

```
from joblib import dump dump(clf, 'model.joblib') dump(vectorizer, 'vectorizer.joblib')
```

The dump function from the joblib library can be used to save the clf model to the model.joblib file. Once the model is saved, it can be loaded in other Python scripts using the load function, as shown in the previous answer.

5. Model Deployment

Finally, we can deploy our model as a web application using the Flask framework. We will create a simple web form where users can input text, and the model will output whether the text is likely to be real or fake news.

```
from flask import Flask, request, render_template from joblib import load from nltk.corpus import stopwords from nltk.tokenize import word_tokenize from nltk.stem import PorterStemmer, WordNetLemmatizer import string stop_words = set(stopwords.words('english')) stemmer = PorterStemmer() lemmatizer = WordNetLemmatizer() clf = load('model.joblib') vectorizer = load('vectorizer.joblib') def preprocess_text(text): # Lowercase the text text = text.lower() # Remove punctuation and digits text = text.translate(str.maketrans('', '', string.punctuation + string.digits)) # Tokenize the text words = word_tokenize(text) # Remove stop words words = [word for word in words if word not in stop_words] # Stem or lemmatize the words words = [stemmer.stem(word) for word in words] # Join the words back into a string text = ' '.join(words) return text app = Flask(__name__) @app.route('/') def home(): return render_template('home.html') @app.route('/predict', methods=['POST']) def predict(): text = request.form['text'] preprocessed_text = preprocess_text(text) X = vectorizer.transform([preprocessed_text]) y_pred = clf.predict(X) if y_pred[0]== 1: result = 'real' else: result = 'fake' return render_template('result.html', result=result, text=text) if __name__ == '__main__': app.run(debug=True)
```

We can save the above code in a file named `app.py.` We also need to create two HTML templates, `home.html` and `result.html`, containing the HTML code for the home page and the result page, respectively.

home.html

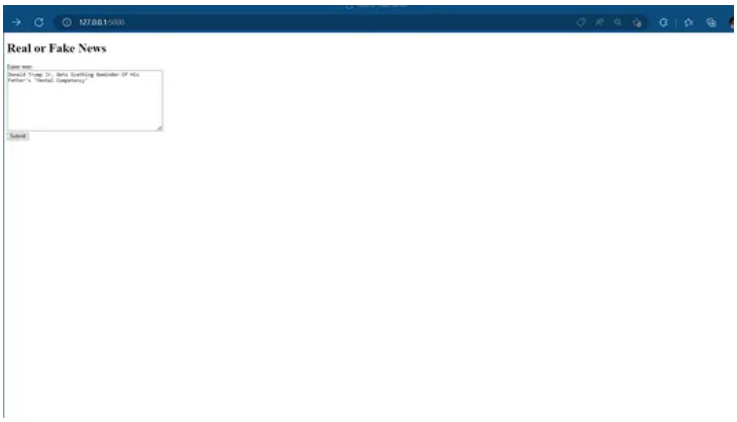
```
<!DOCTYPE html> <html> <head> <title>Real or Fake News</title> </head> <body> <h1>Real or Fake News</h1> <form action="/predict" method="post"> <label for="text">Enter text:</label><br> <textarea name="text" rows="10" cols="50"></textarea><br> <input type="submit" value="Submit"> </form> </body> </html>
```

result.html

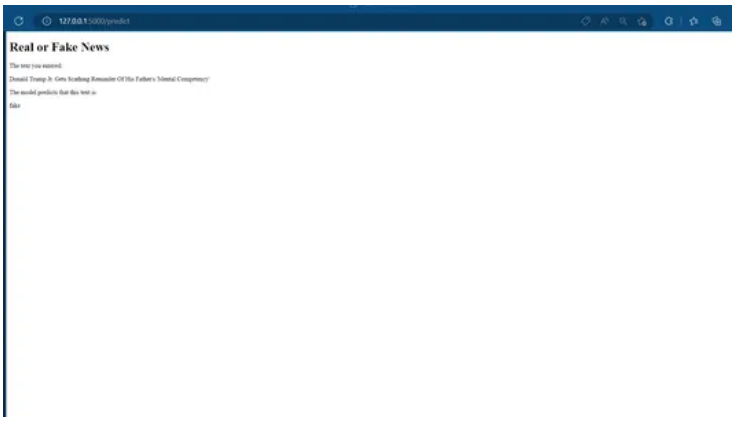
```
<!DOCTYPE html> <html> <head> <title>Real or Fake News</title> </head> <body> <h1>Real or Fake News</h1> <p>The text you entered:</p> <p>{{ text }}</p> <p>The model predicts that this text is:</p> <p>{{ result }} </p> </body> </html>
```

