

Original Article

Fake News Detection: Benchmarking Machine Learning and Deep Learning Approaches

Manan Buddhadev¹, Virtee Parekh²

^{1,2}Department of Computer Science, Rochester Institute of Technology, New York, USA.

Received Date: 13 January 2025

Revised Date: 20 February 2025

Accepted Date: 15 April 2025

Abstract: *Fraudulent articles have cropped up all over the web and spread like wildfire. They constitute falsified facts, phony scientific facts, discriminatory articles, satirical items and misleading articles aimed at demeaning other groups or individuals. It is imperative to contain such articles as they create chaos and lead to unwise decision making. In this project, machine learning and deep learning approaches are used to flag fake news items. Part of the dataset is manually scraped from the web and the other half is publicly available. Feature extraction techniques like Bag of Words, TF-IDF, N-grams, word embeddings like GloVe are explored. Out of the various combinations of feature extraction techniques and models, it was found that implementing CNN-LSTM along with GloVe embeddings gave the best results with 91% testing accuracy.*

Keywords: *Fake News Detection; Deep Learning; Text Classification; Glove Embedding; Machine Learning; CNN-LSTM.*

I. INTRODUCTION

The term 'fake news' caught the world's attention during the 2016 US presidential elections when false news items were circulated over the web and social media to demean other candidates and influence the voters wrongly. Fake news articles spread extremely fast and wide and it became so prevalent that Collins Dictionary added the term to their words of the year list [1]. Fake news can be of several kinds - false connections between headlines and text, manipulated content, satirical pieces, articles with misleading information, fabricated information or information spread with improper context and imitated articles [2]. Examples of fake news include religious propaganda, falsified scientific facts and spurious cures for diseases, prejudiced and misleading information spread with the intention of instigating the general public against a group or individual etc. Such news articles can instigate the people, create chaos and lead to unwise decision making that is harmful to the society. It has become of utmost importance to contain such types of news articles and to be able to identify such articles and their sources.

There are several ways in which artificial intelligence can be used to evaluate the credibility of news articles. Fact checking and verification can be performed to check for the veracity of the articles, the sources of these articles can be examined, the text can be studied for bias and language structure and the stance of the headline versus the stance of the text can also be studied [3].

In this work, we detect between fake and genuine articles by analyzing the headline and the text of the news article and harnessing the power of natural language processing and both, machine learning and deep learning, to build a classifier model. The dataset is manually scraped from over 1500 online sources and the articles are labelled as per the tags found on the websites - OpenSources and MediaBiasFactCheck. Various data processing techniques are applied to prepare the data for feature extraction. Feature extraction techniques are used to convert the raw text into a vectorized format that the models can interpret. A variety of machine learning and deep learning architectures are implemented to find a suitable model. We find that a combination of convolutional neural networks (CNN) and long short term memory(LSTM) models give us the best accuracy.

II. RELATED WORK

One of the major issues in the task of fake news detection is the lack of large, publicly available datasets. To tackle this problem, a dataset called LIAR [4] was developed. LIAR is a dataset consisting of around 12.8k manually labeled records. It consists of snippets from Politifact.com, a website that runs fact checking on statements made by US politicians. The snippets are verified by Politifact editors and are given one of the following six labels - barely true, mostly true, half true, true, false and pants on fire. The snippets belong to speakers who belong to any of the two US political parties and are from a variety of sources like social media posts, press releases, interviews and speeches and cover a host of topics like economy, immigration, taxes, health etc. The developers of LIAR also added metadata features to each record that include the subject, the speaker, the speaker's job title and affiliation, the source and a credit history, along with the snippet and target label. Credit history refers to each speaker's count of barely true, false, mostly true and pants on fire statements made. A hybrid



model of convolutional neural network (CNN) along with bi-directional LSTM was used on the snippet along with the metadata for the purpose of classification and a test accuracy of 0.27 was achieved.

Another set of researchers are working on dealing with this problem by considering both the news article and its social media engagement to evaluate its credibility [5]. They propose that two kinds of data about a news item must be considered - news content based and social context based. News content based features include the title, text, images or videos and the author or publisher of the article. Social context features include - user based features like the credibility of the user, number of followers, groups the user is a part of, previous posts and related demographics, post based features that deal with the topic, stance and credibility of the item and network based features involving social network graphs. Social context features play a key role as usually fake news are generated by bots, are circulated heavily among certain groups etc. The researchers are working on building a dataset that contains all of the above features.

