Adana Alparslan Turkes Science And Technology University

Project Report

<u>Class</u>: Cen-439 Introduction to web application security

<u>Project</u>: Fake-news-detection system

<u>Project Subject</u>: Understanding the news are fake or not

<u>Student Name</u>: Tolunay [redacted]

<u>StudentNu</u>:19[redacted]

1. <u>**Introduction**</u>

- <u>Background and motivation:</u>
    - In today's digital age, the spread of misinformation and fake news has become a significant concern. The ease of sharing information on social media platforms and the lack of fact-checking mechanisms have contributed to the rapid dissemination of false information. This phenomenon has the potential to influence public opinion, shape narratives, and even impact important decisions. Therefore, there is a growing need for reliable methods to detect and combat fake news effectively.
- <u>Problem statement:</u>
    - The problem addressed in this project is the detection of fake news using machine learning models. The aim is to develop a system that can automatically analyze news articles and determine their authenticity. By distinguishing between true and false information, this system can help users make informed decisions and mitigate the spread of misinformation.
- <u>Objectives:</u>
  The main objectives of this project are as follows:

    - Data Collection: Gather a comprehensive dataset consisting of both true and false news articles. This dataset will serve as the foundation for training and evaluating the machine learning models.

    - Data Preprocessing: Implement preprocessing techniques to clean and transform the raw data. This step involves removing irrelevant information, handling missing values, normalizing text, and extracting relevant features.

    - Machine Learning Model Training: Train four different machine learning models on the preprocessed dataset. The models include Logistic Regression, Decision Tree Classifier, Gradient Boosting Classifier, and Random Forest Classifier. Each model will learn to classify news articles as either true or fake based on the provided features.

    - Model Evaluation and Comparison: Evaluate the performance of each trained model using appropriate evaluation metrics such as accuracy, precision, recall, and F1 score. Compare the results of the models to determine their effectiveness in detecting fake news.

    - Model Serialization and Reusability: Save the trained models and vectorization techniques to allow for their reuse in other applications. This step ensures the practicality and efficiency of the developed models.

    - Backend Application Development: Create a Python-based backend application that exposes an API endpoint for detecting fake news. This application will integrate the trained models and provide real-time predictions based on user input.

- Node.js Application for Frontend Communication: Develop a Node.js application that acts as an intermediary between the frontend user interface and the Python backend application. This application will handle user input, communicate with the backend API, and display the prediction results on the frontend.

By accomplishing these objectives, we aim to build an effective and scalable system that can contribute to the detection and mitigation of fake news, promoting informed decision-making and combating the spread of misinformation in the digital landscape.

2. Data Collection and Preprocessing

- Data sources:
  The data used in this project is collected from online sources that provide both true and false news articles. Various reputable news websites, fact-checking organizations, and datasets specifically curated for fake news detection are considered as potential sources. The selection of diverse and reliable sources ensures the representativeness and quality of the dataset.
- Dataset description:
  The dataset consists of a collection of news articles labeled as either true or fake. Each article is associated with relevant metadata, such as the title, author, publication date, and source. The dataset aims to cover a wide range of topics and domains to reflect the diversity of news content found online.
- Data preprocessing techniques
  To prepare the data for training the machine learning models, several preprocessing techniques are applied. These techniques include:

  - Handling Missing Values:
    Missing values, if any, in the dataset are addressed using appropriate strategies. This may involve removing rows or columns with significant missing data or imputing missing values based on statistical measures such as mean, median, or mode.

  - Text Cleaning and Normalization:
    The textual content of the news articles undergoes cleaning and normalization processes. This typically involves removing special characters, punctuation, and numbers. Additionally, stop words (commonly used words with little semantic value) may be removed, and the remaining words are converted to lowercase to ensure consistency.

  - Feature Extraction:
    Feature extraction is performed to convert the text data into numerical representations that can be understood by machine learning algorithms. Techniques such as Term Frequency-Inverse Document Frequency (TF-IDF) or word embeddings like Word2Vec or GloVe may be applied to capture the semantic meaning of words and phrases.

- Data Splitting:
  The preprocessed dataset is divided into training and testing sets. The training set is used to train the machine learning models, while the testing set is employed to evaluate the performance of the trained models. A common split ratio, such as 80:20 or 70:30, is often used to ensure an adequate balance between training and testing data.

  By implementing these data collection and preprocessing techniques, we ensure the availability of a clean and representative dataset for training the machine learning models. The preprocessing steps enable the models to learn from meaningful features and enhance their ability to accurately classify news articles as true or fake.

