Fake News Detection Using Logistic Regression and CNN

Mithat Kus

April 29, 2024

1 Introduction

In recent years, with the increasing volume of information available online, society has become increasingly susceptible to exposure to incorrect or misleading information. In order to prevent the undesirable consequences of fake news dissemination many studies are being conducted to detect fake news. This study will be focusing on using Logistic Regression and a Deep Learning algorithm, namely Convolutional Neural Networks (CNN), together with pre-trained word embeddings to identify real and fake news based on text data. Although typically used for image classification, CNN's have also proven effective in text classification. [14] [18] [10] This study specifically targets fake news articles and uses a prominent dataset used in fake news detection studies named ISOT Fake News Dataset. Although the dataset contains different types of articles on different topics, the majority of articles focus on political and world news topics. The results of this study show that the Logistic Regression model is able to accurately predict the results with up to 94% accuracy and the CNN model with up to 98.5% accuracy. The Google Colab notebook used in this study is available on this Github link and the original ISOT dataset, as well as the pre-processed version of the dataset is available on this Google Drive link

2 Related Works

There are many methods utilized by various studies in fake news classification tasks. This section outlines the main approaches and compares them with the techniques used in this study.

2.1 Content-Based Analysis

This approach relies on the textual content of news articles. It involves the use of Natural Language Processing (NLP) techniques to understand the grammatical structure, context, and sentiment of the text. In these studies, various machine learning algorithms such as Naive Bayes, Support Vector Machines (SVM), and Decision Trees are utilized. These methods require careful feature engineering to identify the most telling features of fake news. [8]

2.2 Social Context Analysis

Beyond the content itself, some methods focus on the social context in which information spreads, such as the structure of social networks, the pattern of shares, and user metadata. This means that the behavioral patterns that accompany the spread of news, such as how frequently an article is shared and who is sharing it, are inspected. This method may require the exploration of important concepts such as Graph Theory or Graph Neural Networks. [16]

2.3 Deep Learning Techniques

Similarly to our study, deep learning algorithms are often preferred for fake news detection studies. Most such studies use deep learning algorithms that are often used for text classification such as Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM) networks, and Transformer-based models like BERT. These models are often used in conjunction with word embeddings, as in our study. [17] [12] [15]

3 Resources

This study specifically targets fake news articles and uses a prominent dataset used in fake news detection studies named ISOT Fake News Dataset. The dataset contains two types of articles: fake and real news. This dataset was collected from real-world sources; the truthful articles were obtained from Reuters.com articles and the fake news articles were collected from unreliable websites that were flagged by Politifact (a fact-checking organization in the USA) and Wikipedia. The dataset contains different types of articles on different topics, however, the majority of articles focus on political and world news topics. The dataset consists of two CSV files. The True.csv file contains the true news and consists of 21417 text instances.

The Fake.csv file contains the fake news and consists of 23481 text instances. Each article contains the following information: article title, text, label, date and subject.

The experiments were conducted using Google Colab through a virtual environment with the following specifications:

- **CPU:** Intel(R) Xeon(R) CPU @ 2.20GHz, 2 cores, 56320 KB cache, running at 2199.998 MHz.
- **Memory:** 13 GB total, with approximately 12.3 GB available during the sessions.
- **Storage:** 108 GB total, with 82 GB available.

In the development of the Convolutional Neural Network (CNN) model for this study, ChatGPT was used as a tool to assist in generating the code and architecture design. Specifically, ChatGPT provided guidance on setting up the hyperparameters and structure of the CNN using the Keras library in TensorFlow. This included defining the model's layers, such as the embedding layer with pre-trained word embeddings, convolutional layers with variable filter sizes and kernel sizes, pooling layers, and dense layers.

4 Research Method

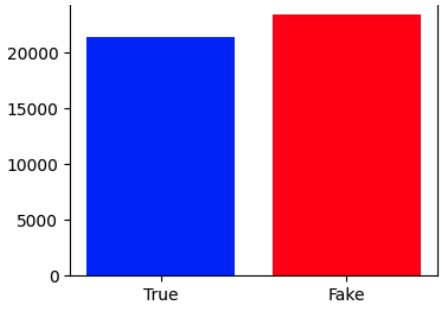
Methods used in this study consist of exploratory data analysis, data pre-processing, creating word embedding vectors, PCA, logistic regression, CNN and the analysis of model results. The rest of the report is dedicated to exploring the details of each of the methods used in this study.

