FYS-STK3155 Project 3

Elaha Ahmadi, Theodor Jaarvik, Herman Scheele

December 2024

Abstract

The rapid spread of misinformation online has raised significant concerns about the reliability of digital information ecosystems. In response, we apply three distinct classification methods—Logistic Regression, a feedforward Neural Network, and a Decision Tree—to the task of fake news detection using a curated Kaggle dataset comprising both true and fake articles. By transforming raw textual data into TF-IDF features, we achieve consistently high accuracy scores, often exceeding 99%, and corresponding ROC AUC values around 0.99. These outstanding results suggest that current models are adept at exploiting easily identifiable lexical cues in the provided dataset. A closer examination using SHAP values and model coefficients reveals that performance can hinge on a few highly discriminative tokens. Even after removing top-impact terms, the Neural Network and Logistic Regression models maintain near-perfect accuracy, while the Decision Tree's performance declines more noticeably. This finding underscores the importance of robust feature analysis and questions the generalizability of models trained on domain-specific indicators. Future work should therefore focus on cross-domain validation, adversarial testing, and incorporating advanced architectures to ensure that detection methods remain effective beyond the confines of a single dataset and static linguistic environment.

Introduction

The widespread proliferation of misleading and deceptive online content—often referred to as fake news—has made reliable automated detection methods increasingly important. As misinformation campaigns grow more sophisticated, their impact extends beyond casual readers, influencing public opinion, policy-making, and social stability. Consequently, the development of robust, generalizable fake news classifiers has become a critical focus within the field of natural language processing and machine learning.

In tackling the challenge of fake news detection, researchers have explored a wide range of algorithms, from classical linear models to complex deep neural networks. While analytical closed-form solutions (such as those available in ordinary least squares regression) provide clarity and interpretability, they often

fail to capture the nuanced linguistic structures and subtle semantic patterns present in textual data. Non-analytical classical methods, like Logistic Regression trained via gradient-based optimization or Decision Trees constructed through recursive feature splitting, offer a crucial balance: they can approximate complex decision boundaries more effectively than purely analytical solutions, yet remain more interpretable and computationally tractable than many deep learning approaches.

Our choice of dataset is guided by the need for high-quality, diverse, and representative textual sources that reflect real-world misinformation. To this end, we utilize a well-established dataset from Kaggle, containing both "true" and "fake" articles. This curated collection spans relevant themes and styles, ensuring that our models encounter realistic writing patterns. Such a dataset not only provides a robust test bed for evaluating model performance but also ensures that insights learned from our experiments can have meaningful implications for applied misinformation detection systems.

In this report, we begin by describing the data, including its origin and preprocessing steps, to highlight how we transform raw textual content into a suitable numerical format for machine learning. We then detail the theoretical foundation and implementation of three distinct models: a Logistic Regression classifier, a feedforward Neural Network for capturing nonlinear relationships and richer representations, and a Decision Tree model for feature importance assessment. Following this, we present empirical results evaluating these models across a range of metrics, including accuracy, ROC AUC, and feature importance measures. We critically analyze the outcomes, discussing how certain words dominate classification decisions, and we explore how removing these key features affects model robustness. Finally, we conclude by reflecting on the broader implications of our findings, comparing them with existing literature, and suggesting directions for future enhancements in modeling and evaluation strategies.

By the end of this report, we hope to provide not just a snapshot of model performance on a single dataset, but also insights into method choice, data selection, and essentials for developing reliable and future-proof fake news detection systems.

Part A – Data Description, Acquisition, and Preprocessing

Context and Source

Identifying and classifying misinformation has become an increasingly critical task in today's media landscape, as unreliable information can spread rapidly and influence public opinion, decision-making and policy. To address this chal-

lenge, we obtained our dataset from the Kaggle platform, a widely recognized repository offering a range of curated open-source machine learning datasets. The Kaggle dataset [2] provides a diverse collection of news articles, ensuring a wide variety of sources, and writing styles, making it a suitable resource for developing and testing our text classification models.

Motivation and Relevance

The motivation for selecting this dataset lies in the growing prevalence of misinformation and the potential impact of automated, data-driven solutions in combating it. By applying machine learning techniques, we can aspire to improve the reliability and trustworthiness of online information ecosystems. As Text classification models improve, it may assist fact-checkers, content moderators, and consumers of digital media in identifying suspicious sources quickly and at scale. Advances in this field can ultimately contribute to more informed societies and the safeguarding the credibility of news sources.

Data Format and Variables

The datasets provided, `true.csv` and `fake.csv`, share the same structure and consist of the following columns:

- **title**: A short, headline-style summary of the article.
- **text**: The main body content of the article.
- **subject**: A categorical variable indicating the topic of the article, such as `politicsNews` in the true dataset or `News` in the fake dataset.
- **date**: The publication date of the article.

Overview of the Datasets

Both datasets are structured as tabular CSV files, with each row representing a unique article in the `text` column. For this classification task we will not include the columns: `title`, `subject` & `date` in our training. This dataset is suitable for tasks such as text classification or topic modeling. A brief overview of each dataset is provided below:

- **True Dataset**: Contains 20826 articles identified as credible, with a primary focus on political news. It uses `politicsNews` as the subject label and includes content dated mostly around late 2017.
- **Fake Dataset**: Includes 22851 articles flagged as non-credible or fake. The `subject` column here is labeled generically as `News`. It also features publication dates around late 2017.
- **Article sizes**: Each article is roughly 300 – 900 words

Usage for Machine Learning

These datasets can be used in a machine learning project to train and evaluate models for detecting fake news, focusing on features such as textual content or headline patterns. Preprocessing steps such as cleaning text data, encoding text data into numerical vectors, should be applied as needed before model development.

Initial Data Quality Inspection

In this study, we focus exclusively on the `text` column from the datasets for training and evaluation. As part of the initial inspection, we assessed the quality of the `text` column to ensure it is suitable for text classification tasks. Below is a summary of the findings:

Missing Values

To identify any missing entries, we examined the `text` column in both datasets. Using the inspection code, the following results were obtained:

- **Fake Dataset**: The `text` column contains no missing values (0 missing entries).
- **True Dataset**: Similarly, the `text` column contains no missing values (0 missing entries).

These results confirm that the `text` column is complete in both datasets, requiring no additional handling for missing data.

Observed Limitations and Biases

- **Varying Text Lengths**: Articles in the `text` column vary significantly in length, from short news snippets to longer reports. This variability could influence model performance.
- **Domain-Specific Language**: The content of the `text` column reflects news topics, with true articles focusing on politics and fake articles labeled more broadly as news. This introduces a domain bias that may limit generalizability to other areas outside of news.
- **Potential Stylistic Differences**: The writing style or tone of "true" and "fake" articles may differ systematically, introducing potential biases. These differences could in principle influence the model's ability to distinguish between the two classes based on stylistic cues.

This inspection establishes the quality of the `text` column, confirming its readiness for further analysis and modeling without requiring additional handling of missing or incomplete data. The next steps will address text representation and vectorization.

Data Preprocessing Steps

To prepare the text data for machine learning tasks, a series of preprocessing steps were applied to ensure data quality and consistency. Below, we describe the main text cleaning procedures:

Filtering

To be fully sure that our data had no missing entries and to ensure generalizability for other datasets, we used filtering to such that only valid text data was included:

- Rows with `NaN` values in the `text` column were removed.
- Rows with empty text fields or entries with zero length were excluded.

