Volume 11 Issue 2 | June 2023 ISSN: 2320-9984 (Online)

International Journal of Modern Engineering & Management Research
Website: www.ijmemr.org

Optimizing Model Accuracy and Execution Time for Fake News Prediction using Machine Learning Algorithms

Deepali Rosta
M.Tech, Research Scholar
Computer Science and Engineering
Takshshila Institute of Engineering and Technology
Jabalpur (M.P.), India
Email: deepalirosta@gmail.com

Swati Soni
Assistant Professor
Department of Computer Science and Engineering
Takshshila Institute of Engineering and Technology
Jabalpur (M.P.), India
Email: swatisoni@takshshila.org

Abstract**—***The time period leading up to the 2016 U.S. presidential election refers to the months and weeks preceding the election day, which was held on November 8, 2016. It was a politically charged period marked by intense campaigning, debates, and media coverage focused on the presidential candidates, their policies, and the overall state of the nation. The election primarily featured two major-party candidates, Donald Trump and Hillary Clinton, who represented the Republican and Democratic parties, respectively. The Source of the Dataset is Kaggle, the dataset used in this study was obtained from Kaggle and consists of 20,800 rows and 5 columns. The columns in the dataset include "id," "title," "author," "text," and "label." Each entry in the dataset is uniquely identified by the "id" column. The "title" column contains the headline or title of the news articles, while the "author" column provides information about the author or source of the articles. The main body of the articles is present in the "text" column, providing further details and information. The "label" column indicates that the dataset is used for a classification task, where each entry is assigned, a label representing a certain category or sentiment (e.g., true/false, reliable/unreliable, etc.). The Study presents the accuracy scores achieved by each model on the training and test data, as well as the execution times for each model. Several machine learning models were created and evaluated using this dataset, and their performance metrics were recorded. The models include Logistic Regression, MultinomialNB, DecisionTreeClassifier, GradientBoostingClassifier, LGBMClassifier, XGBClassifier, and Random Forest Classifier. The modified versions of the Gradient Boosting Classifier and Random Forest Classifier are also examined. Thus, findings provide insights into the performance of different machine learning models applied to the dataset. The accuracy scores represent the models' performance on both the training and test data, while the execution times indicate the time taken by each model to train and make predictions.*

Keywords:**—** *Fake News Prediction, Machine Learning algorithm, NLTK, Logistic Regression, MultinomialNB, DecisionTree Classifier, GradientBoostingClassifier, Random Forest Classifier*

1. INTRODUCTION

The dataset provided is from a Kaggle on fake news detection. The dataset aims to provide a collection of news articles labeled as either "fake" or "real" to facilitate the development and evaluation of machine learning models for fake news detection. It allows researchers to explore various machine learning techniques and develop strategies to identify misleading or fabricated information in news articles. By using this dataset, researchers can perform partition over the Dataset, train machine learning models using the labeled training set and evaluate their performance on the test set. The competition framework encourages the development of effective techniques and models that can accurately classify news articles as "fake" or "real" based on their content.

Machine Learning

Machine learning is a field of artificial intelligence that focuses on the development of algorithms and models that enable computers to learn and make predictions or decisions without being explicitly programmed. It is concerned with creating systems that can automatically learn from and improve with experience. In machine learning, a model is trained on a dataset, which consists of input data and corresponding output labels or target values. The model learns patterns and relationships within the data to make predictions or take actions when presented with new, unseen data. There are several types of machines learning techniques, including:

Supervised Learning: In this approach, the model is trained using labeled data, where the input data is paired with the correct output labels. The model learns to generalize from the training data and make predictions or classify new, unseen data based on the learned patterns.

Unsupervised Learning: Here, the model is trained on unlabeled data, and its objective is to discover inherent patterns or structures within the data. Unsupervised learning techniques include clustering, dimensionality reduction, and anomaly detection.

Semi-Supervised Learning: This is a combination of supervised and unsupervised learning. It uses a small amount of labeled data along with a larger amount of unlabeled data to train the model. The labeled data helps guide the learning process and improve the model's performance.

Reinforcement Learning: In reinforcement learning, an agent learns to make decisions and take actions in an environment to maximize a reward signal. The agent explores the environment, receives feedback in the form of rewards or penalties, and adjusts its actions accordingly to optimize its performance. Machine learning can be utilized for fake news detection by leveraging the patterns and characteristics present in both fake and genuine news articles.

A. Logistic Regression

Logistic regression is a widely used algorithm in machine learning for tackling binary classification problems. It is a supervised learning algorithm that aims to predict the probability of an input belonging to a specific class. Contrary to its name, logistic regression is primarily employed for classification tasks rather than regression.