4.1 Exploratory Data Analysis

During this step, the main objective is to understand the structure of the data, identify patterns, uncover anomalies, and extract useful insights that can guide further analysis and model building. The structure of the data and the sizes of both labels can be visualized as follows:

title	text	subject	date
As U.S. budget fight looms, Republicans flip t...	WASHINGTON (Reuters) - The head of a conservat...	politicsNews	December 31, 2017
U.S. military to accept transgender recruits o...	WASHINGTON (Reuters) - Transgender people will...	politicsNews	December 29, 2017
Senior U.S. Republican senator: 'Let Mr. Muell...	WASHINGTON (Reuters) - The special counsel inv...	politicsNews	December 31, 2017
FBI Russia probe helped by Australian diplomat...	WASHINGTON (Reuters) - Trump campaign adviser ...	politicsNews	December 30, 2017
Trump wants Postal Service to charge 'much mor...	SEATTLE/WASHINGTON (Reuters) - President Donal...	politicsNews	December 29, 2017

Data Structure Overview



Distribution of Class Sizes

The data contains features that are irrelevant to the focus of our study, such as subject and date, which will be dealt with in the data pre-processing step. The most important thing to realize is that the data could be considered balanced, with 21417 True and 23481 Fake news articles, and therefore does not require any augmentation or balancing techniques such as over-sampling or under-sampling.

4.2 Data Pre-processing

The text data contains several features that could lead to bias and overfitting that have to be removed. The pre-processing for this dataset includes:

- Removing the news source, numbers, punctuation, and stop words.
- Changing text to lowercase to standardize the text.
- Removing non-textual elements such as links, URLs, mentions, hashtags, and image credits.
- Stemming each word to its root to reduce noise and simplify the feature space.
- Appending the title of the text at the beginning of the text instance.
- Removing all features except class and text

As a result of these pre-processing steps, the data frame structure is as follows: the `class` column where 0 corresponds to Fake and 1 corresponds to True news, and the `clean_text` column which contains the title and the pre-processed news article.

4.3 Word Embeddings

Pre-trained word embedding is an approach in Natural Language Processing where a large corpus of words have corresponding vectors that represent the semantic meaning and the context of the words. In this study, we are using a 300-dimensional Word2Vec Pre-trained word embedding.

Below is an example of how the word embedding vectors can be visualized on a two-dimensional plane after applying PCA. It is noticeable that words that carry similar meanings are grouped together. This property is very valuable for machine learning models as it allows them to understand that words such as "university" and "school" are very similar in meaning.

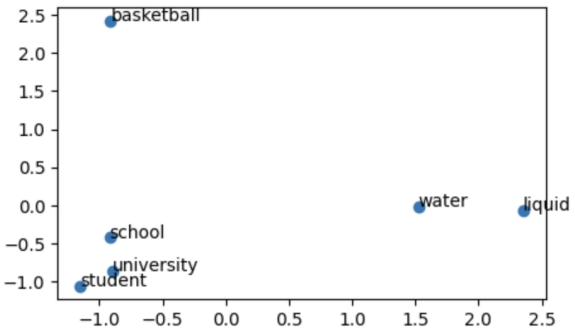


Figure 1: Visualization of word embedding vectors after applying PCA

The similarity between words can be calculated through cosine similarity, which is defined by the following formula:

$$\text{Cosine Similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

where $\mathbf{A}$ and $\mathbf{B}$ are vectors, and $\cdot$ represents the dot product.

4.3.1 Applying Word Embedding on the Dataset For Logistic Regression

This part involves creating the input to our Logistic Regression model. For each text instance, the average of all the 300 dimensional word vectors are calculated and thus a 44851 x 300 matrix is created.

4.4 PCA

Principal Component Analysis (PCA) is a statistical technique used for dimensionality reduction that preserves as much of the data's variability as possible. The goal of PCA is to find projections $\tilde{x}_n$, the orthogonal principal components, of data points x_n that are as similar to the original data points as possible. PCA starts by identifying the direction in which the data varies the most, which is the first principal component. This direction, or vector, represents the greatest variance in the data. The second principal component is calculated in the same way, with the condition that it is orthogonal to the first principal component and captures the next highest variance. This process is repeated for as many components as there are dimensions in the data.

