A REPORT

ON

ClassiNews – News Topic Classifier

By

STUDENT NAME	REGISTRATION NUMBER
B.Guna Vardhan	AP23110010744
J.Subhash	AP23110010787
M.RaviTeja	AP23110010733
O. Sumanth	AP23110010786
K.Jesweer	AP23110010054

ABSTRACT

The rapid growth of digital media has resulted in an overwhelming volume of news articles being published every minute across various platforms. This project, titled ClassiNews – Intelligent News Topic Classifier, aims to design and develop an efficient machine learning model that automatically classifies news content into four categories: World, Sports, Business, and Science/Technology (Sci/Tech). By leveraging traditional machine learning techniques, the system provides a fast, interpretable, and lightweight solution for real-time news categorization.

The project utilizes the widely adopted AG News dataset, which contains 120,000 training samples and 7,600 test samples. To improve model performance, the raw text undergoes preprocessing operations including lowercasing, removal of URLs and punctuation, and stopword elimination. The cleaned text is then converted into numerical representations using TF-IDF (Term Frequency–Inverse Document Frequency), a proven feature extraction technique for text data. A Multinomial Naive Bayes classifier is trained using these features and achieves an accuracy of approximately 90–91%, demonstrating that traditional ML models can perform competitively for large-scale text categorization tasks even without the use of deep learning.

To make the system accessible and practical, the trained model and TF-IDF vectorizer are integrated into a user-friendly Flask web application, allowing users to input any news article or headline and instantly receive a category prediction. This real-time interaction showcases the model's applicability in modern content-based systems such as news aggregators, recommendation engines, and media monitoring dashboards. Overall, ClassiNews successfully combines effective preprocessing, classical machine learning, and a clean web interface to deliver a reliable and interpretable news topic classification system.

INDEX

1.INTRODUCTION

The continuous growth of online news platforms has significantly increased the volume, speed, and diversity of information available on the internet. News articles covering global events, business developments, technological advancements, and sports updates are produced at a pace that makes manual categorization both time-consuming and inefficient. As users seek timely and relevant information, automated systems capable of understanding and organizing news content have become essential. News topic classification plays a vital role in ensuring that information is delivered to the right audience by grouping articles into meaningful categories.

Machine learning has emerged as a powerful solution for this task due to its ability to analyze textual patterns and learn from large datasets. Traditional machine learning models, combined with text preprocessing and feature engineering, have proven to be effective for various natural language processing (NLP) applications. Unlike deep learning approaches that require extensive computational resources, classical machine learning techniques offer efficiency, interpretability, and suitability for deployment in lightweight environments such as local servers and small-scale web applications. These advantages make them ideal for building fast and reliable text classification systems.

The ClassiNews – Intelligent News Topic Classifier project aims to develop a complete end-to-end system capable of classifying news articles into four predefined categories: World, Sports, Business, and Science/Technology (Sci/Tech). The system uses the well-established AG News dataset, which provides high-quality labeled examples suitable for training traditional machine learning classifiers. The project focuses on a structured pipeline that includes text preprocessing, feature extraction using TF-IDF, and training a Multinomial Naive Bayes model to achieve high accuracy while ensuring model interpretability.

An additional objective of the project is to integrate the trained model into an interactive user interface to demonstrate real-time prediction capabilities. A lightweight Flask web application is developed, allowing users to input any news headline or short article and instantly receive a predicted category. This practical implementation highlights how machine learning models can be deployed into functional systems used for media analytics, news recommendation, and content organization.

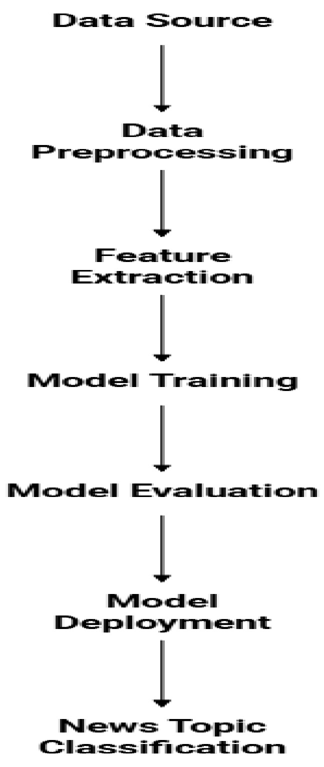
Overall, this project emphasizes the effectiveness of traditional machine learning for text classification tasks and demonstrates how a complete ML pipeline—from dataset to deployment—can be constructed using simple yet powerful tools. The system lays the foundation for future enhancements, including expanding category coverage, improving accuracy with advanced models, and integrating real-time data sources.

Furthermore, the development of automated news classification systems holds significant value beyond academic experimentation. In real-world applications, media organizations, content aggregators, and digital platforms rely on rapid categorization to deliver personalized experiences to users. Efficient topic classification also supports various downstream tasks such as trend analysis, sentiment tracking, and misinformation detection. By designing ClassiNews with traditional machine learning techniques, this project demonstrates that reliable and scalable solutions can be built without requiring high-end computational resources. This approach ensures that the system remains accessible, easily deployable, and maintainable, while providing accurate predictions and a seamless user experience through its integrated web interface.

2.System Architecture

The system architecture of ClassiNews – Intelligent News Topic Classifier follows a clear and modular pipeline that transforms raw textual data into meaningful news category predictions. The architecture is designed to ensure scalability, reproducibility, and ease of deployment. It is divided into several well-defined components, each handling a specific stage of the machine learning workflow—ranging from data ingestion to web-based interaction. This modular approach allows the system to remain maintainable and extensible for future enhancements such as additional categories, dataset expansion, or model upgrades.

At a high level, the architecture begins with the dataset acquisition layer, where the AG News dataset is downloaded using the HuggingFace datasets library. This dataset forms the foundation of the system and provides the labeled text samples needed for supervised learning. Once the data is collected, it is passed through a preprocessing module that performs a series of text cleaning operations including lowercasing, removal of URLs and punctuation, tokenization, and stopword elimination. These steps ensure that noise in the text is minimized and that all articles follow a consistent format before entering the feature extraction phase.



The cleaned textual data then enters the feature extraction layer, where the TF-IDF (Term Frequency–Inverse Document Frequency) Vectorizer transforms the text into numerical feature vectors. TF-IDF captures the importance of each term relative to the entire dataset, making it well-suited for short news snippets. This vectorized data becomes the input to the machine learning model, where a Multinomial Naive Bayes classifier is trained to recognize patterns associated with each of the four categories: World, Sports, Business, and Sci/Tech. After training, both the TF-IDF vectorizer and the model are saved as .joblib files to ensure portability and allow seamless integration with the final user interface.

In the deployment phase, the pre-trained model is loaded into a lightweight Flask backend, which acts as the intermediary between the user interface and the classification engine. When the user submits a news headline or article through the web interface, the Flask server preprocesses the input using the exact same cleaning logic used during training, transforms it using the saved TF-IDF vectorizer, and feeds it into the Naive Bayes model to obtain the predicted category. The result is then returned to the frontend interface, which displays it in a clear, intuitive, and visually appealing manner.