We can now run the Flask app using the command python app.py in the command line. The app should be accessible at http://localhost:5000.

Home Page:



Predict Page:



Related Work

In this section, we discuss some previous work that is related to fake news detection. Fake news can be defined as fabricated information that mimics news media content in form but not in organizational process or intent [1].

In recent years, a lot of automated fake news detection methods have been proposed. For example, Shu, Kai, et al. [2] provided numerous methods to solve the problem of fake news classification, such as user-based, knowledge-based, social network-based, style-based methods, etc. Julio, et al. [3] presented a new set of features and measured the prediction performance of current approaches and features for automatic detection of fake news. Daniel, et al. [4] focused on the analysis of information credibility on Twitter. Heejung, et al. [5] applied the Bidirectional Encoder Representations from Transformers model (BERT) model to detect fake news by analyzing the relationship between the headline and the body text of news.

Mohammad Hadi, et al. [6] applied different levels of n-grams for feature extraction based on the ISOT fake news dataset. Saqib, et al. [7] proposed an ensemble classification model for the detection of fake news that has achieved a better accuracy compared to the state-of-the-art also on the ISOT fake news dataset. Sebastian, et al. [8] used a neural network-based approach to perform text analysis and fake news detection on ISOT fake news dataset as well.

Methodology

The process of fake news detection can be divided into four stages - data preprocessing, word embedding, models, and model fine-tuning.

The Dataset

Link: https://www.uvic.ca/engineering/ece/isot/datasets/fake-news/index.php

The dataset we use is the ISOT Fake News dataset introduced by ISOT Research Lab at University of Victoria in Canada [9]. This dataset is a compilation of several thousand fake news and truthful articles, obtained from different legitimate news sites and sites flagged as unreliable by Politifact.com. To get insight into this dataset, we visualized it with word clouds for real and fake news respectively. Figure 1(a). shows the word cloud of the real news in the dataset, and Figure 1(b). shows the one of the fake news in the dataset.

Figure 1(a).