Consider an i.i.d. dataset $X = \{x_1, \ldots, x_N\}$ with mean 0 that possesses the data covariance matrix: $S = \frac{1}{N}\sum_{n=1}^{N} x_n x_n^T$.
And a low dimensional compressed representation such that $z_n = B^\top x_n$.

Our aim is to find a matrix B that retains as much information as possible which is equivalent to capturing the largest amount of variance in the low-dimensional representation.

We start by seeking a single vector b_1 that maximizes the variance of the projected data. This means that we aim to maximize the variance of the first coordinate of z so that

$$V_1 := V[z_1] = \frac{1}{N}\sum_{n=1}^{N} z_{1n}^2$$

is maximized. This yields:

$$V_1 = \frac{1}{N}\sum_{n=1}^{N}(b_1^\top x_n)^2 = \frac{1}{N}\sum_{n=1}^{N} b_1^\top x_n x_n^\top b_1 = b_1^\top \left(\frac{1}{N}\sum_{n=1}^{N} x_n x_n^\top\right) b_1 = b_1^\top S b_1$$

The vector b_1 that points in the direction of maximum variance can be found by the optimization problem: $\max_{b_1} b_1^\top S b_1$.
Given $Sb_1 = \lambda_1 b_1$ and $b_1^\top b_1 = 1$, we get

$$V_1 = b_1^\top S b_1 = \lambda_1 b_1^\top b_1 = \lambda_1$$

This means that the variance of the data projected onto a one-dimensional subspace equals the eigenvalue that is associated with the basis vector b_1 that spans this subspace. Therefore, to maximize the variance of the low-dimensional representation, we choose the basis vector associated with the largest eigenvalue of the covariance matrix.

The PCA method implemented in this study does exactly the same operation of computing the eigenvectors with the largest eigenvalues from the covariance matrix. Upon applying PCA to the 44851 x 300 matrix that was constructed by taking the average of the word embedding vectors for each text instance, the result is a 44851 x 2 matrix where each vector is a principal component. [9]

4.5 Logistic Regression

The logistic function, also known as the sigmoid function, is used to model the probability $P(y = 1 \mid X)$ of the binary outcome variable y being 1. Mathematically, it is represented as:

$$P(y = 1 \mid X) = \frac{1}{1 + e^{-(b_0 + b_1 x_1 + \ldots + b_m x_m)}}$$

Where x is the vector of input features $[x_1, x_2, \ldots, x_m]$ and the coefficients that the model will learn are denoted as $[b_0, b_1, \ldots, b_m]$.

The goal of logistic regression is to find the set of coefficients b that maximize the likelihood function, which represents the probability of observing the given data. For binary outcomes, this likelihood is the product of probabilities assigned to actual outcomes:

$$L(b) = \prod_{k:y_k=1} P(y_k) \prod_{k:y_k=0} (1 - P(y_k))$$

Taking the logarithm of this function gives the following log-likelihood function:

$$l(b) = \sum_{k:y_k=1} \ln(P(y_k)) + \sum_{k:y_k=0} \ln(1 - P(y_k))$$

To find the coefficients b, we use the iterative optimization algorithm gradient descent. The gradient descent algorithm updates the coefficients in the direction that leads to the maximum increase in the log-likelihood. The gradient of the log-likelihood function with respect to the coefficients b tells us how much we would increase the log-likelihood if we made a small change in b.

The gradient descent is calculated as:

$$\nabla_b l(b) = X^T (Y - P)$$

Where X is the matrix of input features, Y is the vector of observed outcomes, P is the vector of predicted probabilities, and $Y - P$ is the vector of residuals or errors.

During each iteration of gradient descent, the coefficient vector b is updated by subtracting the product of the gradient and a step size (learning rate):

$$b := b - \alpha \nabla l(b)$$

[5] [4]

4.6 Logistic Regression Results

Upon thorough testing of both the original and PCA-reduced dataframes with numerous step sizes, the highest accuracy for the original dataframe was obtained with step size 0.05, and the highest accuracy for the PCA-reduced dataframe was obtained with step size 0.07.