The credibility of a news item on Facebook can be evaluated by evaluating the users who liked the particular post. This paper [6] proposes logistic regression and boolean crowdsourcing algorithms to evaluate whether a post is a hoax or not. Their dataset consists of 15500 Facebook posts and over 900k users. In logistic regression, the user engagement with the post is considered as a feature. Logistic regression will fail to learn about a user if the training set does not contain a post liked by the user. Boolean crowdsourcing is used in this case. The researchers achieved an accuracy of about 99% by their proposed approaches.

Deep learning and machine learning have been used extensively to tackle the problem of fake news detection. One paper [7] proposes a two step method where a recurrent neural network (RNN) captures the temporal pattern of user activity on a news item and the second neural net understands the behaviour of all users that are engaging with the article and a third module that brings together this information to classify the news item. Another paper [8] uses a combination of CNN and LSTMs to evaluate falsified articles on Twitter and have achieved an accuracy of 80%. A group of students built a fake news classifier using a weighted ensemble classifier made of Naive Bayes and AdaBoost [9] and a developer used a simple deep neural network to build a fake news detector [10].

III. DATA

A. Manually Created Dataset

For the purpose of evaluating credibility of news articles, a sufficiently large and labelled dataset consisting of news titles and their corresponding texts is required. Due to the unavailability of a single source of a reasonably sized dataset, we decided to scrape the news data off the web. The list of credible and non-credible news sources is available on the following websites-

a) *OpenSources.co*:

OpenSources [11] is a professionally curated database that consists of around 800 websites and a tag that determines the credibility of news articles published on that website. The database is publicly available for download in the form of JSON and comma separated files. OpenSources assigns one of the following tags to each website - fake for articles that consist of phony and falsified information; satire for articles that make use of excessive sarcasm, irony and humour; bias for articles that are highly prejudiced and opinions are presented as facts; conspiracy for articles promoting conspiracy theories; rumor for articles that comprise of unverified facts and rumors; state for articles that are regulated by the government; junksci for articles that promote spurious scientific theories; hate for news sources encouraging discrimination and hateful content; clickbait for click-bait articles; unreliable for articles that are not fully reliable; political for sources that usually provide reliable content but are usually concerned with politics and credible for reliable news sources.

Only the tags relating to fake news were considered for scraping because the list of websites available for credible sources was not large enough. The state and political tags were discarded because they are not strong indicators of the news being fake. They do publish domain-specific information but there is no strong and clear evidence of the news being fake. The unreliable tag was also discarded because the tag itself means that the developers at OpenSources are not quite sure about these news sources. The other 7 tags comprised 700 news sources. Each of these sources was manually scraped for a news title and its corresponding text. The number of articles scraped was around 9000 and these were assigned the target variable FAKE. The scraping was performed using the Newspaper library in Python. Multi-threading in Python was used to make the process faster.

b) *MediaBiasFactCheck.com*:

MediaBiasFactCheck [12] is a website that reports not only on how politically biased the news website but also on how factually correct its articles are. Each news website thus gets 2 tags - the bias tags that includes center, left-center, left, right-center, right, pro-science, satire, conspiracy and the truthfulness tags that consist of very high, high, mixed, low and very low.

Since the fake articles were scrapped from OpenSources, MediaBiasFactCheck was used to scrape genuine articles only. The tags considered were center, left-center, right-center and pro-science with the truthful quotient being very high or high. These tags ensure articles that have little or no bias and have verified content. center tag implies that the source has no political leaning and left-center/right-center imply that the source usually does not have any political leanings but may have editorials that may favor a certain candidate. MediaBiasFactCheck was first scrapped to get a list of the credible sources and then each source was scrapped to get news titles and their corresponding text. To make class distributions equal, 9000 articles were collected using the Newspaper library and multi-threading in Python. These were assigned the target variable REAL.

B. Publicly Available Datasets

Experiments were run on the above manually created dataset. However, the models were more prone to over-fitting. To increase the dataset, publicly available datasets found on Kaggle [13] and GitHub [14] were added to the original dataset. Adding these datasets doubled the size of the original dataset.