The fundamental concept behind logistic regression is to establish a relationship between the input variables (features) and the probability of the target variable (class label) assuming a particular value. The output of logistic regression is a probability value ranging from 0 to 1, which indicates the likelihood of the input belonging to the positive class.

B. MultinomialNB

The Multinomial Naive Bayes (MultinomialNB) classifier is primarily designed for text classification tasks with discrete features. However, it can also be used for binary classification tasks by converting the binary labels into two distinct classes. The classifier is based on the Naive Bayes principle, which assumes that the features are conditionally independent given the class. MultinomialNB works for binary classification:

C. Decision Tree Classifier

A Decision Tree Classifier for fake

news prediction is a machine learning algorithm that uses a decision tree model to classify news articles as either fake or genuine. The algorithm learns from a labeled dataset of news articles, where each article is labeled with its corresponding class (fake or genuine). The decision tree model represents a flowchart-like structure where each internal node corresponds to a feature or attribute, and each leaf node represents a predicted class (fake or genuine).

The Decision Tree Classifier works by recursively splitting the dataset based on the selected features and splitting criteria to create a tree-like structure. The splitting criteria are chosen to maximize the homogeneity or purity of the resulting subsets. The algorithm determines the most informative features that can effectively separate fake and genuine news.The Decision Tree Classifier is a popular algorithm used for fake news prediction.

D. Random Forest Classifier

Random Forest Classifier is an ensemble learning algorithm that combines multiple decision trees tomake predictions. It is widely used for binary classification tasks. Here's how it works:

1. Data preparation: First, you need to prepare your data by splitting it into a feature matrix (X) and a target variable vector (y). The feature matrix contains the input features or attributes, while the target variable vector contains the corresponding binary labels (0 or 1).

2. Building Decision Trees: Random Forest Classifier creates a collection of decision trees. Each tree is trained on a random subset of the training data, known as a bootstrap sample. Additionally, at each node of the tree, only a random subset of features is considered for splitting. This process helps to introduce randomness and reduce overfitting.

3. Growing Decision Trees: Each decision tree is grown by recursively partitioning the data based on the selected features. The splits are determined by finding the best feature and threshold that maximize the information gain or Gini impurity (a measure of node purity). The process continues until a stopping criterion is met, such as reaching the maximum depth of the tree or having a minimum number of samples in a leaf node.

4. Voting for Predictions: Once all the decision trees are built, predictions are made by each tree individually. For binary classification, each tree votes for the class label (0 or 1) based on the majority class in its leaf node. The final prediction is determined by taking the majority vote of all the trees (mode of the predictions).

5. Handling Missing Data: Random Forest Classifier can handle missing data effectively. When making predictions for a sample with missing values, the algorithm considers all available features and averages the predictions from the trees that do not rely on the missing feature.

***6. Evaluation and Tuning**:* After training the Random Forest Classifier, you can evaluate its performance on a separate validation or test set. Common evaluation metrics for binary classification include accuracy, precision, recall, and F1 score. If necessary, you can adjust the hyperparameters of the Random Forest, such as the number of trees, the maximum depth of each tree, or the number of features considered at each split, to improve its performance.

Random Forest Classifier is known for its robustness, scalability, and ability to handle high-dimensional data. It can handle both numerical and categorical features and provides estimates of feature importance, allowing you to analyze the contribution of each feature in the classification process.

E. Gradient Boosting Classifier

Gradient Boosting Classifier is an ensemble machine learning algorithm that is widely used for fake news prediction. It works by combining multiple weak models, usually decision trees, to create a strong predictive model. Here's a high-level concept of how Gradient Boosting Classifier works for fake news prediction:

Data Preparation: Prepare a labeled dataset of news articles, where each article is labeled as either fake or genuine. Split the dataset into input features (X) and labels (Y).

Initialization: Initially, the Gradient Boosting Classifier creates a weak model, such as a decision tree, to make predictions. The weak model is assigned equal weight to all instances in the dataset.

Predictions and Residuals: The weak model is used to make predictions on the dataset. The difference between the predicted values and the actual labels (residuals) is calculated.

Building the Ensemble: A new weak model, typically another decision tree, is created to predict the residuals. This model is trained on the residuals rather than the original labels. The goal is to capture the patterns in the residuals that were not captured by the previous model.

Weighted Voting: The predictions from all the weak models are combined using weighted voting. The weight assigned to each model is determined by its performance in minimizing the residuals. Models that contribute more to reducing the overall error are assigned higher weights.

Iterative Training: Steps 3 to 5 are repeated iteratively. In each iteration, a new weak model is trained to predict the residuals of the previous ensemble. The ensemble is updated by adding the new model with an updated set of weights.