Logistic Regression Accuracy	**Original DataFrame**	**PCA Reduced DataFrame**
Accuracy	0.9381	0.8322

Table 1: PCA vs no PCA Logistic Regression Accuracy Comparison

PCA is often most effective in scenarios where the dataset contains a mixture of both informative and noisy features. The low accuracy of the PCA-reduced dataframe in our case can be attributed to the fact that each of the 300 dimensions of the word embedding model captures an important feature of the data. As a result, applying PCA on our dataframe most likely removed very important features that do not carry the highest variance, which caused the linear regression model to underperform in the absence of those important features.

4.7 Introduction to CNN and its parameters

4.7.1 Our CNN Model

A CNN is a neural network, an algorithm to recognize patterns in data. Neural Networks are formed by a collection of neurons that are grouped in layers, each with their own learnable weights and biases. [11]

The Input: Each input instance is a matrix that represents a single text document, where the rows of the matrix correspond to the words in the text, and each word is represented by its corresponding vector from our word embedding model. Each text document is first tokenized, then converted into integers since our model needs numerical data to make computations. In order to focus on the most important linguistic features in our data, only the 35000 most frequent words out of the 168818 words in our data are kept in order to reduce the model complexity. After tokenization each token is mapped to its corresponding vector using our embedding matrix.

The sequences of integers representing the words in each document vary in length, depending on the original length of the document. However, neural networks, specifically those involving layers that expect fixed-size inputs like Convolutional Neural Networks, require that all input data must have the same shape.

To address the varying lengths of data and to standardize the input for the neural network, we employ padding. Padding involves adding 0 to the sequence of data until each sequence reaches the same predetermined length. In our model, this length is set to 430. This number was calculated by taking the 90th percentile of the word counts, which means that 90 percent of the text instances have 430 words or fewer. This process ensures that all input matrices are of uniform size, which is crucial for batch processing in neural networks. The padding values are non-informative and have minimal interference with the learning process.

Embedding Layer: The embedding layer is the first layer of the CNN and it is responsible for converting sequences of word indices into their corresponding vector representations. In our implementation, the embedding layer utilizes our pre-established embedding matrix as weights, which are set to be non-trainable in order to retain the pre-learned word associations. This layer outputs a 3D tensor, where each word in a text sequence is represented by its dense vector.

Convolutional Layer: Following the embedding layer, the convolutional layer applies multiple filters to the embedded word vectors. These filters are designed to identify specific features in the text, such as the presence of n-grams. Each filter slides across the word embeddings, applying a convolution operation—element-wise multiplication followed by summation. This results in a feature map that highlights regions of the input text where specific linguistic features are detected. Our convolutional layer uses the Rectified Linear Unit (ReLU) activation function to learn the complex patterns.

Pooling Layer: The pooling layer is employed after the convolutional layer in order to reduce the dimensionality of the feature maps. This layer performs down-sampling by selecting the maximum value (max pooling) from each region of the feature map. This allows extracting dominant features that are robust to noise in the input data in order to simplify the information that the network needs to process.

Flattening Layer: The flattened output is necessary to transition from the 2D feature maps to a 1D feature vector, which can be fed into subsequent dense layers. The flatten layer reshapes the pooled feature maps into a single long vector and maintains the most important features extracted by the convolutional and pooling layers.

Dense Layers: Towards the end of the CNN architecture, dense layers are used to perform high-level reasoning based on the features extracted earlier. These layers are fully connected, which means that each neuron in a dense layer receives input from all neurons in the previous layer. This allows the model to learn non-linear combinations of the features. The final dense layer in our model outputs a probability that indicates whether the input text is likely to be fake news.

4.7.2 Adam: Adaptive Moment Estimation Algorithm

Adam [13] [3] is an adaptive learning rate algorithm that is designed to improve training speeds in neural networks and reach convergence quickly. The algorithm optimizes each parameter's learning rate based on its gradient history which helps the neural network learn very efficiently. In order to understand this algorithm, it's important to understand the basic gradient descent algorithm:

$$\theta := \theta - \alpha \nabla l(\theta)$$

Here θ represents the model parameters, α represents the step size (learning rate), and $\nabla l(\theta)$ represents the gradient of the cost function. This update changes the parameters in the negative direction of the gradient in order to minimize the cost function where α determines the size of the step. In this algorithm the learning rate is fixed which means that we need to

manually change the alpha where a small step size will lead to very slow convergence and a large step size will overshoot and miss the minima.