IV. DATA CLEANING AND PREPARATION

Data in its raw format cannot be passed as an input to any machine learning model. Certain cleaning techniques need to be followed in order to make the data easy to interpret. Special attention needs to be paid to the fact that the text consists of published news articles. Since they are published news articles, chances of finding errors like misspelled words are extremely rare because of prior proof-reading. A comprehensive list of techniques for cleaning news articles can be found here [15]. In this project, NLTK [16] and spaCy [17] have been used. The techniques used for pre-processing the data are listed below -

- Merging the title and body: The title of the news article and the text body are merged to create one feature. The title is an important feature to detect any kind of fake news. Clickbait or misleading titles play an important role in detecting hoax news items. Hence, they are merged with the text to form a composite feature. Any trailing or leading white spaces or lines are eliminated.
- Removal of accented and special characters: News articles may include proper nouns of people who may have accented characters in their names like Renée or André. Such accented and other non-ASCII characters are converted to their ASCII format and all of the text is converted to UTF-8 encoding.
- Expanding contractions: News articles tend to have quotes by other people that will have words like 'don't', 'I'm', 'could've', 'let's' etc. For the sake of consistency, all of these contractions are expanded to their original text like 'do not', 'I am', 'could have' and 'let us' respectively.
- Removal of stopwords: Stopwords are words that occur frequently in a text data and have no major significance like 'a', 'an', 'the', 'is', 'was', 'and' etc. Python's popular natural language processing tool, NLTK, has a comprehensive list of stopwords that can be used for this filtering. Before filtering, each word in the text is tokenized by using NLTK's Toktok Tokenizer.
- Lemmatization: Lemmatization refers to the process of removing inflections from a word and converting them to their root form, which always exists in a dictionary. For example, words like 'crying' and 'cried' get reduced to 'cry'. spaCy, another natural language processing tool in Python, has an excellent method for lemmatization which is used here.

V. MACHINE LEARNING APPROACH

Once the data is cleaned, we now move towards feature engineering and building models for the task of detecting non-credible news articles.

A. Feature Extraction Techniques

Raw textual data cannot be fed to a machine learning algorithm directly. It needs to be vectorized i.e. the raw data needs to be converted to a suitable numeric format for a model to be able to interpret the data. The following feature engineering techniques were used to vectorize the data.

Bag of Words and N-Grams: Bag of Words (BoW) is a fairly easy technique to understand and implement. Let us consider that the dataset has three documents.

- Doc1 - She loves chocolate cake
- Doc2 - She is baking a cake
- Doc3 - He is baking a cake to surprise her

A bag would include all of the unique words in this corpus - 'baking', 'cake', 'chocolate', 'he', 'her', 'is', 'loves', 'she', 'surprise', 'to'. Each of these words is called a *gram*. The BoW model will now count the frequency of each word in a document and transform each document into a vector consisting of the frequencies of each word in that document. It follows

the logic that words that occur frequently in a document are significant. The BoW matrix for the above corpus is shown in Table 1.

Table 1 : Bag of Words Document Matrix

	Doc 1	Doc 2	Doc 3
baking	0	1	1
cake	1	1	1
chocolate	1	0	0
he	0	0	1
her	0	0	1
is	0	1	1
loves	1	0	0
she	1	1	0
surprise	0	0	1
to	0	0	1

The above model is unigram as each word is considered individually. In an N-gram model, N groups of consecutive words are chosen and the above process is repeated. For example, in a bi-gram model [18], the bag for Doc1 would include the terms - 'she loves' , 'loves chocolate', 'chocolate cake'. This is useful in capturing patterns and phrases for which, if the constituent words were used individually, would make no sense. Using N-grams increases the semantic understanding of the model. Python's Scikit-Learn [19] has a method called CountVectorizer() for implementing BoW (with or without N-grams).

Term Frequency - Inverse Document Frequency: Term Frequency - Inverse Document Frequency (TF-IDF) is another technique for vectorization where words are given weights by considering their frequency, not just in the single document, but also in the entire corpus. If a word occurs frequently in a given document, it might be important. But if it occurs frequently across all the documents, it is most likely that the word is not a keyword/significant word [20]. This is a clever technique to identify distinctive keywords and weed out the noise. First, the frequency of the word(TF) in a given document is calculated. IDF is the score of how infrequent the word is across all the documents. Thus, more the frequency of a word across all documents, lesser the TF-IDF score. Considering the same example used in BoW, its TF-IDF document matrix is shown in Table 2.