Final Prediction: The final prediction is obtained by summing the predictions from all the weak models in the ensemble, considering their respective weights. The resulting value is typically transformed using a threshold to classify the news articles as fake or genuine.

2. RELATED WORK

A. Data Preprocessing

The data preprocessing tasks performed on the fake real news prediction dataset include:

1. Handling Missing Values: The code replaces missing values in the dataset, specifically in the text column, with an empty string. This ensures that the subsequent processing steps can handle the data properly.

2. Feature Extraction: Instead of using the entire text column, the code merges the title and author columns to create a new column called "Content." This step aims to reduce the text size and processing time by focusing on specific information that may be relevant for the classification task.

3. Removing a Column: The code removes a column, likely the original text column, from the dataset. This step eliminates unnecessary features that are not used in the subsequent modeling process.

4. Stemming: Stemming is applied to the content column to reduce words to their root forms. The code uses regular expressions to remove digits, special characters, and punctuations. It then converts uppercase letters to lowercase, splits the text into a list of words, applies stemming to each word (excluding stopwords), and finally joins the stemmed words. This process helps to normalize the text data and reduce the vocabulary size.

5. Converting Raw Text Data into Numerical Value: The code uses the TfidfVectorizer from scikit-learn to convert the preprocessed text data into a numerical representation using the TF-IDF scheme. This technique calculates the importance of each word in the dataset based on its frequency in a specific document and across all documents. It generates a matrix of numerical features that can be used for training machine learning models.

B. Natural Language Toolkit (NLTK)

The Natural Language Toolkit is a popular open-source library for working with human language data in Python. It provides a wide range of tools and resources for tasks such as tokenization, stemming, lemmatization, part-of-speech tagging, parsing, semantic reasoning, and more. NLTK is widely used in the fields of natural language processing (NLP) and computational linguistics for research, education, and building NLP applications [14].

Natural Language Processing (NLP) is a field of study that focuses on enabling computers to understand, interpret, and generate human language. NLP techniques are often combined with Machine Learning (ML) algorithms to build models that can process and analyze textual data.Here's a general overview of how NLP works in Machine Learning:

1. Data Preprocessing: The first step in NLP is to preprocess the raw text data. This includes tasks such as tokenization (splitting text into individual words or tokens), removing punctuation and stopwords (commonly used words like “the,” “is,” etc. that do not carry much meaning), and performing other text normalization techniques like stemming or lemmatization to reduce words to their base forms.

2. Feature Extraction: Once the text data is preprocessed, the next step is to convert the textual information into numerical features that ML models can understand. Common feature extraction [3] techniques in NLP include bag-of-words representation, TF-IDF (Term Frequency-Inverse Document Frequency) weighting, word embeddings (such as Word2Vec or GloVe), or more advanced techniques like transformer-based models (e.g., BERT).

3. Model Training: With the numerical features extracted from the text data, ML models can be trained on labeled datasets for various NLP tasks. For example, tasks like sentiment analysis, text classification, named entity recognition, machine translation, question answering, etc. ML models such as Naive Bayes, Support Vector Machines (SVM), decision trees, random forests, or deep learning models like recurrent neural networks (RNNs), convolutional neural networks (CNNs), or transformer models can be used for training.

4. Model Evaluation and Tuning: Once the model is trained, it needs to be evaluated to assess its performance. This is typically done using evaluation metrics specific to the NLP task at hand. Common metrics include accuracy, precision, recall, F1 score, or perplexity, depending on the specific task. If the model's performance is not satisfactory, it may require further tuning or optimization by adjusting hyperparameters, trying different model architectures, or using more advanced techniques like ensemble learning.

5. Deployment and Inference: After the model has been trained and evaluated, it can be deployed to make predictions on new, unseen text data. This involves feeding the raw text into the trained model, which processes the input using the learned patterns and provides predictions or outputs based on the task it was trained for.

C. Feature Selection & Feature Extraction

Feature Selection: Feature selection is the process of selecting a subset of features from the original dataset that are most relevant to the target variable or have the highest predictive power. The main objective of feature selection is to improve model performance, reduce dimensionality, and eliminate unnecessary or redundant features. By selecting the most informative features, we can reduce computational complexity and enhance the interpretability of the model.

Feature Extraction: Transforming the original features into a lower-dimensional space while capturing essential information. It aims to reduce redundancy and noise in the data.

It's important to note that feature extraction is different from feature selection. While feature extraction creates new features, feature selection focuses on selecting a subset of the original features based on their relevance or importance. Both methods aim to improve model performance, reduce computational complexity, and mitigate the risk of overfitting.