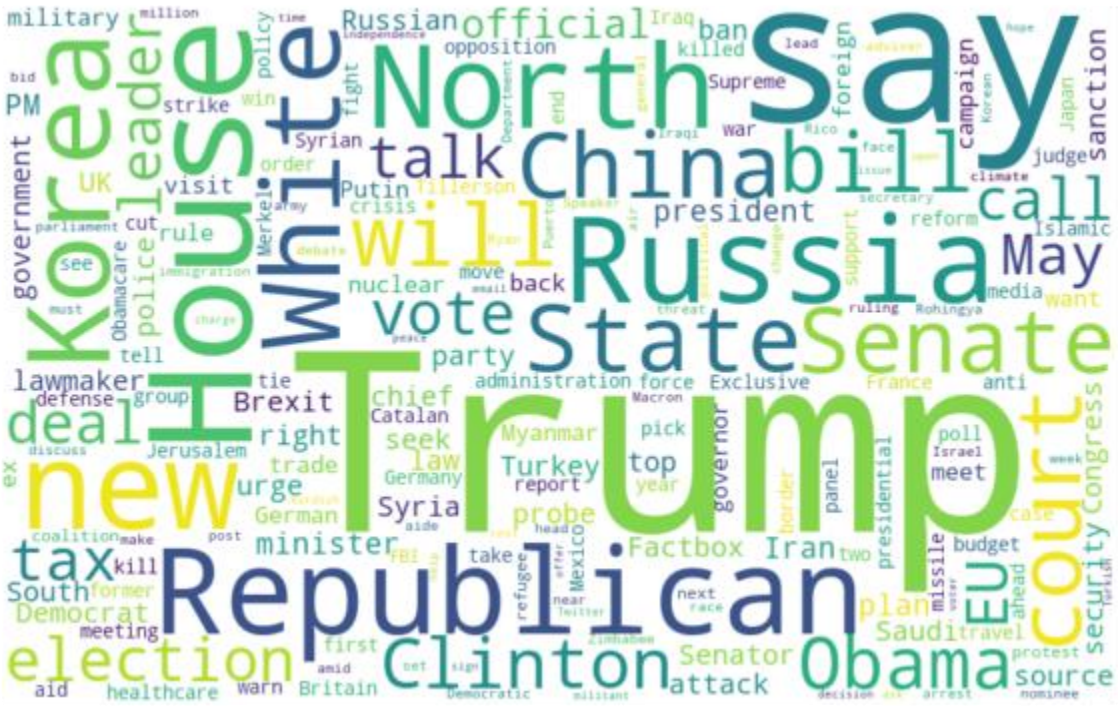
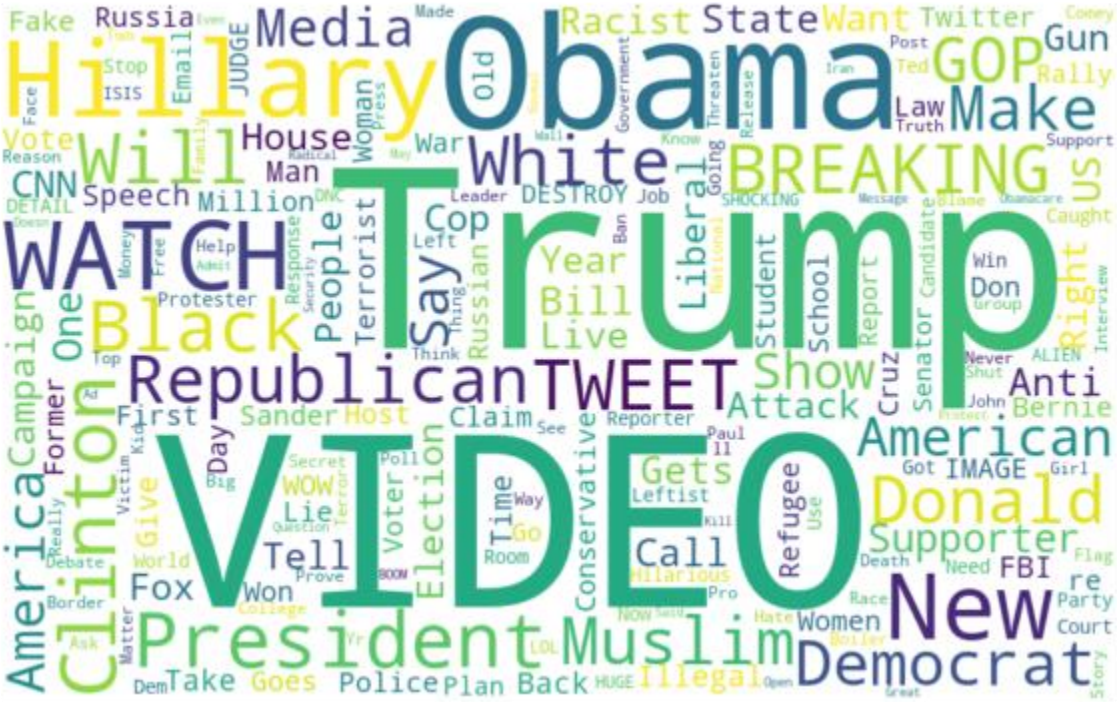


Figure 1(b).



We can see that, in the real news word cloud, 'Trump', 'say', 'Russia', 'House', 'North', and Korea' appeared frequently; while in fake news one, 'VIDEO', 'Trump', 'Obama', 'WATCH', and 'Hillary' appeared the most frequently. 'Say' appears frequently in real news but does not in fake news. 'VIDEO', and 'WATCH' appear frequently in fake news but do not in real news. From these two word clouds, we can get some important information to differentiate the two classes of data. The original form of the dataset is two CSV files containing fake and real news respectively. We combined the dataset and split it into training, validation, and test sets with shuffling at the ratio of 64%:16%:20%. The original combined dataset contains 44,898 pieces of data, and Table 1. shows the distribution of data in the training, validation, and test sets.

