

# **NEWS ARTICLE CLASSIFICATION**

# PROJECT OVERVIEW

- **Objective:** Automatically categorize news articles into predefined categories
- **Key Goals:**
  - Build robust multi-class classifier
  - Preprocess text data and extract meaningful features
  - Evaluate model performance
  - Derive actionable insights

# DATASET OVERVIEW

- **Initial Size:** 50,000 articles
- **Features:**

```
#   Column          Non-Null Count  Dtype
---  -
0   category        50000 non-null    object
1   headline         50000 non-null    object
2   links            50000 non-null    object
3   short_description 50000 non-null    object
4   keywords         47332 non-null    object
dtypes: object(5)
```

- **Categories:** 10 classes including WELLNESS, POLITICS, SPORTS, BUSINESS, etc.
- Target feature: **Category**

# DATA PREPROCESSING

- **Handling Missing Values:** Filled missing keywords with empty strings
- **Duplicate Removal:** Removed 4,251 duplicate articles
- **Link Processing:** Extracted meaningful text from URLs
- **Text Cleaning:**
  - Removed HTML tags, URLs, numbers, special characters
  - Lowercasing and whitespace normalization
  - Tokenization and stopword removal
  - Stemming and Lemmatization

Before:

[https://www.huffingtonpost.com/entry/running-lessons\\_us\\_5b9dc0a4e4b03a1dcc8c72d1](https://www.huffingtonpost.com/entry/running-lessons_us_5b9dc0a4e4b03a1dcc8c72d1)

After:

'running lessons us'

## Link Preprocessing:

```
def clean_links(text):
    text = str(text)
    # Extract everything after '/entry/' if it exists
    text = re.sub(r'.*/entry/', '', text)
    # Replace underscores/hyphens with spaces
    text = re.sub(r'[_\s]', ' ', text)
    # Remove hash-like tokens (letters+digits mixed, usually long)
    text = re.sub(r'\b[a-zA-Z]*\d+[a-zA-Z\d]*\b', '', text)
    # Keep only letters and spaces
    text = re.sub(r'^a-zA-Z\s', '', text)
    # Normalize
    text = text.lower()
    text = ' '.join(text.split())
    return text
```

```
class TextPreprocessor:
```

```
    def __init__(self):
```

```
        self.stop_words = set(stopwords.words('english'))
```

```
        self.stemmer = PorterStemmer()
```

```
        self.lemmatizer = WordNetLemmatizer()
```

```
        self.punctuation = set(string.punctuation)
```

```
    def clean_text(self, text):
```

```
        """Remove HTML, URLs, numbers, special characters,  
        and normalize"""
```

```
        # Remove HTML tags
```

```
        text = re.sub(r'<.*?>', "", text)
```

```
        # Remove URLs
```

```
        text = re.sub(r'http\S+', "", text)
```

```
        # Remove hyphens connecting words
```

```
        text = re.sub(r'(\w)-(\w)', r'\1 \2', text)
```

```
        # Remove numbers & special characters
```

```
        text = re.sub(r'^a-zA-Z\s]', "", text)
```

```
        # Lowercase
```

```
        text = text.lower()
```

```
        # Remove extra spaces
```

```
        text = ' '.join(text.split())
```

```
        return text
```

```
def tokenize_text(self, text):
    return word_tokenize(text)
def remove_stopwords(self, tokens):
    return [t for t in tokens if t not in self.stop_words]
def apply_stemming(self, tokens):
    return [self.stemmer.stem(t) for t in tokens]
def apply_lemmatization(self, tokens):
    return [self.lemmatizer.lemmatize(t) for t in tokens]
```

```
def preprocess_pipeline(self, text, use_stemming=True,
use_lemmatization=True):
    text = self.clean_text(text)
    tokens = self.tokenize_text(text)
    tokens = self.remove_stopwords(tokens)
    if use_stemming:
        tokens = self.apply_stemming(tokens)
    if use_lemmatization:
        tokens = self.apply_lemmatization(tokens)
    return ' '.join(tokens)
```

**headlines:**

**Before:**

'143 Miles in 35 Days: Lessons Learned'

**After:**

'mile day lesson learn'