D. Stopwords

Stopwords are common words that often occur in a text but do not carry significant meaning or contribute much to the overall understanding of the text. In natural language processing (NLP), stopwords are typically removed during the data preprocessing stage to reduce noise and improve the efficiency and effectiveness of text analysis tasks.

Examples of stopwords in English include words like “the,” “is,” “and,” “a,” “an,” “in,” “on,” “of,” etc. These words are frequently used in the language but do not provide much context or information about the specific topic or sentiment of the text. The removal of stopwords is performed to:

Reduce Dimensionality: Stopwords are often the most frequently occurring words in a document, and their presence can result in a high-dimensional feature space. Removing stopwords helps reduce the dimensionality of the dataset and focus on more meaningful words.

Speed up Processing: By eliminating stopwords, the processing time for NLP tasks such as tokenization, parsing, and feature extraction is reduced. This is because stopwords are common and their removal helps speed up the computations.

Improve Accuracy: In some NLP tasks, like sentiment analysis or text classification, stopwords do not contribute much to the sentiment or classification decision. By removing them, the model can focus on more informative words and potentially improve accuracy.

E. PorterStemmer

PorterStemmer: The Porter stemming algorithm, also known as the Porter Stemmer, is a widely used algorithm for stemming English words. Stemming is the process of reducing words to their base or root form, called the stem, by removing suffixes or inflections. The PorterStemmer is a rule-based algorithm that applies a set of predefined rules to transform words into their stems. For example, words like “running,” “runs,” and “ran” would all be stemmed to the base form “run” using the PorterStemmer algorithm. It helps to normalize words and reduce them to a common form, which can be useful for tasks like information retrieval, text mining, and feature extraction in NLP.

Stemming: Stemming is the process of reducing words to their base or root form, called the stem. It involves removing suffixes or inflections from words to obtain the core meaning or essence of a word. The stem may not always be a valid word, but it represents a linguistic or morphological variant of the original word. Stemming is commonly used in NLP to handle variations of words and reduce them to a common form for tasks like indexing, search, information retrieval, and text analysis. Different stemming algorithms, such as the Porter Stemmer, Lancaster Stemmer, and Snowball Stemmer, can be applied based on the specific requirements of the application or language.

F. TfidfVectorizer (Term Frequency-Inverse Document Frequency)

TfidfVectorizer is a popular feature extraction technique used in NLP to convert a collection of raw text documents into a numerical representation suitable for machine learning algorithms. TF-IDF stands for Term Frequency-Inverse Document Frequency. It calculates a weight for each term (word) in the document based on its frequency in the document and its importance in the entire corpus of documents. The TF-IDF value increases with the frequency of the term in the document but is offset by the frequency of the term across the corpus. TfidfVectorizer computes the TF-IDF values for each term in the document collection and represents the documents as vectors in a high-dimensional space. This vector representation can then be used as input for various machine learning models. TfidfVectorizer is commonly used for tasks such as text classification, clustering, and information retrieval.

G. To improve the Execution Time of the Gradient Boosting Classifier

To potentially improve the execution time of the Gradient Boosting Classifier-

1. Reduce the Learning Rate: The learning rate (learning_rate) controls the contribution of each tree in the ensemble. By reducing the learning rate, you can make the boosting process more conservative, potentially improving execution time. However, decreasing the learning rate may require more iterations to achieve comparable accuracy.

ModelGBC = GradientBoostingClassifier (learning_rate=0.1, subsample=0.8) # Adjust the learning_rate as needed

2. Limit the Maximum Depth of Trees: Controlling the maximum depth of the trees (max_depth) in the Gradient Boosting Classifier can help reduce the execution time. Setting a smaller value for max_depth restricts the depth of individual decision trees, resulting in faster training. However, be cautious not to set it too low, as it may lead to underfitting and lower accuracy.

ModelGBC = GradientBoostingClassifier (max_depth=3, subsample=0.8) # Adjust the max_depth as needed

3. Reduce the Number of Estimators: Decrease the number of estimators (n_estimators) in the Gradient Boosting Classifier. The number of estimators determines the number of boosting stages. By reducing the number of boosting stages, the training and prediction time can be reduced. Experiment with different values to find a balance between execution time and accuracy.

ModelGBC = GradientBoostingClassifier (n_estimators=100, subsample=0.8) # Adjust the n_estimators as needed

4. Parallelize Training: Enable parallel training using multiple cores by setting the n_jobs parameter to a higher value. This allows the GradientBoostingClassifier to utilize multiple cores during training, potentially reducing the execution time.

