FilipinoFactCheck: An Analysis of Machine Learning Classifiers for Filipino Fake News Detection

Alister Marc Domilies
University of the Philippines - Diliman
abdomilies@up.edu.ph

Jemar E. Laag
University of the Philippines - Diliman
jelaag@up@edu.ph

ABSTRACT

In today's era, misinformation has become widespread despite the scarcity of manual fact-checking resources. While numerous studies focus on creating fake news detection models, most are tailored exclusively for the English language. This study endeavors to fill this void by developing a fake news detection model explicitly designed for the Filipino language. Taking off from the complex transfer learning models utilized by Cruz et al. [3], which posed deployment challenges, this research adopts an alternative approach. By utilizing dataset used in [3], the study employs the TF-IDF technique to extract essential features for the classifiers used in this study. This study uses readily available classifiers from the sci-kit learn library, encompassing individual, ensemble, and neural network models. Initially, the study utilized default hyperparameters and subsequently engaging in hyperparameter tuning to identify hyperparameters that would yield optimal models for each classifier. Notably, hyperparameter tuning significantly enhances the performance of individual learners while leaving ensemble and neural network classifiers relatively unaffected. Ultimately, the stochastic gradient descent classifier emerged as the overall best-performing model. Not only does it exhibit the highest accuracy, but it also boasts shorter training times compared to other classifiers. Furthermore, it has an accuracy that is almost as accurate as the prior study which made use of relatively way more complex models. The researchers successfully deploy this model, leveraging its simplicity for ease of implementation, marking a significant step towards bridging the gap between research and practical application in combating fake news.

Categories and Subject Descriptors

I.2 [**ARTIFICIAL INTELLIGENCE**]

Keywords

Machine Learning, Tagalog Fake News Detection, Misinformation

1. INTRODUCTION

Based on Pulse Asia's findings [7], nine out of 10 individuals acknowledge that fake news is an ongoing issue in the Philippines. This false information, commonly termed as fake news, predominantly originates from the internet, social media platforms, and television. Moreover, according to the results of a Social Weather Stations (SWS) survey held in December 2021, 51% of Filipinos find it difficult to spot fake news [21].

This surge in misinformation poses a significant challenge in the country, particularly aggravated by the widespread use of social media, where verifying content authenticity remains notably difficult. As time progresses, the spread of fake news across various online platforms is anticipated to increase. Despite the presence of independent media fact-checking organizations like Verafiles and the National Union of Journalists in the Philippines, their effectiveness is limited due to the massive volume of content. The process of manual detection becomes challenging because of the overwhelming abundance of articles.

This paper aims to develop a model capable of determining the authenticity of articles in Filipino language combined with English vernaculars, by employing a supervised machine learning algorithm. The model will rely on words present within the article, leveraging a dataset classified by fact-checking organizations and mainstream news websites. Additionally, this study proposes a simple algorithm that can be readily deployed for easy human input, facilitating the determination of an article's credibility as real or fake.

2. RELATED WORKS

Numerous studies have developed various models to differentiate between fake and real news with many of these models focus primarily on languages like English. For instance, Rajeswari et al. [8] created a Naïve Bayes Algorithm-based model achieving an 81% accuracy in fake news detection. Sharma et al. [20] employed various feature generation techniques including Bag-of-Words, N-grams, and TF-IDF which were fed into three classifiers. After grid search parameter optimization for Logistic Regression, accuracy increased from 65% to 80%.

Khanam et al. [6] utilized the LIAR dataset for fake news detection, applying multiple algorithms such as XGBoost, Random Forest, Naïve Bayes, K-Nearest Neighbors, Decision Tree, and SVM. XGBoost led with over 75% accuracy,

followed by SVM and Random Forest at approximately 73%. Kaur et al. [5] developed a hybrid text classification method combining KNN and Random Forest, resulting in a 94% f1-score, 8% higher than SVM.

Deep learning methods were explored for detecting Filipino fake news, notably in Cruz et al.'s study [3]. They introduced the "Fake News Filipino" benchmark dataset, a first-of-its-kind curated by the researchers. The study utilized transfer learning techniques, including ULMFiT, BERT, and GPT-2, achieving accuracies ranging from 87% to 91%. Additionally, augmenting transformer-based transfer techniques with auxiliary language modeling losses further improved model accuracy by 4% to 6%. However, these highly accurate models present complexities that hinder their easy deployment for consumer-level applications, limiting the swift verification of fake and real news articles by a broader audience.