**short\_description:**

**Before:**

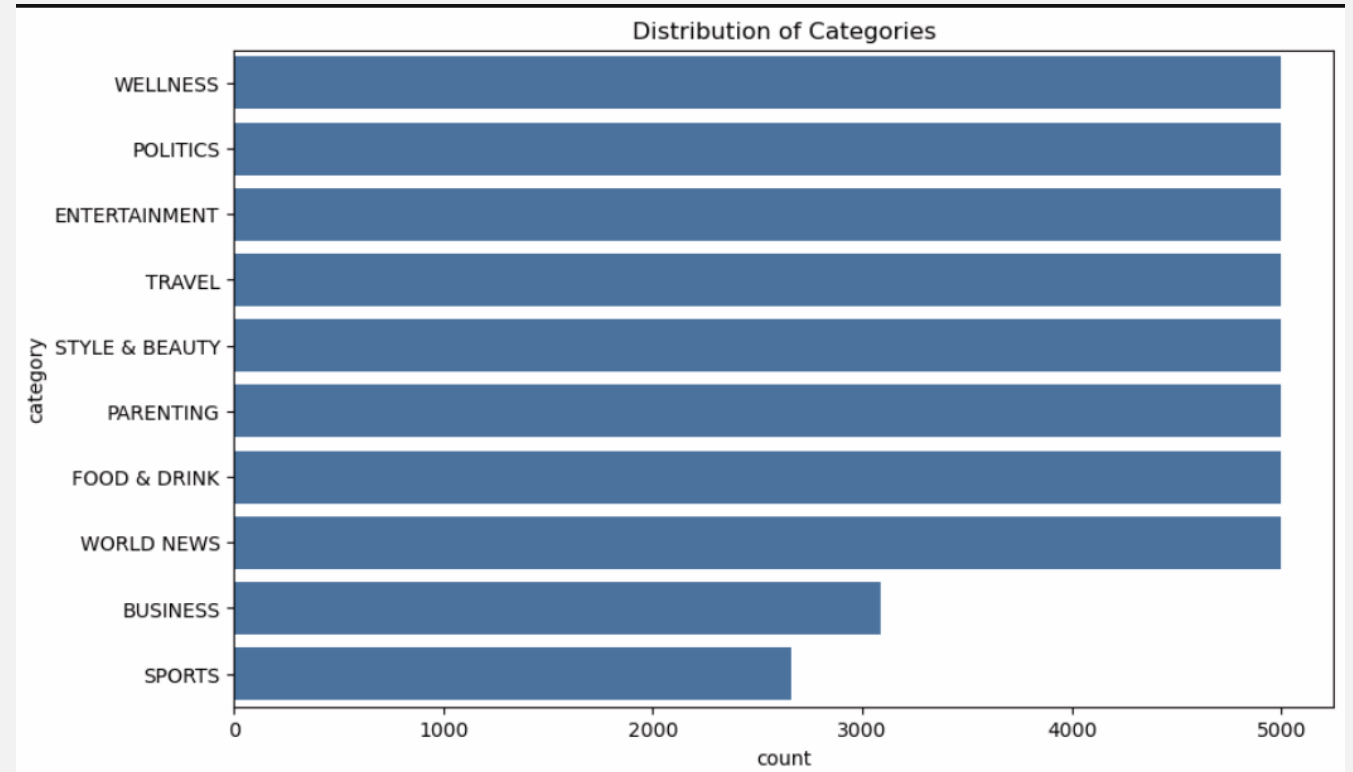
"Resting is part of training. I've confirmed what I sort of already knew: I'm not built for running streaks. I'm built for hard workouts three to five days a week with lots of cross training, physical therapy and foam rolling. But I've also confirmed that I'm stubborn with myself."

**After:**

'rest part train ive confirm sort already knew im built run streak im built hard workout three five day week lot cross train physic therapi foam roll ive also confirm im stubborn'

# EXPLORATORY DATA ANALYSIS (EDA)

- Bar chart showing count per category
- BUSINESS and SPORTS have fewer samples
- **Class Imbalance:** Identified and addressed using SMOTE



# FEATURE EXTRACTION

- **Method Used:** TF-IDF Vectorization
- **Features Extracted From:**
  - Headline (5,000 features)
  - Short Description (5,000 features)
  - Keywords (5,000 features)
- **Total Features:** 15,000 combined TF-IDF features
- **Feature Matrix:** Sparse matrix format for efficiency

```
tfidf_vectorizers = {}  
tfidf_features_list = []  
for col in text_cols:  
    vectorizer = TfidfVectorizer(max_features=5000)  
    # Limit top 5000 words per column  
    tfidf_features = vectorizer.fit_transform(na[col])  
    tfidf_vectorizers[col] = vectorizer  
    tfidf_features_list.append(tfidf_features)
```