This filtering step ensured that all retained entries contained meaningful textual information.

Lowercasing

All text entries were converted to lowercase:

- **Why**: Lowercasing standardizes the text, reducing variability caused by case sensitivity (e.g., "The" and "the" are treated the same by the vectorization process).

Example:

```
Original: "The Politician had No relevent Backgorund."
Processed: "the politician had no relevant background."
```

Removing Punctuation

Punctuation marks and special characters were removed from the text using regular expressions:

- **Why**: Punctuation generally does not contribute to the semantic meaning for many machine learning tasks and can introduce unwanted noise.

Example:

```
Original: "Hello, world! Welcome to NLP."
Processed: "hello world welcome to nlp"
```

Stopword Removal and Stemming

The stopword removal process was not included before generating the final dataset. It was handled by the vectorization framework [6] we used, but we talk about it here to retain the thread-like description of the data preprocessing steps.

- **Stopword Removal**: Removing common words (e.g., "the," "and," "is") that may not contribute significantly to the task. This step was omitted to retain all words for modeling.
- **Stemming**: These techniques reduce words to their root forms (e.g., "running" → "run"). These were not applied to preserve original word forms.

Final Dataset

After text cleaning, the datasets from both `fake.csv` and `true.csv` were processed, labeled, and combined:

- **Labeling**: Fake data was labeled as 0, while true data was labeled as 1.
- **Shuffling**: After simply combining the dataset, we shuffled it to ensure a randomized order, beneficial for training machine learning models.

The final cleaned dataset was saved to `df_final.csv` for subsequent vectorization.

TF-IDF Vectorization

To convert our collection of text documents into a numerical format suitable for machine learning models, we applied the Term Frequency–Inverse Document Frequency (TF-IDF)[6] vectorization technique. The idea behind TF-IDF is to measure how important a word is to a particular document compared to its importance across the whole dataset.

First, we compute the Term Frequency (TF), which captures how often a word appears in a document:

$$\mathrm{TF}(t, d) = \frac{\mathrm{count}(t, d)}{\sum_{t' \in d} \mathrm{count}(t', d)},$$

where $\mathrm{count}(t, d)$ is the number of times term t appears in document d.

Next, we compute the Inverse Document Frequency (IDF), which measures how rare a term is across the entire set of documents D:

$$\mathrm{IDF}(t, D) = \log\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right).$$

A term that appears in many documents will have a lower IDF, reducing its overall importance.

Finally, the TF-IDF score for a term t in a document d is:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t, D).$$

This process transforms each document into a vector of TF-IDF values, one per term in the vocabulary. Common terms that appear in almost all documents receive low weights, while terms that are more specific to certain documents get higher weights. In practice, we used the `TfidfVectorizer` from `scikit-learn` [6] in Python to automatically compute these values and produce the numerical matrix that our machine learning models used as input.

```
self.vectorizer = TfidfVectorizer(stop_words='english',
    max_df=0.7)
```

`max-df` is simply a parameter that specifies the maximum document frequency a term can have to be included in the vocabulary.

Resulting TF-IDF Data Matrix

After applying the TF-IDF vectorization process, each document in our final data was represented as a vector in a high-dimensional space, where each dimension corresponded to a unique term from the vocabulary. Once the TF-IDF values were computed for each term-document pair, we constructed a numerical matrix $M \in R^{|D| \times |V|}$, defined as:

$$M_{ij} = \text{TF-IDF}(t_j, d_i),$$

where d_i is the i^{th} document in D and t_j is the j^{th} term in V.

In this matrix representation, each row corresponds to a single document, and each column corresponds to a single term. The resulting matrix effectively encodes the relative importance of terms for individual documents, down-weighting common, uninformative words and highlighting more distinctive terms.

This TF-IDF data matrix served as the input feature matrix for our subsequent machine learning models. By providing a richer, more nuanced representation of the textual data, we aimed to improve the classification task of the models trained on these features.

Data Splitting

To prepare our data for model training and evaluation, we employed the `train_test_split` function from the `scikit-learn` library [5]. This function allowed us to randomly partition the original dataset into two subsets: a training set used to fit the model parameters, and a separate test set used solely for performance

assessment. Specifically, we allocated 80% of the samples for training and the remaining 20% for testing, ensuring that both subsets maintained similar class distributions by enabling the stratification option. This approach helped mitigate overfitting by ensuring that our model's generalization capability was evaluated on data not seen during the training phase, thus providing a more reliable measure of its real-world performance.

Part B – Methods and algorithms

We discuss three central algorithms employed for the classification of fake and true news: Logistic Regression, Neural Networks, and Decision Trees. Each of these algorithms provides unique strengths and is well-suited for specific types of data and tasks. Below, we detail the theoretical foundations of these algorithms and their application in the context of this project.

Logistic Regression

Logistic Regression is a foundational method for binary classification tasks. Instead of modeling a continuous output like ordinary least squares regression, Logistic Regression models the probability that a given input $\mathbf{x}$ belongs to one of two classes (e.g., fake or true news).

Model Formulation: The model predicts a probability via the logistic (sigmoid) function:

$$P(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-(\beta_0+\beta_1 x_1+\cdots+\beta_n x_n)}}.$$

Here, $\beta_0, \beta_1, \ldots, \beta_n$ are parameters learned from the data, and $\mathbf{x}$ is the vector of input features.

Training: Parameter estimation is achieved through maximum likelihood estimation. We adjust β_i's to maximize the likelihood of the observed data. In practice, this is often done using optimization routines such as gradient descent.

Advantages and Interpretability:

- **Simplicity**: The model is relatively easy to implement and train.
- **Interpretability**: Coefficients β_i directly indicate how each feature influences the log-odds of the prediction.

These qualities make Logistic Regression an excellent baseline. In the context of fake news detection, while it may not capture the full complexity of language like a neural network, it often provides surprisingly strong performance and transparency, helping us understand which features (words) drive classification decisions.

Neural Networks

Neural Networks are a class of flexible, non-linear models inspired by the workings of the human brain. They consist of interconnected layers of computational units, known as neurons, which apply linear transformations followed by non-linear activation functions. Unlike classical parametric models, neural networks can approximate a vast range of complex functions, making them well-suited for tasks involving nuanced patterns, such as language understanding and fake news detection.

Basic Structure and Layers: A feedforward neural network typically includes an input layer, one or more hidden layers, and an output layer. Each hidden layer comprises multiple neurons, each taking a weighted sum of inputs from the previous layer and then applying an activation function. Common activation functions include:

- **Sigmoid** ($\sigma(z) = \frac{1}{1+e^{-z}}$): Squashes values into the range $(0, 1)$, historically popular but prone to vanishing gradients.
- **tanh** ($\tanh(z)$): Maps real numbers to $(-1, 1)$, often preferred over sigmoid for stronger gradients, but still susceptible to vanishing gradients.
- **ReLU** ($\max(0, z)$): The Rectified Linear Unit is computationally simple and tends to mitigate vanishing gradients, thus often improving training stability and speed.

In this project, we employ a shallow feedforward neural network with a single hidden layer. The chosen complexity reflects a balance between model expressiveness and computational efficiency. Although deeper networks (i.e., those with multiple hidden layers) can capture more intricate patterns, they also require more careful hyperparameter tuning, longer training times, and potentially larger datasets to avoid overfitting.