Furthermore, Ahmad et al. [1] conducted a comprehensive study exploring multiple algorithms encompassing shallow learners, ensemble learners, and deep learning techniques. Among these, the bagging classifier emerged as the top-performing algorithm with an accuracy of 94%. Conversely, benchmark algorithms like Wang-CNN and Wang-Bi-LSTM performed poorer compared to other algorithms tested in the study. Notably, traditional machine learning algorithms occasionally outperformed deep learning approaches, as seen in the study's comparison of performance among various techniques.

3. PRELIMINARIES

3.1 Term Frequency – Inverse Document Frequency

Term Frequency – Inverse Document Frequency (TF-IDF) is a technique used for feature extraction. It quantifies the importance of term in a document relative to a document corpus into numerical features suitable for machine learning. This method involves two core components: Term Frequency (TF) and Inverse Document Frequency (IDF).

TF measures a term's significance within an individual document, calculated as the frequency of the term 't' in document d. The Term Frequency (tf) is given by

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in D} f_{t',d}} \tag{1}$$

IDF emphasizes rare terms occurring in fewer documents across the corpus which is computed through the inverse of the frequency of documents containing the term t. The Term Frequency (tf) is given by

$$itf(t,D) = \log\left(\frac{|D|+1}{|\{d \in D : t \in d\}|+1}\right) + 1 \tag{2}$$

Higher TF values denote greater relevance of a term within a specific document, while higher IDF scores indicate that the word occurs less frequently. Overall, TF-IDF assigns higher weights to terms that are frequent in a specific document (TF) but less common across the entire document collection (IDF). A library available in scikit-learn can be easily accessed to automatically compute the TF-IDF scores of tokens.

3.2 Individual and Ensemble Learners vs. Deep Learning Models

Individual and ensemble learners, often termed as traditional machine learning methods, significantly differ from deep learning methods in various aspects within Natural Language Processing (NLP). Common individual learners utilized in NLP include algorithms like Naïve Bayes and Support Vector Machines (SVM), while ensemble learners encompass techniques like Random Forests. Conversely, in the domain of deep learning for NLP, prevalent methods include Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformer-based models such as BERT and GPT.

The architectural design of traditional machine learners is notably simpler in comparison to deep learning models, which predominantly comprise layered neural networks. Traditional machine learners typically necessitate manual feature engineering to identify and retain relevant features, whereas deep learning models excel in automatic feature extraction directly from raw data. However, the complexity inherent in deep learning techniques demands substantial computational resources and extensive datasets for effective training, while individual and ensemble learner models generally require fewer computational resources and offer faster training times in contrast to deep learning models.

In this study, a variety of classifiers spanning individual and ensemble learners, as well as neural network models have been employed. These classifiers encompass commonly used machine learning algorithms for classifications including the Stochastic Gradient Descent (SGD) Classifier, Multinomial Naive Bayes (MultinomialNB), Logistic Regression, K-Nearest Neighbors (KNeighborsClassifier), Support Vector Classification (SVC), Decision Tree Classifier, Random Forest Classifier, Gradient Boosting Classifier, AdaBoost Classifier, and the Multilayer Perceptron Classifier (MLP-Classifier). Each classifier possesses distinct strengths and characteristics, contributing to the diversity of the experimental approach in this study. The following description presents brief details of these classifiers used in the study.

1. *MultinomialNB* (Multinomial Naive Bayes): The MultinomialNB class in scikit-learn implements the Multinomial Naive Bayes algorithm that is commonly used for text classification tasks. This classifier is based on the Naive Bayes assumption of feature independence and is particularly effective for datasets with discrete features, such as word counts. Its simplicity and effectiveness make it a popular choice in natural language processing application. [16]
2. *LogisticRegression*: The LogisticRegression class in scikit-learn is a linear model for binary and multiclass classification tasks. The model employs logistic functions to model the probability of a sample belonging to a particular class. Regularization terms can also be applied to this classifier to prevent overfitting. [9]
3. *RandomForestClassifier*: The RandomForestClassifier

in scikit-learn is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes for classification tasks. It aggregates the predictions of multiple trees that results in improved generalization performance and reduces overfitting. It is robust, handles high-dimensional data, and provides feature importances, making it widely used in practice. [17]