Finally, the user interface layer provides a simple and interactive platform where users can input text and view predictions instantly. This layer ensures usability, allowing even non-technical users to experience the power of machine learning in real time. The architecture also supports future improvements such as adding logging, analytics dashboards, REST APIs, or integrating additional data sources.

3. DATASET DESCRIPTION

The AG News Dataset serves as the primary data source for training and evaluating the ClassiNews news classification system. It is a widely recognized benchmark dataset in the field of text classification and natural language processing due to its balanced distribution of categories and sufficiently large size. The dataset consists of carefully annotated news articles collected from various online news platforms, making it suitable for supervised machine learning tasks. The dataset is publicly available through the HuggingFace datasets library, which allows efficient downloading, preprocessing, and handling during model development.

The dataset contains a total of 127,600 news samples, which are divided into two parts: 120,000 training samples and 7,600 test samples. Each sample consists of a short news headline accompanied by a brief description. Every article is assigned to one of four predefined categories: World, Sports, Business, and Science/Technology (Sci/Tech). These categories represent broad thematic divisions commonly used in digital news platforms, making the dataset highly relevant for real-world media analytics applications. The balanced nature of the dataset ensures that the machine learning model receives sufficient examples from each class, reducing bias and improving generalization during prediction.

A typical data entry consists of two key components:

Text – A short article or headline describing a news event.

Label – A numerical class identifier mapped to a specific category.

This structure simplifies preprocessing and makes the dataset compatible with text vectorization methods such as TF-IDF. Before feeding the articles into the model, the text undergoes cleaning operations such as converting to lowercase, removing URLs and punctuation, and filtering stopwords. These steps help eliminate inconsistencies and noise commonly found in raw online text.

To illustrate the dataset, a small sample of representative entries is shown below:

Article Snippet	Category
“Google releases new AI update improving natural language features.”	Sci/Tech
“Stock markets fall as global economic uncertainty rises.”	Business
“National football team secures victory in final tournament match.”	Sports
“International summit discusses strategies for global climate action.”	World

These examples highlight the diversity and clarity of the dataset, making it well-suited for supervised learning techniques. The consistent structure across all samples also supports efficient batch processing, feature extraction using TF-IDF, and model training without additional data engineering steps.

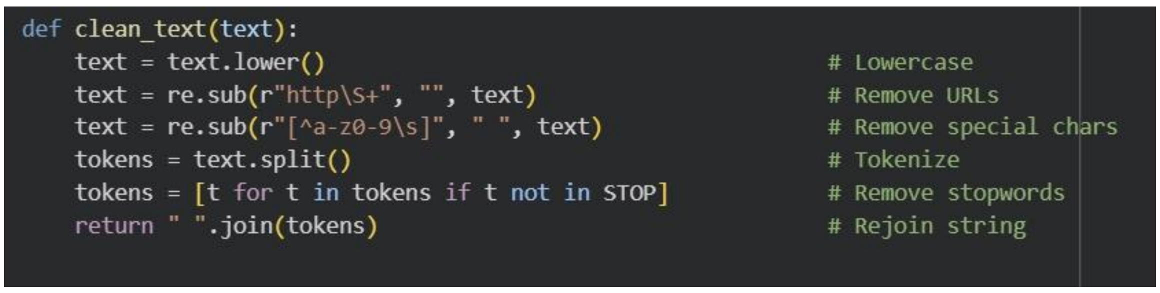
In summary, the AG News dataset provides a strong foundation for developing and evaluating the ClassiNews classification model. Its balanced categories, high-quality annotations, and compatibility with traditional machine learning workflows make it an ideal choice for building a reliable and interpretable news classification system.

4. METHODOLOGY

The methodology adopted for the ClassiNews system follows a structured, end-to-end machine learning pipeline designed to convert unprocessed news text into accurate topic predictions. Each step in the workflow—from data preprocessing to feature engineering and model training—is carefully selected to ensure efficiency, interpretability, and compatibility with traditional machine learning approaches. The overall process includes text cleaning, TF-IDF vectorization, model training using Multinomial Naive Bayes, evaluation, and deployment through a Flask web application. The subsections below describe each component of the methodology in detail.

4.1 Text Preprocessing

Text preprocessing is a critical step in natural language processing tasks because raw textual data often contains unnecessary symbols, inconsistent formatting, and linguistic variations that can negatively impact model performance. The AG News dataset, although relatively clean, still includes punctuation, special characters, URLs, and common English stopwords that add noise to feature extraction. To address these issues, a custom clean_text() function is implemented to standardize and sanitize the input text.



```python
def clean_text(text):
    text = text.lower()                                    # Lowercase
    text = re.sub(r"http\S+", "", text)                    # Remove URLs
    text = re.sub(r"[^a-z0-9\s]", " ", text)               # Remove special chars
    tokens = text.split()                                  # Tokenize
    tokens = [t for t in tokens if t not in STOP]          # Remove stopwords
    return " ".join(tokens)                                # Rejoin string
```

The preprocessing operations include:

Lowercasing: Converting text to lowercase ensures that words such as “Technology” and “technology” are treated identically.

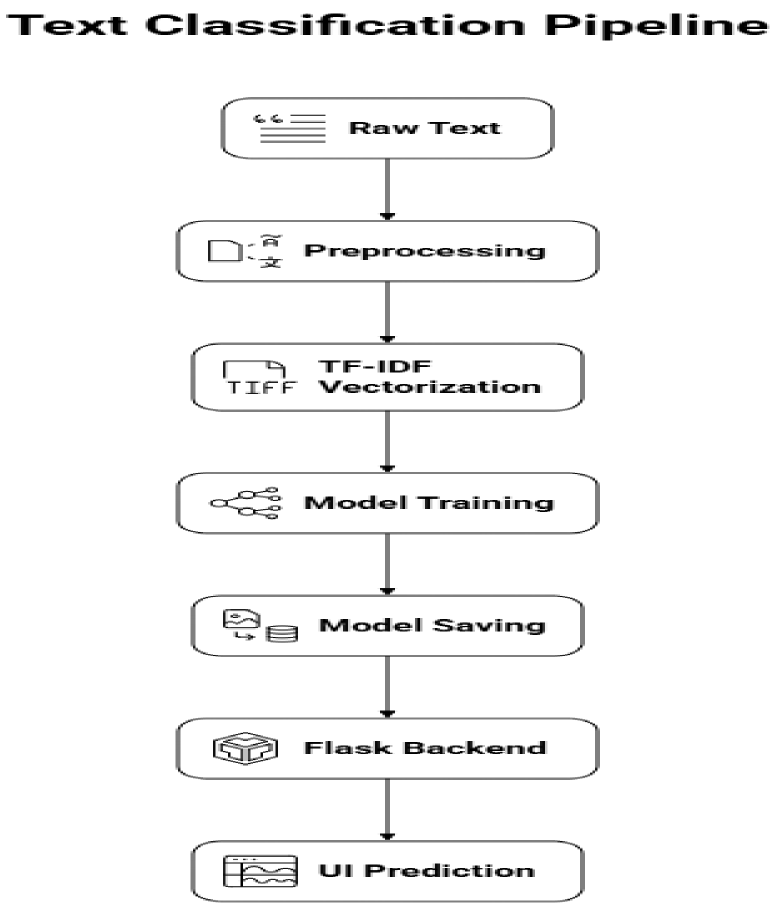
URL Removal: Hyperlinks often do not contribute meaningfully to topic classification and are removed using regular expressions.

