Problem 4: Sport vs Politics Text Classifier

NLU Assignment 1 — B23CS1008

Bhargav Shekokar
B23CS1008
Indian Institute of Technology Jodhpur

February 15, 2026

Contents

1 Introduction

In this assignment, I built a simple but complete text classification system that can distinguish between two types of news sentences: **SPORT** and **POLITICS**.

Instead of using ready-made libraries like scikit-learn, I implemented everything from scratch to properly understand how each algorithm works.

The goal was not just to get high accuracy, but to understand:

- How different feature representations affect performance
- Why some models generalise better than others
- What overfitting looks like in practice
- How dataset size influences model behaviour

I experimented with three classifiers: Naive Bayes, Logistic Regression, and K-Nearest Neighbours, combined with three feature extraction methods: Bag-of-Words, TF-IDF, and Word Bigrams.

2 Dataset

- **Sources:** `sport.txt` (70 sentences) and `politics.txt` (70 sentences).
- Each file contains one sentence per line covering Indian sports and political topics.
- **Total samples:** 140 (*Train:* 110, *Test:* 28, split ratio 80/20, seed=42).
- Class distribution is balanced: 50% SPORT, 50% POLITICS.

Topic coverage.

Sport
: Cricket (IPL, ICC, Tests), Olympics, football (ISL, I-League), badminton, boxing, hockey, athletics, wrestling.

Politics
: Parliament (Lok Sabha, Rajya Sabha), elections, party politics (BJP, Congress, AAP, TMC, DMK), Supreme Court, government schemes, state-level politics.

3 Project Structure

```
NLP_Classifier/
 main.py            # Entry point (--auto or interactive menu)
 load_data.py       # Dataset loading & train/test splitting
 preprocess.py      # Tokenisation, NLTK stopwords, feature extractors
 train_models.py    # NB, LR, KNN classifiers (from scratch)
 evaluate.py        # Accuracy, Precision, Recall, F1, confusion matrix
 visualize.py       # Matplotlib charts → output/
 sport.txt          # Dataset
```

```
politics.txt       # Dataset
README.md
output/            # Generated charts (created on first run)
```

4 Methodology

4.1 Text Preprocessing

Before feeding sentences into any model, I cleaned and simplified the text. This step is very important because raw text contains noise that can confuse models.

Each sentence goes through:

1. Converting everything to lowercase so that "Parliament" and "parliament" are treated the same.
2. Removing punctuation since symbols do not help in this task.
3. Splitting the sentence into individual words (tokenisation).
4. Removing common stopwords like "the", "is", "and" using NLTK's English stopword list.

Removing stopwords was especially useful because they appear frequently in both SPORT and POLITICS sentences and do not help in distinguishing between the two classes.

4.2 Feature Extraction

After preprocessing, the text needs to be converted into numerical vectors so that machine learning algorithms can process it.

I experimented with three different representations:

1. Bag-of-Words (BoW) This is the simplest method. It counts how many times each word appears in a sentence. While simple, it ignores how important a word is across the whole dataset.

2. TF-IDF TF-IDF improves over BoW by reducing the weight of very common words and giving higher importance to words that are more unique to a class. For example, words like "wicket" or "parliament" become strong signals for SPORT and POLITICS respectively.

In my experiments, TF-IDF clearly performed better than plain counts.

3. Word Bigrams Instead of single words, this method looks at pairs of consecutive words. The idea is to capture short phrases like "Lok Sabha" or "World Cup".

However, since the dataset is small, most bigrams appear only once, making the feature space very sparse. This hurt performance.

4.3 Classifiers

I implemented all classifiers from scratch to understand their internal working instead of treating them as black boxes.

4.3.1 Multinomial Naive Bayes

Naive Bayes assumes that words occur independently given the class. Although this assumption is unrealistic, it often works surprisingly well for text classification.

It calculates:

$$P(c \mid \mathbf{x}) \propto P(c) \prod_j P(x_j \mid c)$$

Laplace smoothing was added to avoid zero probabilities.

