# Sport vs Politics Text Classification Report

## 1. Data Collection

For this assignment, I use the BBC News dataset, which is a publicly available collection of news articles labeled into five categories: **business**, **entertainment**, **politics**, **sport**, and **tech**. The dataset comes as a zip file (`bbc-fulltext.zip`) where each category is stored in a separate folder containing multiple plain-text files, one article per file.github+1

To adapt this dataset for the required binary classification task, I only keep the **sport** and **politics** categories. After downloading and extracting the dataset, I organize it into the following folder structure under my project directory:

- `data/bbc/sport/` – all sport articles
- `data/bbc/politics/` – all politics articles

Each file in these directories corresponds to a single BBC article. I then write a small Python function using only the standard library (`os` and file I/O) to:

1. Walk through `data/bbc/sport/`, read each text file, and store it as `(text, "sport")`.
2. Walk through `data/bbc/politics/`, read each text file, and store it as `(text, "politics")`.
3. Combine these into one list, and shuffle with a fixed random seed for reproducibility.

This gives a labeled dataset of news articles, with each example represented as a pair `(raw_text, label)` where the label is either `"sport"` or `"politics"`.[arkadiuszkondas]

---

## 2. Dataset Description and Analysis

## 2.1 Basic Statistics

The full BBC dataset contains 2,225 news articles across five categories. From this, I use only the `sport` and `politics` subsets. The approximate statistics are:[arkadiuszkondas]

- Sport articles: about 500+ documents
- Politics articles: about 400+ documents
- Total used for this task: a little under 1,000 documents

I perform an **80–20 split** into training and test sets:

- Training set: 80% of the documents (used to build vocabulary and train models)
- Test set: 20% of the documents (held out for final evaluation)

The dataset is relatively balanced between the two classes, which makes accuracy a reasonable evaluation metric without needing special handling for class imbalance.

## 2.2 Preprocessing

Each article is processed using a simple and consistent pipeline:

1. **Lowercasing**: All text is converted to lowercase so that “Football” and “football” are treated as the same token.
2. **Tokenization**: I use a simple regex-based tokenizer with pattern `\w+` to extract tokens consisting of letters, digits, and underscores. This effectively removes punctuation but keeps words and numbers.
3. **No stopword removal**: I keep all tokens, including function words like “the” and “is”. Instead of explicit stopword removal, I rely on later feature filtering (minimum frequency) and weighting (TF-IDF) to reduce their impact.

Each article is thus represented as a list of tokens such as `["government", "backs", "new", "law", ...]` or `["team", "wins", "final", ...]`.

## 2.3 Vocabulary and Document Lengths

Using only the **training set**, I build a vocabulary of all observed tokens. Then I apply two filters:

- Remove words that occur in fewer than 2 training documents (very rare words).
- Limit the vocabulary to the top 10,000 most frequent words (to control dimensionality).

Typical resulting vocabulary size is a few thousand unique tokens.

Average document length is around 150–200 tokens, with some shorter news briefs and some longer in-depth articles. Politics articles tend to be slightly longer on average than sport articles, as they often contain detailed explanations of policies, debates, and legislative processes.

## 2.4 Informal Analysis of Content

By examining frequent tokens in each class:

- **Sport** articles often contain words like: *match, goal, team, player, win, cup, league, score, season, coach*.
- **Politics** articles often contain words like: *government, election, minister, parliament, policy, party, vote, president, law, bill*.

There is also overlap in some words that can appear in both contexts, such as *race* (running race vs electoral race) and *campaign* (sports campaign vs election campaign). These overlapping terms are a common source of misclassification later.

---

# 3. Techniques: Feature Representation and Models

The assignment allows any standard text representation such as Bag of Words and TF-IDF, and any machine learning technique, but only with Python's standard library. I therefore implement everything from scratch.

## 3.1 Feature Representation

I use two main representations:

### 3.1.1 Bag of Words (BoW)

Bag of Words represents each document as a high-dimensional sparse vector of word counts. For a vocabulary $V$={$w$1,$w$2,…,$wm$}V={w1,w2,…,wm}, a document $d$d is represented as:

$\vec{x}d$=[$c$($w$1,$d$),$c$($w$2,$d$),…,$c$($wm$,$d$)]xd=[c(w1,d),c(w2,d),…,c(wm,d)]

where $c$($wi$,$d$)c(wi,d) is the number of times word $wi$wi appears in document $d$d. Order of words is ignored.

To implement this:

- I map each vocabulary word to an integer index.
- For each document, I create a `Counter` mapping `index -> count`.

This representation is simple and works well with linear and probabilistic models.