# MODELLING & HANDLING CLASS IMBALANCE

```
y = na['category'] #Target column
X_train, X_test, y_train, y_test = train_test_split(X_text, y, test_size=0.2,
random_state=42, stratify=y)

smote = SMOTE(random_state=42)X_train_res, y_train_res =
smote.fit_resample(X_train, y_train)

print("Before SMOTE:", np.bincount(y_train.factorize()[0]))

# original counts

print("After SMOTE:", np.bincount(y_train_res.factorize()[0])) # resampled
counts
```

Before SMOTE: [4000 4000 2126 4000 4000 4000 2473 4000 4000 4000]

After SMOTE: [4000 4000 4000 4000 4000 4000 4000 4000 4000 4000]

# LOGISTIC REGRESSION

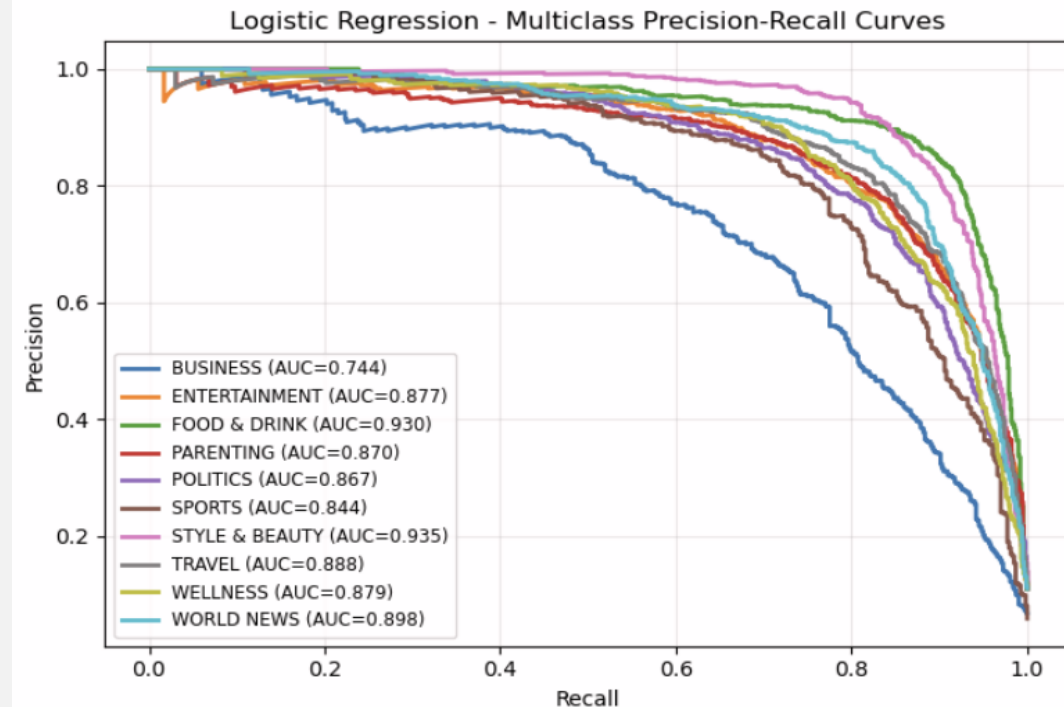
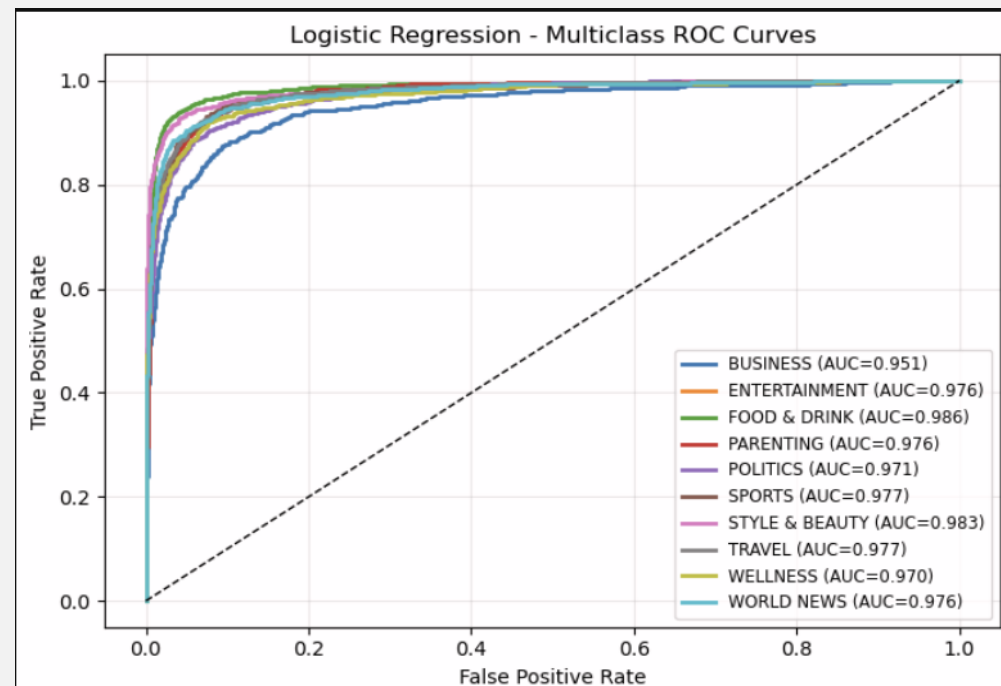
- Best Logistic Regression Parameters: {'C': 1, 'penalty': 'l2', 'solver': 'liblinear'}
- **81% Accuracy & F1-Score**
- Consistent across **8/10** categories
- Excellent in **(Style & Beauty: 0.87 F1)**