Table 2 : TF-IDF Document Matrix

	Doc 1	Doc 2	Doc 3
baking	0	0.52	0.32
cake	0.34	0.40	0.25
chocolate	0.58	0	0
he	0	0	0.42
her	0	0	0.42
is	0	0.52	0.52
loves	0.55	0	0
she	0.44	0.52	0
surprise	0	0	0.42
to	0	0	0.42

Words like 'cake' and 'baking' occur in almost all of the documents and thus are given lower weights than all other words. 'surprise' has one of the highest TF-IDF scores because it occurs least frequently across all documents. Python's Scikit-Learn [19] has a method called TfidfVectorizer() for implementing TF-IDF.

Hashing Vectorizers: TF-IDF and BoW, both require the construction of a vocabulary. Sometimes, the vocabulary is humongous and cannot be stored or accessed at the same time. In such cases, hashing vectorizers are useful. They use the hashing function to map words to their integer values. The term frequency is usually the value to be hashed. The con of this technique is that there is no way to reverse the process and inspect the vocabulary. Thus, we cannot view the most important features of the corpus. Python's Scikit-Learn has a method called HashingVectorizer() for implementing the above.

B. Machine Learning Models

Numerous combinations of classification algorithms like logistic regression, Naive Bayes classifier and stochastic gradient descent with support vector machines with the above feature extraction techniques were implemented using Python's Scikit-Learn [19]. The experiments and results are explained in further sections.

VI. DEEP LEARNING APPROACH

In the past few years, deep learning has made rapid advancements in the field of artificial intelligence and has proved to be a great tool for tasks like text classification. The power of deep learning is harnessed here to see if it can improve the performance of the conventional machine learning models.

A. Feature Engineering using Word Embedding

Like machine learning models, deep learning models also require raw text in a vectorized format for processing. In the above feature extraction techniques, the context and semantics of the text are lost. Consider the two sentences, 'I hate desserts' and 'I dislike sweets'. Both sentences mean the same but their BoW or TF-IDF representations are independent of each other and do not store the similarity that is inherent in these sentences. To overcome this obstacle, word embeddings are used. Word embeddings are vector representations of the corpus that aim to preserve its similarity, context and semantics [21].

Popular word embedding techniques include the global matrix factorization method in Latent Semantic Analysis (LSA) and word2vec. In LSA, it is assumed that words that have a similar meaning tend to occur in related articles. A mapping of word counts per document is created and singular value decomposition is used to reduce this matrix while preserving the similarities. word2vec uses neural networks to predict a word, given its surrounding words or predict the surrounding context, given a word [21]. word2vec takes in only local context which is a limitation as it disregards patterns or repetitive words that might be visible globally [22]. While LSA does look at the corpus globally, it does not produce as good word analogies as word2vec does [23]. GloVe aims to merge the above two methods to tackle these problems. It takes in global statistics while preserving the meaning of the words. GloVe builds a co-occurrence matrix which indicates the number of times one word appears in the context of another. Vectors are formed by using the matrix to derive co-occurrence ratios (probability) of 2 words with respect to a context word where the ratios indicate the difference in these vectors. GloVe is trained on the function that the dot product of the 2 word vectors must equal the log of the co-occurrence ratios [23]. GloVe has results that are comparable to word2vec and was thus chosen as the word embedding technique for this project.

B. Deep Learning Architectures

The following deep learning architectures were implemented using the Keras [24] library in Python -

- Long Short Term Memory (LSTMs): LSTMs are a specialized version of recurrent neural networks (RNNs). RNNs have the ability to allow information to persist. However, they are not capable of handling long term dependencies in text. LSTMs were developed for this very purpose [25]. Since news articles can tend to be lengthy and it is important to keep track of the pattern of the text, right from the beginning, LSTMs are used here.
- Gated Recurrent Unit (GRU): GRUs also solve the vanishing gradient problem like an LSTM but are computationally simpler and as efficient as an LSTM [25].
- Convolutional Neural Net with Long Short Term Memory (CNN-LSTMs): This research proved that using CNNs with hyper-parameter tuning can work wonders in natural language processing tasks as well [26]. Adapting their approach, a CNN along with a LSTM is used in this system.