3. Machine Learning Models

- Logistic Regression

  - **Model overview**
    Logistic Regression is a supervised learning algorithm used for binary classification tasks. It is a statistical model that estimates the probability of a binary outcome based on input features. In the context of fake news detection, Logistic Regression can predict whether a given news article is true or fake by learning the underlying patterns in the dataset.
  - Training process
    1. Load the preprocessed training data.
    2. Create an instance of the Logistic Regression model.
    3. Fit the model to the training data using the vectorized features obtained through TF-IDF or other feature extraction techniques.
    4. During the training process, the model adjusts its internal parameters to minimize the loss function and optimize the classification performance.
    5. Iterate the training process for a specified number of epochs or until convergence criteria are met.
    6. Upon completion of training, the model is ready to make predictions on new, unseen data.
- Decision Tree Classifier

  - Model overview
    A Decision Tree Classifier is a supervised learning algorithm that uses a binary tree structure to make decisions based on feature values. It partitions the data into subsets based on the values of the input features and predicts the target variable's class by following the decision path in the tree. In the context of fake news detection, a Decision Tree Classifier can learn hierarchical rules and make predictions based on different features to determine if a news article is true or fake.
  - Training process
    1. Load the preprocessed training data.

        2. Create an instance of the Decision Tree Classifier model.
        3. Fit the model to the training data using the vectorized features obtained through TF-IDF or other feature extraction techniques.
        4. During the training process, the model builds a tree structure by recursively partitioning the data based on feature values. The partitioning is done to minimize impurity or maximize information gain, depending on the specific algorithm used.
        5. Repeat the partitioning process for each node in the tree until a stopping criterion is met, such as reaching the maximum tree depth or having a minimum number of samples in the leaf nodes.
        6. Upon completion of training, the model is ready to make predictions on new, unseen data by traversing the decision path in the tree.

- Gradient Boosting Classifier

    - Model overview
      Gradient Boosting Classifier is an ensemble learning algorithm that combines multiple weak classifiers (decision trees) to create a strong predictive model. It works by training the weak classifiers sequentially, where each subsequent classifier focuses on correcting the mistakes made by the previous classifiers. In the context of fake news detection, Gradient Boosting Classifier can effectively capture complex relationships between features and improve the overall prediction accuracy.
    - Training process
        1. Load the preprocessed training data.
        2. Create an instance of the Gradient Boosting Classifier model.
        3. Specify the hyperparameters for the model, such as the number of boosting stages (number of weak classifiers to train) and the learning rate (controls the contribution of each weak classifier).
        4. Fit the model to the training data using the vectorized features obtained through TF-IDF or other feature extraction techniques.
        5. During the training process, the model sequentially trains weak classifiers, where each subsequent classifier focuses on minimizing the errors made by the previous classifiers.
        6. The training process involves iteratively adding weak classifiers to the ensemble and updating the weights of the samples based on their classification errors.
        7. The weak classifiers are typically decision trees with shallow depth to avoid overfitting. Each tree is trained to predict the residuals (the differences between the actual and predicted values) of the ensemble's previous predictions.
        8. Repeat the boosting process until reaching the specified number of boosting stages or achieving a desired level of performance.

9. Upon completion of training, the ensemble of weak classifiers forms the Gradient Boosting Classifier model, which can be used for making predictions on new, unseen data.

- Random Forest Classifier

  - Model overview
    Random Forest Classifier is an ensemble learning algorithm that combines multiple decision trees to create a powerful predictive model. It works by training each decision tree on a random subset of the training data and features, and then combining their predictions through voting or averaging. In the context of fake news detection, Random Forest Classifier can capture complex relationships and provide robust predictions.
  - Training process
    1. Load the preprocessed training data.
    2. Create an instance of the Random Forest Classifier model.
    3. Specify the hyperparameters for the model, such as the number of trees in the forest, the maximum depth of the trees, and the number of features to consider for each split.
    4. Fit the model to the training data using the vectorized features obtained through TF-IDF or other feature extraction techniques.
    5. During the training process, multiple decision trees are independently trained on random subsets of the training data and features.
    6. Each decision tree is trained using a subset of the training data, typically selected through bootstrap sampling, which involves randomly sampling the training data with replacement.
    7. At each split in the decision tree, a subset of features is randomly selected, and the best split is determined based on criteria such as Gini impurity or information gain.
    8. The process of training decision trees is repeated until the specified number of trees is reached.
    9. The predictions of the individual decision trees are combined through majority voting (for classification) or averaging (for regression) to obtain the final prediction of the Random Forest Classifier.
    10. The randomization in the training process helps to reduce overfitting and improve the model's generalization ability.