Logistic Regression Results:

	precision	recall	f1-score	support
BUSINESS	0.69	0.69	0.69	618
ENTERTAINMENT	0.79	0.83	0.81	1000
FOOD & DRINK	0.86	0.88	0.87	1000
PARENTING	0.79	0.82	0.80	1000
POLITICS	0.81	0.79	0.79	1000
SPORTS	0.79	0.77	0.78	532
STYLE & BEAUTY	0.90	0.84	0.87	1000
TRAVEL	0.82	0.82	0.82	1000
WELLNESS	0.81	0.80	0.80	1000
WORLD NEWS	0.84	0.84	0.84	1000
accuracy			0.81	9150
macro avg	0.81	0.81	0.81	9150
weighted avg	0.81	0.81	0.81	9150

Confusion Matrix:

```
[[429 25 10 23 40 7 9 25 33 17]
 [ 10 826 11 30 29 26 25 17 16 10]
 [  5 15 882 12  8  7 12 33 24  2]
 [ 24 35 16 816 13 11 13 23 43  6]
 [ 47 29  3 16 785 14  3 21 17 65]
 [  9 44  5  9 14 408  6  8  9 20]
 [ 18 34 14 26  7  8 841 24 22  6]
 [ 19 16 37 25 18 14 12 819 15 25]
 [ 30 11 44 63 15 11  8 13 798  7]
 [ 30 15  5 12 46 12  2 21 14 843]]
```