Training via Gradient-Based Optimization: The core idea behind training a neural network is to iteratively adjust its weights to minimize a chosen loss function. For classification tasks, the categorical cross-entropy loss is commonly used to measure the discrepancy between predicted probabilities and true labels. The training process involves:

1. **Forward Propagation**: Input data is passed through the network layer by layer, producing a final prediction.
2. **Loss Computation**: The prediction is compared to the true label, and a scalar loss value is computed.
3. **Backward Propagation (Backpropagation)**: The error is propagated backward through the network. Using the chain rule, partial derivatives of the loss with respect to each weight are computed. This provides a clear direction for how to update the weights to reduce the loss.

Why Neural Networks for Fake News Detection? While simpler models (like Logistic Regression) can capture linear relationships and provide interpretability, they may fail to represent the complex, context-dependent patterns of human-written text. Neural networks, with their layered transformations and non-linear activations, can model subtle semantic and syntactic cues. This allows them to pick up on more holistic language features that might distinguish well-written, credible articles from those crafted to deceive, thereby potentially achieving higher accuracy and robustness.

Decision Trees

Decision Trees are intuitive, non-parametric models that classify instances by successively splitting the feature space. Each internal node of the tree applies a rule, dividing the data into subsets that are increasingly homogeneous in terms of the target variable.

Splitting Criteria: To determine where to split, the tree chooses features and thresholds that best reduce impurity. Two common measures are:

- **Gini Index**:
$$\text{Gini}(D) = \sum_{k=1}^{C} p_k(1 - p_k),$$
where C is the number of classes and p_k is the proportion of class k in the subset D.

- **Entropy**:
$$\text{Entropy}(D) = -\sum_{k=1}^{C} p_k \log_2(p_k).$$

Lower values of Gini or Entropy indicate more homogeneous subsets, and thus each split is chosen to yield a substantial reduction in impurity.

Advantages: Decision Trees are:

- **Interpretable**: Their hierarchical structure can be visualized and understood intuitively.
- **Versatile**: They natively handle non-linear boundaries and mixed data types.

In the context of fake news detection, a decision tree might highlight a small set of words or phrases that strongly differentiate credible articles from deceptive ones. Although such a tree may not match the accuracy of more advanced methods, its transparency helps us understand the linguistic indicators the model relies on.

Logistic Regression, Neural Networks, and Decision Trees were implemented and evaluated for the classification of fake and true news. Logistic Regression provided simplicity and efficiency, Neural Networks offered flexibility and the ability to model non-linear patterns, while Decision Trees excelled in interpretability and accuracy. The combination of these methods highlights the versatility of machine learning algorithms in addressing real-world classification problems.

Part C – Implementation and tests

The primary goal of this project was to build and evaluate machine learning models for classifying text data into binary categories, such as distinguishing between fake and credible news articles. This process involved preprocessing the raw text data, constructing and training multiple machine learning models, and rigorously evaluating their performance using various metrics.

The dataset, sourced from Kaggle [2], was first preprocessed and vectorized using scikit-learn's TF-IDF vectorizer [6]. The vectorized data served as input for three models:

- Logistic Regression (a simple and interpretable baseline model),
- Neural Network (a more complex architecture capturing non-linear relationships)
- Decision Tree (an intuitive and interpretable tree-based model).

These models were evaluated using a consistent set of metrics and techniques to ensure a fair comparison and robust validation.

Evaluation Metrics

- Accuracy: The proportion of correctly classified instances, supplemented by a confusion matrix for additional insights [5].
- Classification Report: Includes precision, recall, F1-score, and support for each class [5].
- ROC AUC: Area under the ROC curve to assess the trade-off between true positive and false positive rates [5].
- Cross-Validation: Assesses the performance of the model across different data splits. (Cross-Validation was not used the Neural Network because of implementation issues) [5].
- Bootstrapping: Tests the robustness of the model by sampling subsets of data and calculating the average performance.[5]

- Feature Importance: Highlights the most impactful characteristics driving the model's predictions (coefficients, Shapley Additive explanations(SHAP)[3] values, or impurity reduction, depending on the model).

Logistic regression

Logistic Regression, a probabilistic linear model, was chosen as the baseline due to its simplicity, efficiency, and interpretability. It models the probability that an instance belongs to a particular class using the logistic function.

Implementation Details: In our implementation, we utilized the `LogisticRegression` class from the `scikit-learn` [5] library. We first fitted the model to the training set, which had been vectorized into TF-IDF representations. By setting `max_iter=1000`, we ensured that the optimization process had sufficient iterations to converge given the complexity of our dataset.

```
def logistic_regression(self):

        # Training a Logistic Regression model

        self.lr = LogisticRegression(max_iter=1000)
        self.lr.fit(self.X_train_tfidf, self.y_train)
        self.y_pred_lr = self.lr.predict(self.X_test_tfidf)

        self.accuracy_lr = accuracy_score(self.y_test, self.y_pred_lr)
        print(f'Logistic Regression Accuracy: {self.accuracy_lr:.2f}')
```

Performance evaluation: The model's accuracy and ROC AUC scores were evaluated on the test set. Additionally, cross-validation and bootstrap resampling were performed to validate robustness and reduce the influence of specific data splits. Feature importance was analyzed using the model's coefficients, with positive coefficients indicating features contributing to the positive class (e.g., fake news) and negative coefficients contributing to the negative class (e.g., credible news).

Visualizations

- Confusion Matrix: Provided a summary of true and false classifications.
- Feature Importance: Visualized the top positive and negative features based on coefficients.
- ROC Curve: Illustrated the model's discriminative performance across thresholds

Neural Network

For the neural network, a simple feed forward neural network was created using Keras [1]. It's shallow nature is beneficial for text and binary classification problems, where the tasks have relatively simple patterns. As the code below shows, it first sets the input layer to the size of the number of columns in "X_train_tfidf". It then adds a hidden layer with 64 nodes, this layer uses the ReLu activation function for learning more complex relationships. The output layer consists of two neurons, representing the binary output of the model. The softmax activation function is used, as it outputs a probability distribution between the two cases.

Implementation Details:

```
def build_neural_network(self, activation='relu', optimizer='adam'):

        # building the neural network

        input_dim = self.X_train_tfidf.shape[1]
        self.model = Sequential([
            Dense(64, input_dim=input_dim, activation=activation),
            Dense(2, activation='softmax')
        ])
```

The model is trained in 5 epochs with a batch size of 64, this was found to be a sweet spot, as the accuracy is high even with only a few epochs, meaning there is a lower chance of the model being overfitted. The "to_categorical" function is used to convert the y_train dataset into a one hot encoded dataset, ensuring that the model interprets y_train as distinct categories and not integers. The model is then fitted, using the paramaters specified above, verbose is just a helpful addition, giving real time feedback on the training process.

```
def train_neural_network(self, epochs=5, batch_size=64):

        # training the neural network

        self.y_train_nn = to_categorical(self.y_train)
        self.model.fit(self.X_train_tfidf, self.y_train_nn,
            epochs=epochs, batch_size=batch_size, verbose=1)
```

Performance evaluation: Metrics such as accuracy, ROC AUC, and classification reports were used to evaluate the network. Bootstrap sampling was applied to validate the model's robustness. Feature importance was analyzed using SHAP [3] values, which quantify each feature's contribution to individual predictions.

