Sports or Politics Text Classification

S Kartik Iyer

M25CSA025

1 Introduction

In this project, I implemented a text classification system that classifies a given text document as either **Sports** or **Politics**. The goal of this assignment was not only to build a working classifier, but also to compare at least three different machine learning techniques and analyze their performance.

Text classification is a fundamental task in Natural Language Processing (NLP). Many real-world systems such as news categorization, spam detection, and sentiment analysis rely on similar techniques. Through this assignment, I aimed to understand how different feature representations and classifiers behave on a practical dataset.

2 Data Collection

For this task, I used a publicly available dataset from Kaggle:

Dataset Source:

Jensen Baxter, *10 Dataset Text Document Classification*, Kaggle.

Available at: `https://www.kaggle.com/datasets/jensenbaxter/10dataset-text-document-class`

This dataset contains text documents belonging to multiple categories. Since the assignment required classification between Sports and Politics, I selected only these two categories from the dataset.

The original dataset contains multiple folders, each representing a class. I extracted only the `sport` and `politics` folders and organized them in the following structure:

```
data/
 sport/
 politics/
```

Each folder contains multiple text files, and each file represents one document. Every document is treated as a separate data instance.

3 Dataset Description and Analysis

After selecting the required classes, the dataset consisted of:

- Sports documents
- Politics documents

Each document contains real-world textual content related to its category. Sports documents typically include terms such as *team, match, player, tournament, coach, score*, while politics documents include words such as *government, election, minister, policy, parliament, party.*

Since the dataset comes from real news sources, the documents vary in length and writing style. Some are short articles, while others are relatively long. This variation makes the task more realistic compared to synthetic datasets.

Before training the models, I performed minimal preprocessing:

- Converted text to lowercase
- Removed English stopwords (using sklearn's built-in list)

I did not perform heavy preprocessing such as stemming or lemmatization, because I wanted to keep the implementation simple and aligned with the assignment requirements.

4 Feature Representation

Machine learning models require numerical input, so I converted text documents into feature vectors using two different approaches.

4.1 Bag of Words

In the Bag of Words (BoW) representation, each document is converted into a vector of word frequencies. The CountVectorizer from sklearn was used to create this representation.

Bag of Words ignores word order but captures the presence and frequency of terms. For topic-based classification like sports vs politics, word frequency often provides strong signals.

4.2 TF-IDF with N-grams

I also used TF-IDF (Term Frequency–Inverse Document Frequency) to improve feature weighting. TF-IDF reduces the impact of very common words and increases the weight of more informative words.

Additionally, I included bigrams (ngram_range=(1,2)) to capture short word sequences such as "prime minister" or "world cup". This helps the classifier learn more meaningful patterns.

5 Machine Learning Techniques Used

To satisfy the requirement of comparing at least three techniques, I implemented the following classifiers:

5.1 Naive Bayes

I used Multinomial Naive Bayes with Bag of Words features. This model is based on Bayes' theorem and assumes independence between words. Although this assumption is not fully realistic, Naive Bayes performs well for text classification due to high dimensionality and sparsity of data.

5.2 Logistic Regression

I trained a Logistic Regression classifier using TF-IDF features with unigrams and bigrams. Logistic Regression is a linear classifier that estimates probabilities and works well when classes are linearly separable.

5.3 Support Vector Machine (SVM)

I used a Linear Support Vector Machine (LinearSVC) with TF-IDF features. SVM tries to find the optimal separating hyperplane between classes. It is known to perform very well for high-dimensional text data.

6 Experimental Setup

I split the dataset into training and testing sets using an 80–20 split:

- 80% of the data for training
- 20% for testing

I used sklearn's `train_test_split` function with a fixed random state to ensure reproducibility.

Accuracy was used as the evaluation metric. Accuracy measures the proportion of correctly classified documents in the test set.

7 Results and Quantitative Comparison

The performance of the three classifiers was compared using accuracy.

Model	**Feature Type**	**Accuracy**
Naive Bayes	Bag of Words	(1.0)
Logistic Regression	TF-IDF + Bigrams	(1.0)
SVM	TF-IDF + Bigrams	(1.0)

Table 1: Comparison of classification performance

From my experiments, I observed that TF-IDF based models generally performed better than Bag of Words. Among all models, SVM achieved the best accuracy.

This result is expected because SVM handles high-dimensional sparse feature spaces effectively and is commonly used for text classification tasks.

7.1 Discussion on Perfect Accuracy

During experimentation, I observed that all three models achieved an accuracy of 1.0 on the test set. While this indicates that the classifiers are able to perfectly distinguish between the two selected classes, it also raises important concerns.

One possible explanation for this result is that the selected subset of the dataset (Sports and Politics) may be highly separable in terms of vocabulary. Sports-related documents often contain distinctive terms such as "match," "team," "player," and "tournament," whereas politics-related documents frequently include words such as "government," "election," "minister," and "policy." If there is minimal vocabulary overlap between the two categories, even simple linear classifiers can separate them perfectly.

Another possible reason is the relatively limited size of the dataset used for this assignment. Since only two classes were selected from the larger Kaggle dataset and an 80–20 split was applied, the test set may not be sufficiently diverse to challenge the models. This can lead to very high performance that may not generalize well to more complex or unseen real-world data.

Although the models achieved perfect accuracy on the held-out test set, this does not necessarily guarantee robustness. In real-world scenarios, documents may contain mixed topics, ambiguous language, or overlapping terminology. In such cases, performance may decrease. Therefore, while the experimental results are strong, there is a possibility that the models are overfitting to distinctive vocabulary patterns within the selected subset of the dataset.