# NAIVE BAYES (MULTINOMIAL)

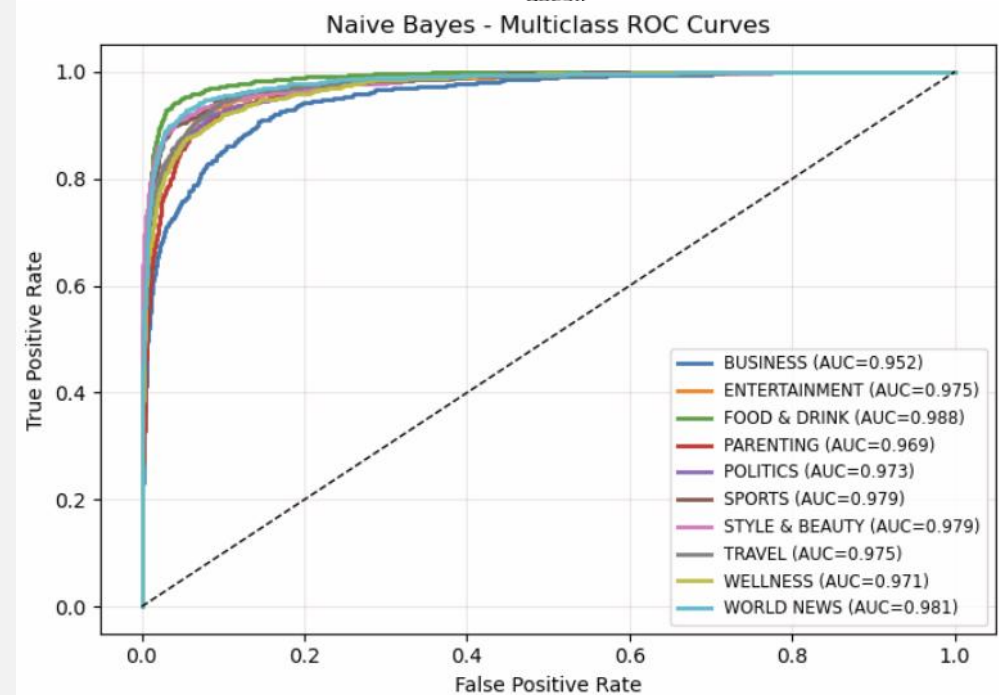
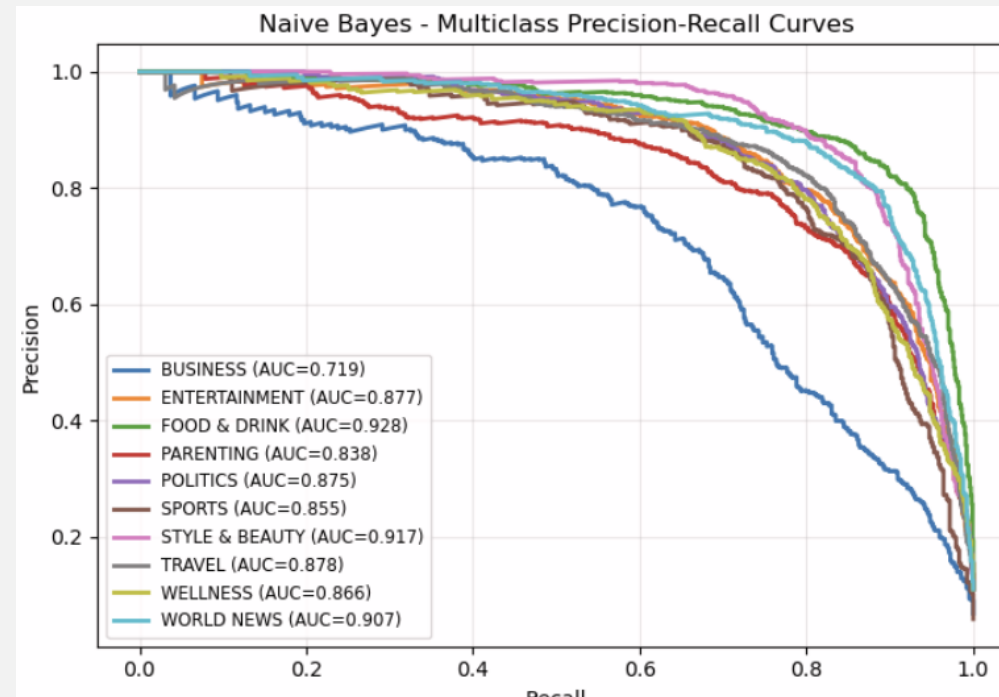
- Strong in Food & Drink (0.87 FI) and World News (0.84 FI)
- Competitive with Logistic Regression
- 80% Accuracy & FI-Score
- **Weaker Areas:** Business (0.67 FI), Parenting (0.76 FI)
- Minor confusion between similar topics (Business ↔ Politics)
- Fast training and prediction

Naive Bayes Results:

	precision	recall	f1-score	support
BUSINESS	0.66	0.67	0.67	618
ENTERTAINMENT	0.82	0.78	0.80	1000
FOOD & DRINK	0.86	0.87	0.87	1000
PARENTING	0.71	0.81	0.76	1000
POLITICS	0.81	0.78	0.79	1000
SPORTS	0.80	0.77	0.79	532
STYLE & BEAUTY	0.87	0.83	0.85	1000
TRAVEL	0.79	0.81	0.80	1000
WELLNESS	0.80	0.79	0.79	1000
WORLD NEWS	0.85	0.84	0.84	1000
accuracy			0.80	9150
macro avg	0.80	0.79	0.80	9150
weighted avg	0.80	0.80	0.80	9150