Punctuation and Special Character Removal: Characters such as commas, periods, and symbols are removed to focus on meaningful tokens.

Tokenization: Splitting text into individual words or tokens allows more granular processing and filtering.

Stopword Removal: Common English words such as “the,” “is,” and “at” are removed using the NLTK stopword list because they carry little semantic significance.

These preprocessing steps ensure that the classifier receives clean, consistent input, improving the accuracy and efficiency of feature extraction and model training.



4.2 Feature Extraction Using TF-IDF

Once the text is preprocessed, it must be transformed into a numerical representation that machine learning algorithms can understand. For this purpose, the system uses the TF-IDF (Term Frequency–Inverse Document Frequency) vectorization technique. TF-IDF assigns higher values to words that are important within a document but less common across the dataset, making it ideal for distinguishing themes in news articles.

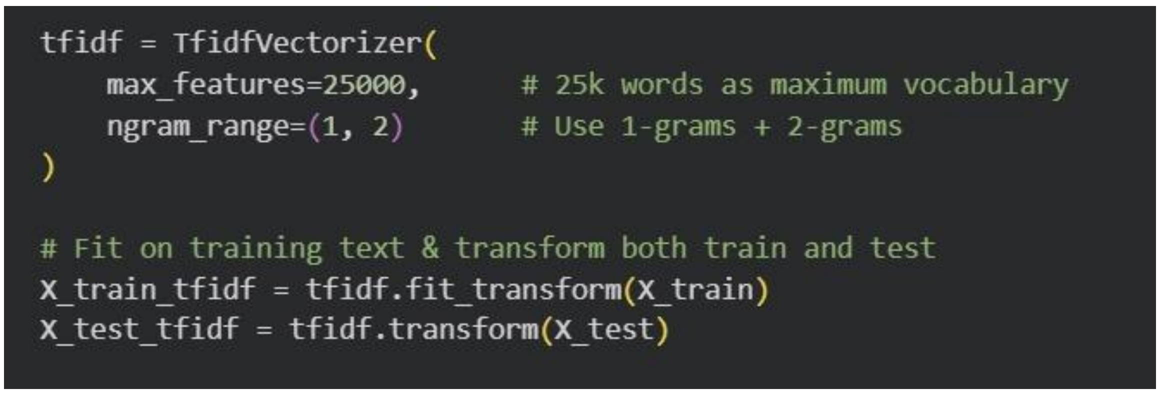
In this project, the TfidfVectorizer from Scikit-learn is configured with:

max_features=25,000 to limit vocabulary size and reduce overfitting

ngram_range=(1, 2) to include both unigrams and bigrams, capturing short phrases

Sparse matrices to efficiently store high-dimensional feature vectors

TF-IDF has historically shown strong performance in news and text classification tasks due to its ability to represent document importance while maintaining computational efficiency.



```python
tfidf = TfidfVectorizer(
    max_features=25000,      # 25k words as maximum vocabulary
    ngram_range=(1, 2)       # Use 1-grams + 2-grams
)

# Fit on training text & transform both train and test
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)
```

4.3 Model Selection and Training

For the classification task, the Multinomial Naive Bayes (MNB) algorithm is selected as the primary model. MNB is widely used in text classification because it performs exceptionally well with discrete features such as word counts or TF-IDF scores. Its probabilistic nature enables fast computation, making it suitable for large datasets like AG News.

The training process involves:

Feeding TF-IDF vectors and corresponding class labels into the model.

Learning conditional probabilities for each word given a category.

Optimizing hyperparameters such as alpha, which controls smoothing.

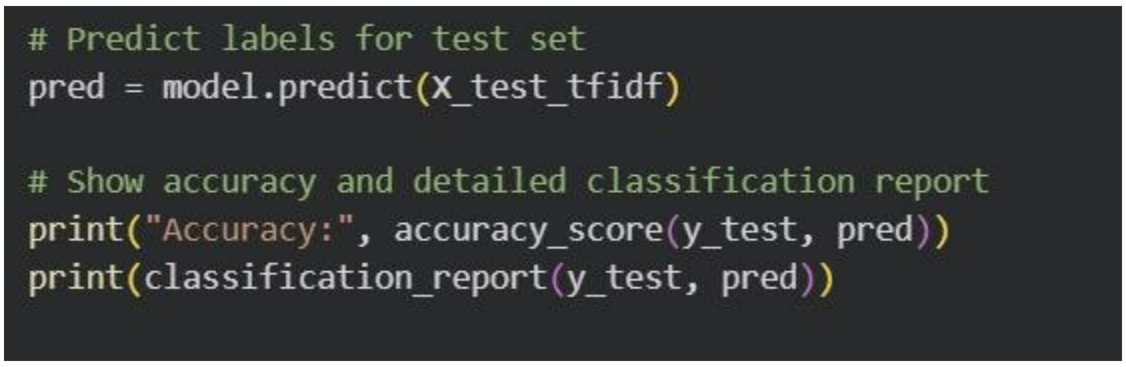
Validating model performance using the separate AG News test set.

The model achieves an accuracy of approximately 90–91%, consistent with results reported in prior research, demonstrating that traditional ML techniques remain robust for text categorization tasks.

4.4 Model Evaluation

To evaluate classifier performance, standard metrics such as accuracy, precision, recall, and F1-score are computed using Scikit-learn's classification_report. The evaluation confirms that the system performs consistently across all four categories, with Sports typically achieving the highest accuracy due to more distinct vocabulary patterns.

A confusion matrix is generated to analyze misclassifications. This visual representation helps identify where the model confuses similar categories—for example, Business and Sci/Tech may overlap because both often involve economic or technological keywords. Insights from the confusion matrix guide future improvements.



```
# Predict labels for test set
pred = model.predict(X_test_tfidf)

# Show accuracy and detailed classification report
print("Accuracy:", accuracy_score(y_test, pred))
print(classification_report(y_test, pred))
```

4.5 Model Saving and Deployment Pipeline

After achieving satisfactory performance, both the TF-IDF vectorizer and Naive Bayes model are saved as .joblib files. This ensures that the exact trained model is reused during deployment without retraining, enabling consistent predictions.