Table 1. Distribution of Data

Training	Validation	Test
64%	16%	20%
28734	7184	8980

Data Preprocessing

The main goal of this part is to use NLP techniques to preprocess the input data and prepare for the next step to extract the proper features.
The data we use contains news titles and texts. Each of the titles is about 12.45 words long, while each of the texts is about 405.28 words long. In our project, we only use the titles for the fake news detection because the texts are too large for us to train efficiently. Also, the text contains too many details and information for a piece of news, which may distract the models during training.
We built a preprocessing pipeline for each statement to eliminate the noise in the fake news dataset. The preprocessing pipeline includes the following 3 sub-parts:

1. Replaced characters that are not between a to z or A to Z with whitespace.
2. Converted all characters into lower-case ones.
3. Removed the inflectional morphemes like “ed”, “est”, “s”, and “ing” from their token stem. Ex: confirmed → “confirm” + “ -ed”

We also cropped the titles into sentences with a maximum length of 42 in order to train the model on a dataset with sentences of reasonable lengths, and also eliminate titles with an extreme length that may let the model fit on unbalanced data.

Word Embedding

This part is important because we need to convert the dataset into a form that models can handle. We use different types of word embedding for different models we built. For LSTM, Bidirectional LSTM, and CNN, we first create a Tokenizer to tokenize The words and create sequences of tokenized words. Next, we zero-padded each sequence to make the length of it 42. Then, we utilized the Embedding layer that initialized with random weights to let it learn an embedding for all of the words in the training dataset. The Embedding layer [10] will convert the sequence into a distributed representation, which is a sequence of dense, real-valued vectors. For BERT, we utilized BERT tokenizer to tokenize the news titles in the dataset first. BERT uses Google NMT’s WordPiece Tokenization [11] to separate the words into smaller pieces to deal with words that are not contained in the dictionary. For example, embedding is divided into ['em', '##bed', '##ding'].
Also, there are two other special tokens that BERT needed, which are [CLS] and [SEP]. [CLS] is put at the beginning of a sentence representing the front of an input series. [SEP] is put at the middle of two sentences when they are combined to a single input series, or put at the end of a sentence if the input series is only one sentence.

Then, for the input of BERT, we need to convert the original statements into three kinds of tensors, which are token tensors, segment tensors, mask tensors. Token tensors represent the indices of tokens, which are obtained by the tokenizer. Segment tensors represent the identification of different sentences. Mask tensors represent the concentration of tokens including information after zero-padding the data into the same length.

Models

- **LSTM**
  Long-Short Term Memory (LSTM) is an advanced version of Recurrent Neural Network (RNN), which makes it easier to remember past data in memory. LSTM is a well-suited model for sequential data, such as data for NLP problems. Thus, we utilized LSTM to perform fake news detection.

- **Bidirectional LSTM**
  Bidirectional LSTM (BiLSTM) [12] consists of two LSTMs: one taking the input from a forward direction, and the other in a backward direction. BiLSTM effectively increases the amount of information available to the network, improving the context available to the algorithm.

- **CNN-BiLSTM**
  In this section, we used Convolutional Neural Network (CNN) as the upper layer of the bidirectional LSTM. That is, the output of the CNN is the input of the BiLSTM. This architecture extracts the maximum number of features and information of the input text with convolutional layers, and also utilizes the bidirectional benefit of BiLSTM to ensure that the network can output based on its entire input text.