ModelGBC = GradientBoostingClassifier (n_estimators=100, subsample=0.8, n_jobs=-1) # Adjust n_jobs as needed

5. LightGBM: LightGBM is a high-performance gradient boosting framework developed by Microsoft. It is known for its efficiency and faster execution time compared to scikit-learn's implementation. You can install LightGBM using pip install lightgbm and then use it as follows:

import lightgbm as lgb

ModelGBC = lgb.LGBMClassifier (n_estimators=100)

# Fit Model

start_timeGBC = time.time()

ModelGBC.fit(X_train, Y_train)

GBC_time = time.time() - start_timeGBC

6. XGBoost: XGBoost is another popular gradient boosting library that provides efficient and scalable implementation. You can install XGBoost using pip install xgboost and then use it as follows:

import xgboost as xgb

ModelGBC = xgb.XGBClassifier (n_estimators=100)

# Fit Model

start_timeGBC = time.time()

ModelGBC.fit(X_train, Y_train)

GBC_time = time.time() - start_timeGBC

H. To improve the Execution Time of the Random Forest Classifier

To improve the execution time of the Random Forest Classifier the following modifications can be considered:

1. Reduce the Number of Trees: The execution time of the Random Forest Classifier is directly proportional to the number of trees in the forest. By reducing the n_estimators parameter, you can decrease the execution time. However, keep in mind that reducing the number of trees may affect the accuracy of the model.

Example: ModelRFC = RandomForestClassifier(n_estimators=100, random_state=0)

2. Limit the Depth of Trees: Controlling the maximum depth of the trees in the Random Forest can help reduce the execution time. By setting the max_depth parameter to a smaller value, you can reduce the complexity of the trees.

Example: ModelRFC = RandomForestClassifier(max_depth=10, random_state=0)

3. Use Subset of Features: Random Forests are capable of handling a large number of features, but using a subset of features can help reduce the execution time. You can set the max_features parameter to a smaller value, limiting the number of features considered for each split.

Example: ModelRFC = RandomForestClassifier(max_features=10, random_state=0)

4. Parallelize Training: Enable parallel training using multiple cores by setting the n_jobs parameter to a higher value. This allows the RandomForestClassifier to utilize multiple cores during training, potentially reducing the execution time.

ModelRFC = RandomForestClassifier (n_estimators=100, random_state=0, n_jobs=-1) # Adjust n_jobs as needed

5. Consider Memory Optimization: If memory usage is a concern, you can experiment with reducing the memory footprint by specifying the max_features parameter. Setting max_features to a lower value limits the number of features to consider at each split, which can reduce memory requirements and improve execution time.

ModelRFC = RandomForestClassifier (max_features="sqrt", random_state=0) # Adjust max_features as needed

3. PROPOSED WORK AND IMPLEMENTATION

1. Used Dataset

Data Source **-** https://www.kaggle.com/competitions/fake-news/data

About Dataset

Content

train.csv: This file contains the training set of news articles, where each row represents an article and includes the following columns:

- id: A unique identifier for each article.
- title: The headline or title of the news article.
- author: The author or source of the article.
- text: The main body of the article.
- label: The label indicating whether the article is "fake" or "real."

Outcome: Class variable (0 or 1)

Class Distribution:

1 represents: Unreliable / **Fake News**

0 represents: Reliable / **Real News**

train Data Set Rows / Features – 20800

Rows * 5 Columns

File –1, file Size - 94.1 MB, Type – csv

2. Platform and Methodology

IDE - Google Collaborator

Python – Python 3

Applied Binary Classification on Labeled Data

3. Implementation

A. Mounting the Drive

B. Importing the Dependencies

C. Downloading package stopwords to /root/nltk_dat

D. Print Stopwords in English

E. Loading the Dataset to a Pandas Dataframe

F. Find the Shape of the DataFrame

i. Print 1st 5 Rows of DataSet

ii. Number of Missing Values

G. Data Preprocessing

i. We can Drop these Missing Values, Here Replacing the Null Values with empty string

H. Feature Extraction

i. As Text Column may have Large Text, which can take very much time for processing. Thus, here we are Not Using Text Column & Using title & Author Column. Merging the author name & title, Creating new Column as Content.

