Fake News Detection

- **TUSHAR AHUJA**

Abstract:-

In this research summary, we explore the application of NLP techniques to the detection of ‘fake news’, that is, misleading news stories that come from non-reputable sources. Using a dataset obtained from a list of sources, we apply term frequency-inverse document frequency (TF-IDF) to a corpus of about 7000 articles. We are skeptical about the generalizability of these findings, however, and include ample discussion on next steps for exploration in this space.

Introduction:-

In 2016, the prominence of disinformation within American political discourse was the subject of substantial attention, particularly following the surprise election of President Trump. The term ‘fake news’ became common parlance for the issue, particularly to describe factually incorrect and misleading articles published mostly for the purpose of making money through pageviews. In this project, we seek to produce a model that can accurately predict the likelihood that a given article is fake news.

Facebook has been at the epicenter of much critique following media attention. They have already implemented a feature for users to flag fake news on the site; however, it is clear from their public announcements that they are actively researching their ability to distinguish these articles in an automated way. Indeed, it is not an easy task. A given algorithm must be politically unbiased--since fake news exists on both ends of the spectrum--and also give equal balance to legitimate news sources on either ends of the spectrum.

Classification Models

Most classification models have been simple approaches, which have worked substantially well. It is unclear the extent to which more complex modeling approaches have been tested for success. We implement the following models:

1. **Decision Trees (DT)** :-

Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision node are where the data is split.

An example of a decision tree can be explained using above binary tree. Let's say you want predict whether a person is fit given their information like age, eating habit, and physical activity etc. The decision nodes here are questions like 'What's the age?' 'Does he exercise?' 'Does he eat a lot of pizzas'? And the leaves, which are outcomes like either 'fit' or 'unfit'. In this case this was a binary classification problem (a yes no type problem).

2. **Random Forests (RF) :-**

Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. A random forest is a classifier consisting of a collection of tree structured classifiers.Briefly, Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

3. k-nearest neighbors algorithm :-

k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. Both for classification and regression, a useful technique can be to assign weights to the contributions of the neighbors, so that the nearer neighbors

contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/*d*, where *d* is the distance to the neighbor.
The neighbors are taken from a set of objects for which the class (for *k*-NN classification) or the object property value (for *k*-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

4. Naive Bayes :-

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features

DATASET

FAKENEWSDETECT_DATASET is a cleaned dataset which contains 6968 articles. 131 (1.88%) are labeled as fake. The reliable articles come from 9 unique sources. The unreliable articles come from 3 unique sources. This is not an ideal approach; we would like to have a more authoritative, vetted source for establishing reliable versus unreliable sources, and we would like our examples to be from a wider variety of sources.

This dataset was created by using some beautiful libraries of python termed as **Scrapy, BeautifulSoup**.

Datasets are also called **data corpus** and **data stock.**

Our code relies upon the implementation of these methods in SciKit-Learn. More information about these implementations can be found on their website.

First, we scrub the articles of any mention of the name of the source. Because the reliable/unreliable classification is determined at the source level, this step is necessary to ensure the model doesn't just learn the mappings from known sources to labels.

We use SciKit Learn to generate the relevant features. We utilize SciKit Learn's Pipeline feature to create fit-transform and transform methods that can be used on the training data and then applied to the test set.

Confusion Matrix

A confusion matrix is a summary of prediction results on a classification problem.
The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix.
The confusion matrix shows the ways in which your classification model is confused when it makes predictions.
It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.

	Class 1 Predicted	*Class 2 Predicted*
Class 1 Actual	**TP**	**FN**
Class 2 Actual	**FP**	**TN**

Here,
- Class 1: REAL(0)
- Class 2 : FAKE(1)

Definition of the Terms:

- Positive (P) : News is REAL
- Negative (N) : News is FAKE
- True Positive (TP) : News is REAL, and is predicted to be REAL.
- False Negative (FN) : News is REAL, but is predicted FAKE.
- True Negative (TN) : News FAKE, and is predicted to be FAKE.
- False Positive (FP) : News is FAKE, but is predicted TRUE.