Confusion Matrix:

```
[[416 11 10 50 42 6 7 20 38 18]
 [ 16 776 9 44 29 22 53 28 14 9]
 [ 5 11 867 27 5 4 11 46 24 0]
 [ 22 30 15 811 13 12 19 26 49 3]
 [ 66 17 3 24 775 17 3 16 11 68]
 [ 11 41 3 18 11 412 5 8 7 16]
 [ 13 28 16 41 4 5 828 29 31 5]
 [ 17 20 30 40 15 17 9 812 15 25]
 [ 28 4 47 70 13 12 10 18 792 6]
 [ 39 9 4 16 49 8 4 21 13 837]]
```



# SUPPORT VECTOR MACHINE (SVM)

## Performance:

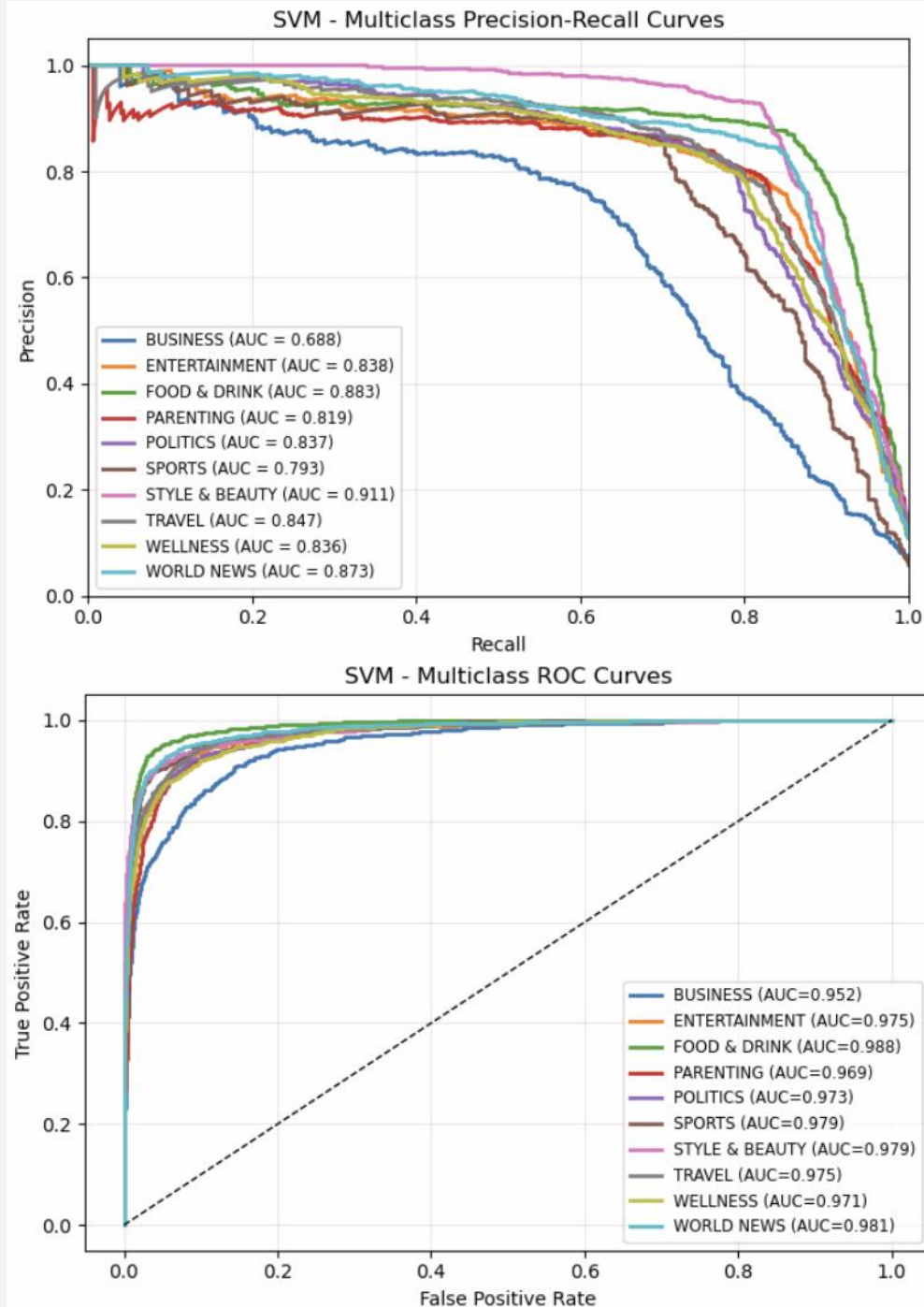
- **Accuracy:** 0.81
- **Macro F1-score:** 0.80
- **Weighted F1-score:** 0.80