Visualizations:

- Confusion Matrix: Provided a summary of true and false classifications.
- Feature Importance: Plots using SHAP [3] values highlighted the most impactful features for both positive and negative classes. As well as overall importance (Absolute values)
- ROC Curve: Illustrated the model's discriminative performance across thresholds

Decision tree

Implementation details: The descision tree was implemented using sckitit's [5] descision tree classifier. It was implemented with a max depth of 8 to reduce overfitting. The descision tree was optimized by analyzing the descision tree architecture, as well as the feature importances.

```
def train_decision_tree(self):

      # Training the decision tree

       self.dt = DecisionTreeClassifier(max_depth=8)
       self.dt.fit(self.X_train_tfidf, self.y_train)
       self.y_pred_dt = self.dt.predict(self.X_test_tfidf)

       self._print_metrics(self.y_test, self.y_pred_dt,
           model_name="Decision Tree")
```

Performance evaluation: To optimize the descision tree model, feature importance [5] and tree depth were used, as these are important aspects of a descision tree and give insights on how well the model is classifying. When running the descision tree without any restrictions, it performs with good accuracy scores. However taking a look at the tree architecture, it is evident that the model is overfitting on a single path and does not generalize well. Adding a maximum depth to the model helps it to strengthen other paths, which in turn generalizes the model.

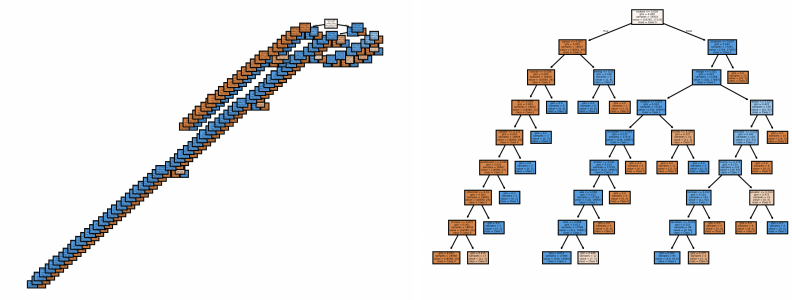


Figure 1: Descision tree models. Left figure: No depth restrictions, Right figure: max depth is 8. Accuracy both: 1.00

Again, we used the same metrics for evaluating performance, so evaluating accuracy, classification reports (precision, recall, F1-score), and ROC AUC to measure performance and class discrimination. Cross-validation ensured robustness by averaging performance across data splits. Additionally, feature importance highlighted the most impactful features based on impurity reduction, providing insights into the tree's decision-making process.

Visualizations:

- Tree Structure: Visualized the learned splits and thresholds, offering insights into the decision-making process.
- Confusion Matrix: Provided a summary of true and false classifications.
- ROC Curve: Illustrated the model's discriminative performance across thresholds Feature Importance: Highlighted the top features based on their contribution to reducing impurity.

Part D – Results and Findings

Introduction: In this section, we present the outcomes of our classification experiments on detecting fake versus credible news articles using three distinct models: Logistic Regression, a Neural Network classifier, and a Decision Tree. By comparing their performances and examining the most influential features, we gain insight into how textual patterns inform classification decisions. We also relate our findings to the broader literature, which reports similar success using both traditional and deep learning-based methods for fake news detection [8]. Notably, we investigate how omitting or down-weighting highly discriminative features affects model behavior, providing further context on the robustness and domain-dependence of these approaches.

Impressive initial results: After training a Logistic Regression, Neural Network and a Decision Tree on the unchanged final dataset from the TF-IDF vectorizer, we achieved quite impressive results. We achieved a score of `0.99 accuracy` for all models. This made us suspicious of how easy it was for the models to learn the underlying patterns that differentiate true and fake articles. To learn more we decided to measure the importance of individual words using metrics from the SHAP (SHapley Additive exPlanations) values, which quantify the contribution of each feature to a given prediction [3].

Removal of High-Impact Features: After establishing the strong baseline performances of our models, we conducted the feature removal experiment. Our aim was to uncover whether the classification decisions were robust to the removal of high-impact terms.

In Figure 10, we present the top 10 most important features ranked by their absolute SHAP values. These features, drawn from the neural network model's output, highlight a handful of terms that exert high influence on the final classification outcome.

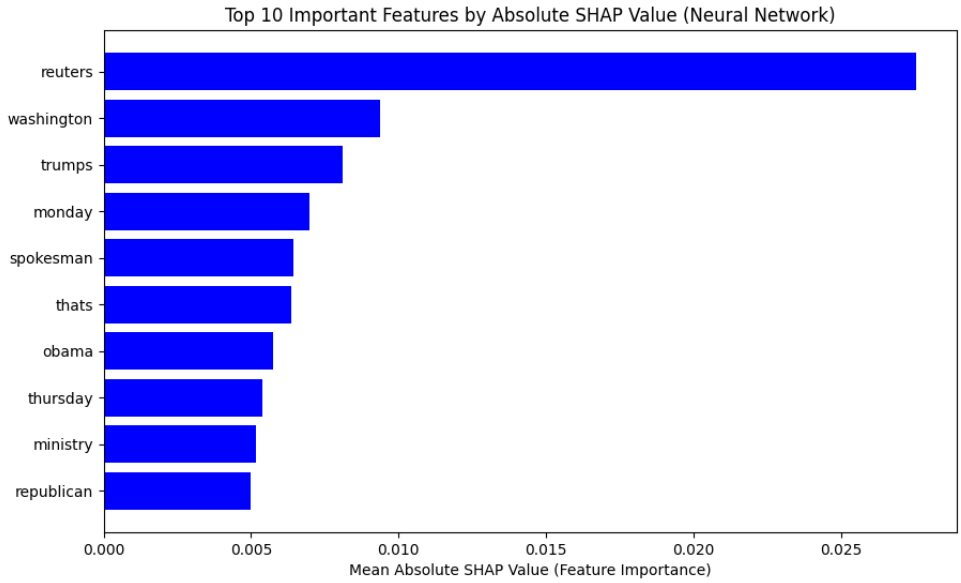


Figure 2: Top 10 Features by Absolute SHAP Value. This visualization highlights the key lexical markers that contribute most significantly to the classification decisions. Each feature's bar represents its average impact on prediction, and longer bars correspond to greater influence.

As we can see, the terms `reuters` emerged as consistently high-impact tokens across multiple bootstrap samples, implying that the model had learned to associate these words strongly with one class (either fake or true). Below is another plot that describes the top positive and negative features that drove the `logistic regression` model the most to classify as true or fake.