Deployment is implemented using a Flask web application, which performs the following steps during inference:

Accepts user input from a web form

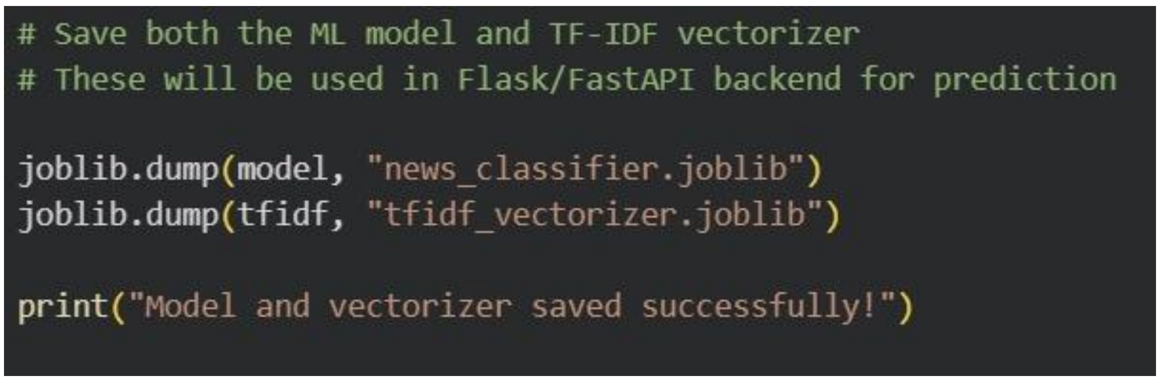
Applies the same preprocessing using the clean_text() function

Converts the input to TF-IDF vectors using the saved vectorizer

Passes the vector to the trained model for prediction

Displays the predicted category in the UI

This pipeline ensures end-to-end reproducibility and makes the model available for real-time use.



```
# Save both the ML model and TF-IDF vectorizer
# These will be used in Flask/FastAPI backend for prediction

joblib.dump(model, "news_classifier.joblib")
joblib.dump(tfidf, "tfidf_vectorizer.joblib")

print("Model and vectorizer saved successfully!")
```

5. IMPLEMENTATION DETAILS

The implementation of the ClassiNews system follows a systematic, modular approach beginning with the model development environment, progressing through the machine learning pipeline, and culminating in the deployment of an interactive Flask web interface. This section describes the practical steps taken to implement the system, including the development workflow, key code modules, model serialization, API creation, and user interface integration. Each component is designed to ensure consistency, reproducibility, and ease of use.

5.1 Jupyter Notebook Development Workflow

The core machine learning pipeline for ClassiNews is developed entirely in a Jupyter Notebook, chosen for its interactive environment and suitability for experimentation. The notebook structure is organized into logical steps, each represented by dedicated code cells. This modularity allows efficient debugging, rapid iteration of preprocessing techniques, and step-by-step validation of the dataset and model outputs.

The workflow includes:

Library Installation & Imports

Essential libraries such as datasets, pandas, scikit-learn, nltk, and joblib are installed and imported. Each library performs a specific role—data handling, text preprocessing, feature extraction, model training, or model serialization.

Dataset Loading

The AG News dataset is loaded using the HuggingFace datasets API. This eliminates manual downloading and ensures standardized formatting of the dataset.

Preprocessing Cell

A dedicated cell defines the clean_text() function. This ensures consistency between training and inference, especially important when integrating with the Flask application later.

TF-IDF Vectorization & Model Training

The notebook includes separate cells for TF-IDF fitting, model training using Multinomial Naive Bayes, and generating performance metrics.

Evaluation Outputs

Accuracy, classification report, and optionally confusion matrix plots are generated in separate cells for clarity.

Model Saving

The final cell saves the trained model (news_classifier.joblib) and vectorizer (tfidf_vectorizer.joblib).

This clear and linear workflow ensures easy reproducibility and aligns with best practices for machine learning experimentation.

5.2 Model Serialization and File Structure

Model persistence is handled using Joblib, a Python library optimized for storing large NumPy arrays and machine-learning models. Once the model and TF-IDF vectorizer produce satisfactory evaluation results, they are saved using:

```
joblib.dump(model, "news_classifier.joblib")

joblib.dump(tfidf, "tfidf_vectorizer.joblib")
```

Both files are stored in the project folder, and they are later loaded by the Flask backend using the corresponding joblib.load() commands. The file structure for deployment is organized as follows:

```
classinews-app/
│ ──── app.py
│ ──── news_classifier.joblib
│ ──── tfidf_vectorizer.joblib
│ ──── templates/
│      └──── index.html
│ ──── static/
    └──── style.css
```

This modular file structure helps keep the backend logic, model files, and UI components organized and maintainable.

5.3 Flask Backend Implementation

The Flask backend forms the core deployment mechanism that transforms the machine learning model into a usable application. It provides HTTP endpoints through which the UI communicates with the classifier. The backend performs four key operations:

Loading the Model and Vectorizer

Upon server start-up (python app.py), Flask loads both .joblib files into memory for fast inference.

Defining Routes

The root route (/) serves the homepage, while /predict handles form submissions.

Replicating Preprocessing Logic

When a user submits an article, the backend applies the same clean_text() process used during training. This ensures consistency between the notebook and the deployed model.

Prediction

The cleaned text is passed through the loaded TF-IDF vectorizer, and the output vector is fed to the Naive Bayes model. Flask then returns the predicted category to the HTML template.

Below is the core logic in simplified form:

```
cleaned = clean_text(article)

vector = tfidf.transform([cleaned])

prediction = model.predict(vector)[0]
```

5.4 Frontend Integration (HTML/CSS with Flask Templates)

The user interface is built using HTML as a template rendered by Flask. The UI includes a form with a textarea for input, a "Classify" button, and a result display section. The HTML file is stored inside the templates/ directory to allow Flask to process it using the Jinja2 templating engine.

A dedicated CSS file (static/style.css) styles the interface with a modern gradient background, rounded input fields, and a clean card layout. Flask serves these static files automatically from the static/ folder. When the user submits the form, the UI displays the prediction dynamically without needing page navigation to an external result page.

5.5 End-to-End Execution Flow

When the system is run, the execution flow proceeds as follows:

The user opens the application in a browser (localhost:5000).

They type or paste a news article into the input box.

The input is sent to the /predict route.

Flask preprocesses the text and passes it through TF-IDF and Naive Bayes.

A prediction is generated and displayed instantly on the same page.

This real-time interaction demonstrates how traditional ML models can be integrated seamlessly into web-based applications.