VII. EXPERIMENTS AND RESULTS

A. Machine Learning Approach

In the first step, machine learning models were implemented along with feature extraction techniques mentioned in Section V-A. The results are displayed in Table 3. Numerous combinations of feature extraction techniques along with the models were implemented.

Originally, only the manually scraped and labelled dataset was used. The four classification algorithms chosen were Multinomial Naive Bayes, Stochastic Gradient Descent with Support Vector Machines (SGDC w/ SVM) and logistic regression. Multinomial Naive Bayes (MultinomialNB) work very well with text classification problems. According to the Scikit-Learn documentation, Naive Bayes works better with the bag of words (BoW) model as compared to TF-IDF [19]. This conclusion echoes in the results achieved here where the BoW model performs much better than the TF-IDF model. This is because the multinomial distribution in MultinomialNB uses discrete counts such as frequencies or word counts. TF-IDF values are weights which are continuous in nature as seen in Table 2. The performance of the SGDC with SVM model is also at par with the above model. This is because the model runs for a certain number of epochs, and in each epoch fits a SVM to the data and uses stochastic gradient descent for optimizing the loss i.e. the loss for each data sample is calculated and the model is

updated, with respect to the learning rate. To increase the efficiency of the model, L2 regularization was added and the type of loss used is hinge which creates the SVM. This model works for a sparse matrix with floating point values [19] and the TF-IDF, as seen in Table 2, is a sparse matrix with weights as floating point values, hence the higher accuracy. Logistic regression, another stable classifier also works well in this case.

Table 3 : Machine Learning Approach Results

Model	Feature Extraction	Manual Dataset Accuracy (%)		Extended Dataset Accuracy (%)	
		Train	Test	Train	Test
MultinomialNB	BoW w/ bi-gram	98.9	88.3	99.9	90.8
SGDC w/ SVM	TF-IDF w/ bi-gram	98.5	87.4	94.1	89.4
Logistic Regression	BoW w/ bi-gram	98.9	85.7	99.9	88.8
Logistic Regression	TF-IDF w/ bi-gram	96.0	85.1	96.3	89.2
SGDC w/ SVM	Hashing Vect.	94.5	84.9	93.5	88.0
MultinomialNB	TF-IDF w/ bi-gram	94.3	82.7	94.7	86.3

However, it is obvious that there is a large amount of over-fitting that is taking place. To overcome this, hyper-parameter tuning was implemented. MultinomialNB does not have many parameters to train other than a smoothing factor. For SGDC-SVM, different values were tried out for its number of epochs, regularization value and learning rate with the help of grid search cross validation (GridSearchCV in Scikit-Learn). Different values of training-testing split were also experimented with. Unfortunately, these were the best results possible. Finally, it was decided to increase the size of the dataset. To add more variance in the data and due to time constraints, two publicly available datasets were added to double the size of the original dataset. The results after adding more data are seen in Table 3. It is observed that over-fitting reduces to a much more acceptable measure. SGDC with SVM can be chosen as a top performer for this dataset as it has minimum over-fitting when compared to the other classifiers and an acceptable accuracy.

B. Deep Learning Approach

To improve on the performances of machine learning models, various deep learning architectures were tried. Since embeddings will be used here, in the data cleaning process, stopwords were not removed and the data was not lemmatized. This ensures that the model gets a full understanding of the data and the embeddings are able to capture semantics accurately. Python's Keras library was used for implementation. Keras has a Tokenizer API which provides the functionality of converting text to integers or converting the text into a vectorized version of desired format like TF-IDF, one-hot encoded format, count of words in the document etc. Initially, a simple multi-layer perceptron was developed, with the vectorized text as its input. The training accuracy was 98.6% and the test accuracy was 86.5%. This was created to serve as a *baseline model* for the system. Clearly, there is a large amount of overfitting in this case too.

To improve upon this, word embeddings are used instead of simple vectorized versions of texts. Keras provides an embedding layer that converts the above created vectorized version of text to embeddings. It learns from the training data and creates embeddings based on it. This embedding layer can then be passed on to LSTMs, GRUs etc. for further processing. Initially, this model was trained on the manually created dataset only. For all of the architectures, the activation function used is ReLu and for the output layer it is softmax. Like the machine learning results, there is over-fitting happening in this case. Measures like increasing the dropout rate to 50% and adding maximum pooling to the convolutional network were taken. A dropout rate of 0.5 implies that 50% of the neurons from that layer are deleted so as to generalize the model better. Max pooling performs a similar function in case of CNNs where it reduces the dimensionality of the feature matrix. Dropout was added after each layer and L2 regularization was also added, which did not lead to much success. Hence, the size of the training data was doubled. The results are displayed in Table 5. On adding additional data, the accuracy improves significantly and there is reduction in over-fitting too, when compared to the previous dataset. The LSTM on the extended data performs the best in this scenario.