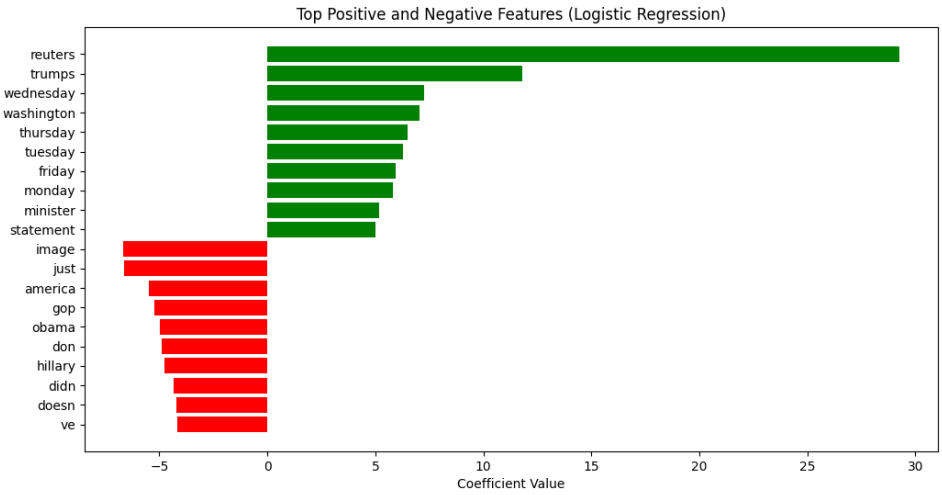
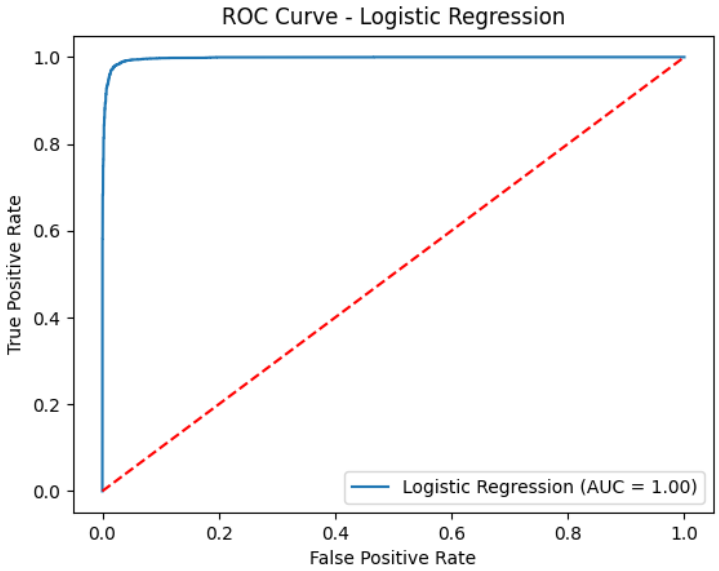


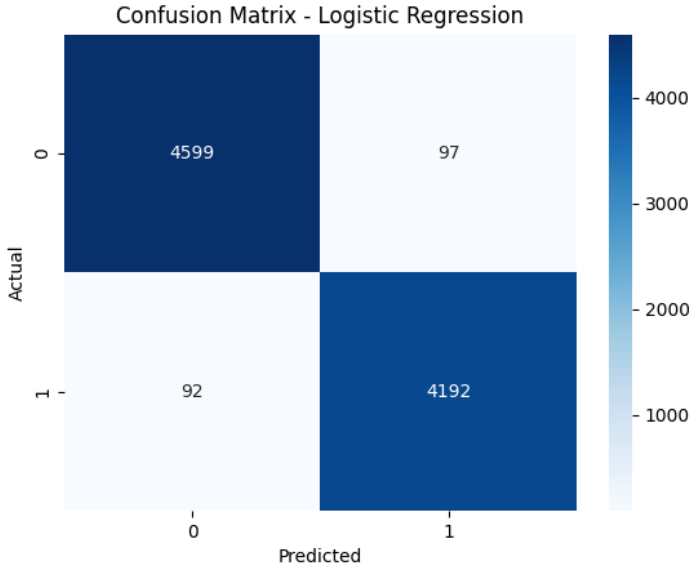
Figure 3: Top positive and negative features for Logistic regression. The values are the coefficients value in the linear combination for the logistic regression equation. The red are the values that drive the prediction to False. The green drives the prediction to True.

Motivated by these findings, we decided to remove the words `reuters`, `trump`, `image` and `image`. These words had the highest prediction impact and by depriving the training process of these major textual "cues", we forced the classifiers to rely on more subtle or context-dependent patterns. The underlying question was whether the models, particularly the neural network and logistic regression, could still achieve satisfactory accuracy and maintain robust classification capabilities. After re-training all our models with the new final dataset we achieved the following results.

Logistic Regression Results: Our Logistic Regression model exhibited robust performance, achieving an accuracy of approximately 0.98 without resampling and confirming its consistency with an average accuracy of 0.98 over 20 bootstrap samples. A cross-validation score of about 0.96 further supports the model's stability. The ROC AUC of 0.98 and near-perfect precision, recall, and F1-scores indicate that the model adeptly discerns between the two classes.



(a) The ROC curve for Logistic Regression demonstrates the trade-off between the true positive rate and the false positive rate at different thresholds. The area under the curve provides a summary metric for model performance, where higher AUC values indicate better classification ability.



(b) The confusion matrix shows the counts of true positives, true negatives, false positives, and false negatives, offering insights into the classification model's performance and error distribution.

Accuracy (without bootstrapping): 0.97
Average Accuracy over 20 Bootstraps: 0.98
Cross Validation Score: [0.96, 0.96, 0.95, 0.96, 0.96]
Cross Validation Score (mean): 0.96
ROC AUC: 0.98

	precision	recall	f1-score	support
0	0.98	0.98	0.98	4696
1	0.98	0.98	0.98	4284
accuracy			0.98	8980
macro avg	0.98	0.98	0.98	8980
weighted avg	0.98	0.98	0.98	8980

A closer examination of the top positively and negatively weighted features reveals how certain lexical cues drive the classification. Terms like `trumprussia` and `washhere` strongly predict credible news, while words like `jussi` and `america` lean towards the fake category. (Even though we removed `trumps` from the dataset, we still ended up with a high-impact word including the same word we removed. This is a result from the fact that `trumprussia` initially was `trump-russia`, but was combined in the data preprocessing steps.)

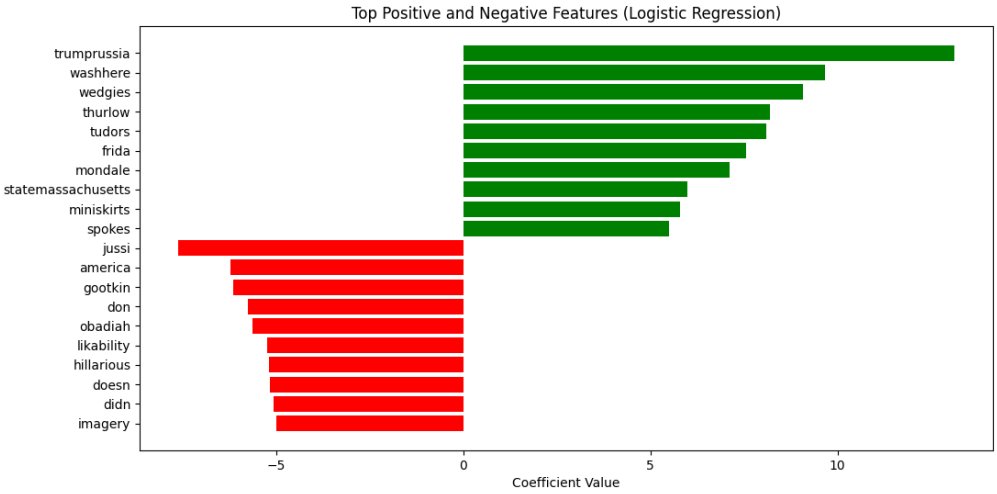
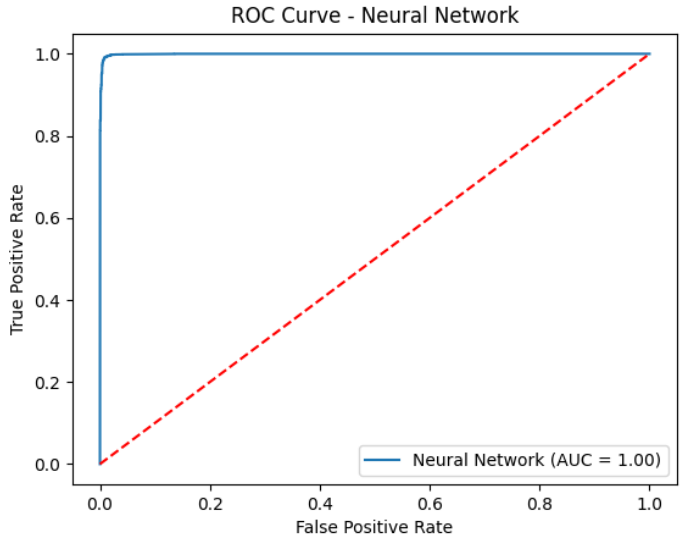