Classification Rate/Accuracy:

Classification Rate or Accuracy is given by the relation:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

However, there are problems with accuracy. It assumes equal costs for both kinds of errors. A 99% accuracy can be excellent, good, mediocre, poor or terrible depending upon the problem.

Underfitting:

A statistical model or a machine learning algorithm is said to have underfitting when it cannot capture the underlying trend of the data. Underfitting destroys the accuracy of our machine learning model. Its occurrence simply means that our model or the algorithm does not fit the data well enough. It usually happens when we have less data to build an accurate model and also when we try to build a linear model with a non-linear data. In such cases the rules of the machine learning model are too easy and flexible to be applied on such a minimal data and therefore the model will probably make a lot of wrong predictions. Underfitting can be avoided by using more data and also reducing the features by feature selection.

So, here at my dataset there is a drastic affect of Underfitting. Due to lack of data the accuracy is degraded. Which shows the performance of our dataset is not up to extent or good.

Comparison of top unreliable and reliable source by article frequency

Top Three Unreliable News Sources		Top Five Reliable News Sources	
OANN	160	INDIA TODAY	1896
NEWS18	126	BBC	830
BVA	90	CNN	765

Feature Generation

Our approach evaluates the performance of models trained on three feature sets:

1. Bigram Term Frequency-Inverse Document Frequency
2. Count Vectorizer

Term Frequency-Inverse Document Frequency :-

The first feature set is vectorized bigram Term Frequency-Inverse Document Frequency. This is a weighted measure of how often a particular bigram phrase occurs in a document relative to how often the bigram phrase occurs across all documents in a corpus.
Because of the political nature of our corpus, we want to limit the model's knowledge of the people and institutions mentioned in the article text. Otherwise, we risk the model simply learning patterns such as "Clinton corrupt" which describe the topic and viewpoint of the text, rather than the outcome of interest (is this source reliable or not). Additionally, these patterns will be highly sensitive to the particular

news cycle. To address this concern, we introduce a step during tokenization to use Spacy's named entity recognition to replace all mentions of named entities with a placeholder, e.g. <-NAME-> or <-ORG->.

We use SKLearn to calculate the TFIDF for each bigram within each document and build a sparse matrix of the resulting features. To keep the dimensionality of our data to a manageable size, we limit the vocabulary to only consider the top 3000 terms ordered by term frequency across the entire corpus. We did not experiment with different methods or thresholds for selecting the terms included in the vocabulary, or with different lengths of n-grams, but this may be an area to explore in future work.

Count Vectorizer :-

The most straightforward one, it counts the number of times a token shows up in the document and uses this value as its weight.

Count Vectorizer builds a count matrix where rows are occurrences counts of different words taking into account the high-dimensional sparsity.

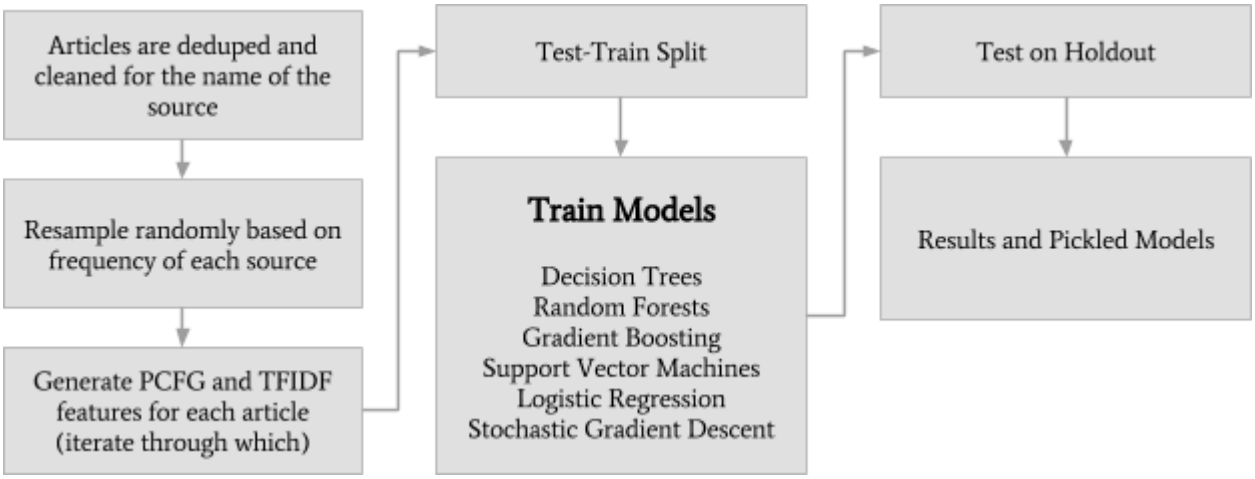
If you want to play around with CountVectorizer, you can fiddle with three parameters. First of them is max_df. This pretty much sets how many features, or in this case words, you want CountVectorizer to count. Setting this parameter can be incredibly useful when dealing with a large amount of documents as you'll run into speed issues with computation.