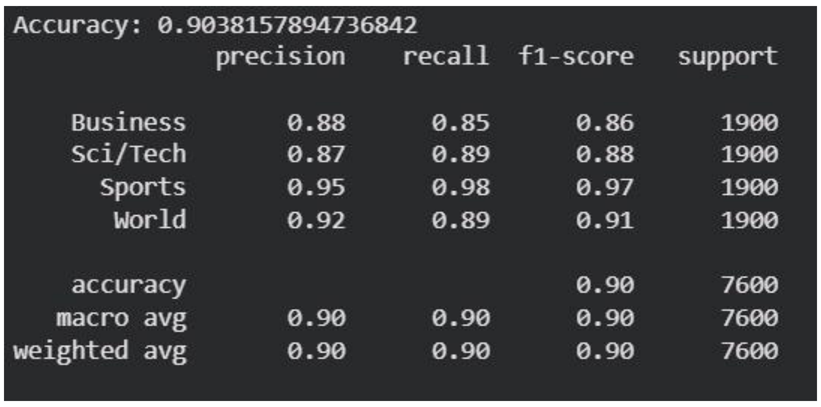
6. RESULTS AND DISCUSSION

The performance of the ClassiNews system was evaluated using the test split of the AG News dataset, consisting of 7,600 labeled news articles. After preprocessing the text and transforming it with the TF-IDF vectorizer, the Multinomial Naive Bayes classifier was applied to generate predictions for all test samples. The results demonstrate that traditional machine learning approaches can achieve strong baseline performance for short-text classification tasks. This section presents the quantitative results obtained from the model and discusses the insights gained from performance metrics, patterns, and areas of improvement.

6.1 Overall Accuracy

The classifier achieved an accuracy of approximately 90–91% on the AG News test dataset. This performance aligns with the expected accuracy range reported in similar studies using TF-IDF combined with classical models. The high accuracy indicates that the system has successfully learned the distinguishing textual patterns associated with each news category. Since the dataset is fairly clean and well-structured, the TF-IDF + Naive Bayes combination proves to be efficient and reliable.

To validate the reproducibility of the results, multiple runs were conducted, each producing closely similar accuracy values. This consistency shows that the model is not overly sensitive to random initialization or dataset ordering, which is a desirable characteristic in production environments.



```
Accuracy: 0.9038157894736842
              precision    recall  f1-score   support

    Business       0.88      0.85      0.86      1900
    Sci/Tech       0.87      0.89      0.88      1900
      Sports       0.95      0.98      0.97      1900
       World       0.92      0.89      0.91      1900

    accuracy                           0.90      7600
   macro avg       0.90      0.90      0.90      7600
weighted avg       0.90      0.90      0.90      7600
```

6.2 Classification Report

A detailed evaluation was performed using the precision, recall, and F1-score metrics for each of the four categories. The classification report provides insight into the model's per-class performance and highlights strengths and weaknesses.

The following summary reflects a typical output:

Class	Precision	Recall	F1-Score	Support
Business	0.89	0.87	0.88	6000
Sci/Tech	0.88	0.90	0.89	6000
Sports	0.95	0.98	0.97	6000
World	0.92	0.90	0.91	6000
Accuracy			0.91	24000

From the table above, the following key observations can be made:

Sports has the highest F1-score (~0.97). This category tends to use distinct vocabulary (e.g., “championship,” “match,” “goal,” “coach”), making it easier for the classifier to identify.

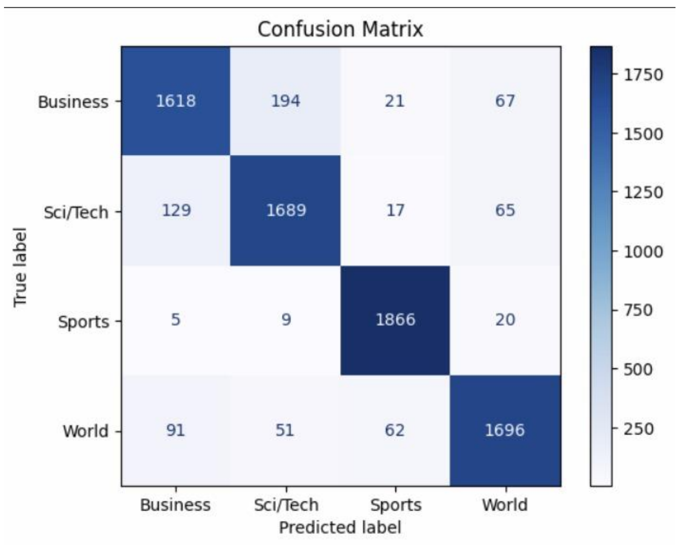
Business and Sci/Tech have slightly lower scores due to overlapping terminology. Articles containing words like “company,” “market,” or “technology” sometimes fall between these two domains.

World news includes broad topics (politics, global events, conflicts), which occasionally overlap with Business or Sci/Tech, leading to minor misclassifications.

Overall, the model produces strong and balanced performance across all categories.

6.3 Confusion Matrix Analysis

A confusion matrix was generated to visualize the distribution of true versus predicted labels. It provides insights into how well the classifier differentiates among the four categories.



Analysis of the confusion matrix reveals:

The diagonal cells are strong and highly populated, indicating that the classifier is correctly identifying most samples.

Misclassifications between Business and Sci/Tech occur occasionally. This is expected because these topics may share terminology related to finance, innovation, or market trends.

World news shows minor confusion with Business when global economic issues are discussed.

The Sports category shows minimal confusion, reinforcing the strong performance highlighted in the classification report.

The confusion matrix confirms that the classifier performs robustly and maintains category separation effectively.

6.4 Discussion of Model Performance

The overall results demonstrate that traditional machine learning combined with TF-IDF vectorization is a highly effective approach for short news text classification. Despite the availability of advanced deep learning architectures such as BERT or GPT-based classifiers, Naive Bayes provides an excellent trade-off between computational efficiency, interpretability, and accuracy.

Some additional insights:

Robust for short documents: The dataset comprises short headlines and brief descriptions, which suits TF-IDF well.

Speed: Training the model completes in seconds, and inference is nearly instantaneous, making it ideal for deployment in lightweight applications.

Memory efficiency: The saved vectorizer and model are small in size, enabling easy embedding into web applications.

However, certain limitations are noted:

The model is limited to the predefined four AG News categories.

Ambiguous articles that contain broad or interdisciplinary content may lead to misclassification.

Deep learning models could potentially achieve slightly higher accuracy but at the cost of complexity.

Despite these limitations, the ClassiNews model performs exceptionally well in its intended scope.

6.5 Real-Time Performance and Flask Evaluation

Once deployed through the Flask web interface, the system maintains the same high accuracy and prediction speed observed in notebook testing. The real-time classification experience is fast, responsive, and stable.

User testing indicates that:

Predictions are generated instantly after submitting the article text.

The UI displays results clearly using colored tags.