- **BERT**
  In the Natural Language Processing field, Transformers become more and more dominant. BERT [13], the acronym for Bidirectional Encoder Representations from Transformers, is a transformer-based machine learning technique that changed the NLP world in recent years due to its state-of-the-art performance. Its two main features are that it is a deep transformer model so that it can process lengthy sentences effectively using the 'attention' mechanism, and it is bidirectional so that it will output based on the entire input sentence.
  We used BERT to handle the dataset and construct a deep learning model by fine-tuning the bert-based-uncased pre-trained model for fake news detection. Training a model for natural language processing is costly and time-consuming

because of the large number of parameters. Fortunately, we have pre-trained models of BERT that enable us to conduct transfer learning efficiently. We choose the pre-trained model of bert-base-uncased from a lot of models with different kinds of parameters. The chosen one consists of a base amount of parameters and does not consider cases of letters (upper-case and lower-case).

Model Fine-Tuning

For different downstream tasks, we need to conduct different fine-tuning approaches. Thanks to HuggingFace, we have the models for different downstream tasks. In our project of fake news detection, which is the classification of statements, we used bertForSequenceClassification to fine-tune our pre-trained BERT model. The modules of the model contain a BERT module handling various embeddings, a BERT transformer encoder, a BertPooler, a dropout layer, and a linear classifier that returns logits of the 2 classes.

Experiments

Imbalanced Data

Our dataset is balanced between real news and fake news. However, in the real world, real news and fake news are not as balanced as what the dataset shows. We assumed that the amount of real news is way larger than the amount of fake news to reflect the real-life situation. This, we did two experiments for each model with balanced and imbalanced datasets respectively to compare the performance between them. To build the imbalanced dataset, we shrunk the original fake dataset into one-tenth of the original size to imitate the real-world situation. Table 2. shows the true/fake distribution of data in the original training set, imbalanced training set, original validation set, imbalanced validation set, original test set, and imbalanced test set.

Data	Original Training Set	Imbalanced Training Set	Original Validation Set	Imbalanced Validation Set	Original Test Set	Imbalanced Test Set
True	13765	13765	3409	3409	4243	4243

Data	Original Training Set	Imbalanced Training Set	Original Validation Set	Imbalanced Validation Set	Original Test Set	Imbalanced Test Set
Fake	14969	1497	3775	378	4737	474

Results

To evaluate the performance of the models we constructed for the fake news detection problem, we used the most commonly used metrics:

- True Positive (TP): when predicted fake news is actually fake news
- True Negative (TN): when predicted true news is actually true news
- False Negative (FN): when predicted true news is actually fake news
- False Positive (FP): when predicted fake news is actually true news

We can define the following metrics based on the value of the above 4 situations.

$$
\begin{aligned}
Precision &= \frac{|TP|}{|TP| + |FP|} \\
Recall &= \frac{|TP|}{|TP| + |FN|} \\
F1 &= 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \\
Accuracy &= \frac{|TP| + |TN|}{|TP| + |TN| + |FP| + |FN|}
\end{aligned}
$$

These 4 metrics are the most widely used in the world of machine learning, especially for classification problems. It allows us to evaluate the performance of a classifier from

different perspectives. Normally, 'Accuracy' is the most representative metric for the evaluation because it reflects the situation of classification completely.

Table 3(a). and Table 3(b). show the performance of the 4 different models we used on balanced datasets and imbalanced datasets, with different metrics mentioned above, respectively. The results show that training on imbalanced datasets is slightly better than training on balanced datasets for our fake news detection task. The results also show that Transformer-based models are considerably better than the other models. And deep learning models with more attributes perform better than those with fewer attributes. That is, BiLSTM performs better than LSTM, and CNN-BiLSTM performs better than simple BiLSTM.