I. Separating the Data and Label

i. To Remove a Column from Dataset axis = 1

ii. To Remove a Row from Dataset axis = 0

J. Fake News Classification Using Pie Chart Where 0-Real News 1-Fake News

K. Stemming

i. Stemming is the process of reducing the words to its Root Word.

ii. Example: actor, actress, acting --> act

iii. re = Regular Expression

iv. sub = Substituting Some Values

v. ^ Excluding Everything (Digits/ Special Character/Punctuations All will beReplaced by White Space) Taking Only a--z and A--Z from content Column

vi. Then Converting All Upper Case Letters to Lower Case Letters

vii. Then Spliting it into List

viii. Stemming - Converting words into Rootword Not in Stopwords

ix. Then Joining the Words

x. Getting Stemmed Content

L. Applying Stemming on content

i. No Upper-Case Letters & All the Root Words are there

M. Separating the Features & Label

N. Converting Raw Text Data into Numerical Value

i. TfidfVectorizer() Term Frequency Inverse Document Frequency

ii. Fitting it On X Not on Y, y is Already a Number

O. Splitting Data set into Training & Test

i. 20% of Total Data is Used as Testing Data and 80% is Used for Training

ii. stratify = Y Used?? 1: Fake News 0: Real News, If we Don’t Use stratify = Y, Then Real & Fake New Won't be Segregated in Equal Proportion after Splitting Data as were in Total Data