## Observations:

- **Highest precision:**  
STYLE & BEAUTY (0.92), FOOD & DRINK (0.86)
- **Strong recall:**  
ENTERTAINMENT (0.85), WORLD NEWS (0.85)
- **Confusion between related categories:**  
BUSINESS & POLITICS, TRAVEL & STYLE & BEAUTY

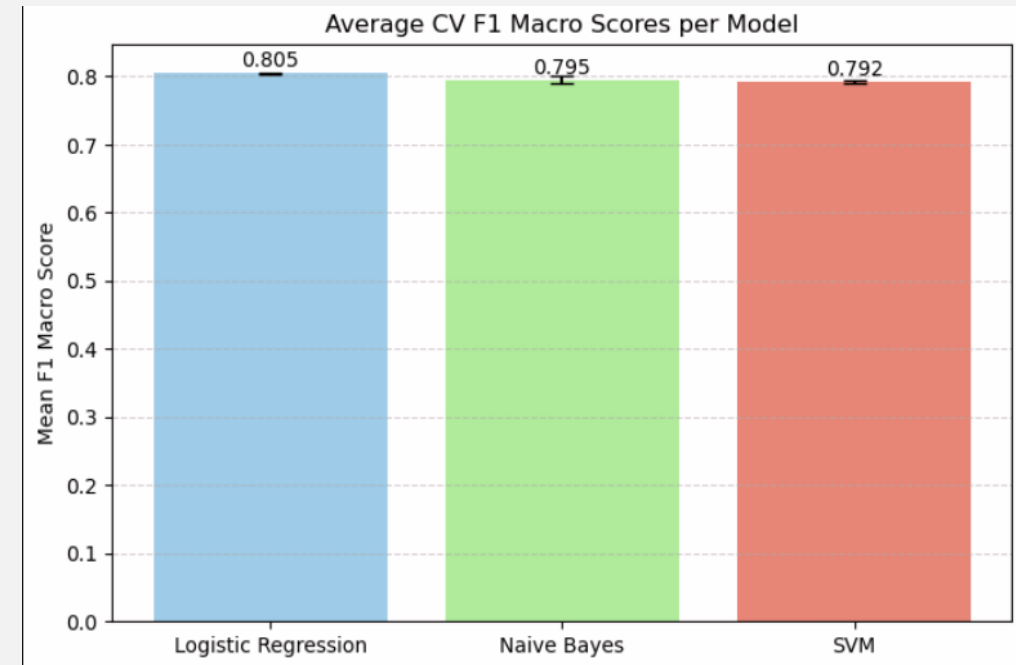
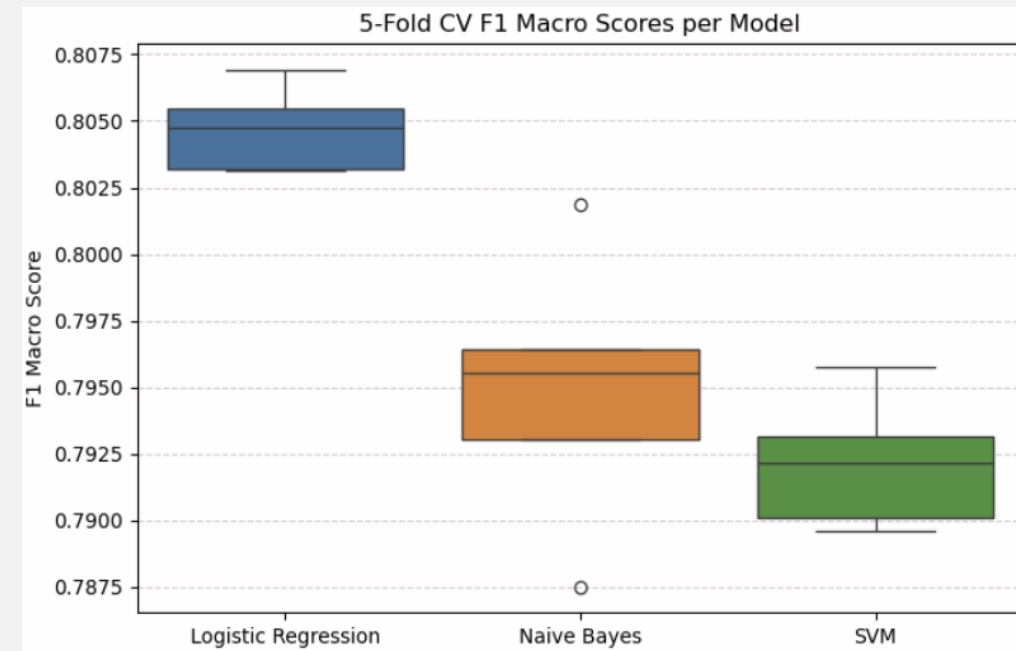
SVM Results:				
	precision	recall	f1-score	support
BUSINESS	0.73	0.63	0.68	618
ENTERTAINMENT	0.74	0.85	0.79	1000
FOOD & DRINK	0.86	0.86	0.86	1000
PARENTING	0.78	0.82	0.80	1000
POLITICS	0.78	0.79	0.79	1000
SPORTS	0.85	0.70	0.77	532
STYLE & BEAUTY	0.92	0.82	0.87	1000
TRAVEL	0.78	0.81	0.79	1000
WELLNESS	0.78	0.80	0.79	1000
WORLD NEWS	0.83	0.85	0.84	1000
accuracy			0.81	9150
macro avg	0.81	0.79	0.80	9150
weighted avg	0.81	0.81	0.80	9150