4. *DecisionTreeClassifier*: The DecisionTreeClassifier in scikit-learn is an implementation of decision tree-based classification. Decision trees recursively split the feature space based on the most informative features, creating a tree-like structure for making decisions. This classifier is interpretable, handles both numerical and categorical data, and can capture complex relationships in the data.[11]

5. *GradientBoostingClassifier*: The GradientBoostingClassifier in scikit-learn is an ensemble method that builds a sequence of weak learners to create a strong predictive model. It sequentially corrects errors made by previous models, leading to high predictive accuracy. This classifier is effective in capturing complex relationships and is robust against overfitting. [13]

6. *AdaBoostClassifier*: The AdaBoostClassifier in scikit-learn is an ensemble learning method that combines multiple weak learners to create a strong classifier. It assigns weights to data points that focus on misclassified samples in each iteration. This classifier is particularly effective in boosting the performance of weak models, achieving high accuracy even with simple base classifiers. It is widely used in practice for classification tasks. [10]

7. *SVC* (Support Vector Classification): The SVC class in scikit-learn represents the Support Vector Classification algorithm, which is a powerful and versatile method for both binary and multiclass classification. It aims to find a hyperplane that maximally separates classes in the feature space. The classifier supports different kernel functions to handle non-linear relationships and is effective in high-dimensional spaces. [19]

8. *KNeighborsClassifier*: The KNeighborsClassifier in scikit-learn is an implementation of the k-nearest neighbors (KNN) algorithm which is a simple and effective non-parametric classification method. This classifier assigns a class label to a sample based on the majority class among its k-nearest neighbors in the feature space. [14]

9. *SGDClassifier* (Stochastic Gradient Descent): The Stochastic Gradient Descent in scikit-learn is an implementation of linear classifiers using Stochastic Gradient Descent (SGD) optimization.e default loss function for classification is the 'hinge' loss, which corresponds to a linear Support Vector Machine (SVM). It excels in large-scale machine learning tasks as it updates model parameters iteratively with small batches of data. Moreover, it is particularly well-suited for large-scale and sparse datasets, making it efficient and scalable.[18]

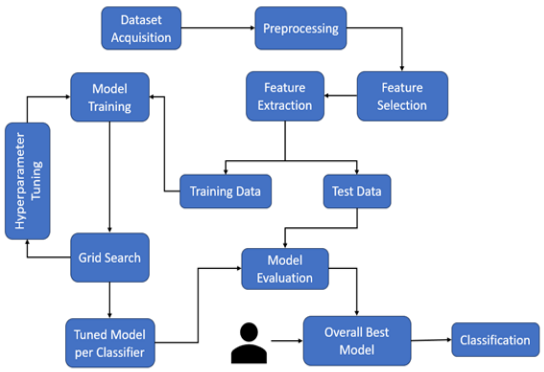

Figure 1: Flowchart Overview of the Methodology

10. *MLPClassifier* (Multilayer Perceptron Classifier): The MLPClassifier in scikit-learn represents a multilayer perceptron, a type of artificial neural network used for classification tasks. It consists of an input layer, one or more hidden layers, and an output layer. The model is trained using backpropagation and gradient descent, making it capable of learning complex non-linear relationships in the data. It is a versatile classifier suitable for various applications, but its performance may depend on the choice of hyperparameters and the amount of training data available.[15]

4. CONTRIBUTION

Taking inspiration from Cruz et al.'s work [3], this study endeavors to develop a simplified model capable of predicting fake news in Filipino language, with a level of accuracy similar to their research. Utilizing the 'Fake News Filipino' benchmark dataset employed in Cruz et al.'s study, this research will leverage readily available algorithms in scikit-learn. These encompass individual/base learners including Stochastic Gradient Descent Classifier, Naïve Bayes Classifier, Logistic Regression, K-Nearest Neighbors Classifier, Support Vector Classifier, Decision Tree Classifier, ensemble learners including the Random Forest Classifier, Gradient Boosting Classifier, Adaboost Classifier, and a neural network classifier: Multilayer Perceptron Classifier.

This study will identify and select the best-performing model based on the overall highest accuracy achieved. This entails finding the optimal classifier along with an ideal combination of feature selection and hyperparameter tuning to maximize predictive performance. Furthermore, this study involves deploying this model to address the challenges encountered by Cruz et al. while utilizing deep learning techniques.