P. Logistic Regression Model

i. Create Model

ii. Fit the Model on Training Data

iii. Calculate Model Execution Time

iv. Model Evaluation - Accuracy_score Prediction on Training Data

v. Model Evaluation - Accuracy_score Prediction on Test Data

vi. Plot the training and testing accuracies

vii. Prepare Confusion Matrix to Evaluate Model Performance

viii. Calculate the AUC-ROC score to Evaluate Model Performance

ix. Manual Model Testing

Q. MultinomialNB Classifier

i. Create Model

ii. Fit the Model on Training Data

iii. Calculate Model Execution Time

iv. Model Evaluation - Accuracy_score Prediction on Training Data

v. Model Evaluation - Accuracy_score Prediction on Test Data

vi. Plot the training and testing accuracies

vii. Prepare Confusion Matrix to Evaluate Model Performance

viii. Calculate the AUC-ROC score to Evaluate Model Performance

R. Decision Tree Classifier

i. Create Model

ii. Fit the Model on Training Data

iii. Calculate Model Execution Time

iv. Model Evaluation - Accuracy_score Prediction on Training Data

v. Model Evaluation - Accuracy_score Prediction on Test Data

vi. Plot the training and testing accuracies

vii. Prepare Confusion Matrix to Evaluate Model Performance

viii. Calculate the AUC-ROC score to Evaluate Model Performance

S. Grdient Boosting Classifier

i. Create Model

ii. Fit the Model on Training Data

iii. Calculate Model Execution Time

iv. Model Evaluation - Accuracy_score Prediction on Training Data

v. Model Evaluation - Accuracy_score Prediction on Test Data

vi. Plot the training and testing accuracies

vii. Prepare Confusion Matrix to Evaluate Model Performance

viii. Calculate the AUC-ROC score to Evaluate Model Performance

T. Random Forest Classifier

i. Create Model

ii. Fit the Model on Training Data

iii. Calculate Model Execution Time

iv. Model Evaluation - Accuracy_score Prediction on Training Data

v. Model Evaluation - Accuracy_score Prediction on Test Data

vi. Plot the training and testing accuracies

vii. Prepare Confusion Matrix to Evaluate Model Performance

viii. Calculate the AUC-ROC score to Evaluate Model Performance

4. RESULTS

A. Gradient Booster Classifier

Model Execution Time
Model Accuracy Percentage

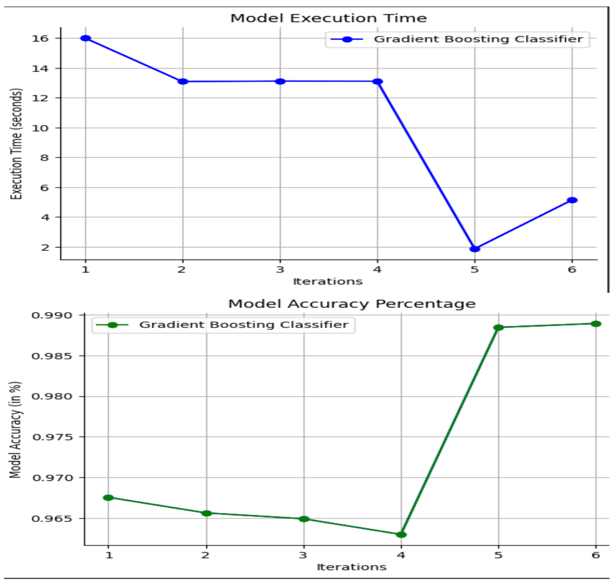

Figure 1: Gradient Boosting Classifier Model Execution Time & Accuracy Graph

B. Model Accuracy & Execution Time Comparisons

Table 1: Existing Vs Current Model Accuracy & Execution Time Comparison

Sr. No.	Models Tested	Test Data Accuracy	Model Execution Time (in Sec)
1	Logistic Regression Model	0.9790865384615385	0.32551074028015137
2	MultinomialNB Classifier	0.9550480769230769	0.01177072525024414
3	Decision Tree Classifier	0.9925480769230769	0.908332347869873
Proposed Work to Optimize Model Accuracy and Execution Time for Fake News Prediction Grdient Boosting Classifier			
1	GradientBoostingClassifier(max_depth=3)	0.9675480769230769	16.015048027038574
2	GradientBoostingClassifier(learning_rate=0.1, subsample=0.8)	0.965625	13.089359521865845
3	GradientBoostingClassifier(max_depth=3, subsample=0.8)	0.9649038461538462	13.114485502243042
4	GradientBoostingClassifier(n_estimators=100, subsample=0.8)	0.9629807692307693	13.107226848602295
5	LGBMClassifier(n_estimators=100)	0.9884615384615385	1.8740756511688232
6	XGBClassifier(n_estimators=100)	0.9889423076923077	5.1496899127960205
Random Forest Classifier			
1	RandomForestClassifier(random_state=0)	0.9942307692307693	22.227541208267212
2	RandomForestClassifier(n_estimators=100, random_state=0)	0.9942307692307693	25.408351182937622
3	RandomForestClassifier(max_depth=10, random_state=0)	0.8730769230769231	1.450249195098877
4	RandomForestClassifier(n_estimators=100, random_state=0, n_jobs=-1)	0.9942307692307693	15.483426809310913
5	RandomForestClassifier(max_features="sqrt", random_state=0)	0.9942307692307693	19.244366884231567

C. Random Forest Classifier

Model Execution Time
Model Accuracy Percentage

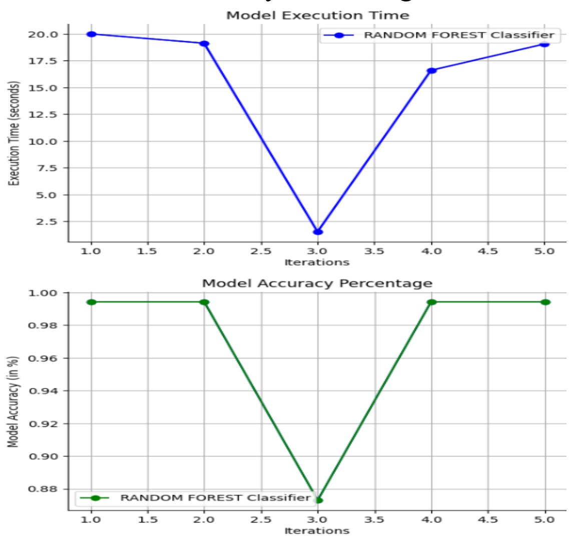

Figure 2: Random Forest Model Execution Time & Accuracy Graph

5. CONCLUSION

Dataset: The dataset contains 20,800 records with 5 features. It includes information about real and fake news, with 1 representing fake news and 0 representing real news.

Data Preprocessing Techniques: Several data preprocessing techniques have been applied to the dataset, including handling missing values, feature extraction, removing columns, stemming, and using TfidfVectorizer from scikit-learn. These techniques are essential for preparing the data before training the models.

Model Evaluation: Various classification models have been tested on the preprocessed dataset, including logistic regression, multinomial Naive Bayes, decision tree classifier, gradient boosting classifier, and random forest classifier. The models have been evaluated based on their test data accuracy and execution time.

Accuracy and Execution Time: The accuracy scores and execution times vary among the tested models. It is observed that the Random **Forest Classifier** generally achieves high accuracy scores **99.42%,** while the logistic regression and multinomial Naive Bayes models have shorter execution times. The decision tree classifier also demonstrates high accuracy, while the gradient boosting classifier exhibits a trade-off between accuracy and execution time.It can be observed that the **Execution Times of the GradientBoostingClassifier and** RandomForestClassifier **models varied with different parameter configurations**.

By Changing the parameters resulted in improved execution times. For example, in the case of the GradientBoostingClassifier, models 2, 3, and 4 had slightly lower execution times compared to model 1. Similarly, in the case of the RandomForestClassifier, model 3 had a significantly lower execution time compared to models 1 and 2.

REFERNCES:

[1] Nihel Fatima Baarir, Abdelhamid Djeffal, "Fake News detection Using Machine Learning", Workshop on Human-Centric Smart Environments for Health and Well-being (IHSH) | 978-1-6654-4084-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/ IHSH51661.2021.9378748.

[2] Amit Neil Ramkissoon, Shareeda Mohammed, "2020 International Conference on Data Mining Workshops (ICDMW) | 978-1-7281-9012-9/20/$31.00 ©2020 IEEE | DOI: 10.1109/ ICDMW51313.2020.00022".