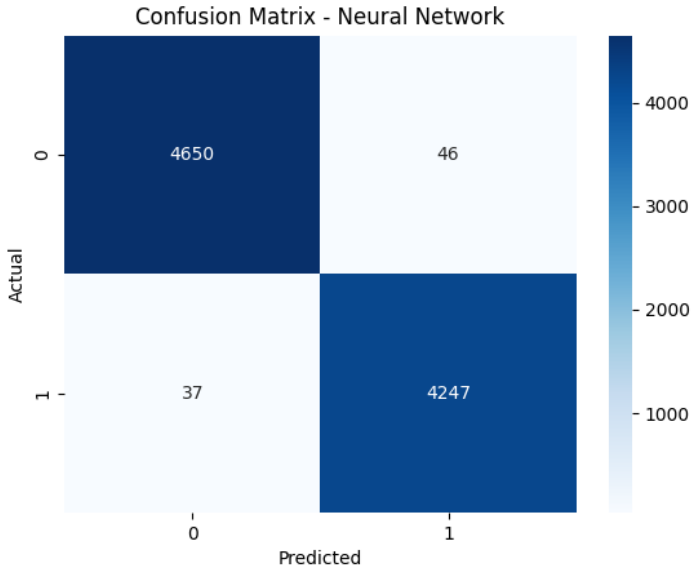
Figure 5: These are the non-absolute SHAP [3] values. Top positive and negative features for Logistic regression. The red are the values that drive the prediction to False. The green drives the prediction to True.

Such reliance on specific tokens echoes concerns in existing literature that high performance in controlled datasets may, in part, stem from dataset-specific artifacts [11]. By potentially removing or down-weighting additional high-impact terms, we can guide the generalizability of the model, highlighting whether strong performance is still contingent on a few dominant features or whether the classifier captures more intrinsic patterns of true articles.

Neural Network Results: The Neural Network model we used to generate these results have the parameters described in the implementation part. This model demonstrated even stronger discrimination than the logistic regression, with an accuracy of approximately 0.99 and a similarly high ROC AUC of 0.99. These results are consistent with contemporary research that emphasizes the power of representation learning in neural architectures [9]. The network's near-perfect performance across bootstrapped samples and its comprehensive classification report reflect its capacity to learn nuanced patterns in the textual data.



(a) The ROC curve for Neural Network demonstrates the trade-off between the true positive rate and the false positive rate at different thresholds. The neural net consist of one hidden-layer with 64 nodes.



(b) The confusion matrix shows the counts of true positives, true negatives, false positives, and false negatives, offering insights into the classification model's performance and error distribution.

Accuracy (without bootstrapping): 0.990
Average Accuracy over 20 Bootstraps: 0.99
ROC AUC: 0.99

	precision	recall	f1-score	support
0	0.99	0.99	0.99	4696
1	0.99	0.99	0.99	4284
accuracy			0.99	8980
macro avg	0.99	0.99	0.99	8980
weighted avg	0.99	0.99	0.99	8980

Despite its impressive metrics, the examination of feature importance underlines a subtlety: the Neural Network's top contributing features often show near-zero or extremely small absolute contributions when isolated. This may indicate that the network's predictive ability emerges from more intricate, distributed representations rather than reliance on a few key terms.

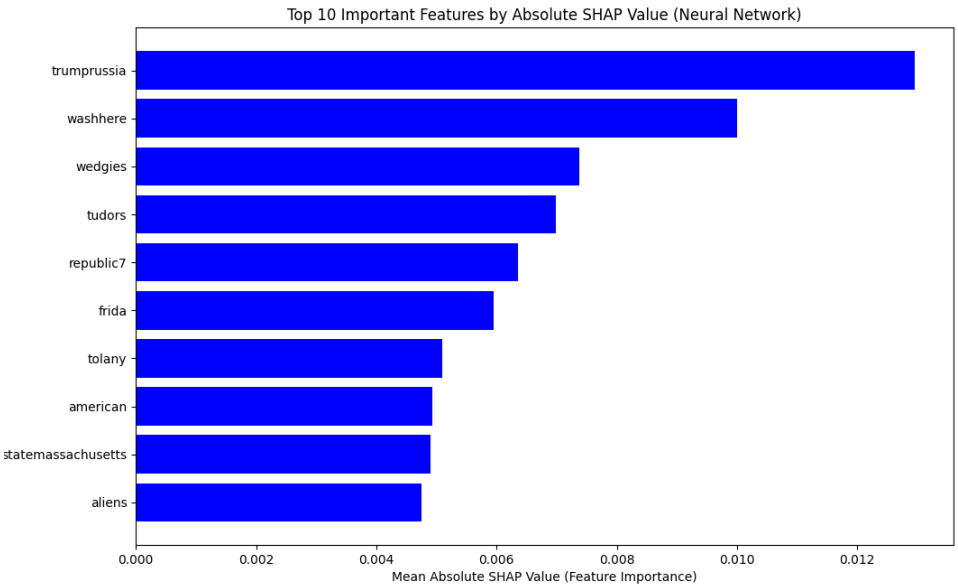


Figure 7: Absolute SHAP values for the top 10 most important features of our trained Neural Network. Note: We get very low values indicating low classification dependability of textual cues.

Such a finding mirrors insights from recent deep learning studies, where robust performance commonly arises from intricate patterns rather than individual cues [9]. Thus, while the Neural Network does not change substantially when omitting certain words, it suggests that a more holistic text representation could be the reason, potentially enhancing model resilience to adversarial modifications to articles.

Decision Tree Results: In contrast to the near-perfect performance of the linear and neural models, the Decision Tree achieved an accuracy of about 0.85, a ROC AUC of 0.85, and a cross-validation score around 0.84. Although these results are still strong, they highlight the tree's relatively lower capacity to capture complex decision boundaries in text. Nevertheless, such performance remains competitive and interpretable.