Evaluation metrics:

1. To assess the performance of the Logistic Regression model, several evaluation metrics are commonly used. These metrics provide insights into how well the model is classifying the news articles. Some commonly used evaluation metrics for binary classification tasks include:

2. Accuracy: It measures the overall correctness of the model's predictions by calculating the ratio of correctly classified samples to the total number of samples.

3. Precision: It quantifies the proportion of correctly classified positive predictions (true positives) out of all predicted positives (true positives + false positives). Precision represents the model's ability to avoid false positives.

4. Recall (Sensitivity or True Positive Rate): It calculates the proportion of correctly classified positive predictions (true positives) out of all actual positive samples (true positives + false negatives). Recall represents the model's ability to identify true positives and avoid false negatives.

5. F1 Score: It is the harmonic mean of precision and recall, providing a balanced measure of the model's performance. F1 Score considers both precision and recall and is often used as an overall evaluation metric.

   These evaluation metrics will be calculated and reported to assess the performance of the Logistic Regression model in classifying news articles as true or fake.

4. <u>**Model Evaluation and Comparison**</u>

Evaluation Metrics Used:

- Precision: It measures the accuracy of the positive predictions. For both classes (fake news and not fake news), precision is calculated.
- Recall: It measures the ability of the model to identify the positive cases correctly. Again, it is calculated for both classes.
- F1-score: It represents the harmonic mean of precision and recall, providing a balanced measure of model performance.
- Support: It indicates the number of samples in each class.

Comparison of Model Performance:

Logistic Regression:

- Precision: 0.99 for both classes
- Recall: 0.99 for both classes
- F1-score: 0.99 for both classes
- The Logistic Regression model shows high precision, recall, and F1-score for both classes, indicating excellent performance in classifying fake and not fake news.

Decision Tree Classifier:

- Precision: 1.00 for both classes
- Recall: 1.00 for both classes
- F1-score: 1.00 for both classes
- The Decision Tree Classifier achieves perfect precision, recall, and F1-score for both classes, indicating optimal performance in classifying fake and not fake news.

Gradient Boosting Classifier:

- Precision: 0.99 for class 0, 1.00 for class 1
- Recall: 0.99 for class 0, 1.00 for class 1
- F1-score: 0.99 for class 0, 1.00 for class 1
- The Gradient Boosting Classifier demonstrates high precision, recall, and F1-score for both classes, with a slightly higher performance in classifying not fake news.

Random Forest Classifier:

- Precision: 0.99 for both classes
- Recall: 0.99 for both classes
- F1-score: 0.99 for both classes
- The Random Forest Classifier achieves high precision, recall, and F1-score for both classes, indicating strong performance in classifying fake and not fake news.

Discussion of Results:

- All models show excellent performance with high precision, recall, and F1-scores for both classes.
- The Decision Tree Classifier achieves perfect scores for all evaluation metrics, indicating optimal performance on the given dataset.
- Logistic Regression, Gradient Boosting Classifier, and Random Forest Classifier also demonstrate exceptional performance, with scores close to perfect.
- The choice of the best model depends on various factors such as the specific requirements of the application, interpretability, and computational efficiency.
- Overall, these models provide reliable means of classifying news articles as fake or not fake, with high accuracy and performance.

5. <u>Model Serialization and Reusability</u>

Saving trained models:

- After training the machine learning models (LR, DT, GB, RF), the models can be saved for future use using the pickle library in Python.
- The pickle.dump() function is used to save the trained models to separate files.

- The trained LR model is saved to 'lrModel.sav', DT model to 'dtModel.sav', GB model to 'gbModel.sav', and RF model to 'rfModel.sav'.

Saving vectorization techniques:

- The vectorization technique used for transforming text data into numerical feature vectors can also be saved for reusability.
- The pickle.dump() function is used to save the vectorization object.
- The vectorization object is saved to 'vectorization.sav'.
- Code snippet for saving trained models and vectorization object:

$$pickle.dump(LR, open('lrModel.sav', 'wb'))$$
$$pickle.dump(DT, open('dtModel.sav', 'wb'))$$
$$pickle.dump(GB, open('gbModel.sav', 'wb'))$$
$$pickle.dump(RF, open('rfModel.sav', 'wb'))$$
$$pickle.dump(vectorization, open('vectorization.sav', 'wb'))$$

By using the above code, the LR, DT, GB, RF models, and the vectorization object will be saved as binary files in the current working directory.

The saved models and vectorization object can be loaded in the backend application for making predictions on new data.

