**Part B : News Article Classification**

**Objective:**

The primary objective of this project is to develop a machine learning-based text classification system that reads the content of a news article and automatically classifies it into predefined categories such as:

* Sports
* Politics
* Business
* Technology
* Health

This approach eliminates the need for manual tagging and enhances the speed, accuracy, and scalability of content categorization.

**Real-World Motivation:**

**Why do we need news classification?**

1. Information Overload: Thousands of news articles are published daily across platforms. Manual categorization is impractical.
2. Personalization: Users want to see articles relevant to their interests (e.g., sports lovers want sports news).
3. Efficient Navigation: Apps, websites, and aggregators can offer better filtering and UI if articles are properly tagged.
4. Content Moderation: Automatic classification helps in flagging or highlighting specific content (e.g., sensitive political topics).

**Dataset Description:**

The dataset used includes:

* **Article**: A string containing the full text of a news article.
* **Category**: The class label for the article (target variable).

It includes thousands of labeled samples for supervised learning.

**Data Preprocessing Steps (Detailed):**

Text data is noisy and unstructured. To prepare it for model training, the following steps were applied:

**🔹 Step 1: Lowercasing**

All text was converted to lowercase to ensure uniformity ("India" and "india" are treated the same).

**🔹 Step 2: Punctuation and Special Character Removal**

Used regular expressions (re.sub) to remove unwanted characters such as: !, @, #, .

**🔹 Step 3: Tokenization**

Breaking down sentences into individual words using nltk.word\_tokenize().

**🔹 Step 4: Stopword Removal**

Common but uninformative words (like "the", "is", "in") were removed using:

from nltk.corpus import stopwords

stopwords.words('english')

**🔹 Step 5: Stemming**

Reduced words to their root form (e.g., "running" → "run") using:

from nltk.stem import PorterStemmer

stemmer = PorterStemmer()

**🔹 Final Output:**

Each article was transformed into a cleaned and stemmed version for vectorization.

**Feature Engineering Using TF-IDF:**

To feed text into machine learning models, it must be converted into numbers.

Why TF-IDF?

* TF (Term Frequency) – How frequently a word appears in a document.
* IDF (Inverse Document Frequency) – Reduces the weight of common words and increases rare but important words.

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(max\_features=5000)

X = tfidf.fit\_transform(processed\_articles)

* Output: A sparse matrix of shape [n\_samples, 5000] representing each article as a 5000-dimensional vector.

Model Training and Algorithms Used:

Split data into training and testing sets:

python

CopyEdit

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

🔸 Algorithms Trained:

1. Logistic Regression – Best performer, simple yet powerful for text classification.
2. Multinomial Naive Bayes – Fast and suitable for high-dimensional text data.
3. Random Forest Classifier – An ensemble method using decision trees.
4. Support Vector Machine (SVM) – High performance but computationally expensive.

**Evaluation Metrics:**

Evaluated using:

* Accuracy – Overall correctness
* Precision – Correct positive predictions / total predicted positives
* Recall – Correct positive predictions / total actual positives
* F1 Score – Harmonic mean of precision and recall

Example Code:

from sklearn.metrics import classification\_report, accuracy\_score, confusion\_matrix

print(classification\_report(y\_test, y\_pred))

Confusion Matrix:

* Shows the actual vs predicted categories.
* Diagonal elements indicate correct predictions.

**Results & Analysis:**

**🔹 Best Model: Logistic Regression**

* Accuracy: **88-89%**
* F1-Score: Consistently high across categories
* Naive Bayes and SVM also performed well.

**🔹 Observations:**

* TF-IDF worked effectively for feature extraction.
* Stemming improved generalization.
* Ensemble methods like Random Forest were slower and overfit in some cases.

**Deployment Ideas:**

* Deploy the model using **Flask or FastAPI** for inference.
* Build a **web app** where users paste news content and get the predicted category.
* Integrate with a **news API** to auto-tag live news feeds

**Future Improvements:**

1. **Use Deep Learning (BERT, LSTM):** Models like BERT understand context better than traditional ML models.
2. **Use Word Embeddings (Word2Vec, GloVe):** Richer representation of words based on context.
3. **Multi-label Classification:** Extend to allow articles to belong to more than one category.
4. **Language Detection and Multilingual Support**

**Final Conclusion:**

**This project successfully demonstrated the power of NLP and machine learning in automating a tedious and critical task — news article classification. With careful preprocessing and model selection, high performance was achieved using classical machine learning models.**