I noticed that Naive Bayes tends to perform well when features are frequency-based and when the dataset is not too small.

4.3.2 Logistic Regression

Logistic Regression is a discriminative model. Instead of modelling probabilities of words per class, it directly learns a decision boundary between SPORT and POLITICS.

Weights are updated using gradient descent to minimise cross-entropy loss.

In my results, Logistic Regression performed the best overall, especially with TF-IDF features. However, it sometimes showed signs of overfitting due to high training accuracy.

4.3.3 K-Nearest Neighbours (KNN)

KNN is a lazy learning algorithm. It does not learn a model during training. Instead, it stores the training data and compares distances during prediction.

I used Euclidean distance and $k = 5$.

KNN worked well with TF-IDF (where vectors are normalised), but struggled with raw BoW features due to high dimensionality and noisy distances.

5 Results

5.1 Overall Comparison

Table 1 summarises train/test accuracy for all 9 combinations.

Table 1: Train and test accuracy for all feature–classifier combinations.

Feature	Classifier	Train Acc (%)	Test Acc (%)
BoW	Naive Bayes	99.09	82.14
BoW	Logistic Regression	99.09	85.71
BoW	KNN	69.09	64.29
TF-IDF	Naive Bayes	99.09	82.14
TF-IDF	Logistic Regression	**100.00**	**89.29**
TF-IDF	KNN	95.45	**89.29**
Bigram	Naive Bayes	86.36	60.71
Bigram	Logistic Regression	93.64	64.29
Bigram	KNN	61.82	60.71

Best combination: TF-IDF + Logistic Regression and TF-IDF + KNN (both **89.29%** test accuracy).

5.2 Per-Class Precision, Recall, and F1

Table 2 reports per-class metrics on the test set for the top models.

Table 2: Per-class metrics for selected model combinations (test set).

		POLITICS			SPORT		
Feature	**Classifier**	P	R	F1	P	R	F1
TF-IDF	Log. Reg.	1.00	0.80	0.89	0.81	1.00	0.90
TF-IDF	KNN	0.93	0.87	0.90	0.86	0.92	0.89
BoW	Log. Reg.	0.82	0.93	0.88	0.91	0.77	0.83
BoW	Naive Bayes	1.00	0.67	0.80	0.72	1.00	0.84

5.3 Visualizations

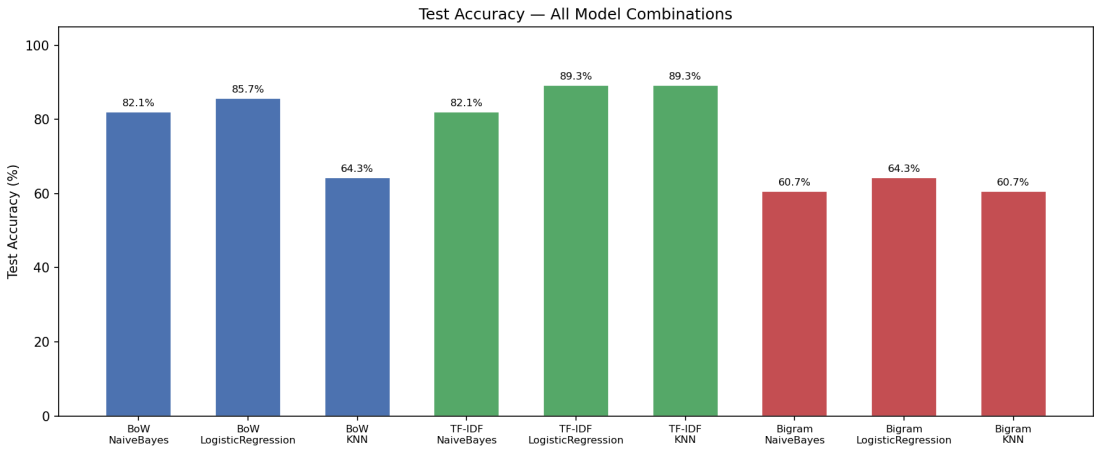

Figure 1: Test accuracy for all 9 model combinations.

