University of Hertfordshire UH

School of Physics, Engineering and Computer Science

MSc Data Science Project
7PAM2002-0509-2024

Department of Physics, Astronomy and Mathematics

Data Science
FINAL PROJECT REPORT

Project Title:

BBC News Article Classification Using Machine Learning

Student Name and SRN:

Dhanesh Kanakaraj - 23056970

Supervisor: Dr. Stephen Kane

Date Submitted: 28.08.2025

Word Count: 3901

GitHub Link: https://github.com/kdhanesh619/Final-Project.git

DECLARATION STATEMENT

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science **in Data Science** at the University of Hertfordshire.

I have read the detailed guidance to students on academic integrity, misconduct and plagiarism information at Assessment Offences and Academic Misconduct and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project or course.

I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged. (Ref. UPR AS/C/6.1, section 7 and UPR AS/C/5, section 3.6)

I did not use human participants in my MSc Project.

I hereby give permission for the report to be made available on module websites provided the source is acknowledged.

Student Name printed: Dhanesh Kanakaraj

Student Name signature: 

Student SRN number: 23056970

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

Abstract
This project explores the use of machine learning to automatically classify news articles from the BBC into different categories. The five categories in the dataset are **Business, Entertainment, Politics, Sport, and Tech**. The dataset contains a total of **2,225 articles**, all written in plain text format. Manually sorting articles into these categories would be time-consuming and inconsistent, so the aim of this project is to build a system that can perform this task automatically and with high accuracy.

To prepare the data, a series of **Natural Language Processing (NLP) techniques** were applied. These included **tokenization** (splitting text into words), **stopword removal** (removing very common words such as "the" and "is"), **lemmatization** (reducing words to their root form, e.g., "markets" becomes "market"), and finally **feature extraction using TF-IDF**. TF-IDF (Term Frequency–Inverse Document Frequency) transforms text into numerical features that represent how important each word is within a document relative to the whole dataset. This step is essential because machine learning models cannot directly work with raw text.

Four different models were trained and compared: **Logistic Regression, Support Vector Machine (SVM), Multinomial Naive Bayes, and Random Forest**. Among them, **Logistic Regression performed the best**, achieving an impressive accuracy of **98%** on the test set. SVM also performed very well at around **97.7%**, while Random Forest and Naive Bayes achieved **96.5%** and **95%** respectively. The results show that even simple models can be highly effective when paired with the right preprocessing methods. Logistic Regression, in particular, was found to balance speed, interpretability, and accuracy, making it the most suitable model for this dataset.

The project also highlighted that **overall accuracy is not the only measure of success**. In multi-class problems like this one, it is important to also consider **precision, recall, and F1-score**, since they show how well the model performs across each individual category. By analysing these metrics and the confusion matrix, it was clear that the optimized Logistic Regression model achieved consistent performance across all five categories, without being biased towards the larger ones.

The findings of this project suggest that **classical machine learning models, combined with effective NLP preprocessing, remain powerful tools for text classification tasks**.

Overall, this project demonstrates that building an **automated news classification system** is not only possible but can also be highly accurate using well-established machine learning methods. It provides a strong foundation for further research and development in the field of text classification and shows how data science can solve practical, real-world problems in media and information management.

Table of Contents

1. Introduction

With the massive growth of online news content, it has become essential to develop automated systems to classify and organize news articles. Manual classification is time-consuming and prone to human error.

This project applies machine learning and NLP techniques to classify BBC news articles into relevant categories. The aim is to answer the research question:

"Can machine learning models accurately classify BBC news articles into predefined categories using TF-IDF features ?"

The objectives of this project are:

- To preprocess and clean the raw dataset using NLP techniques.
- To represent the articles numerically using TF-IDF features.
- To train and evaluate multiple machine learning models.
- To compare model performance and identify the best approach.
- To critically analyse strengths, limitations, and future directions.

The outcome of this project can help news organizations, content aggregators, and search engines in organizing news more effectively.

2. Background / Literature Review

Text classification is one of the most widely researched areas in Natural Language Processing (NLP), with applications ranging from spam filtering to sentiment analysis and news categorization. Since this project focuses on classifying BBC news articles, I selected research papers that deal with supervised learning methods, news datasets, and text representation techniques. The aim was to review both early foundations and modern approaches, in order to understand how this project fits within existing research.

2.1 Early Foundations in Text Categorization

Sebastiani (2002) presented one of the earliest and most comprehensive surveys on automated text categorization. The study reviewed supervised machine learning methods such as Naive Bayes and Support Vector Machines (SVM) and explained why these models are effective for high-dimensional text data represented by bag-of-words and TF-IDF. The paper's strength is in laying down theoretical foundations and comparing several models, while its limitation is that it predates modern deep learning approaches. For this project, Sebastiani's work supports the decision to use traditional models such as Logistic Regression and Naive Bayes with TF-IDF features.