5. METHODOLOGY

This methodology details the steps in conducting the experiment, encompassing data acquisition, preprocessing, feature extraction, modeling, evaluation, and deployment.

5.1 Dataset Description and Preprocessing

The dataset utilized in this study originated from the dataset introduced by Cruz et al. [3], consisting of 3,206 news articles categorized as real or fake, with a 50/50 split between

the two. Fake news articles were sourced from online platforms tagged as fake news sites by the independent media fact-checking organization Verafiles and the National Union of Journalists in the Philippines (NUJP). Real news articles were collected from prominent mainstream news websites in the Philippines, such as Pilipino Star Ngayon, Abante, and Bandera.

Both training and test data set underwent the preprocessing phase which involves the removal of stop words, converting text to lowercase and excluding of punctuations marks. However, misspelled words were retained to maintain the original information within the datasets. A collection of 147 stop words in Filipino language was sourced from GitHub to facilitate this process.

5.2 Feature Selection and Feature Extraction

Both training and test data set underwent feature extraction with the Term Frequency-Inverse Document Frequency (TF-IDF) technique. TF-IDF directly processes text data to generate numerical representations from the documents without requiring tokenization.

The TfidfVectorizer function from the sklearn library was employed to implement TF-IDF for feature extraction. This method efficiently managed stop words, converted text to lowercase and excluded punctuation marks during preprocessing and generated TF-IDF features for subsequent analysis.

The initial part of the study involved standardized features for TF-IDF vectorization, maintaining a maximum document frequency of 90% and utilizing unigrams (single words). Subsequently, the second phase involved feature selection for the TF-IDF vectorization, exploring different values for tfidf_max_df and various settings for ngram_range.

The tfidf_max_df parameter was utilized to filter out words occurring in the TF-IDF matrix in over 70%, 80%, and 90% of the documents. Meanwhile, ngram_range was set to explore both unigrams (1, 1) and bigrams (1, 2) across all classifiers, representing single words and pairs of consecutive words, respectively.

The dataset was randomly shuffled and divided into an 80% training dataset and a 20% testing dataset. The training dataset was utilized to train various classifiers to identify the best model. Meanwhile, the testing dataset served the purpose of evaluating the performance of the tuned models.

5.3 Model Training and Hyperparameter Tuning

The extracted features were subjected to various classifiers for model training, including Naïve Bayes Classifier, Logistic Regression, K-Nearest Neighbors Classifier, Support Vector Classifier, Decision Tree Classifier, Random Forest Classifier, Gradient Boosting Classifier, Adaboost Classifier, Multilayer Perceptron Classifier, and Stochastic Gradient Descent Classifier which were all sourced from the scikit-learn library. The initial phase of the study did not involve any classifier hyperparameter tuning. Subsequently, the second phase included specific classifier hyperparameter tuning for each classifier. Table 2 presents both predefined hyperparameters values for each specific classifiers and the explored selected hyperparameter values during the analysis.

The initial phase exclusively utilized models trained during the training phase with default features for the TF-IDF vectorization and relied on default hyperparameter values as predefined by the library's developers.

In the subsequent phase involving feature selection and hyperparameter tuning, the gridsearchCV methodology was employed to identify the most suitable model for each classifier. This systematic evaluation aimed to determine the optimal combination of features and hyperparameters for each classifier under assessment. Due to the expansive search space, only the combinations of hyperparameters that constituted the best model for each classifier were retained which were assessed based on accuracy computed from the validation dataset through 5-fold cross-validation.

5.4 Model Selection and Model Evaluation

For each considered classifier, a tuned model was determined which incorporated iterative feature selection for TF-IDF vectorization and classifier hyperparameter tuning as determined from the validation dataset through gridsearchCV.

These tuned models underwent further evaluation using the test dataset to assess performance metrics including accuracy, precision, recall, F1 score, and specificity. The objective was to gauge their generalization capability, robustness, and overall predictive performance in real-world scenarios. Afterwards, these tuned models were compared to determine the overall best model based on the highest accuracy achieved in the test dataset.

5.5 Model Deployment

Finally, the overall best-performing model, as determined by the overall highest accuracy on the test dataset, will be integrated into a Flask Application. This platform will enable users to input text and instantly predict whether it is classified as "fake" with a simple click.

6. RESULTS AND DISCUSSION

In this section, we discuss the results acquired training the dataset in different classifiers. Different accuracy metrics such as accuracy, Recall, Precision, f1-score, and Specificity were recorded and analysed and evaluated.

