FAKE NEWS DETECTION USING NLP

INTRODUCTION:

With the rise of social media platforms, spreading of fake news have increased exponentially. The internet is flooded with fake accounts and fake posts, coming from all around the world. Fake news is a type of propaganda which is purposefully spread with an intention to mislead the public. It is hard for a common man to differentiate between fake and real news. It is however important to understand the difference between fake and bias news.
Many technology companies like Google and Facebook have started to take significant steps to address this fake news issue. They have recently integrated news rankings prioritization and fact checking systems into their platforms respectively.

MOTIVATION:

Our main motivation to carry out this project is to identify and classify news article as fake or not, using the concept of Natural Language Processing and Machine Learning.

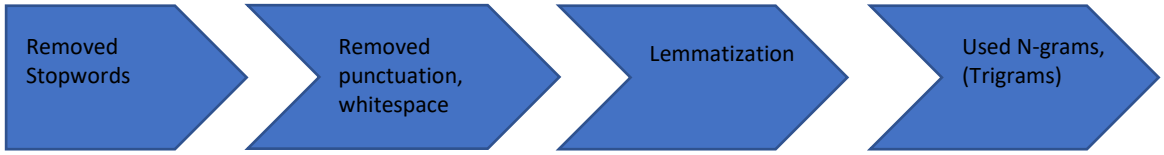
WORKFLOW:

DATA COLLECTION:

For this project, we started by collecting news articles classified as fake news from Kaggle, FAKE NEWS. These articles are derived using B.S. Detector, which is a browser extension which searches links of unreliable sources and cross-checks them against a third-party list of domains. Then, we collected real news also from Kaggle, REAL_NEWS . To ensure we do not contain news articles from questionable websites, we filtered the news articles from Real news dataset and selected only the well-known and reliable websites like New York Times, Washington Post, Forbes etc. We also filtered the real news during November, 2016 because the fake news dataset articles were also collected during this time.
So, finally in total we had around 24,000 news articles containing both fake and real news articles. We also created two columns in this dataset, first column being the news articles and the second column fake or not column, where fake being 1 and not fake being 0.

TEXT PREPROCESSING:

As we know that the performance of machine learning models for NLP is highly dependent on words and their features inside the corpus. So, in the text preprocessing part, we removed stop words such as 'the', 'there', 'some' etc. These words act as noise and increase the feature dimensionality of the document. So, we used Python packages such as spaCy genism to tokenize our text, thereby performing the preprocessing steps. These steps were very significant as they helped reduce the size of our corpus and also helped in adding context to the text. Also, we removed the unnecessary white spaces and punctuations from our documents to reduce dimensions. So, next in feature conversion such as lemmatization, we converted the words into their root form, such as 'running' to 'run' and 'flattened' to 'flatten' .This converted different words into a single representation. We also used N-gram specially trigrams for our project, where we combined nearby words into single features, thereby giving meaning to words which have little significance on their own.



CONVERTING TEXT TO FEATURES:

After the text preprocessing, in order to analyze and prepare the data for modelling, we need to convert it into features. There are several methods which can be used such as TF-IDF (Term Frequency- Inverse Document Frequency and Word2Vec. We did this using TF-IDF method.
In general, TF-IDF is a statistic whose main motive is to tell how important a word is for a document inside a corpus. With the increase in the appearance of words in a document, it increases proportionally, but it is offset by its frequency in the total corpus.
Using TF-IDF, we found the relative importance of words in our news datasets.
The formula for TF-IDF is:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

CLASSIFICATION:

After cleaning and pre-processing the data the last step for predicting is the classification. For this we implemented various models such as Logistic Regression, Random Forest, XGBoost, Stochastic Gradient Descent, and AdaBoost Classifier.

All the classifiers gave a varied response;

Model	Accuracy
Logistic Regression	87.56%
Random Forest	92.31%
XGBoost	89.47%
Stochastic Gradient Descent	95.67%
AdaBoost Classifier	94.45%

FUTURE WORK:

We also tried the 'Text conversion to features' step using Word2Vector method. The Word2Vec technique converts text to features when maintaining the original position and relationship between words in a corpus. It is a combination of two techniques namely the Continuous bag of words and the skip-gram model. Both are neural networks (shallow) which does mapping of words into a target variable, which is also a word. These techniques learn weights which act as word vector representations.

So for our project we the Google News dataset (300 dimensional vectors for 3 million words and phrases) to train our models. The greater the amount of words it is trained against, better are the quality of word vectors formed.

But we are facing some issues in applying classification models to this. But we are working on it.