However, the accuracy of the model can still be improved. At this stage, the results are at par with the machine learning models, but not better. Deep learning models are also computationally expensive. It feels like it is better to use the machine learning approach since it provides a similar accuracy, with lesser over-fitting and takes much less time to run. In the above Keras embeddings, the embeddings are trained on the training data which is a much smaller set when compared to the English language as a whole. If embeddings are unable to capture all of the semantics correctly, the model will not perform better. Hence, we turn towards GloVe embeddings. GloVe are pre-trained vectors, thus they

reduce the training time of the model and can be used on the model directly. GloVe has been trained on billions of Wikipedia articles and web data, thus can capture semantics in an excellent and wholesome manner [23]. We use GloVe embeddings on the same architectures as above and both the datasets. On the extended dataset, we achieve much higher accuracies and the over-fitting has reduced drastically. The results are displayed in Table 5. The CNN-LSTM architecture performs the best with minimum over-fitting and maximum accuracy out of all the architectures. The architecture of the model is shown in Figure 1.

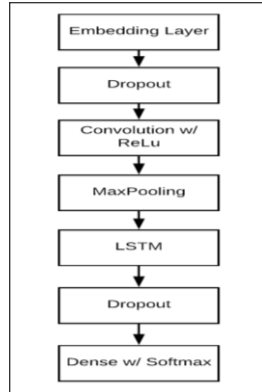


Figure 1 : CNN-LSTM Architecture

Table 4 : Deep Learning Using Keras Embedding

Model	Manual Dataset Accuracy (%)		Extended Dataset Accuracy (%)	
	Train	Test	Train	Test
CNN w/ LSTM	98.2	83.3	98.4	89.5
GRU	95.8	78.6	97.6	86.8
LSTM	96.5	82.5	93.2	85.5

Table 5 : Deep Learning Using Glove Embedding

Model	Manual Dataset Accuracy (%)		Extended Dataset Accuracy (%)	
	Train	Test	Train	Test
CNN w/ LSTM	92.3	85.6	93.3	91.1
GRU	95.1	84.3	92.2	90.0
LSTM	91.5	82.1	92.8	87.8

VIII. CONCLUSION

Both machine learning and deep learning approaches were used to evaluate the credibility of news articles. While machine learning articles achieve a decent accuracy, they still suffer from some over-fitting and feature extraction techniques like Bag of Words, TF-IDF etc. do not provide the optimal solution in this case. Word embeddings like GloVe can be used instead as they capture the semantics of the text much better. When used with deep learning models, they provide better accuracy than machine learning models. The only disadvantage of deep learning architectures is that they are computationally expensive and require powerful processors for faster computation.

Future work would involve building a web application to deploy the model. This way the model can be put into actual use. For achieving better performance of the deep learning models, deeper architectures with finer hyper-parameter tuning can be done.

IX. REFERENCES

[1] Singh, Anita. "'Cuffing Season' and 'Corbynmania' Are Named Words of the Year by Collins Dictionary." The Telegraph, 2 Nov. 2017, <https://www.telegraph.co.uk/news/2017/11/02/cuffing-season-corbynmania-named-words-year-collins-dictionary/>. Accessed 20 Apr. 2025.

[2] Wardle, Claire. "Fake News. It's Complicated." First Draft, 16 Feb. 2017, <https://firstdraftnews.org/fake-news-complicated/>. Accessed 20 Apr. 2025.