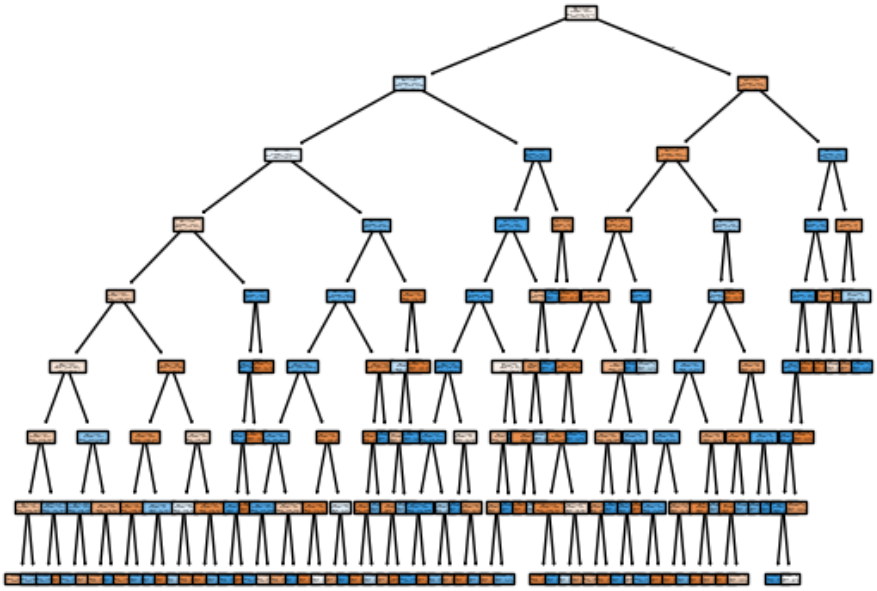
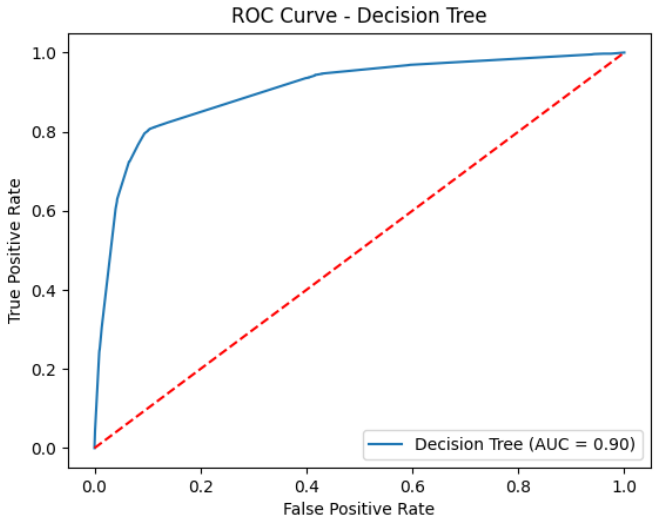
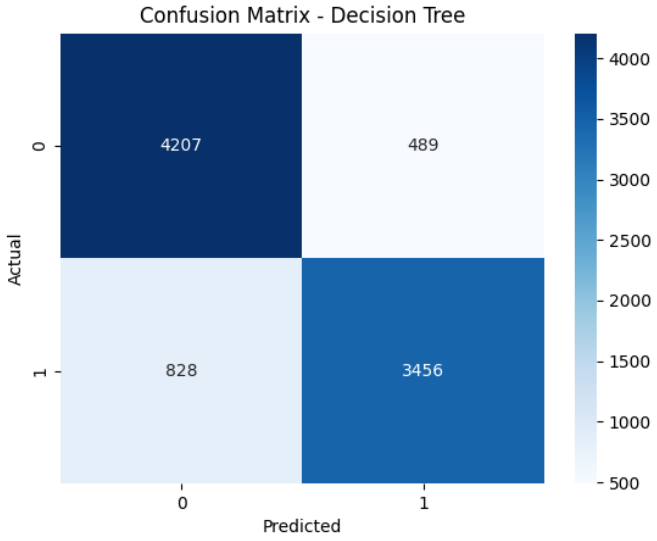


Figure 8: Decision Tree Visualization. Each node represents a feature split, with colors indicating the majority class (orange for fake news and blue for true news). The hierarchical structure reflects the decision-making process, where deeper splits refine classification based on additional features.



(a) The ROC curve for Decision Tree demonstrates the trade-off between the true positive rate and the false positive rate at different thresholds. As we can tell, the area under the curve is smaller compared to LR and NN models indicating a lower accurate classification ability.



(b) The confusion matrix shows the counts of true positives, true negatives, false positives, and false negatives, offering insights into the classification model's performance and error distribution.

Accuracy (without bootstrapping): 0.85
Average Accuracy over 20 Bootstraps: 0.85
Cross Validation Score: 0.84
ROC AUC: 0.85

	precision	recall	f1-score	support
0	0.84	0.90	0.86	4696
1	0.88	0.81	0.84	4284
accuracy			0.85	8980
macro avg	0.86	0.85	0.85	8980
weighted avg	0.85	0.85	0.85	8980

The top ten important features for the Decision Tree (e.g., `jussi`, `washhere`, `miniskirts`, and `trumprussia`) confirm that this model, too, relies heavily on certain discriminative words. These results are consistent with the literature, where decision trees often serve as interpretable baselines but may require ensemble methods to achieve performance levels comparable to neural models [7].

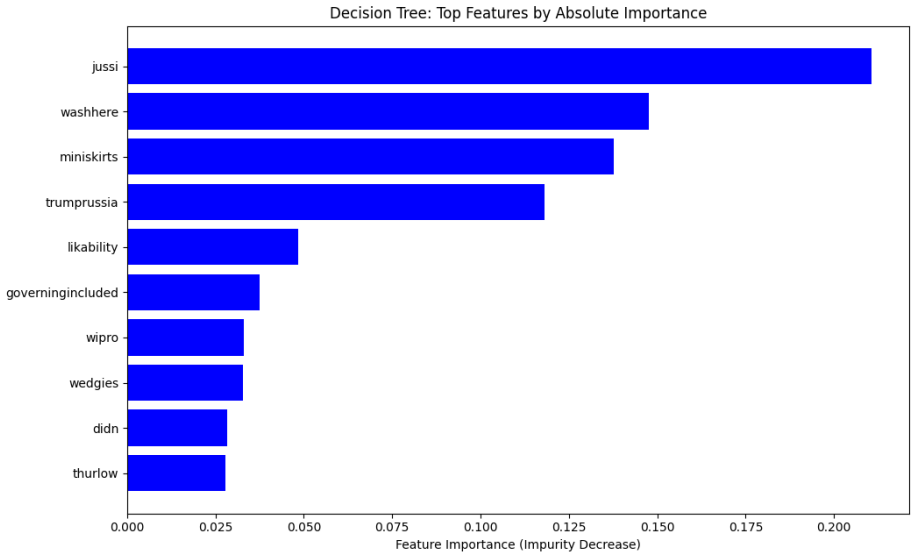


Figure 10: Absolute SHAP values for the top 10 most important features of our trained Neural Network. Note: Higher values indicating higher classification dependability of textual cues.

Unlike the Neural Network, the Decision Tree's performance appears more sensitive to the removal of dominant features. Such sensitivity suggests a weaker ability to generalize beyond the top contributing terms, emphasizing the role that highly weighted features play in guiding classification.

