Assignment 1

Link for the github file: https://github.com/b22ai049/NLU-Sports-Politics-Classifier/tree/main

1. Data Collection: Where it all started

To be honest, I initially thought about manually copying news headlines, but I realized that wouldn't give me enough data for a decent model. I ended up finding a dataset on GitHub that had exactly what I needed: raw text from news stories labeled as either "sport" or "politics."

I wrote a Python script using the requests library to fetch the data directly from the raw URL. This was way better than downloading it manually because now anyone running my code can get the same data instantly. One thing I ran into was that some labels were uppercase and some weren't, so I had to force everything to lowercase during the "cleaning" phase so the model wouldn't think "Sport" and "sport" were different things.

2. Dataset Description and Analysis

After filtering out the categories I didn't need, I was left with **6,436 articles**.

The Numbers:

- **Total Samples:** 6,436
- **Training Set:** 80% (approx. 5,148 samples)
- **Test Set:** 20% (1,288 samples)

Looking at the data, I noticed the "Politics" side was a bit heavier than "Sports" (746 vs 542 in my test set). The language in the Politics section was very formal—lots of words like *legislation* and *minister*. The Sports side was much more casual and action-packed, with words like *injury, striker,* and *victory*.

3. Feature Representation (The "Secret Sauce")

I used **TF-IDF (Term Frequency-Inverse Document Frequency)** to turn the text into numbers. I chose this because a simple word count (Bag of Words) was giving too much importance to boring words like "the" and "is." TF-IDF is smarter because it rewards words that are rare across the whole dataset but frequent in one category.

I also experimented with **Bi-grams**. This was a game-changer. For example, the word "Prime" or "Minister" alone is okay, but "Prime Minister" is a massive clue for Politics. Same with "Home Run" for sports. Using pairs of words made the accuracy jump up significantly.

4. The "Battle" of the Models

I compared three different techniques to see which one was the most reliable. I kept the same TF-IDF features for all three to keep it a fair fight.

ML Technique	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Multinomial Naive Bayes	89.21%	0.89	0.88	0.89
Logistic Regression	90.14%	0.90	0.89	0.90
SVM (Linear Kernel)	**90.68%**	**0.91**	**0.90**	**0.91**

Results Table:Why SVM won: SVM (Support Vector Machine) was the winner. I think this is because text data usually has a lot of dimensions (thousands of unique words), and SVM is really good at finding a clean "border" between two groups in that kind of high-dimensional space.

5. Limitations (What my system gets wrong)

Even though ~91% accuracy is pretty good for a first try, it’s not perfect.

- **Boundary Cases:** The biggest issue is articles that talk about both. Like if a story is about "Government funding for the local football stadium," it has words from both worlds. My model usually flips a coin on those.
- **Lack of Context:** Because I'm using TF-IDF, the model doesn't actually "read." It just counts. It doesn't know the difference between a "political campaign" and a "championship campaign."
- **No Lemmatization:** I didn't have time to implement a full stemmer/lemmatizer. So the model treats "runs," "running," and "ran" as three totally different words, which probably makes the vocabulary larger than it needs to be.