- [3] Álvarez, Miguel M. "How Can Machine Learning and AI Help Solving the Fake News Problem?" Miguel M. Álvarez, 23 Mar. 2017, <https://miguelmalvarez.com/2017/03/23/how-can-machine-learning-and-ai-help-solving-the-fake-news-problem/>. Accessed 20 Apr. 2025.
- [4] Wang, William Yang. "'liar, liar pants on fire': A new benchmark dataset for fake news detection." arXiv preprint arXiv:1705.00648 (2017).
- [5] Shu, Kai, et al. "Fake news detection on social media: A data mining perspective." ACM SIGKDD explorations newsletter 19.1 (2017): 22-36.
- [6] Tacchini, Eugenio, et al. "Some like it hoax: Automated fake news detection in social networks." arXiv preprint arXiv:1704.07506 (2017).
- [7] Ruchansky, Natali, Sungyong Seo, and Yan Liu. "CSI: A Hybrid Deep Model for Fake News Detection." Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, 2017, pp. 797-806.
- [8] Ajao, Oluwaseun, Deepayan Bhowmik, and Shahrzad Zargari. "Fake News Identification on Twitter with Hybrid CNN and RNN Models." Proceedings of the 9th International Conference on Social Media and Society, 2018, pp. 226-230.
- [9] Singh, UMBER, Talieh Hajzargarbashi, Sasanka Gandavarapu, and Brennan Borlaug. "Make News Credible Again." UC Berkeley School of Information, 2017, <https://www.ischool.berkeley.edu/projects/2017/make-news-credible-again>. Accessed 20 Apr. 2025.
- [10] Estela, Zach. "D4D Project Highlight: Are You Fake News?" Medium, Data for Democracy, 27 Nov. 2018, <https://medium.com/data-for-democracy/d4d-project-highlight-are-you-fake-news-2c8b3930f804>. Accessed 20 Apr. 2025.
- [11] Szpakowski, Maciej. "FakeNewsCorpus." GitHub, 24 Jan. 2020, <https://github.com/several27/FakeNewsCorpus>. Accessed 20 Apr. 2025.
- [12] Media Bias/Fact Check. Media Bias/Fact Check, 2025, <https://mediabiasfactcheck.com/>. Accessed 20 Apr. 2025.
- [13] Kaggle. "Fake News Detection." Kaggle, <https://www.kaggle.com/c/fake-news/data>. Accessed 20 Apr. 2025.
- [14] McIntire, George. "fake_real_news_dataset." GitHub, 24 Jan. 2020, https://github.com/GeorgeMcIntire/fake_real_news_dataset. Accessed 20 Apr. 2025.
- [15] Sarkar, Dipanjan. "A Practitioner's Guide to Natural Language Processing (Part I): Processing & Understanding Text." Towards Data Science, 20 June 2018, towardsdatascience.com/a-practitioners-guide-to-natural-language-processing-part-i-processing-understanding-text-9f4abfd13e72. Accessed 20 Apr. 2025.
- [16] Loper, Edward, and Steven Bird. "Nltk: The natural language toolkit." arXiv preprint cs/0205028 (2002).
- [17] "spaCy: Industrial-Strength Natural Language Processing in Python." spaCy, Explosion AI, <https://spacy.io/>. Accessed 20 Apr. 2025.
- [18] D'Souza, Jocelyn. "An Introduction to Bag-of-Words in NLP." GreyAtom, 3 Apr. 2018, <https://medium.com/greyatom/an-introduction-to-bag-of-words-in-nlp-ac967d43b428>. Accessed 20 Apr. 2025.
- [19] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.
- [20] Fueyo, Enrique. "WTF is TF-IDF?" KDnuggets, 6 Aug. 2018, <https://www.kdnuggets.com/2018/08/wtf-tf-idf.html>. Accessed 20 Apr. 2025.
- [21] Karani, Dhruvil. "Introduction to Word Embedding and Word2Vec." Medium, 1 Sept. 2018, <https://medium.com/data-science/introduction-to-word-embedding-and-word2vec-652doc2060fa>. Accessed 20 Apr. 2025.
- [22] Kurita, Keita. "Paper Dissected: 'Glove: Global Vectors for Word Representation' Explained." Machine Learning Explained (2018).
- [23] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation." Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1532-1543.
- [24] Keras. "Keras: Deep Learning for Humans." Keras, <https://keras.io/>. Accessed 20 Apr. 2025.
- [25] Olah, Christopher. "Understanding LSTM Networks." colah's blog, 27 Aug. 2015, <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>. Accessed 20 Apr. 2025.
- [26] Kim, Yoon. "Convolutional Neural Networks for Sentence Classification." arXiv, 25 Aug. 2014, <https://arxiv.org/abs/1408.5882>. Accessed 20 Apr. 2025.