Adam solves this issue by adapting the step size α for each parameter θ. Adam has two important concepts that make it very effective.

First is momentum. Momentum is essentially adding a fraction of the previous gradient to the current one. The momentum term proportional to the previous gradients accumulate and accelerate the optimization in that direction.

$$v_t = \gamma v_{t-1} + \eta g_t$$
$$\theta = \theta - v_t$$

The momentum vector at time t, v_t, is a function of the previous momentum vector v_{t-1}. γ is the momentum decay that exponentially decays the previous momentum vector and η is the learning rate used to take the step in the opposite direction of the gradient.

RMSProp, which stands for Root Mean Squared Propagation, is another optimization algorithm that Adam takes advantage of. In RMSProp, we look at the steepness of the error surface for each parameter to update the learning rate adaptively.

$$E[g_t^2] = \gamma E[g_{t-1}^2] + (1-\gamma) g_t^2$$
$$\theta = \theta - \left(\frac{\eta}{\sqrt{E[g_t^2] + \epsilon}} \right) g_t$$

The first equation is a weighted moving average of squared gradients, which gives the variance of gradients. When updating the θ, the learning rate is divided by the moving average of the squared gradients.

The Adam algorithm, combining the two concepts, can be represented as the following equations:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$
$$\theta = \theta - \left(\frac{\alpha m_t}{\sqrt{v_t + \epsilon}} \right)$$

The following graphs are the visualizations of the algorithm trajectory for fairly straightforward optimization tasks. They help visualize how momentum and RMSProp differ in their objective of reaching the minima, and how Adam combines the characteristics of the two.

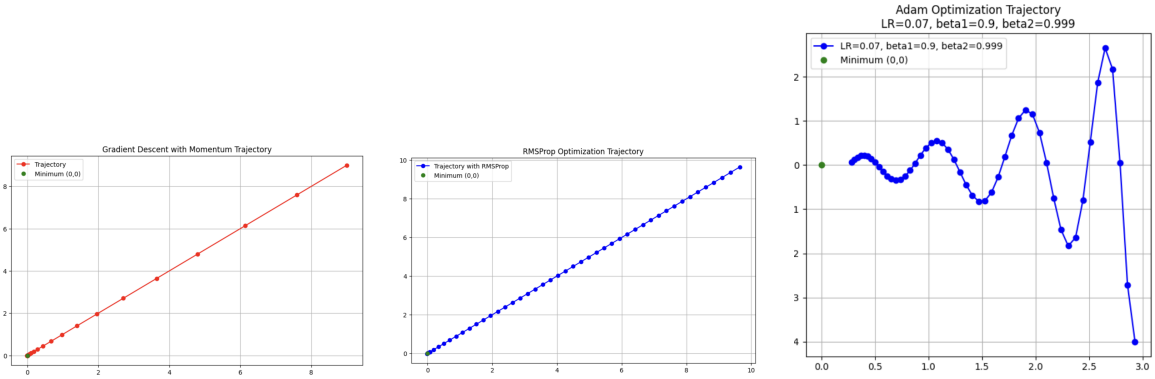


Figure 2: Comparison of optimization algorithms: Momentum, RMSProp, and ADAM.

4.7.3 Binary Cross Entropy

Entropy is a measure of the uncertainty associated with a given distribution $q(y)$. In binary classification, each instance is supposed to belong to one of two classes, labeled as 1 or 0. The binary cross-entropy loss is a measure of how well the model's predicted probabilities align with the actual labels. The formula for binary cross-entropy loss for a single data point is:

$$L(y, p) = -y \log(p) - (1-y) \log(1-p)$$

The intuition behind the formula comes from a simple motivation. Since we're trying to compute a loss, we need to penalize bad predictions and reward good predictions. Let's consider both scenarios. In a good prediction (Probability = 1,

True Label = 1), plugging the values into the loss function gives $L(1, 1) = 0$. In a poor prediction (Probability = 0.01, True Label = 1), the function outputs $L(1, 0.01) = -\log(0.01)$ where $\log(0.01)$ is a negative number, and therefore the result is a large positive number which indicates a big loss. The same idea applies when the True Label = 0.