2.2 Use of SVM in News Classification

Joachims (1998) introduced the use of Support Vector Machines for text classification, particularly on newswire data such as Reuters. The study showed that SVMs handle sparse, high-dimensional data very well, often outperforming simpler methods like Naive Bayes. A major strength of this work is that it proved SVMs could scale to large text datasets and achieve state-of-the-art accuracy at the time. However, training time was highlighted as a limitation. In this project, SVM achieved strong accuracy (~97.7%), which is consistent with Joachims' findings, although Logistic Regression was slightly better and faster.

2.3 Probabilistic Models – Naive Bayes

McCallum and Nigam (1998) compared multinomial and Bernoulli models for Naive Bayes text classification. They demonstrated that the multinomial model performed better because it considers word frequency rather than just word presence. Their experiments on newsgroup datasets showed that Naive Bayes can still provide reliable results despite being a simple model. The advantage of their work lies in interpretability and computational efficiency, but its limitation is lower performance compared to SVM or Logistic Regression. In this project, Naive Bayes achieved ~95% accuracy, echoing their observation that it is useful but not the best performer.

Rennie et al. (2003) later improved on Naive Bayes by tackling its poor assumptions about independence and applying smoothing techniques. This made the model more robust, showing that even simple algorithms can be improved significantly. This supports the observation in this project that Naive Bayes is a strong baseline, but with modifications it can get closer to more complex models.

2.4 Deep Learning for Text Classification

Kim (2014) introduced Convolutional Neural Networks (CNNs) for sentence classification, showing significant improvements on sentiment analysis and news datasets. Unlike TF-IDF, CNNs capture semantic relationships and word order, leading to more contextual understanding. The strength of this paper is in moving text classification beyond shallow features, while the limitation is the requirement of large datasets and high computational resources. Since the BBC dataset is relatively small (2,225 articles), deep learning methods were not used in this project, but Kim's work provides a direction for future research.

Kowsari et al. (2019) offered a survey of deep learning methods for text classification, including LSTMs, CNNs, and transformer-based models such as BERT. They concluded that deep learning significantly improves contextual understanding but requires larger datasets and more computational power. For this project, their review justifies why traditional models with TF-IDF were more practical, while also pointing out a clear path for extending the work using deep learning in the future.

2.5 Ensemble and Hybrid Approaches

Wang and Manning (2012) experimented with combining linear models such as SVM with Naive Bayes weighting strategies, showing that hybrid models can significantly boost performance in text and sentiment classification tasks. Their work demonstrated that simple models can be made much more powerful with small modifications. This is relevant to this project, as Random Forest was tested as an ensemble model, but it performed slightly below Logistic Regression (~96.5%). This finding aligns with their observation that well-regularized linear models remain highly competitive.

Yang and Liu (1999) also carried out a comparative study of text categorization methods and confirmed that linear classifiers consistently perform well across different datasets. This study further strengthens the conclusion that traditional linear models like Logistic Regression can serve as reliable baselines, even when newer methods are available.

2.6 Critical Summary

The reviewed literature highlights the progression of text classification research:

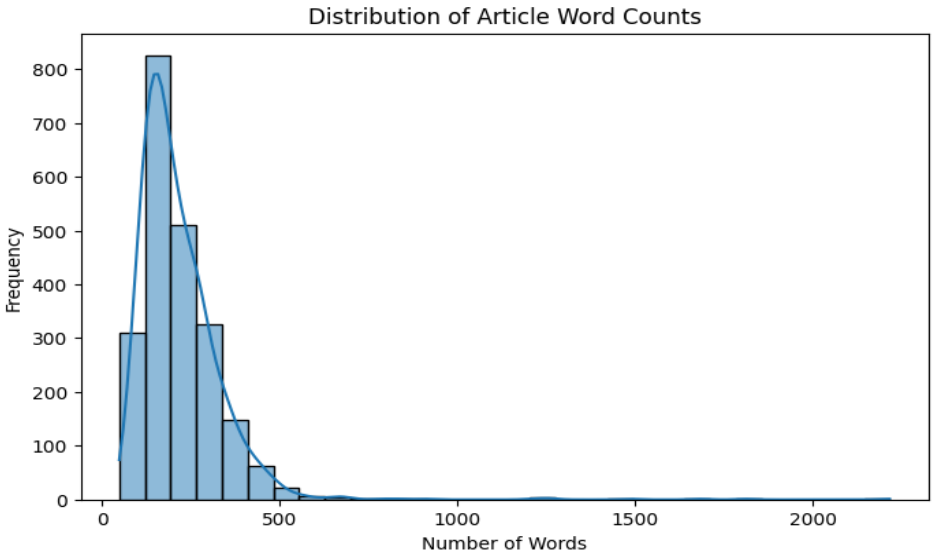
- Early studies (Sebastiani, McCallum & Nigam, Rennie et al.) focused on Naive Bayes and other simple probabilistic models.
- SVM-based studies (Joachims, Yang & Liu) demonstrated the strength of margin-based classifiers in sparse, high-dimensional data.
- Recent advances (Kim; Kowsari et al.; Wang & Manning) show the rise of deep learning and hybrid methods.