Modeling and Evaluation

Our Pipeline

After cleaning the data and generating features, we execute a 75/25 random test-train split on the dataset and feed it into a modeling pipeline. This pipeline iteratively fit models varying the tuning parameters with which they are executed up to 50 times, depending on the number of possible permutations for that model. These models are then tested on the 10% holdout data to understand their performance.

Pipeline representation



Baseline Models for Comparison: Naive and Random

As a baseline comparison for understanding the performance of our models, we look to two methods. First, a naive model that predicts all majority class, in this case that all articles are from reliable news sources. Second, a model that randomly selects a classification for each article as either reliable or unreliable based on the posterior probability of that class in the training set.

Combined Features Model Performance

Combining both FAKENEWSDETECT_DATASET.csv and resized_v2.csv sets, our models perform well above our baseline.

Average model performance with TF-IDF bi-gram features at 0.25 score threshold for categorization.

Model	Test Accuracy	Validation Accuracy
Naïve Bayes	99.2%	13%
Random Forests	99.6%	3%
KNeighbors	100%	13.36%

Combining both FAKENEWSDETECT_DATASET.csv and fake_or_real_news.csv sets, our models perform well above our baseline.

Average model performance with TF-IDF bi-gram features at 0.25 score threshold for categorization.

Model	Test Accuracy	Validation Accuracy
Naïve Bayes	99.4%	50.2%
Random Forests	100%	56.34%
KNeighbors	100%	63.36%

We note that our best models tended to be K Neighbors, which, given that they tend to perform well with sparse and highly dimensional data, is not surprising.

Results and Next Steps

Results

Our discussion has touched on the following conclusions:

- Overall, the best performing model by overall ROC AUC is K Neighbors TFIDF feature set only.
- TFIDF shows promising potential predictive power, even when ignoring named entities, but we remain skeptical that this approach would be robust to changing news cycles. This would require a more complete corpus, however.

Limitations

There are substantial limits to our analysis that prevent broader generalizability.

- We evaluated our models using absolute probability thresholds, which may not be the most reliable for models where probability scoring is not well-calibrated. With more time, we would re-run evaluation using relative probabilities for labeling for better comparison.
- While TFIDF performs better, we are possibly overfitting to topics/terms important in this news cycle . A vectorized approach like the one we pursued unfortunately makes it technically hard to see what individual features are most important, hampering our analysis.
- With limited verified sources of fake news in our dataset, we cannot claim that this approach would generalize to unseen sources.

Conclusions and Next Steps

Initial results show that term frequency is potentially predictive of fake news; an important first step toward using machine classification for identification. However, we remain concerned about overfitting and learning topical patterns that predict the partisan split of the legitimate vs. fake sources . Further research should take seriously the challenge of finding a corpus that satisfies the conditions . Ultimately an objective way of classifying ‘fake’ from legitimate news continues to be barrier that will make adoption difficult.

We suggest the following paths for further research:

- Curation of a more carefully selected article corpus.
- Focus on testing models across time--this will indicate whether the term frequencies are capable of processing unseen articles from new news cycles in addition to the training set.
- Layering on of additional techniques on top of term frequency.

Time Spent and Code Responsibilities

It is difficult to quantify precisely how much time was spent on this project. We would estimate conservatively approximately 140 hours over the course of an month.