Confusion Matrix:										
[	387	31	11	26	58	3	6	24	47	25]
[	10	854	9	27	30	13	15	20	10	12]
[	5	20	861	16	8	4	13	42	27	4]
[	17	35	15	823	17	8	10	27	40	8]
[	30	29	4	22	794	8	3	23	21	66]
[	8	76	6	8	19	372	3	11	10	19]
[	12	44	12	35	9	4	822	26	31	5]
[	16	25	37	25	20	10	10	806	20	31]
[	21	16	41	57	19	8	6	27	797	8]
[	22	20	4	12	41	7	2	24	15	853]]



# CROSS-VALIDATIONS

- Logistic Regression 5-Fold CV F1 Macro Scores: [0.80315809 0.80692264 0.80318348 0.80477424 0.8054712 ]
  - Logistic Regression Mean CV F1 Macro Score: **0.8047**
- Naive Bayes 5-Fold CV F1 Macro Scores: [0.78750686 0.79551489 0.79305324 0.80187797 0.79643708]
  - Naive Bayes Mean CV F1 Macro Score: **0.7949**
- SVM 5-Fold CV F1 Macro Scores: [0.79009784 0.79217083 0.78962214 0.79575064 0.79317024]
  - SVM Mean CV F1 Macro Score: **0.7922**



## OBSERVATIONS:

Model	Accuracy	Macro F1	Observations
<b>Logistic Regression</b>	<b>0.81</b>	<b>0.805</b>	Consistently strong across classes, best average F1, balanced performance.
<b>Naive Bayes</b>	0.80	0.795	Slightly lower overall, performs well on FOOD & DRINK and ENTERTAINMENT but weaker on BUSINESS.
<b>SVM</b>	0.81	0.792	Accuracy similar to Logistic Regression, but slightly lower macro F1, some classes like SPORTS have lower recall.

### Conclusion:

- **Logistic Regression** is the best choice for your multiclass prediction task.
- It has the highest **mean CV F1 macro** (0.8047),
- Balanced precision/recall across categories, and consistent 5-fold performance.

THE END

Video link: <https://drive.google.com/file/d/1vEfIXcUDxKKs4l0TCbGcmhBnv0pjqDE9/view?usp=sharing>