Plotting the loss function as a function of the probability p while keeping the true label constant, we get the following graphs that perfectly represent our intuition:

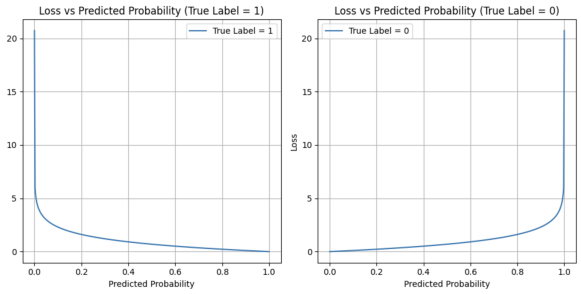


Figure 3: Binary Cross-Entropy Loss as a function of predicted probability p for a given true label.

As a result, Binary Cross-Entropy is a powerful metric for evaluating the performance of classification models as it offers insights into how well the model's predictions align with the actual labels. [7]

4.7.4 Activation Functions

An activation function [1] is a function that is added into a neural network in order to help the network learn complex patterns in the data. It takes in the output signal from the previous layer and converts it into a form that can be taken as input to the next layer. In order for the model to learn the non-linear patterns, specific nonlinear layers (activation functions) are added in between.

ReLU The Rectified Linear Unit [6] is a commonly used activation function in deep learning models. The function returns 0 if it receives a negative input, and returns the input directly if it is positive. The function can be written as f(x)=max(0,x) and can be visualized as follows:

The ReLU function can be visualized as follows:

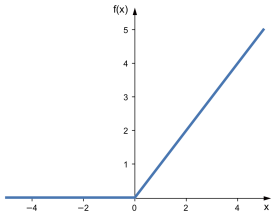


Figure 4: The Rectified Linear Unit (ReLU) activation function.

The ReLU function has a derivative of 0 for negative inputs and 1 for positive inputs. It accelerates the convergence of gradient descent towards the global minimum of the loss function due to its linear property. Since the output is zero for all negative inputs, it causes some nodes to completely die and not learn anything, which is referred to as the "dying ReLU" problem. However, when training on a reasonably sized batch, there will usually be some data points that provide positive values to any given node, therefore the average derivative is rarely close to 0, which allows gradient descent to keep progressing.

4.8 Analysis of our CNN Model

In this study, the dataset was divided using an 80-20 split. The training data consists of a large subset of the preprocessed and padded feature vectors, which the model uses to learn and adjust its parameters. [2]

4.8.1 Hyperparameter Tuning

The best hyperparameters are determined during the training phase. The process consists of iteratively adjusting the model's architecture and parameters to find the combination that yields the best performance on a validation set. Below is the figure representing our model's best parameters, which means that this exact combination lead to the smallest amount of loss.

```
Best number of filters in the first Conv1D layer: 96
Best kernel size for the first Conv1D layer: 7
Best pooling size for the first MaxPooling1D layer: 5
Best number of units in the first Dense layer: 224
```

Figure 5: Optimal hyperparameters for the CNN model.

4.8.2 The Confusion Matrix

The confusion matrix outcomes indicate that the model correctly identified 4193 true news articles as True Positives (TP) and 4653 fake news articles as True Negatives (TN). However, there were 54 instances where fake news articles were incorrectly classified as true news, labeled as False Positives (FP), and 74 instances where true news articles were incorrectly classified as fake news, labeled as False Negatives (FN).

The performance metrics derived from the confusion matrix are as follows:

- Accuracy (98.57%): The model correctly classifies news articles as fake or true the majority of the time.
- Precision (98.73%): The probability that the model predicts true news as fake news is very low.
- Recall (98.27%): The model is very effective at identifying true news articles

4.8.3 Model Performance Curves

The Precision-Recall and Receiver Operating Characteristic (ROC) curves are graphical representations that illustrate the performance of a binary classification task.