In summary, while our models perform well on the given dataset, achieving outstanding accuracy, precision, and recall, these findings must be interpreted with caution. The top-performing models, particularly the Neural Network, seem more resilient to feature omissions, indicating richer learned representations. Yet, further research is necessary to confirm these observations in more varied and dynamic contexts, ensuring that tomorrow's misinformation detection tools remain effective in the ever-evolving information landscape.

Part E – Critical assessment of methods and links with existing literature

The high accuracy and robustness scores obtained by our Logistic Regression, Neural Network, and Decision Tree models comes with a careful and critical examination. At first glance, achieving such near-perfect classification performance on a fake news detection task might suggest that these methods are highly adept at identifying subtle cues separating genuine articles from deceptive ones. However, these results should be interpreted with caution. A closer look at the models feature usage and the experimental setup reveals several important considerations that align with current discussions in the literature.

Overreliance on Dataset-Specific Cues: Our results show that even after removing top-impact tokens such as `"reuters"`, `"trumps"`, and `"image"`, the neural network and the logistic regression models remained extremely accurate. This suggests that the dataset may still contain multiple, easily exploitable lexical cues that strongly correlate with one of the classes. Existing research frequently highlights the risk of high reported accuracy stemming from dataset artifacts rather than semantic understanding [8]. In other words, our classifiers may have learned to rely on publisher-specific language patterns, article formatting, or stylistic markers rather than developing a robust grasp of the factuality or credibility of the underlying content even though our neural network may have hinted to it.

Model Complexity versus Interpretability: The Logistic Regression model, despite its simplicity, achieved accuracy levels that rivaled more complex methods. While linear models are known for their proficiency in text classification tasks, the near-perfect performance here raises the question of whether the problem presented by the chosen dataset is too simple. If a linear model can nearly solve the task, it suggests the presence of strong linear separability—perhaps

due to dataset biases or topic-specific vocabulary. The Decision Tree, though inherently interpretable, demonstrated lower performance after feature removal, implying that it was more sensitive to these highly discriminative terms. This aligns with literature warning against overfitting to a handful of features and encourages more robust evaluation frameworks [10].

Neural Networks and Distributed Representations: The Neural Network's resilience to feature removal and its ability to maintain extremely high accuracy is consistent with the literature, where deep learning models often excel at identifying complex, non-linear patterns. However, the fact that its top contributing features, as measured by SHAP [3] values, did not dramatically shift performance when removed suggests that the network might be capturing more distributed representations of fake news. While this indicates a form of robustness, it does not guarantee that the learned patterns will extend beyond the training. Without testing on more diverse datasets, it remains unclear if the network's success generalizes well enough to tackle real-world scenarios.

Dataset Bias and Domain Generalization: High performance on a single dataset does not guarantee generalization across different news sources, languages, or time periods. Prior studies emphasize the importance of cross-domain evaluation and temporal robustness, as model performance can degrade sharply when facing new types of misinformation or shifts in language styles [7]. The consistent use of terms and topics in our dataset may have created ideal conditions for classification—a scenario that may not reflect the broader information ecosystem. Thus, while the methods studied appear powerful, their true effectiveness remains uncertain.

Future Directions and Recommendations: To better align with the guidance from existing literature and to ensure that our findings have practical relevance, several steps should be taken:

- **Cross-Domain and Longitudinal Testing:** Evaluating the models on datasets from different sources, time periods, and domains would provide a clearer picture of their adaptability and real-world performance.
- **Adversarial Evaluation:** Testing models against intentionally manipulated texts, such as machine-generated misinformation, would reveal their resilience and highlight weaknesses that could be exploited.
- **Incorporating Contextual and Multimodal Cues:** Beyond text-only approaches, integrating metadata, social network signals, or image and video content could foster more robust and contextually aware models, reflecting insights from multimodal research [4].
- **Leveraging Advanced Architectures:** Exploring transformer-based language models and few-shot learning approaches could improve gen-

eralization capabilities, as evidenced by recent breakthroughs in natural language understanding tasks.

Conclusions

In this report, we investigated the task of fake news classification using three distinct models—Logistic Regression, a feedforward Neural Network, and a Decision Tree—trained and evaluated on a curated Kaggle dataset. Our best-performing models consistently achieved accuracies above 98%, with the Neural Network and Logistic Regression models frequently surpassing the 99% threshold. The ROC AUC scores were similarly high (commonly above 0.98), signaling strong discriminative power. These results demonstrate that even relatively simple architectures can identify textual patterns predictive of veracity, suggesting that certain linguistic or stylistic cues within the data are highly correlated with fake or credible news.

Beyond metrics, we learned that model interpretability is crucial for understanding why these classifiers perform so well. Logistic Regression offered immediate insights through its linear coefficients, revealing specific words with strong positive or negative influence on predictions. The Decision Tree's feature importance indicated that a handful of high-impact terms drove much of the model's reasoning. Although the Neural Network's decisions are more complex, SHAP values helped highlight important tokens, suggesting the network's reliance on a richer, more distributed set of textual cues.

Despite these impressive results, the report underscores the importance of critically assessing model performance. Our experiments showed that removing top predictive features only slightly reduced accuracy for some models, implying that strong performance may be partially due to dataset-specific artifacts rather than a robust, domain-independent understanding of credibility. Future work should address this by testing models on diverse datasets, exploring adversarial robustness, and incorporating more sophisticated architectures (e.g., transformer-based language models) that can better generalize to unseen domains.

In summary, while we achieved near-perfect classification on this dataset, the main takeaway is that evaluating a model's adaptability, interpretability, and resilience to feature manipulations is as important as achieving high numerical scores. By adopting these assessment criteria and considering more challenging data sources, we can guide the development of more trustworthy and generalizable fake news detection systems.

References

[1] Francois Chollet et al. Keras, 2015.

[2] Bhavik Jikadara. Fake news detection dataset, 2023.

[3] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, pages 4765–4774, 2017.

[4] Federico Monti, Fabrizio Frasca, David Eynard, Daniele Mannion, and Michael M. Bronstein. Fake news detection on social media using geometric deep learning. *arXiv preprint arXiv:1902.06673*, 2019.

[5] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Vincent Dubourg, Jake Vanderplas, Ana Passos, Rudolf Weiss, and Tom Keepers. Scikit-learn: Machine learning in python, 2011.

[6] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Vincent Dubourg, Jake Vanderplas, Ana Passos, Rudolf Weiss, and Tom Keepers. Tfidfvectorizer by scikit learn, 2011.

[7] Kai Shu, Deepak Mahudeswaran, Suhang Wang, Dongwon Lee, and Huan Liu. Fakenewsnet: A data repository with news content, social context and dynamic information for studying fake news on social media. In *2018 IEEE/ACM International Conference on Big Data (BigData)*, pages 2886–2895. IEEE, 2019.

[8] William Yang Wang. "liar, liar pants on fire": A new benchmark dataset for fake news detection. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 422–426, 2017.

[9] Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Defending against neural fake news. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 9054–9065, 2019.

[10] Xiaoyan Zhou and Reza Zafarani. Fake news: A survey of research, detection methods, and opportunities. *arXiv preprint arXiv:1812.00315*, 2018.

[11] Chenghui Zou, Yezhou Dong, Chen Liu, Yaliang Li, Jia Xu, and Guanfeng Chen. Fake news detection via nlp is vulnerable to adversarial attacks. *arXiv preprint arXiv:1805.08773*, 2018.