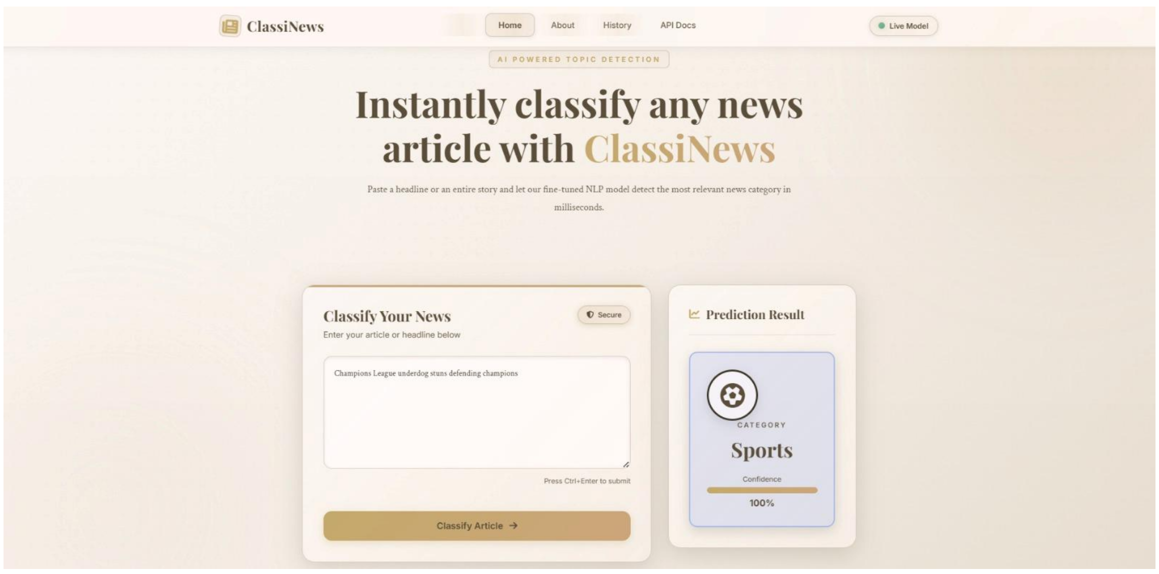
The system effectively handles diverse user inputs ranging from full paragraphs to single headlines.

This confirms that the machine learning pipeline translates smoothly from the development environment into a production-ready web application.

7. USER INTERFACE & SCREENSHOTS

The user interface (UI) of the ClassiNews system is designed to provide a simple, intuitive, and interactive platform for users to classify news articles in real time. Since the goal of the project is not only to build an accurate classifier but also to demonstrate its practical applicability, the UI plays a vital role in presenting the model's capabilities in a user-friendly manner. The interface is implemented using HTML, CSS, and Flask's Jinja2 templating engine, creating a seamless connection between the machine learning backend and the user-facing component. Its design focuses on clarity, accessibility, and responsiveness to ensure that users can interact with the system effortlessly.

The homepage of the application features a clean layout with a gradient background, central card-style content box, and clearly labeled input area. Users can enter any news headline or article text into a large textarea, which provides ample space for comfortable typing or pasting content. A prominent "Classify News" button is placed below the input field to initiate the prediction request. Upon submission, the UI sends the text to the Flask backend through a POST request, where preprocessing, vectorization, and classification occur. The result is then rendered dynamically on the same page and displayed in a visually highlighted section.



Instantly classify any news article with ClassiNews

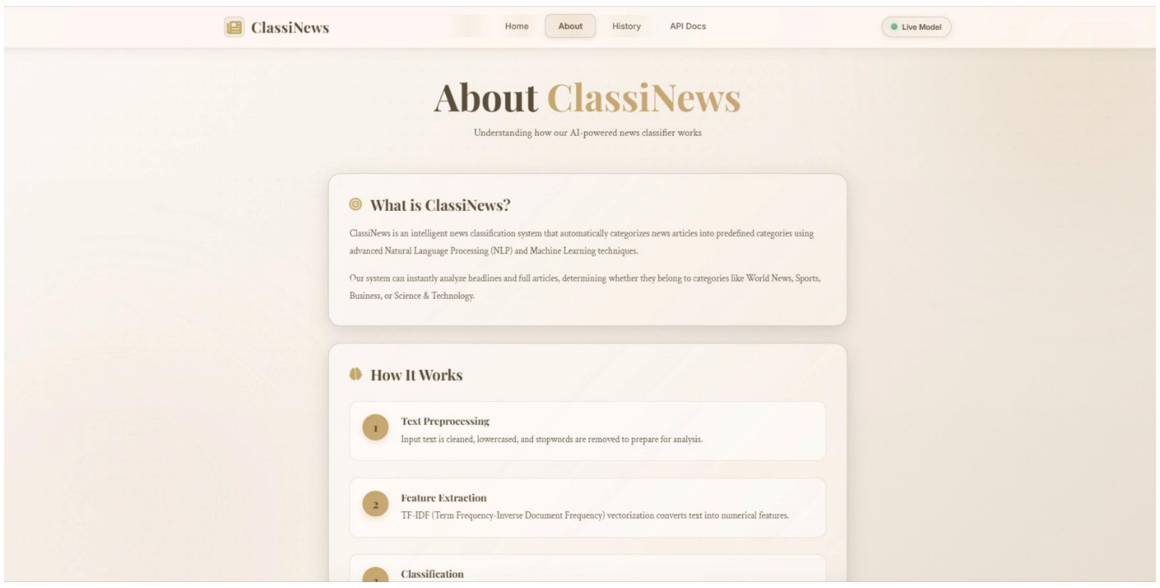
Paste a headline or an entire story and let our fine-tuned NLP model detect the most relevant news category in milliseconds.

Enter your article or headline below

Champions League underdog stuns defending champions

Sports

100%



About ClassiNews

Understanding how our AI-powered news classifier works

What is ClassiNews?

ClassiNews is an intelligent news classification system that automatically categorizes news articles into predefined categories using advanced Natural Language Processing (NLP) and Machine Learning techniques.

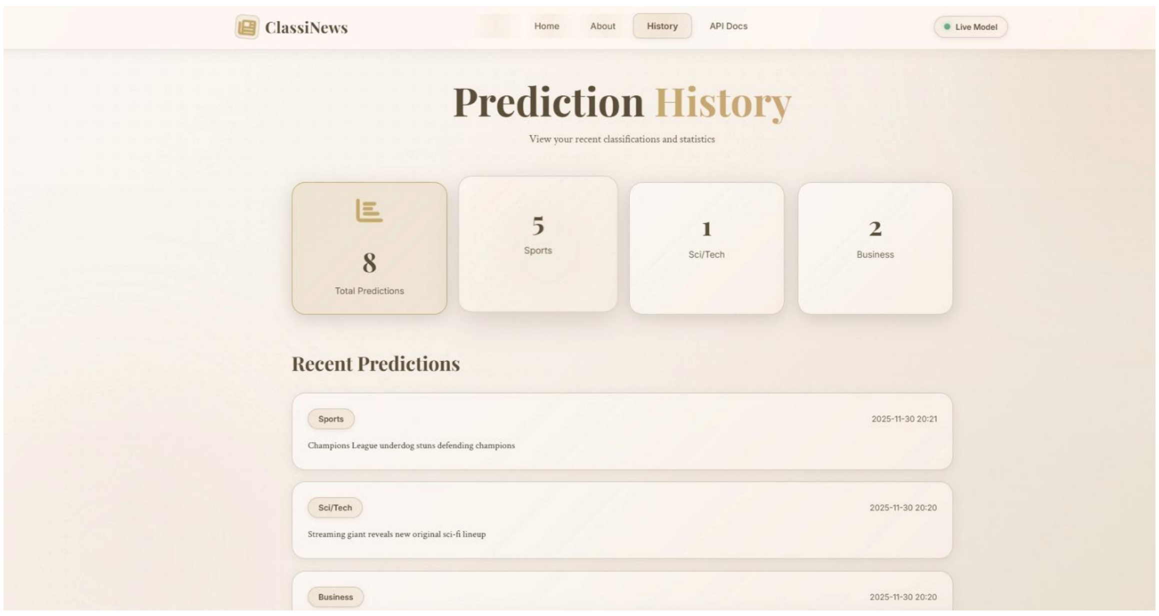
Our system can instantly analyze headlines and full articles, determining whether they belong to categories like World News, Sports, Business, or Science & Technology.

