Part B : News Category Classification

Final Report

1. Objective

The aim of this part was to build a machine learning model that can read a short news article (title + description) and classify it into one of **10 categories** like *Politics*, *Sports*, *Business*, *Entertainment*, etc.

Compared to Part A, this was a **multi-class classification problem**, which makes it a little more complex since the model has more than two options to choose from.

2. Understanding the Dataset

The dataset came in a CSV file named `data_news.csv` and had 50,000 rows with 5 columns:

- `category`: The target label
- `headline`: The news title
- `short_description`: A brief summary of the article
- `links` and `keywords`: Not useful for our task

Initial Checks

- There were exactly 5000 articles for each of the 10 categories — so the dataset was perfectly balanced.
- No missing values in the columns we cared about (`headline`, `short_description`, and `category`).
- 2,668 rows had missing keywords — but we didn't use them, so this didn't affect us.

3. Data Preparation

a. Combine headline and short description

I created a new column called **`text`** by combining the `headline` and `short_description`. The idea was that combining both gives the model more context to learn from, while still keeping things short and manageable.

I also removed the `links` and `keywords` columns since they weren't useful for this task.

4. Preprocessing the Text

This step was very similar to what I did in Part A, but still important to repeat for a new dataset. Here's what the preprocessing function did:

- **Removed HTML** using BeautifulSoup\
- **Converted to lowercase**
- **Removed special characters and punctuation** (only kept letters)
- **Tokenized** each sentence into words
- **Removed stopwords**
- **Lemmatized** each word to reduce different forms of the same word

The final cleaned version was stored in a column called **`clean_text`**.

5. Converting Text to Numbers (Feature Extraction)

Again, I used **TF-IDF Vectorization** to convert the cleaned text into numerical values.

6. Encoding the Categories

The `category` column had 10 text labels. Since machine learning models need numbers, I used **Label Encoding** to convert each category into a number between 0 and 9.

This allowed the model to understand that there are 10 different classes.

7. Model Training

Just like in Part A, I used **Logistic Regression**.

Train-Test Split

I split the dataset:

- 80% for training
- 20% for testing

This ensures we're evaluating the model on data it hasn't seen before.

Training the Model

- I increased `max_iter` to 200 to give the model more time to converge since we're dealing with 10 classes.
- Then, I made predictions on the test set and evaluated performance.

8. Results

Accuracy: 79.8%

- It's a multi-class problem
- We didn't use any advanced models or word embeddings

Classification Report Highlights:

Category	Precision	Recall	F1-Score
BUSINESS	0.73	0.78	0.75
ENTERTAINMENT	0.77	0.78	0.78
FOOD & DRINK	0.85	0.76	0.80
POLITICS	0.79	0.80	0.79
SPORTS	0.87	0.86	0.86
TRAVEL	0.83	0.83	0.83
STYLE & BEAUTY	0.86	0.78	0.82

The performance was slightly lower for some classes like **BUSINESS** and **FOOD & DRINK**, and stronger for **SPORTS**, **TRAVEL**, and **STYLE & BEAUTY**.

9. Conclusion

This project gave me solid hands-on experience with multi-class text classification.

- Understanding data before modeling is key
- Clean text always leads to better results
- Even simple models like Logistic Regression can perform surprisingly well