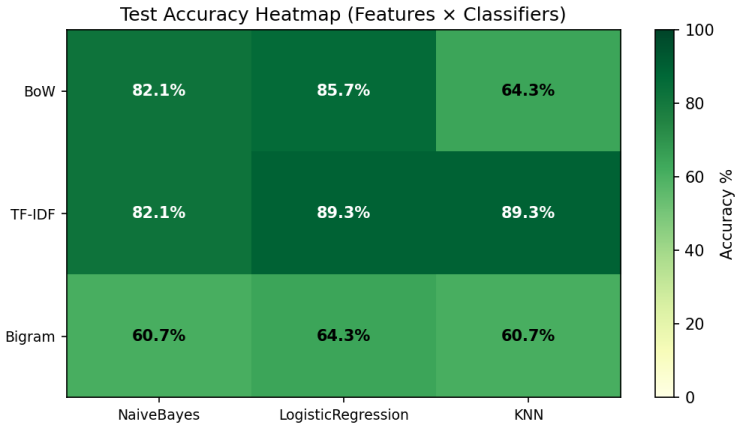

Figure 2: Accuracy heatmap — features (rows) vs classifiers (columns).

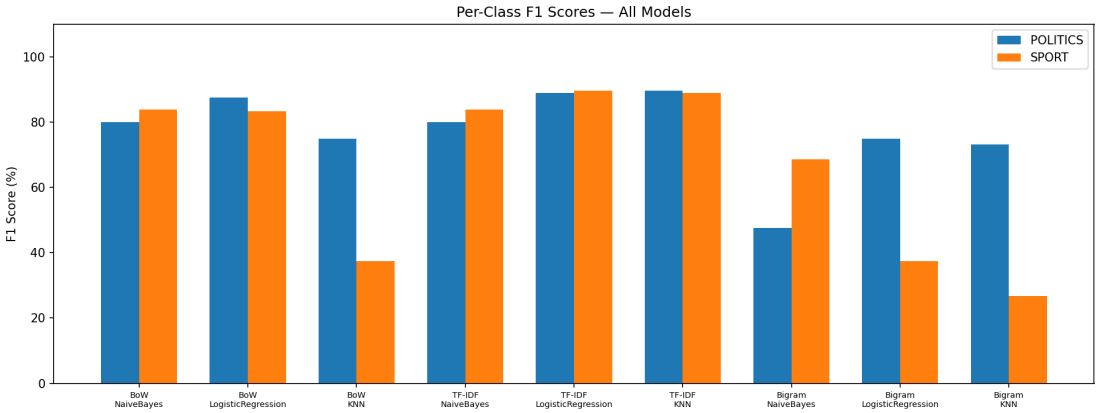

Figure 3: Per-class F1 scores across all model combinations.