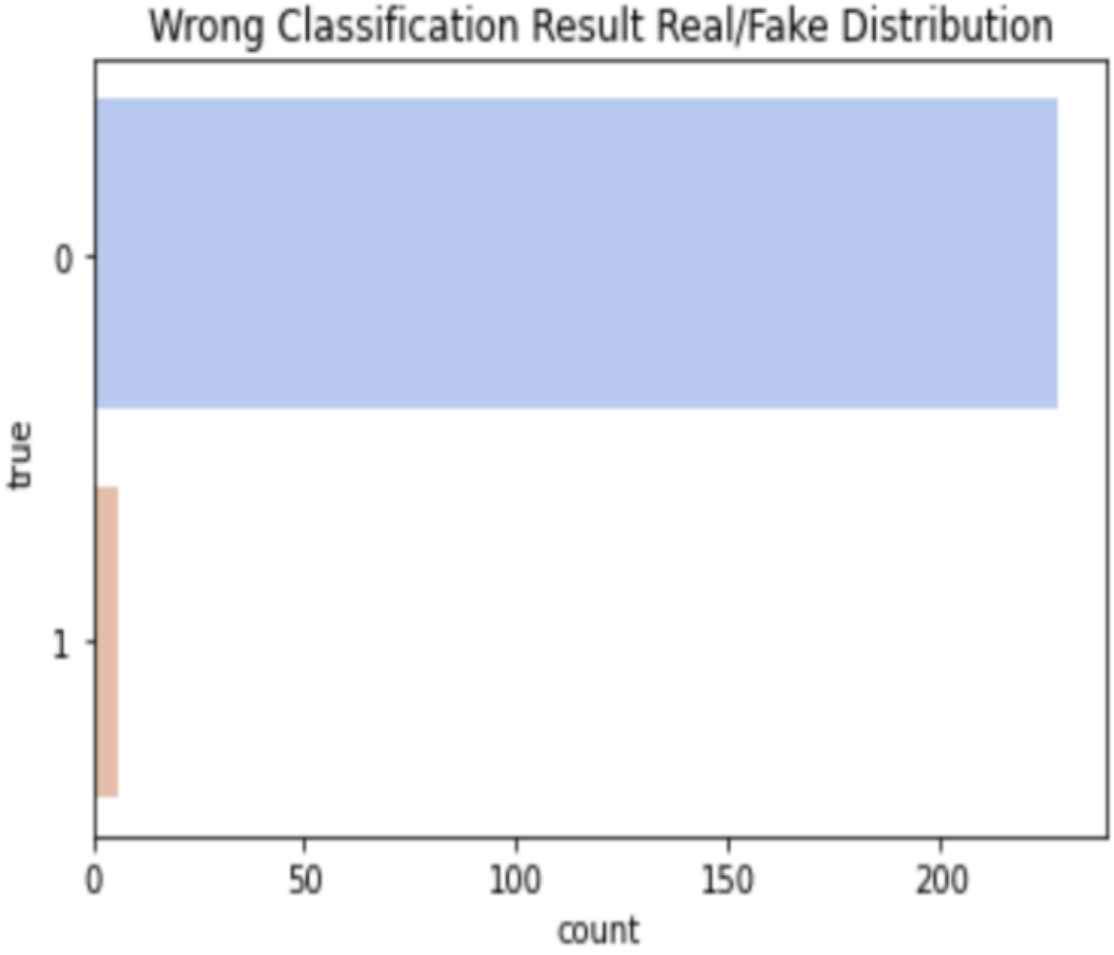
On Balanced Dataset

Model	Accuracy	Precision	Recall	F1-Score
LSTM	0.9697	0.97	0.97	0.97
BiLSTM	0.9710	0.97	0.97	0.97
C-BiL	0.9722	0.97	0.97	0.97
BERT	0.9874	0.99	0.99	0.99

On Imbalanced Dataset

Model	Accuracy	Precision	Recall	F1-Score
LSTM	0.9780	0.98	0.98	0.98
BiLSTM	0.9799	0.98	0.98	0.98
C-BiL	0.9791	0.98	0.98	0.98
BERT	0.9903	0.99	0.99	0.99

Figure 2. Class Distribution of wrongly classified data



code :

```
# This Python 3 environment comes with many helpful analytics libraries
installed
# It is defined by the kaggle/python Docker image:
https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load
import warnings
warnings.filterwarnings('ignore')
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import nltk
from nltk.sentiment import SentimentIntensityAnalyzer
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will
list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/)
that gets preserved as output when you create a version using "Save & Run
All"
# You can also write temporary files to /kaggle/temp/, but they won't be
saved outside of the current session
```

```
/kaggle/input/fake-and-real-news-dataset/True.csv
/kaggle/input/fake-and-real-news-dataset/Fake.csv
```

Loading Data

In [2]:

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

In [3]:

```
fake['Category'] = 'fake'
fake
```

Out[3]:

	title	text	subject	date	Category
0	Donald Trump Sends Out Embarrassing New Year'...	Donald Trump just couldn t wish all Americans ...	News	December 31, 2017	fake
1	Drunk Bragging Trump Staffer Started Russian ...	House Intelligence Committee Chairman Devin Nu...	News	December 31, 2017	fake
2	Sheriff David Clarke Becomes An Internet Joke...	On Friday, it was revealed that former Milwauk...	News	December 30, 2017	fake
3	Trump Is So Obsessed He Even Has Obama's Name...	On Christmas day, Donald Trump announced that ...	News	December 29, 2017	fake
4	Pope Francis Just Called Out Donald Trump Dur...	Pope Francis used his annual Christmas Day mes...	News	December 25, 2017	fake
...	...	...	...	...	...
23476	McPain: John McCain Furious That Iran Treated ...	21st Century Wire says As 21WIRE reported earl...	Middle-east	January 16, 2016	fake
23477	JUSTICE? Yahoo Settles E-mail Privacy Class-ac...	21st Century Wire says It s a familiar theme. ...	Middle-east	January 16, 2016	fake
23478	Sunnistan: US and Allied 'Safe Zone' Plan to T...	Patrick Henningsen 21st Century WireRemember ...	Middle-east	January 15, 2016	fake
23479	How to Blow $700 Million: Al Jazeera America F...	21st Century Wire says Al Jazeera America will...	Middle-east	January 14, 2016	fake

23480	10 U.S. Navy Sailors Held by Iranian Military ...	21st Century Wire says As 21WIRE predicted in ...	Middle-east	January 12, 2016	fake

23481 rows × 5 columns

In [4]:

```
true['Category'] = 'true'
true
```

Out[4]:

	title	text	subject	date	Category
0	As U.S. budget fight looms, Republicans flip t...	WASHINGTON (Reuters) - The head of a conservat...	politicsNews	December 31, 2017	true
1	U.S. military to accept transgender recruits o...	WASHINGTON (Reuters) - Transgender people will...	politicsNews	December 29, 2017	true
2	Senior U.S. Republican senator: 'Let Mr. Muell...	WASHINGTON (Reuters) - The special counsel inv...	politicsNews	December 31, 2017	true
3	FBI Russia probe helped by Australian diplomat...	WASHINGTON (Reuters) - Trump campaign adviser ...	politicsNews	December 30, 2017	true
4	Trump wants Postal Service to charge 'much mor...	SEATTLE/WASHINGTON (Reuters) - President Donal...	politicsNews	December 29, 2017	true
...	...	...	...	...	...
21412	'Fully committed' NATO backs new U.S. approach...	BRUSSELS (Reuters) - NATO allies on Tuesday we...	worldnews	August 22, 2017	true
21413	LexisNexis withdrew two products from Chinese ...	LONDON (Reuters) - LexisNexis, a provider of l...	worldnews	August 22, 2017	true
21414	Minsk cultural hub becomes haven from authorities	MINSK (Reuters) - In the shadow of disused Sov...	worldnews	August 22, 2017	true
21415	Vatican upbeat on possibility of Pope Francis ...	MOSCOW (Reuters) -	worldnews	August 22, 2017	true

		Vatican Secretary of State ...			
21416	Indonesia to buy $1.14 billion worth of Russia...	JAKARTA (Reuters) - Indonesia will buy 11 Sukh...	worldnews	August 22, 2017	true

21417 rows × 5 columns

In [5]:

```
#Now let's combine the whole dataset into one
data = pd.concat([fake, true], ignore_index = True)
data
```