6.1 Model Selection

Table 1 shows the training accuracies for various classifiers using both predefined hyperparameters and after applying hyperparameter tuning. Initially, the models trained with predefined settings already demonstrated high accuracy, notably the Stochastic Gradient Descent Classifier at 0.9578, the Multilayer Perceptron Classifier at 0.9512, and the Support Vector Classifier at 0.9450. However, some classifiers achieved lower accuracy, such as Naïve Bayes and K-Nearest Neighbors Classifier, scoring 0.8686 and 0.8690, respectively, though still presenting decent performance.

By implementing hyperparameter tuning, several classifiers exhibited increased training accuracies. Particularly, the

Table 1: Training Accuracies: Predefined Hyperparameter vs. Tuned Hyperparameter

Classifier	Predefined	Tuned
Multinomial Naive Bayes	0.8686	0.9181
Logistic Regression	0.9356	0.9524
Random Forest	0.9411	0.9419
Decision Tree	0.913	0.9142
Gradient Boosting	0.929	0.936
AdaBoost	0.9341	0.9431
Support Vector Machine	0.945	0.9571
K Neighbors	0.869	0.883
Stochastic Gradient Descent	0.9579	0.9579
Multilayer Perceptron	0.9512	0.9512

Naïve Bayes Classifier, Logistic Regression, K-Nearest Neighbors Classifier, and Support Vector Classifier showed improvements of approximately 5%, 1.7%, 1.4%, and 1.2%, respectively. This observed improvement in hyperparameter tuning primarily affected the individual learners among the classifiers which highlights their sensitivity to hyperparameter changes. In contrast, ensemble methods and neural networks, due to their inherent complexity and architectural differences, might showcase more robust behavior or lesser sensitivity to certain hyperparameters, leading to relatively less impact on their performance through tuning.

Table 2 shows the training times for both predefined hyperparameters and hyperparameter tuning across various classifiers. As observed, individual learners typically demonstrate shorter training times, ranging from 1.2 seconds to 11.2 seconds. Meanwhile, ensemble learners require comparatively longer training times, spanning from 8.8 seconds to 46.8 seconds while the Multilayer Perceptron Classifier, a neural network classifier, exhibited the longest training time of 292.7 seconds.

This difference in training times shows the varying complexities among classifier types. Individual learners, being simpler models, generally demand less computational time for training. In contrast, ensemble methods and neural network classifiers inherently involve more complex operations, requiring increased computational resources and resulting in longer training durations.

Moreover, training times with implemented hyperparameter tuning are notably higher compared to the training times with predefined parameters. This outcome aligns with expectations since the tuning process involved testing various hyperparameter combinations, leading to increased iterations during training to explore these combinations thoroughly. While one contributing factor to longer training times is the inherent complexity of the classifier, the increase in training durations primarily is an effect from the increased number of combinations explored during hyperparameter tuning.

Table 2: Training Times(in seconds): Predefined Hyperparameter vs. Hyperparameter Tuning

Classifier	Predefined	Tuned
Multinomial Naïve Bayes	4.97	18.97
Logistic Regression	3.31	38.34
Random Forest	14.7	205.6
Decision Tree	4.32	201.3
Gradient Boosting	46.8	2765
AdaBoost	8.76	1040
Support Vector Machine	11.2	197.7
K-Nearest Neighbors	9.38	402.8
Stochastic Gradient Descent	1.21	35.61
Multilayer Perceptron	293	11700

Table 3: Test Metrics for Tuned Parameters: Accuracy, Recall, and Precision Scores

Classifier	Accuracy	Recall	Precision
Multinomial Naïve Bayes	0.9174	0.9414	0.892
Logistic Regression	0.9595	0.9316	0.9828
Random Forest	0.9486	0.9283	0.9628
Decision Tree	0.9081	0.8958	0.9106
Gradient Boosting	0.9377	0.9088	0.9588
AdaBoost	0.9361	0.9153	0.9493
Support Vector Machine	0.9611	0.9446	0.9732
K-Nearest Neighbors	0.8723	0.9349	0.8223
Stochastic Gradient Descent	0.9626	0.9446	0.9764
Multilayer Perceptron	0.9611	0.9642	0.9548

To choose the model for deployment, the tuned hyperparameters performance were tested using the test data. Notably, four classifiers namely Logistic Regression, Stochastic Gradient Descent, Support Vector Machine (SVC), and the Multilayer Perceptron (MLP) Classifier achieved accuracies exceeding 95% while most of the remaining classifiers has achieved at least 90% accuracies.This is comparable to the accuracy of models in [3]. Furthermore, recall, precision, f1-score and specificity, as presented in Tables 3 and 4, indicate strong performance across various accuracy metrics for almost all models.