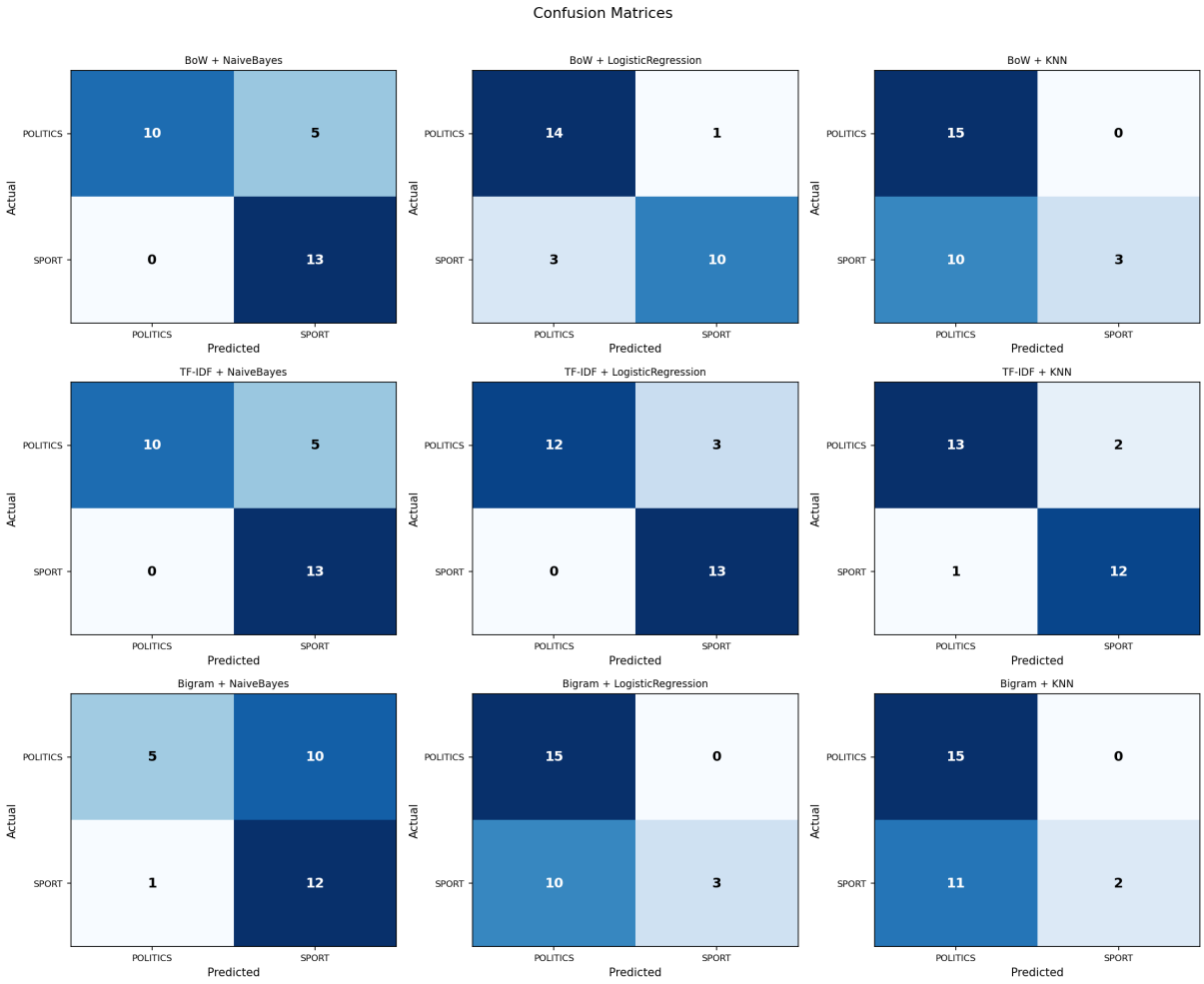

Figure 4: Confusion matrices for each feature–classifier pair.