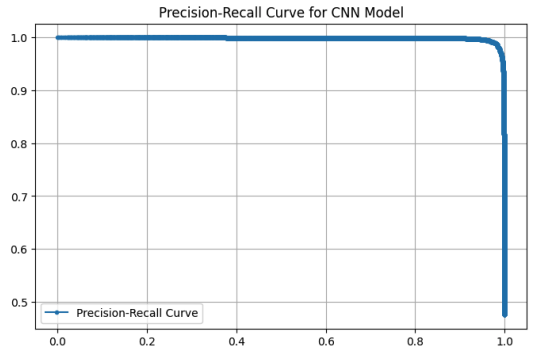


Figure 6: Precision-Recall Curve

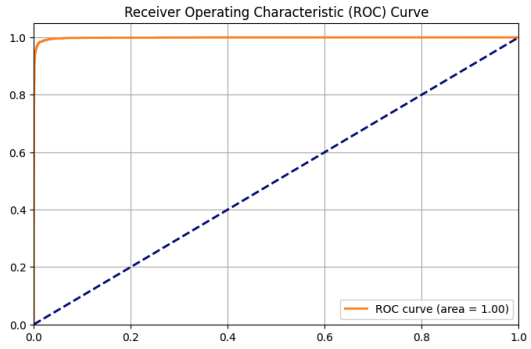


Figure 7: ROC Curve

The precision-recall curve demonstrates the trade-off between precision and recall for various threshold levels. Our curve maintains a high precision at varying levels of recall which suggests that the model does not significantly compromise on precision to improve recall. The ROC curve, on the other hand, plots the true positive rate against the false positive rate. Our curve closely follows the top left corner of the plot which means there is an almost excellent separation between the classes.

5 Conclusion

In conclusion, this study utilizes Logistic Regression and Convolutional Neural Networks together with pre-trained word embeddings to detect fake news articles. The most important and powerful aspect of this study is the use of pre-trained word embeddings. They allow the models to uncover certain features of the articles that would otherwise have been impossible to bring out. The embeddings represent each word so powerfully that even a simple logistic regression model was able to reach accuracies of up to 94%. Nevertheless, the Convolutional Neural Network model excelled further and achieved an impressive accuracy of up to 98.5%. This highlights the effectiveness of CNNs in handling complex patterns in text data, which makes them particularly suited for text classification tasks such as fake news detection.

References

[1] Activation functions in neural networks. *Towards Data Science*.

[2] Building a convolutional neural network (cnn) in keras. *Towards Data Science*.

[3] Complete guide to the adam optimization algorithm. *BuiltIn*.

[4] Logistic regression. *Wikipedia*.

[5] Logistic regression simply explained. *Datalab*.

[6] Rectifier, neural networks. *Wikipedia*.

[7] Understanding binary cross-entropy / log loss: a visual explanation. *Towards Data Science*.

[8] Sajjad Ahmed, Knut Hinkelmann, and Flavio Corradini. Development of fake news model using machine learning through natural language processing, 2022.

[9] Marc Peter Deisenroth, Cheng Soon Ong, and Aldo A. Faisal. *Mathematics for Machine Learning*. Cambridge University Press, 2021.

[10] Bao Guo, Chunxia Zhang, Junmin Liu, and Xiaoyi Ma. Improving text classification with weighted word embeddings via a multi-channel textcnn model. *Neurocomputing*, 2019.

[11] Aurélien Géron. *Hands-on machine learning with scikit-learn, Keras, and tensorflow: Concepts, tools, and techniques to build Intelligent Systems*. O'Reilly Media, 2023.

[12] Rohit Kumar Kaliyar, Anurag Goswami, Pratik Narang, and Soumendu Sinha. Fndnet – a deep convolutional neural network for fake news detection. *Cognitive Systems Research*, 2020.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[14] Yuandong Luan and Shaofu Lin. Research on text classification based on cnn and lstm. 2019.

[15] Jamal Abdul Nasir, Osama Subhani Khan, and Iraklis Varlamis. Fake news detection: A hybrid cnn-rnn based deep learning approach. *International Journal of Information Management Data Insights*, 2021.

[16] Van-Hoang Nguyen, Kazunari Sugiyama, Preslav Nakov, and Min-Yen Kan. Fang: Leveraging social context for fake news detection using graph representation. ACM, 2020.

[17] I. Kadek Sastrawan, I.P.A. Bayupati, and Dewa Made Sri Arsa. Detection of fake news using deep learning cnn–rnn based methods. *ICT Express*, 2022.

[18] Ming Wang Xiliang Zhang Weidong Zhao, Lin Zhu and Jinming Zhang. Wtl-cnn: a news text classification method of convolutional neural network based on weighted word embedding. *Connection Science*, 2022.