Overall, the stochastic gradient descent classifier emerges as the top-performing model, boasting the highest test accuracy. Despite its simplicity compared to other classifiers, it not only excelled in accuracy but also demonstrated the shortest training time. This highlights that even the simplest classifiers can yield higher accuracy compared to more complex counterparts. Further evaluation using the test dataset confirms the model's ability to generalize, maintaining high accuracy at 0.9626. Additionally, consistent high values across various performance metrics (ranging from 0.94 to 0.98) signify the robustness and generalizability of the model.

Table 4: Test Metrics for Tuned Parameters: f1-score and Specificity

Classifier	f1-score	Specificity
Multinomial Naïve Bayes	0.916	0.8955
Logistic Regression	0.9565	0.9851
Random Forest	0.9453	0.9672
Decision Tree	0.9031	0.9194
Gradient Boosting	0.9331	0.9642
AdaBoost	0.932	0.9552
Support Vector Machine	0.9587	0.9761
K-Nearest Neighbors	0.875	0.8149
Stochastic Gradient Descent	0.9603	0.9791
Multilayer Perceptron	0.9595	0.9582

6.2 Overall Best Model Deployment

The superior model was deployed via a Flask application as shown in Figures 2 and 3. This allows users to input text and predict whether it is classified as "fake" or not with a simple click. The article in Figure 2 is an article tagged as Fake News and the model predicted it as a Fake news also.

Here is the translation for the fake news article. "Netizens are currently criticizing the move of Vice President Leni Robredo at a day care center in Tondo, Manila. During her storytelling session with the youth in Tondo, Leni read a book to the students titled "Digong Yellow." The story of the book revolves around a young boy named Digo who has a fondness for the color yellow. Leni is a member of the Liberal Party, known for its association with the color yellow."

This marks as a starting point for the venture into creating a consumer-level application. The application provides a streamlined user experience, enabling effortless interaction and immediate predictions.

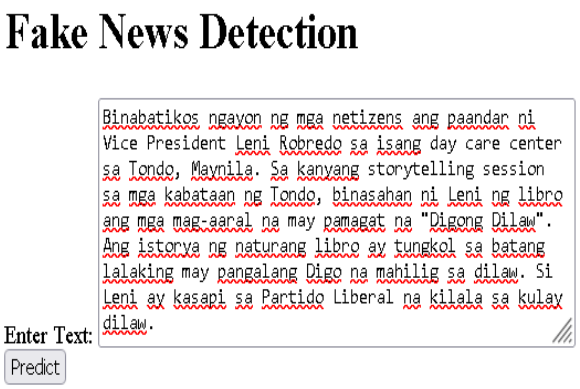

Figure 2: Sample interface of Fake News Detection by the Model Deployed

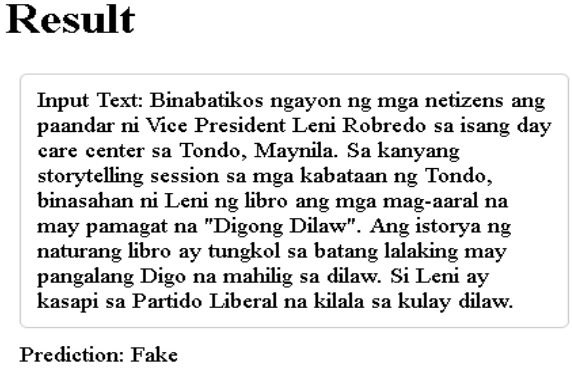

Figure 3: Sample Result of Fake News Detection by the Model Deployed

7. CONCLUSION

In response to the pervasive challenge of misinformation in the Philippines, this study aimed to develop a simple yet effective model for accurately identifying fake news in the Filipino language. The research extensively tested various classifiers available in scikit-learn, exploring individual, ensemble, and neural network models while utilizing TF-IDF extracted features.