6. <u>Backend Application Development</u>

Architecture and design: The backend application is developed using Flask, a lightweight web framework in Python. The application follows a client-server architecture, where the frontend (client) communicates with the backend (server) to make predictions on news articles. The design of the backend application includes the following components:

- Flask: The web framework used for handling HTTP requests and responses.
- Flask-CORS: A Flask extension that enables Cross-Origin Resource Sharing (CORS) to allow communication between different domains (e.g., frontend and backend).
- Machine learning models: Logistic Regression (LR), Decision Tree (DT), Gradient Boosting (GB), and Random Forest (RF) classifiers, which are trained and saved using pickle.
- Text preprocessing: The wordopt() function is used to preprocess the news text by converting it to lowercase, removing special characters, URLs, HTML tags, punctuation, newlines, and digits.
- Vectorization: The TfidfVectorizer is used to transform the preprocessed text data into a numerical feature matrix for machine learning models.
- API endpoint: The '/detect' endpoint is defined to handle POST requests from the frontend and return the predictions for the given news text.

API endpoint for prediction:

- The '/detect' endpoint is configured to accept POST requests.
- The endpoint expects a JSON payload in the request body, containing the news text as the 'text' field.
- The news text is preprocessed using the wordopt() function to clean and normalize it.

- The preprocessed text is transformed into a feature vector using the TfidfVectorizer, matching the vectorization performed during model training.
- The machine learning models (LR, DT, GB, RF) make predictions on the transformed feature vector.
- The predicted labels are converted into human-readable form using the output_label() function.
- The predictions, along with the original news text, are returned as a JSON response.

Integration of machine learning models:

- The trained machine learning models (LR, DT, GB, RF) are loaded using pickle.
- The vectorization is performed using the TfidfVectorizer, which is fitted on the training data and transformed during prediction.
- Error handling and response format:
- Error handling is implemented to catch any exceptions that may occur during the prediction process.
- If an error occurs, a JSON response with an appropriate error message and a 500 status code is returned.
- The predictions, along with the original news text, are formatted into a JSON object and returned as the response.

7. <u>**Node.js Application for Frontend Communication**</u>

Purpose and functionality: The Node.js application serves as a backend server to facilitate communication between the frontend and the Python backend. It handles incoming requests from the frontend, processes them, and communicates with the Python backend to obtain predictions for fake news detection. The main purpose and functionality of the Node.js application are as follows:

- Receive HTTP POST requests from the frontend containing news text data.
- Extract the text data from the request body.
- Send a POST request to the Python backend API, passing the text data for prediction.
- Receive the prediction results from the Python backend.
- Send the prediction results back to the frontend as a response.

Communication with the Python backend:

- The Node.js application uses the Axios library to send HTTP requests to the Python backend.
- The Python API's URL is set as "http://127.0.0.1:5000/detect" and can be modified to the appropriate address.
- When a POST request is received from the frontend, the Node.js application makes a corresponding POST request to the Python backend API with the text data.
- The Node.js application awaits the response from the Python backend and receives the prediction results as JSON data.
- The prediction results are then sent back to the frontend as a JSON response.

Handling user input and requests:

- The Node.js application listens for incoming HTTP requests on a specified port (in this case, port 3000).
- The Express middleware is used to handle JSON parsing and serve static files from the "frontend" directory.
- The "/detect" endpoint is defined to handle POST requests from the frontend.
- When a request is received at the "/detect" endpoint, the text data is extracted from the request body.
- A POST request is made to the Python backend API using the extracted text data.
- The response from the Python backend is received and sent back to the frontend as a JSON response.
- Error handling is implemented to catch any errors that occur during the communication with the Python backend.

Note: Ensure that the Python backend is running and accessible at the specified address (e.g., "http://127.0.0.1:5000/detect") for successful communication between the Node.js application and the Python backend.

8. Frontend User Interface

HTML page structure: The frontend user interface is implemented using HTML, CSS, and JavaScript. The HTML page structure includes the following elements:

- The <head> section contains the necessary <title> tag and the link to the Bootstrap CSS file for styling.
- The <body> section contains a <div> element with the class "container" to provide a responsive layout.
- Inside the container, there is an <h2> heading for the form title and a <form> element to capture user input.
- The form includes a <textarea> input field with the id "text" for users to enter the news text.
- The form also has a submit button for users to submit the form.
- Below the form, there is a <div> element with the id "result" to display the prediction results. Initially, it has the "d-none" class to hide it.