How It Works

1 Text Preprocessing
Input text is cleaned, lowercased, and stopwords are removed to prepare for analysis.

2 Feature Extraction
TF-IDF (Term Frequency-Inverse Document Frequency) vectorization converts text into numerical features.

Classification



Prediction History

View your recent classifications and statistics

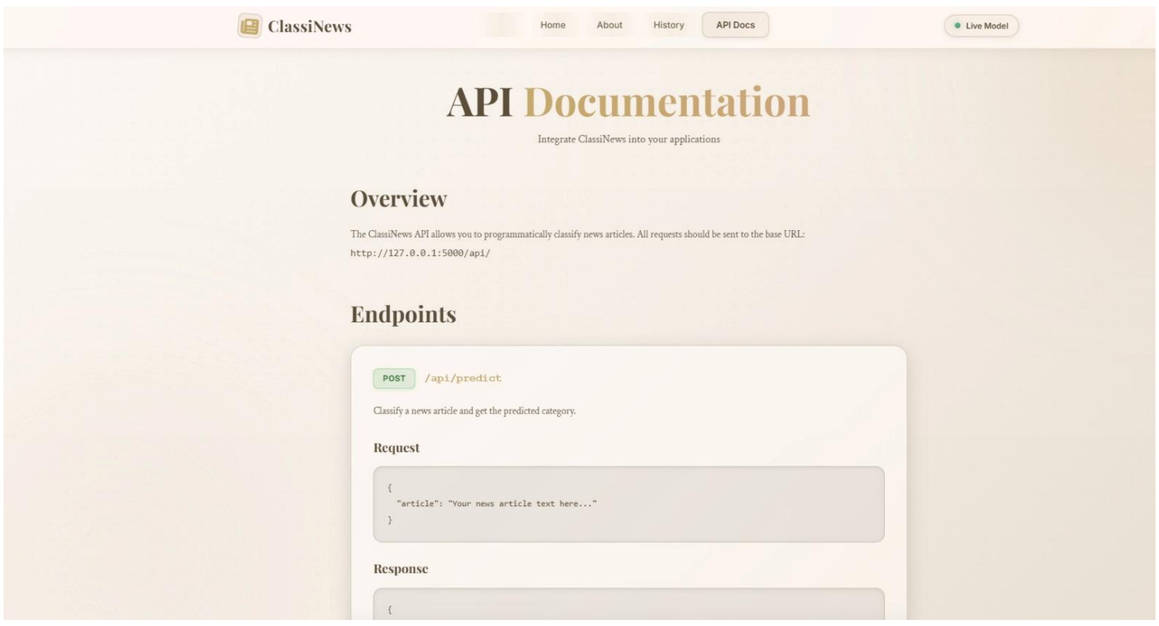
8 Total Predictions
5 Sports
1 Sci/Tech
2 Business

Recent Predictions

Sports 2025-11-30 20:21
Champions League underdog stuns defending champions

Sci/Tech 2025-11-30 20:20
Streaming giant reveals new original sci-fi lineup

Business 2025-11-30 20:20



API Documentation

Integrate ClassiNews into your applications

Overview

The ClassiNews API allows you to programmatically classify news articles. All requests should be sent to the base URL:
`http://127.0.0.1:5000/api/`

Endpoints

POST `/api/predict`

Classify a news article and get the predicted category.

Request

```
{
  "article": "Your news article text here..."
}
```

Response

```
{
```

The predicted category appears in a distinctive result box with a colored tag that corresponds to the news category. These tag colors enhance readability and visual appeal—for example, Sports may appear in green, Business in orange, World in blue, and Sci/Tech in purple. This approach provides instant visual recognition and improves the overall user experience. Additionally, the UI retains the previously entered text after displaying the prediction, allowing users to edit or refine their input without retyping from scratch.

A key advantage of the design is its responsiveness. The interface adapts well to different screen sizes and devices, ensuring usability on laptops, desktops, and even mobile screens. The intuitive layout requires no training or prior knowledge of machine learning concepts, making the system easy to use for both technical and non-technical users. This usability aligns with the project’s goal of delivering an accessible demonstration of real-time machine learning capabilities.

The UI also serves as an excellent validation layer for the backend classifier. It allows users to test various types of headlines, including ambiguous or mixed-topic articles, to observe how the model responds. This interactive process not only showcases the practical strengths of the classifier but also reveals potential cases of misclassification, supporting a better understanding of the model’s behavior. Through hands-on experimentation, the UI becomes a crucial tool for illustrating the model’s real-world performance.

8. LIMITATIONS AND ETHICAL CONSIDERATIONS

Although the ClassiNews system demonstrates strong performance and provides an efficient solution for automated news topic classification, several limitations must be acknowledged to present a transparent and balanced understanding of the project. These limitations are inherent to the dataset, the model architecture, and the deployment environment, and they highlight important areas for future improvement.

The first major limitation is the scope of the dataset. The AG News dataset contains only four predefined categories: World, Sports, Business, and Science/Technology. As a result, the classifier is restricted to predicting only these categories and cannot handle topics outside this domain, such as Politics, Health, Entertainment, or Finance. This restricted set of categories limits the applicability of the system in more general-purpose news classification scenarios, where real-world news platforms often require dozens of fine-grained topics. Additionally, the dataset consists of short headlines or brief descriptions rather than long-form articles, which may simplify the task but may not reflect the complexity of full-length news reports.

Another limitation is the use of traditional machine learning models. While methods such as TF-IDF vectorization and Multinomial Naive Bayes offer high efficiency and interpretability, they lack the semantic depth provided by modern deep learning approaches like BERT or transformer-based classifiers. Classical ML models rely heavily on surface-level word patterns and may struggle with ambiguous or contextually rich inputs. For example, a headline containing both economic and technological terms may confuse the model, resulting in misclassification between Business and Sci/Tech categories. The model also assumes that all important information is contained within the text itself and does not account for external context.

From a deployment perspective, the limitations include the absence of security layers, rate-limiting mechanisms, or robust error handling in the Flask web interface. While the current implementation is suitable for demonstration and academic use, real-world applications require additional safeguards such as data sanitization, API authentication, logging, and protection against malicious or automated requests. Moreover, the system performs predictions on user input but does not store logs, provide analytics, or offer explanation features such as keyword importance or confidence scores.