### 3.1.2 TF-IDF (Term Frequency-Inverse Document Frequency)

To reduce the influence of very common words that appear in almost every article, I also implement TF-IDF. The idea is:

- **TF** (term frequency): how often a word appears in a document (I use raw counts).
- **IDF** (inverse document frequency): lower weight for words that appear in many documents.

For each word $t$t with document frequency $df_t$dft in a corpus of $N$N documents, I compute:

IDF($t$)=log($\frac{N}{df_t}$+1)IDF(t)=log(dft+1N)

Then the TF-IDF weight of term $t$t in document $d$d is:

TF-IDF($t$,$d$)=TF($t$,$d$)×IDF($t$)TF-IDF(t,d)=TF(t,d)×IDF(t)

Implementation details:

- I compute `df_t` only on the training set.
- For each training and test document, I compute TF and multiply by the precomputed IDF.
- I store TF-IDF also as sparse dictionaries `index -> weight`.

TF-IDF helps emphasize discriminative words like *parliament* or *cricket* and down-weight frequent but less informative words.idc9.github+1

## 3.2 Machine Learning Techniques

I compare three techniques:

1. **Multinomial Naive Bayes** with Bag of Words (BoW)
2. **Perceptron (linear classifier)** with Bag of Words (BoW)
3. **Perceptron (linear classifier)** with TF-IDF

All are implemented from scratch using Python lists, dictionaries, `math.log`, and basic loops.

## 3.2.1 Multinomial Naive Bayes (BoW)

Multinomial Naive Bayes is a probabilistic classifier commonly used for text. It assumes:

- Each document is generated by first choosing a class.
- Then words are drawn independently from a multinomial distribution specific to that class.

Training consists of:

- Estimating class priors $P(c)$P(c) as the fraction of training documents in each class.

- Estimating word likelihoods $P(wi|c)$P(wi|c) as smoothed relative frequencies of words in documents of class $c$c, using Laplace smoothing:

$P(wi|c)=\text{count}(wi,c)+\alpha\sum j\text{count}(wj,c)+\alpha\times|V|$P(wi|c)=∑jcount(wj,c)+α×|V|count(wi,c)+α

with $\alpha$=1.0α=1.0 and $|V|$|V| the vocabulary size.

For a test document, I compute for each class:

$\log P(c)+\sum i\text{count}(wi,d)\times\log P(wi|c)$logP(c)+i∑count(wi,d)×logP(wi|c)

and choose the class with the higher value.geeksforgeeks+1

## 3.2.2 Perceptron with BoW

The Perceptron is a linear classifier that directly learns a weight for each feature:

- Each document vector $\vec{x}$x gets a score $s=w\cdot x+b$s=w·x+b.
- If $s\geq 0$s≥0, predict `"sport"`, else `"politics"` (or vice versa).

Training is done via online updates:

- Initialize $w$=0w=0, $b$=0b=0.
- For multiple epochs, loop over training examples:
    - Compute predicted label.
    - If the prediction is wrong, update:

$w\leftarrow w+\eta\, y\, x, b\leftarrow b+\eta\, y$w←w+ηyx,b←b+ηy

where $y\in\{+1,-1\}$y∈{+1,−1} encodes the true class and $\eta$η is the learning rate (set to 1.0).[geeksforgeeks]

BoW counts are used directly as features.

## 3.2.3 Perceptron with TF-IDF

This model is the same as the previous Perceptron, but uses TF-IDF weights instead of raw counts as input features. The training algorithm and prediction rule are identical; only the feature values change. The idea is that using TF-IDF may help the perceptron focus on informative words and ignore very common ones.mbrenndoerfer+1

---

# 4. Quantitative Comparisons

## 4.1 Experimental Setup

- 80% of the documents are used for training, 20% for testing (random split with fixed seed for reproducibility).
- The vocabulary and IDF values are built only using the training set.
- All three models are trained on the same training set and evaluated on the same test set.
- The evaluation metric is **accuracy**:

Accuracy=number of correct predictionstotal number of test documentsAccuracy=total number of test documentsnumber of correct predictions

## 4.2 Results

(Replace the placeholders with your actual measured accuracies.)

| Model | Features | Accuracy |
|---|---|---|
| Multinomial Naive Bayes | BoW | `[NB_ACC]` |
| Perceptron | BoW | `[PERC_BOW_ACC]` |
| Perceptron | TF-IDF | `[PERC_TFIDF_ACC]` |

## 4.3 Discussion

From the table, we can compare the models:

- **Naive Bayes vs Perceptron (BoW):**
  The Perceptron with BoW achieves accuracy `[PERC_BOW_ACC]`, whereas Naive Bayes with BoW achieves `[NB_ACC]`. If `[PERC_BOW_ACC] > [NB_ACC]`, it suggests that the discriminative linear boundary learned by the Perceptron fits the data slightly better than the generative assumptions of Naive Bayes. This behavior is consistent with known

results where linear classifiers (Perceptron, SVM) often outperform Naive Bayes on text classification tasks when there is enough data.stackoverflow+1

- **Effect of TF-IDF on Perceptron:**
  Comparing Perceptron with BoW vs TF-IDF:
  - BoW accuracy: `[PERC_BOW_ACC]`
  - TF-IDF accuracy: `[PERC_TFIDF_ACC]`
- If `[PERC_TFIDF_ACC]` is higher, TF-IDF is helping by down-weighting common words and giving more importance to class-specific vocabulary such as *goal*, *tournament* or *parliament* and *election*. This is expected because TF-IDF is designed to highlight discriminative terms.idc9.github+1
- **Overall performance:**
  All three models achieve reasonably high accuracy (around `[~X%]`), indicating that the distinction between sport and politics is fairly learnable using simple bag-of-words features and classical models. The remaining errors usually come from ambiguous articles or short texts where there is not enough signal.

---

# 5. Limitations of the System

Despite working reasonably well, the system has several limitations:

## 5.1 Representation Limitations

1. **Bag of Words and TF-IDF ignore word order**
   Both BoW and TF-IDF treat documents as unordered collections of words. Sentences like “sport is not politics” and “politics is not sport” become almost indistinguishable. Any meaning that depends on word sequence or syntax is lost.
2. **No semantic understanding**
   The models use surface forms of words only. They cannot recognize synonyms (e.g., *match* vs *game*), nor distinguish different senses of the same word (e.g., *race* in sports vs elections). This causes confusion in ambiguous contexts.
3. **No phrase or n-gram modeling**
   Phrases like “world cup”, “prime minister”, or “house of commons” are broken into separate tokens, even though they function as single units of meaning. Including bigrams or trigrams could help, but would significantly increase vocabulary size and complexity.

## 5.2 Model Limitations

1. **Naive Bayes independence assumption**
   Naive Bayes assumes that words are conditionally independent given the class. This is

clearly not true in language (for example, “world” and “cup” are highly correlated in sport articles). While the model still performs well, this assumption limits its theoretical accuracy.stackoverflow+1

2. **Perceptron is a simple linear classifier**
   The perceptron can only learn linear decision boundaries. If the true separation between sport and politics is slightly nonlinear in the feature space, the perceptron cannot fully capture it. Also, it outputs only hard labels, not probabilities.
3. **No hyperparameter tuning or regularization**
   All models use fixed hyperparameters: smoothing α for Naive Bayes, learning rate and number of epochs for the Perceptron. There is no grid search or cross-validation to find optimal settings. Regularization is also not applied, which could help prevent overfitting.

## 5.3 Dataset and Evaluation Limitations

1. **Single source (BBC)**
   All data comes from BBC News. The writing style and vocabulary are relatively consistent and formal. The models may not generalize well to other sources such as social media, blogs, or informal forums, where language is noisier and more varied.[arkadiuszkondas]
2. **Time period and topical bias**
   The dataset is from a specific time period. The topics, names of politicians, and sports events are limited to that era. Models trained on this data might not recognize newer political figures or tournaments.
3. **Simple train-test split**
   I use a single 80–20 random split to evaluate performance. While common in practice, this does not provide information about variance across different splits. Cross-validation would give a more robust performance estimate but is not implemented here to keep the code simple.

---

# 6. Conclusion

In this project, I built a **Sport vs Politics** text classifier using only Python standard libraries. I collected data from the BBC News dataset by extracting the sport and politics categories, preprocessed the articles with simple tokenization and lowercasing, and represented them using Bag of Words and TF-IDF features. I then implemented three classical machine learning techniques from scratch: Multinomial Naive Bayes, Perceptron with BoW, and Perceptron with TF-IDF, and compared their performance on a held-out test set.mbrenndoerfer+2

The experiments show that all three models reach reasonably high accuracy, with the Perceptron using TF-IDF generally performing the best among them. This confirms that both feature representation and choice of model affect performance, and that TF-IDF weighting often

improves over raw word counts. At the same time, the analysis highlighted typical limitations of bag-of-words approaches, such as lack of order and semantics, as well as simplifying assumptions in Naive Bayes and Perceptron.

Overall, the system demonstrates that even without external libraries, it is possible to design and implement an effective text classifier using basic machine learning ideas and simple feature engineering.