Form design for news input: The form design is implemented using Bootstrap CSS classes. The form has a clean and responsive layout with the following features:

- The form elements are wrapped inside a <div> element with the class "mb-3" to add margin-bottom spacing.
- The <label> element with the class "form-label" is used to label the textarea input field.
- The textarea input field has the class "form-control" to style it as a Bootstrap textarea.

- The textarea has the placeholder "Enter your news..." to provide a hint to the users.
- The textarea has the attribute "rows" set to "4" to display four rows of text.
- The submit button has the class "btn btn-primary" to style it as a Bootstrap primary button.

Interaction with the backend:

- The form submission is handled by the processForm JavaScript function, which is triggered when the form is submitted.
- The event.preventDefault() prevents the default form submission behavior, allowing us to handle the form submission using JavaScript.
- The text input value is retrieved from the textarea element with the id "text".
- The form data is created as a JavaScript object with the text value.
- The form data is sent to the backend by making a POST request to the "/detect" endpoint using the fetch API.
- The response from the backend is received as JSON data.
- The prediction results are dynamically displayed in the <div> element with the id "result" using the innerHTML property.
- The prediction results are formatted using HTML markup and the retrieved data from the response.
- The resultDiv.classList.remove("d-none") is used to remove the "d-none" class from the result <div>, making it visible.
- Note: This code assumes that the frontend application is hosted on the same domain as the backend application, and the backend API is accessible via the "/detect" endpoint.

9. Conclusion

Summary of the Project: The project aimed to develop a machine learning-based system for classifying news articles as fake or not fake. The project utilized four different models: Logistic Regression, Decision Tree Classifier, Gradient Boosting Classifier, and Random Forest Classifier. The models were trained on a dataset consisting of fake and true news articles, and their performance was evaluated using precision, recall, and F1-score metrics. The trained models were integrated into a backend application using Flask, and a frontend application was developed to facilitate user interaction.

Achievements and Limitations:

a. The models achieved excellent performance with high precision, recall, and F1-scores for both classes, indicating their effectiveness in classifying news articles.
b. The backend and frontend applications were successfully developed, enabling users to input news articles and receive predictions on their authenticity.
c. The project successfully demonstrated the use of machine learning techniques for fake news detection and showcased the ability to integrate the models into a practical application.
d. However, there are some limitations to consider:
e. The models were trained and evaluated on a specific dataset, and their performance may vary when applied to different datasets or real-world scenarios.
f. The system relies solely on textual features and does not consider other factors such as source credibility or fact-checking mechanisms.
g. The models may encounter difficulties in handling new or evolving types of fake news that were not present in the training data.
h. The system's performance is dependent on the quality and representativeness of the training data, which may introduce biases or limitations.

Future Enhancements:

- Enhance the system by incorporating additional features, such as source reputation, social media signals, or fact-checking information, to improve the accuracy of the predictions.
- Continuously update and retrain the models using new datasets to adapt to emerging types of fake news.
- Explore advanced natural language processing techniques, such as deep learning models or transformers, to capture more nuanced patterns and improve the models' performance.
- Develop a feedback loop where user feedback on the predictions can be utilized to improve the models over time.
- Expand the system to support multiple languages and international news sources to provide broader coverage and usefulness.
- In conclusion, the project successfully developed a machine learning-based system for fake news detection, showcasing the effectiveness of the trained models. While the system has limitations and areas for improvement, it serves as a foundation for further research and development in the field of fake news detection.

10. References

List of resources and materials used:

- Dataset:
  - Dataset: Fake and real news dataset (https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset)
  - Kaggle: Fake and real news dataset (https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset)
- Libraries and Frameworks:
  - Flask (https://flask.palletsprojects.com/)
  - scikit-learn (https://scikit-learn.org/)
  - Pandas (https://pandas.pydata.org/)
  - NumPy (https://numpy.org/)
- Machine Learning Models:
  - Logistic Regression: scikit-learn documentation (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)
  - Decision Tree Classifier: scikit-learn documentation (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)
  - Gradient Boosting Classifier: scikit-learn documentation (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html)
  - Random Forest Classifier: scikit-learn documentation (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)
- Serialization and Reusability:
  - Pickle: Python documentation (https://docs.python.org/3/library/pickle.html)
- Text Preprocessing and Feature Extraction:
  - Regular Expressions: Python documentation (https://docs.python.org/3/library/re.html)
  - TfidfVectorizer: scikit-learn documentation (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html)
- Flask and API Development:
  - Flask: Flask documentation (https://flask.palletsprojects.com/)
  - Flask-CORS: Flask-CORS documentation (https://flask-cors.readthedocs.io/)