Out[5]:

	title	text	subject	date	Category
0	Donald Trump Sends Out Embarrassing New Year'...	Donald Trump just couldn t wish all Americans ...	News	December 31, 2017	fake
1	Drunk Bragging Trump Staffer Started Russian ...	House Intelligence Committee Chairman Devin Nu...	News	December 31, 2017	fake
2	Sheriff David Clarke Becomes An Internet Joke...	On Friday, it was revealed that former Milwauk...	News	December 30, 2017	fake
3	Trump Is So Obsessed He Even Has Obama's Name...	On Christmas day, Donald Trump announced that ...	News	December 29, 2017	fake
4	Pope Francis Just Called Out Donald Trump Dur...	Pope Francis used his annual Christmas Day mes...	News	December 25, 2017	fake
...	...	...	...	...	...
44893	'Fully committed' NATO backs new U.S. approach...	BRUSSELS (Reuters) - NATO allies on Tuesday we...	worldnews	August 22, 2017	true
44894	LexisNexis withdrew two products from Chinese ...	LONDON (Reuters) - LexisNexis, a provider of l...	worldnews	August 22, 2017	true

44895	Minsk cultural hub becomes haven from authorities	MINSK (Reuters) - In the shadow of disused Sov...	worldnews	August 22, 2017	true
44896	Vatican upbeat on possibility of Pope Francis ...	MOSCOW (Reuters) - Vatican Secretary of State ...	worldnews	August 22, 2017	true
44897	Indonesia to buy $1.14 billion worth of Russia...	JAKARTA (Reuters) - Indonesia will buy 11 Sukh...	worldnews	August 22, 2017	true

44898 rows × 5 columns

In [6]:

```
data.shape
```

Out[6]:

```
(44898, 5)
```

Preprocessing

In [7]:

```
data['Category'].value_counts()
```

Out[7]:

```
Category
fake    23481
true    21417
Name: count, dtype: int64
```

In [8]:

```
#Transforming category values to numerical
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
data['Category'] = encoder.fit_transform(data['Category'])
```

In [9]:

```
data['Category']
```

Out[9]:

```
0        0
1        0
2        0
3        0
```

```
4        0
        ..
44893    1
44894    1
44895    1
44896    1
44897    1
Name: Category, Length: 44898, dtype: int64
```

In [10]:

```
vectorizer = TfidfVectorizer()
title = vectorizer.fit_transform(data['title'])
title
```

Out[10]:

```
<44898x20896 sparse matrix of type '<class 'numpy.float64'>'
      with 546512 stored elements in Compressed Sparse Row format>
```

Modeling

In [11]:

```
from sklearn.model_selection import train_test_split
X = title
y = data['Category']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state = 42)
```

In [12]:

```
model = SVC()
model.fit(X_train, y_train)
```

Out[12]:

SVC

```
SVC()
```

In [13]:

```
y_pred = model.predict(X_test)
print('Classification Report: ')
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support
```

0	0.97	0.96	0.96	4733
1	0.95	0.97	0.96	4247
accuracy			0.96	8980
macro avg	0.96	0.96	0.96	8980
weighted avg	0.96	0.96	0.96	8980

Conclusion

- Preprocessing is an essential step in natural languages processing tasks such as text classification, and techniques such as lowercasing, removing stop words, and stemming/lemmatizing can significantly improve the performance of models.
- CountVectorizer is a powerful tool for converting text data into a numerical representation that can be used in machine learning models.

- The choice of a machine learning algorithm can significantly impact the performance of a text classification task. In this project, we compared the performance of logistic regression and support vector machines and found that logistic regression had the best performance.
- Model evaluation is critical to understanding the performance of a machine learning model and identifying areas for improvement. In this project, we used metrics such as accuracy, precision, recall, and F1 score to evaluate our models.
- Finally, this project demonstrates the potential of machine learning for automated fake news detection and its potential applications in the media industry and beyond.

In this blog post, we learned how to train a simple logistic regression model to classify news articles as real or fake and how to deploy the model as a web application using the Flask framework. We used the sklearn library for preprocessing and modeling the data and created a simple web form using HTML and Flask.

The dataset we used for this project was the Fake and real news dataset from Kaggle, which contains 23481 real news articles and 21417 fake news articles. We preprocessed the text by removing stop words, punctuation, and numbers and then used a bag-of-words approach to represent each article as a vector of word frequencies. We trained a logistic regression classifier on this data and achieved an accuracy of over 99%.

Overall, this project demonstrates how machine learning can be used to tackle the problem of fake news, which is becoming an increasingly important issue in today's society.

thank you

submitted by ;

p.udayakumar