Initially, these models exhibited high accuracies using predefined hyperparameter values. Through subsequent hyperparameter tuning, significant accuracy improvements were observed in individual learner models, revealing their sensitivity to parameter adjustments and emphasizing the robustness of ensemble and neural network models. However, this robustness came with increased training times due to the inherent complexity of these models compared to individual classifiers.

Despite being an individual classifier, the stochastic gradient descent classifier emerged as the optimal model, displaying both the highest accuracy and remarkably shorter training times among all tested classifiers. Further evaluation through the test dataset shows the model's robustness and generalizability which has maintained its consistent high accuracy and high performance across various metrics.

Finally, this study achieved a significant milestone by easily deploying the optimal model which marks a considerable advancement from the deployment challenges faced in Cruz et al.'s study. Despite the simplified model architecture, this study achieved a remarkably high accuracy level that closely aligns with the findings of prior study conducted by Cruz et al. This initial implementation serves as a promising starting point for developing a consumer-level application.

8. RECOMMENDATION

Implementation of a strategy for continuous updates on news articles in the dataset is strongly advised to ensure the model remains adaptive to evolving patterns and trends in information. Regularly updating the dataset will enhance the model's ability to capture and understand new patterns, thereby improving its predictive accuracy. Additionally, labeling the dataset more comprehensively will contribute to a clearer understanding of the data and facilitate more accurate predictions. Furthermore, we suggest exploring additional datasets, particularly those containing localized posts on social media platforms such as Facebook and YouTube.

Incorporating data from these platforms will enable the model to detect and respond to the unique characteristics of misinformation prevalent in these social contexts. Furthermore, it is recommended to include other classifiers that were not used in this study to allow for a comparative analysis of their strengths and weaknesses in handling misinformation detection tasks, enabling the selection of the most suitable model or a combination of models to achieve optimal results. Additionally, broadening the scope of hyperparameter tuning by testing an extensive range of parameters can further refine the model's performance and enhance its ability to effectively detect and classify misinformation.

9. REFERENCES

[1] I. Ahmad, M. Yousaf, and S. Yousaf. Fake news detection using machine learning ensemble methods. *Complexity*, pages 1–11, 2020.

[2] J. Avelino, E. Felizmenio, and P. J. Naval. Unraveling COVID-19 misinformation with latent dirichlet allocation and catboost. In *14th International Conference on Computational and Collective Intelligence*, Hammamet, Tunisia, Sept 28-30 2022.

[3] J. C. B. Cruz, J. A. Tan, and C. Cheng. Localization of fake news detection via multitask transfer learning. pages 2589–2597. European Language Resources Association, May 2020.

[4] C. Janiesch, P. Zschech, and K. Heinrich. Machine learning and deep learning. *Electronic Markets*, 31(3):685–695, 2021.

[5] P. Kaur, R. S. Boparai, and D. Singh. Hybrid text classification method for fake news detection. *International Journal of Engineering and Advanced Technology (IJEAT)*, 8(5):2388, 2019.

[6] Z. Khanam, B. Alwasel, H. Sirafi, and M. Rashid. Fake news detection using machine learning approaches. In *IOP Conference Series: Materials Science and Engineering*, volume 1099, page 012040. IOP Publishing, March 2021.

[7] G. Pabico Lalu. 'fake news' a problem in ph? 9 in 10 filipinos agree, says pulse asia. *Inquirer.net*, October 11 2022.

[8] D. M. Rajeswari and R. C. C. Fake news detection using naive bayes algorithm with machine learning. *Journal of Emerging Technologies and Innovative Research (JETIR)*, 9(6):138, 2022.

[9] scikit-learn. LogisticRegression - scikit-learn 0.24 documentation. Online, 2022.

[10] scikit-learn contributors. AdaBoostClassifier. Online, n.d. Accessed on January 19, 2024.

[11] scikit-learn contributors. Decision Trees. Online, n.d. Accessed on January 19, 2024.

[12] scikit-learn contributors. Feature extraction. Online, n.d. Accessed on December 29, 2023.

[13] scikit-learn contributors. GradientBoostingClassifier. Online, n.d. Accessed on January 19, 2024.

[14] scikit-learn contributors. KNeighborsClassifier. Online, n.d. Accessed on January 19, 2024.

[15] scikit-learn contributors. MLPClassifier. Online, n.d. Accessed on January 19, 2024.

[16] scikit-learn contributors. Multinomialnb. Online, n.d. Accessed on January 19, 2023.