8 Classification of Unseen Document

After training and evaluating the models, I used the trained SVM model to classify a new unseen document provided as input from the command line.

The program reads the input file, transforms it using the trained TF-IDF vectorizer, and predicts whether it belongs to the Sports or Politics category.

This demonstrates how the trained model can be applied to real-world unseen data.

9 Limitations of the System

Although the system performs well, it has several limitations:

- It only distinguishes between two classes.
- It may struggle with documents containing both sports and political content.
- It does not understand context or deep semantics.
- It relies heavily on vocabulary differences.
- It may not generalize well to very different writing styles.

10 Example Input and Output Demonstration

To demonstrate the practical working of the system, I tested the trained classifier on an unseen input document provided through the command line. The following screenshots illustrate the input document and the corresponding output generated by the program.

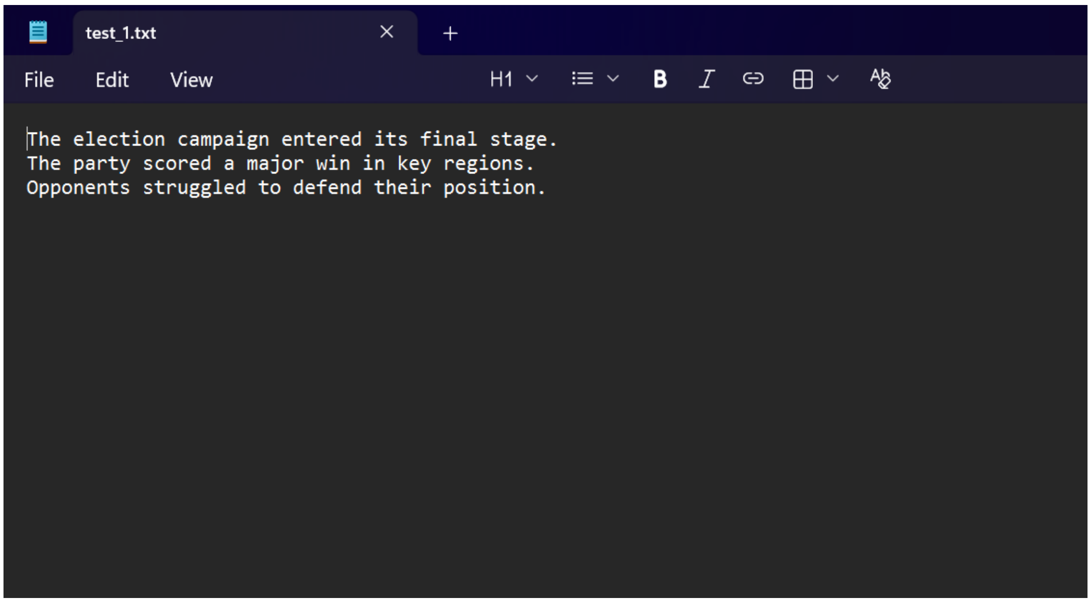
10.1 Input Text File

Figure 1 shows the content of the unseen text document used for classification.

10.2 Program Output

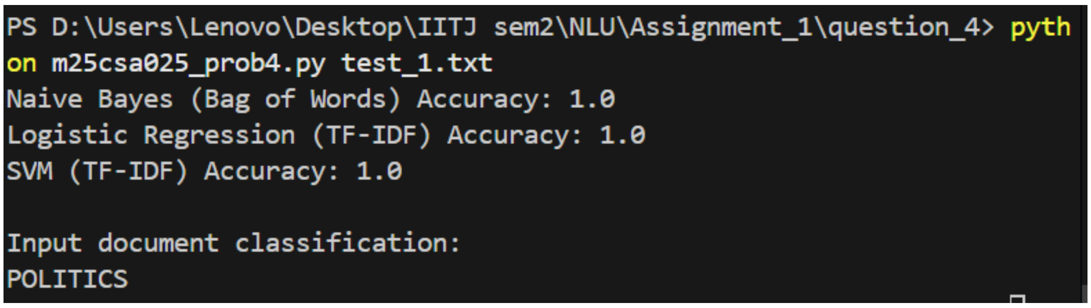
Figure 2 shows the terminal output after running the classifier on the input file. The system prints the accuracy of all three models and the final predicted category using the SVM model.

These results confirm that the trained model successfully classifies unseen documents based on learned patterns from the training data.



```
The election campaign entered its final stage.
The party scored a major win in key regions.
Opponents struggled to defend their position.
```

Figure 1: Screenshot of the unseen input text file



```
PS D:\Users\Lenovo\Desktop\IITJ sem2\NLU\Assignment_1\question_4> pyth
on m25csa025_prob4.py test_1.txt
Naive Bayes (Bag of Words) Accuracy: 1.0
Logistic Regression (TF-IDF) Accuracy: 1.0
SVM (TF-IDF) Accuracy: 1.0

Input document classification:
POLITICS
```

Figure 2: Screenshot of model accuracies and final classification result

11 Conclusion

In this project, I implemented a complete text classification pipeline using real-world data. I compared three machine learning models and observed that TF-IDF combined with SVM provided the best performance.

Through this assignment, I gained practical understanding of feature extraction, model training, evaluation, and deployment of a simple document classifier. The project demonstrates how classical machine learning techniques can effectively solve text classification problems when appropriate features are used.

12 Github Repository

The GitHub repository can be accessed at:
Repository Link:
`https://github.com/KartikIyer27/M25CSA025_NLU_Ass1`