[3] Vanya Tiwari, Ruth G. Lennon, Thomas Dowling. "Not Everything You Read Is True!Fake News Detection using Machine Learning Algorithms", 978-1-7281-9418-9/20/ $31.00 ©2020 IEEE.

[4] Arush Agarwal, Akhil Dixit, "Fake News Detection: An Ensemble Learning Approach", Proceedings of the International Conference on Intelligent Computing and Control Systems (ICICCS 2020), IEEE

Xplore Part Number:CFP20K74-ART; ISBN: 978-1-7281-4876-2.

[5] Karishnu Poddar, Geraldine Bessie Amali D, Umadevi K S, "Comparison of Various Machine Learning Models for Accurate Detection of Fake News", 978-1-5386-8190-9/19/ $31.00 ©2019 IEEE.

[6] Sherry Girgis, Eslam Amer, Mahmoud Gadallah, "Deep Learning Algorithms for Detecting Fake News in Online Text", 978-1-5386-5111-7/18/$31.00 ©2018 IEEE.

[7] Akshay Jain, Amey Kasbe, "Fake News Detection", 2018 IEEE International Students' Conference on Electrical, Electronics and Computer Sciences, 978-1-5386-2663-4/18/ $31.00 ©2018 IEEE.

[8] Iftikhar Ahmad, Muhammad Yousaf, Suhail Yousaf, Muhammad Ovais Ahmad, "Fake News Detection Using Machine Learning Ensemble Methods", Hindawi Complexity, Volume 2020, Article ID 8885861, 11 pages, https:// doi.org/10.1155/2020/8885861.

[9] Ms. Ch. Uma Devi, R. Priyanka, B.S. Priyanka, Ch. N.D.L. Nikhila, "Fake News Detection Using Machine Learning", © 2019 JETIR April 2019, Volume 6, Issue 4 www.jetir.org (ISSN-2349-5162).

[10] P. Yogendra Prasad, Dr.G. Nagalakshmi, P. Siva Kumar, "Fake News Detection Using Machine Learning", Journal of Pharmaceutical Negative Results, Volume 13, Issue 4, 2022.

[11] Z Khanam, B N Alwasel, H Sirafi1, M Rashid, "Fake News Detection Using Machine Learning Approaches", IOP Conf. Series: Materials Science and Engineering, 1099 (2021) 012040, doi:10.1088/1757-899X/1099/1/012040.

[12] Juliane Köhler, Gautam Kishore Shahi, Julia Maria Struß, Michael Wiegand, Thomas Mandl, Mina Schütz, "Overview of the CLEF-2022 Check That! Lab: Task 3on Fake News Detection", CLEF 2022 – Conference and Labs of the Evaluation Forum, September 5-8, 2022, Bologna, Italy.

[13] Khaled M. Fouad, Sahar F. Sabbeh, Walaa Medhat, "Arabic Fake News Detection Using Deep Learning", Computers, Materials & Continua, DOI:10.32604/cmc.2022.021449.

[14] Puja Sunil Erande, Monika Dhananjay Rokade, "Fake News Prediction Using Machine Learning for Social Media Dataset", Volume 6, Issue 4, April 2021, ISSN (Online) 2456-0774.

[15] Thasniya KP, Muneer V.K, Mohamed Basheer K.P, Rizwana K T., "Fake News Detection and Prediction Using Machine Learning Algorithms", Thasniya KP et al/ (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 12 (3), 2021, 81-85.

[16] Pinky Saikia Dutta, Meghasmita Das, Sumedha Biswas, Mriganka Bora, Sankar Swami Saikia, "Fake News Prediction: A Survey", International Journal of Scientific Engineering and Science, ISSN (Online): 2456-7361, Volume 3, Issue 3, pp. 1-3, 2019.

[17] Ms. Alpana A. Borse, Dr. G. K. Kharate, "Fake News Prediction using Hierarchical Attention Network and Hypergraph", International Conference on Contents, Computing & Communication (ICCCC-2022).

[18] G. Senthil Kumar, D. Ashok Kumar, “A Comparative Study on Various Machine Learning Algorithms for the Prediction of Fake News Detections Using Bring Feed New Data Sets”, International Journal of Scientific Research in Computer Science, Engineering and Information Technology, ISSN: 2456-3307 (www.ijsrcseit.com), doi: https:// doi.org/10.32628/CSEIT228691.

[19] Sudhakar Murugesan, Kaliyamurthie K.P, “Estimation of Precision in Fake News Detection Using Novel Bert Algorithm and Comparison with Random Forest”, Authorea. May 12, 2022, DOI:10.22541/ au.165237518.82791368/v1.