[17] scikit-learn contributors. RandomForestClassifier. Online, n.d. Accessed on January 19, 2024.

[18] scikit-learn contributors. SGDClassifier. Online, n.d. Accessed on January 19, 2024.

[19] scikit-learn contributors. SVC - Support Vector Classification. Online, n.d. Accessed on January 19, 2024.

[20] U. Sharma, S. Saran, and S. M. Patil. Fake news detection using machine learning algorithms. *International Journal of Creative Research Thoughts (IJCRT)*, 8(6):509–518, 2020.

[21] L. Tuquero. 51% of filipinos find it difficult to spot fake news on media – sws. *Rappler*, February 26 2022.

Table 5: Appendix A: Hyperparameters for Different Classifiers

Classifier	Default Hyperparameters	Selected Hyperparameter Values Explored
Stochastic Gradient Descent Classifier	learning rate, alpha = 0.0001, penalty term = l2	learning rate, alpha = [0.0001, 0.001], penalty term = ['l2', 'l1', 'elasticnet']
Naïve Bayes Classifier	smoothing parameter alpha = 1.0	smoothing parameter alpha = [0.1, 0.5, 1.0]
Logistic Regression	inverse regularization strength, C = 1.0	inverse regularization strength, C = [0.1, 1.0, 10.0]
K-Nearest Neighbors Classifier	number of neighbors = 5, weight scheme = uniform	number of neighbors = [3, 5, 7], weight scheme = ['uniform', 'distance']
Support Vector Classifier	C = 1.0, kernel = rbf	C = [0.1, 1.0, 10.0], kernel = ['linear', 'rbf']
Decision Tree Classifier	max depth = None, min samples split = 2, min samples leaf = 1	max depth = [None, 10, 20], min samples split = [2, 5], min samples leaf = [1, 2]
Random Forest Classifier	number of estimators = 100	number of estimators = [50, 100, 200]
Gradient Boosting Classifier	number of estimators = 100, learning rate = 0.1, max depth = 3	number of estimators = [50, 100], learning rate = [0.01, 0.1], max depth = [3, 5]
Adaboost Classifier	number of estimators = 50, learning rate = 1.0	number of estimators = [50, 100, 200], learning rate = [0.01, 0.1, 1.0]
Multilayer Perceptron Classifier	momentum coefficient, alpha = 0.0001, learning rate = 0.001	momentum coefficient, alpha = [0.0001, 0.001], learning rate = [0.001, 0.1]

Table 6: Appendix B: Best Hyperparameters for Different Classifiers

Classifier	Best Hyperparameters
SGDClassifier	'classifier__alpha': 0.0001, 'classifier__penalty': 'l2', 'tfidf__max_df': 0.7, 'tfidf__ngram_range': (1, 1)
MultinomialNB	'classifier__alpha': 0.1, 'tfidf__max_df': 0.7, 'tfidf__ngram_range': (1, 1)
LogisticRegression	'classifier__C': 10.0, 'tfidf__max_df': 0.7, 'tfidf__ngram_range': (1, 1)
KNeighborsClassifier	'classifier__n_neighbors': 7, 'classifier__weights': 'distance', 'tfidf__max_df': 0.7, 'tfidf__ngram_range': (1, 2)
SVC	'classifier__C': 10.0, 'classifier__kernel': 'linear', 'tfidf__max_df': 0.7, 'tfidf__ngram_range': (1, 1)
DecisionTreeClassifier	'classifier__max_depth': 20, 'classifier__min_samples_leaf': 1, 'classifier__min_samples_split': 5, 'tfidf__max_df': 0.7, 'tfidf__ngram_range': (1, 1)
RandomForestClassifier	'classifier__n_estimators': 200, 'tfidf__max_df': 0.7, 'tfidf__ngram_range': (1, 1)
GradientBoostingClassifier	'classifier__learning_rate': 0.1, 'classifier__max_depth': 5, 'classifier__n_estimators': 100, 'tfidf__max_df': 0.7, 'tfidf__ngram_range': (1, 1)
AdaBoostClassifier	'classifier__learning_rate': 1.0, 'classifier__n_estimators': 200, 'tfidf__max_df': 0.7, 'tfidf__ngram_range': (1, 1)
MLPClassifier	'classifier__alpha': 0.0001, 'classifier__learning_rate_init': 0.001, 'tfidf__max_df': 0.7, 'tfidf__ngram_range': (1, 1)