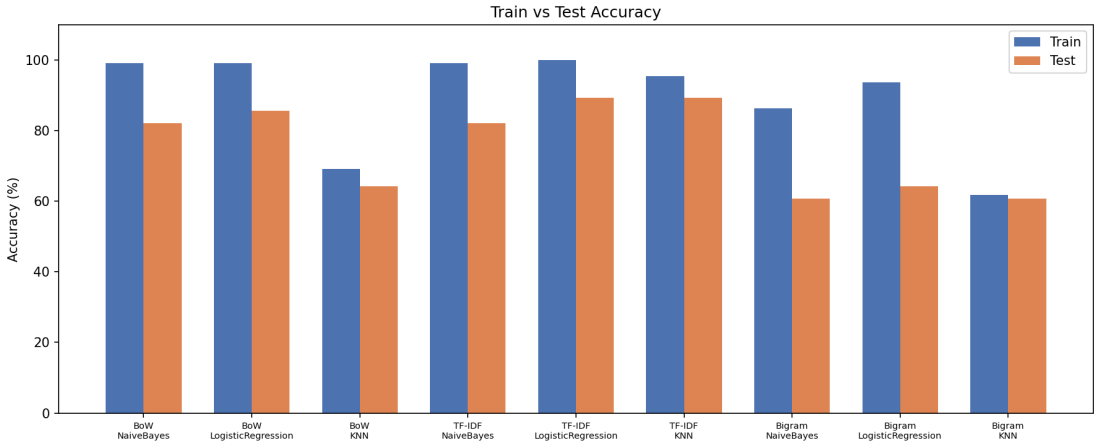

Figure 5: Train vs test accuracy comparison. Note overfitting in Bigram + LR.

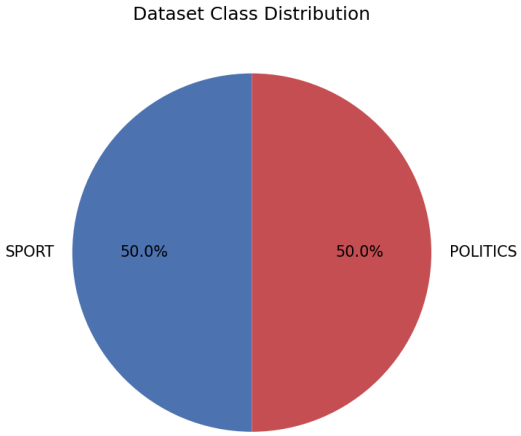

Figure 6: Class distribution in the full dataset (balanced 50–50).

6 Analysis & Discussion

After running all 9 combinations, a few important patterns became clear:

1. **TF-IDF clearly performs best.**

   TF-IDF consistently outperformed Bag-of-Words and Bigrams. This makes sense because TF-IDF reduces the effect of common words and highlights class-specific terms. In small datasets like this, that weighting becomes very important.

2. **Logistic Regression generalises well.**

   Logistic Regression with TF-IDF achieved the highest test accuracy (89.29%). It learned a strong linear boundary between SPORT and POLITICS. However, its 100% training accuracy indicates mild overfitting.

3. **KNN benefits from normalised features.**

   KNN performed poorly with raw counts but improved significantly with TF-IDF. Since TF-IDF includes normalisation, distance comparisons become more meaningful.

4. **Bigrams suffer due to small dataset size.**

   With only 140 sentences, most bigrams appear very rarely. This creates a sparse feature space and makes learning unreliable.

5. **Dataset size is the biggest limitation.**

   The models often achieved very high training accuracy, but test accuracy remained below 90%.

Overall, this experiment helped me understand that:

- Feature engineering matters more than algorithm choice in small datasets.
- Simpler models can perform very well with good features.
- Overfitting becomes obvious when training accuracy is near 100%.

7 How to Run

```
# Install dependencies
pip install nltk matplotlib numpy

# Auto mode: train + evaluate + generate all charts
python3 main.py --auto

# Interactive mode (menu-driven)
python3 main.py
```

Listing 1: Setup and execution.

Charts are saved to `output/`. The interactive menu also allows custom sentence prediction (option 5).

8 Dependencies

- Python 3.8+
- `nltk` — English stopword list
- `matplotlib` — chart generation
- `numpy` — used only in `visualize.py`

9 References

The dataset sentences draw factual context from publicly available Indian news sources. They are original paraphrases, not direct copies.

1. ESPN Cricinfo — https://www.espncricinfo.com/
   Cricket statistics and match reports.
2. Olympics.com — https://olympics.com/
   Indian athletes' Olympic achievements.
3. NDTV — https://www.ndtv.com/
   Indian political news coverage.
4. The Hindu — https://www.thehindu.com/
   Parliamentary proceedings and election reporting.
5. Indian Express — https://indianexpress.com/
   State politics and policy coverage.
6. Press Information Bureau, Govt. of India — https://pib.gov.in/
   Government scheme announcements.
7. Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
   (TF-IDF formulation and Naive Bayes classifier reference.)

8. Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
(Logistic Regression and KNN algorithmic reference.)