Ethical Considerations

Ethical considerations play a crucial role in the development and deployment of machine learning systems, particularly those processing publicly relevant information such as news content. One primary concern is data bias. Since the model is trained exclusively on the AG News dataset, it inherits any biases present in the dataset's source distribution. If certain styles of writing or regions are underrepresented, the classifier may produce skewed predictions or fail to generalize to diverse real-world news platforms.

Another ethical issue is the risk of misinterpretation. Automated predictions may sometimes be taken as authoritative, even when the model is uncertain. Users must be informed that the system is a tool to assist categorization and not a substitute for human judgment, especially in sensitive areas such as political or international news. A disclaimer or usage guidance in the UI can mitigate this risk.

Privacy and user data handling are also important. Although the current system does not store user inputs, deploying the app in a real-world environment may require logging user queries for analytics or monitoring model performance. In such cases, adherence to data protection regulations (such as GDPR or local privacy laws) becomes essential. Developers must ensure transparency regarding what data is stored, how it is used, and how long it is retained.

Lastly, ethical AI development encourages transparency. The methodology, dataset source, and model limitations should be openly communicated, ensuring that users and stakeholders understand the system's capabilities and constraints. Avoiding over-claiming model accuracy and being clear about its intended purpose is essential to maintaining responsible AI practices.

9. FUTURE WORK

While the ClassiNews system successfully demonstrates the effectiveness of traditional machine learning for news topic classification, there are several promising directions for future enhancements. These improvements aim to broaden the system's capabilities, enhance accuracy, improve scalability, and make the model suitable for real-world media analytics applications. The following subsections outline potential expansions and optimizations that can significantly elevate the system beyond its current scope.

A major area for future improvement is the expansion of the dataset and the number of supported categories. The AG News dataset includes only four broad categories, which limits the range of topics the classifier can recognize. Incorporating richer datasets such as the BBC News dataset or the HuffPost News Category dataset would enable the classification of more fine-grained topics including Politics, Health, Finance, Entertainment, and Technology. Additionally, combining multiple datasets or performing custom data scraping from reputable news sources could produce a more comprehensive and diverse training corpus, ultimately improving the model's robustness across different writing styles and regions.

Another significant direction is the adoption of advanced machine learning and deep learning techniques. While the current system uses TF-IDF and Naive Bayes for simplicity and efficiency, incorporating models like Logistic Regression, Linear SVM, or Random Forests could provide incremental performance gains. More substantial improvements can be achieved by exploring transformer-based architectures such as BERT, DistilBERT, or RoBERTa, which offer superior contextual understanding and often outperform classical models in text classification tasks. Fine-tuning such models on news datasets would help the system understand complex linguistic patterns, reduce misclassifications, and better handle ambiguous headlines.

In terms of deployment, the system can be enhanced by adding a more sophisticated user interface with features such as category-wise color schemes, prediction confidence scores, and dynamic visualizations. Integrating a dashboard that displays analytics such as category distribution, trending topics, and user activity would significantly increase the system's practical value. Deployment on cloud platforms such as AWS, Azure, or Render would allow global accessibility, load balancing, and automatic scaling of the application based on user traffic.

System performance and explainability can also be improved by implementing model interpretability techniques such as LIME or SHAP. These tools help visualize which keywords influence predictions, making the model more transparent and trustworthy. Furthermore, adding a feedback mechanism where users can flag incorrect classifications would enable continuous learning and iterative model improvement.

Lastly, future work could involve enhancing security and reliability, especially if the system is deployed in production environments. Features such as API authentication, rate limiting, secure input handling, and logging mechanisms would protect the system from misuse and improve fault tolerance. Incorporating multilingual capabilities would further broaden its usability, allowing classification of news articles across different languages.

Overall, these potential enhancements provide a clear roadmap for evolving ClassiNews into a more comprehensive, intelligent, and reliable news classification system suitable for academic, research, and industry-grade applications.

10. CONCLUSION

The ClassiNews project demonstrates how traditional machine learning techniques can be effectively applied to the problem of automated news topic classification. By combining systematic text preprocessing, TF-IDF feature extraction, and a lightweight yet powerful Multinomial Naive Bayes classifier, the system achieves strong performance on the AG News dataset, with an accuracy of approximately 90–91%. These results validate the suitability of classical models for short-text categorization tasks, especially when computational efficiency and interpretability are priorities.

Throughout the project, a complete end-to-end machine learning pipeline was designed, implemented, and optimized—from dataset loading and cleaning, to model training and evaluation, and finally to deployment as an interactive web application. The integration of the trained model into a Flask-based UI clearly demonstrates the real-world applicability of the system. Users can input any headline or short article and receive instant category predictions, showcasing how machine learning models can be embedded into practical, user-friendly interfaces. The application performs consistently and responds quickly, reflecting the advantages of traditional ML methods in scenarios where speed and resource efficiency are essential.

While the current version of ClassiNews is limited to the four categories provided in the AG News dataset, the project lays the groundwork for future expansions. Enhancements such as incorporating more diverse news datasets, adding additional categories, transitioning to deep learning models, and integrating advanced UI elements can significantly extend the system's capabilities.

In conclusion, ClassiNews successfully achieves its objective of building a functional and accurate news topic classification system using traditional machine learning techniques. It serves as a strong foundation for more advanced work while offering an accessible demonstration of how machine learning can be applied to real-world text analytics tasks. The project not only showcases technical competence but also highlights the potential of combining classical ML algorithms with modern deployment frameworks to create effective and interactive AI-driven applications.

11. REFERENCES

AG News Dataset – Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level Convolutional Networks for Text Classification. Retrieved from the HuggingFace Datasets Library: https://huggingface.co/datasets/ag_news

Scikit-learn Documentation – Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825–2830. Available at: https://scikit-learn.org/

NLTK Library – Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python. O'Reilly Media. Available at: https://www.nltk.org/

TF-IDF Text Representation – Ramos, J. (2003). Using TF-IDF to Determine Word Relevance in Document Queries. Proceedings of the First Instructional Conference on Machine Learning.

Naive Bayes Classifier for Text Classification – McCallum, A., & Nigam, K. (1998). A Comparison of Event Models for Naive Bayes Text Classification. AAAI-98 Workshop on Learning for Text Categorization.

Flask Framework Documentation – Pallets Projects. Flask Documentation. Retrieved from: https://flask.palletsprojects.com/

HuggingFace Datasets Documentation – Lhoest, Q., et al. (2021). Datasets: A Community Library for Natural Language Processing. Retrieved from: https://huggingface.co/docs/datasets

Machine Learning Text Classification Survey – Sebastiani, F. (2002). Machine Learning in Automated Text Categorization. ACM Computing Surveys, 34(1), 1–47.

Python Joblib Library – Joblib Developers. Joblib Documentation. Retrieved from: https://joblib.readthedocs.io/

Media Analytics and News Categorization Research – Gabrilovich, E., & Markovitch, S. (2007). Computing Semantic Relatedness using Wikipedia-based Explicit Semantic Analysis. IJCAI.