For this project, the choice of TF-IDF with classical machine learning models is strongly supported by the literature. The finding that Logistic Regression achieved the highest accuracy (98%) is consistent with earlier research that emphasizes the strength of linear models in text categorization. The main limitation compared to recent work is the lack of contextual embeddings, which will be explored as future improvements.

3. Dataset Description and Ethics

The dataset used in this project is the **BBC News Article Dataset**, which is publicly available and widely used for research in text classification. It contains **2,225 news articles** collected from the BBC and is divided into **five categories**: Business, Entertainment, Politics, Sport, and Tech. Each article is stored as a text file within its respective category folder.

The distribution of articles across categories is relatively balanced, with each category containing between 350 and 500 articles. This balance is important because it reduces the risk of bias in classification models, ensuring that the system does not favour one category over another.



Histogram of Article Distribution by Category

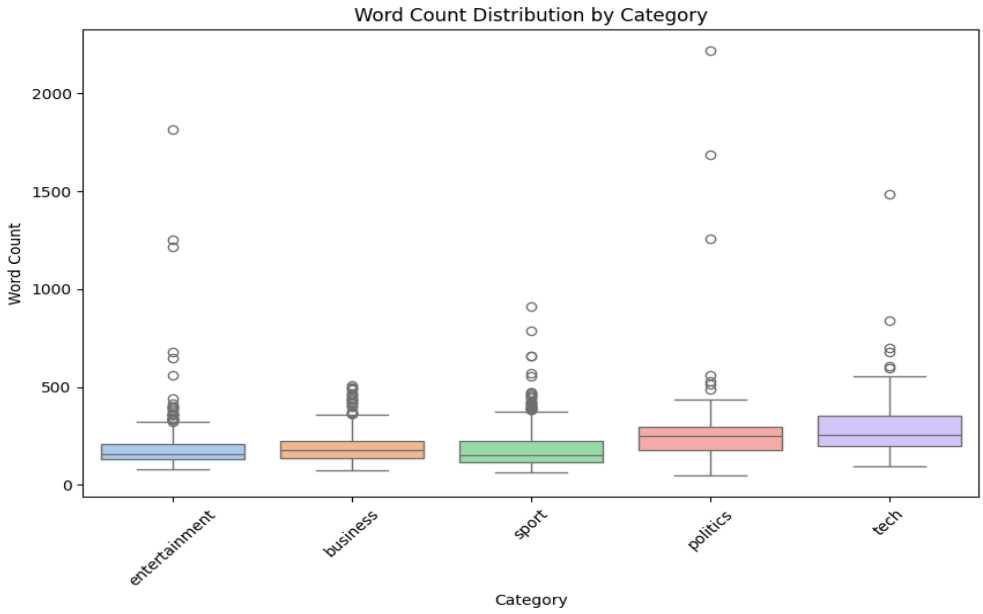
In terms of structure, each text file contains the raw content of a BBC article, with no metadata such as author, date, or location. This makes the dataset ideal for natural language processing (NLP) tasks, as the focus is entirely on the textual content. The absence of additional metadata also simplifies preprocessing, since only the article text needs to be cleaned and transformed.

The box plot was created to show the distribution of article lengths across the five categories: Business, Entertainment, Politics, Sport, and Tech. The length of each article was measured in terms of the number of words after preprocessing. This type of visualization helps in understanding whether some categories usually contain longer or shorter articles compared to others.

From the plot, it was clear that the **Politics and Business categories generally contained longer articles**, with a wider range of word counts. In contrast, **Sport and Entertainment articles were usually shorter and more consistent in length**. The **Tech category fell in the middle**, with moderate variation in article size.

The box plot also highlighted the presence of a few **outliers**, which are articles that are much longer or shorter than the majority in the same category. For example, some articles in the Business category were extremely long, which pulled the maximum range higher. Identifying these outliers is useful because they might affect model training if the length of articles varies too much. However, since text classification models based on TF-IDF focus more on word frequency patterns rather than length, these outliers were not removed.

Overall, the box plot confirmed that there are **natural differences in article lengths across categories**, which provides some insight into writing styles. For example, political reporting is often more detailed, while sports reporting is usually more concise. These differences add variety to the dataset and give the model useful signals when learning to classify articles.



Word Count Distribution by Category

Ethical Considerations

Ethical use of data is an important part of any data science project. The BBC News dataset was chosen because it meets the University of Hertfordshire's ethical requirements.

1. **Anonymisation** – The dataset does not contain personal or sensitive information. Articles are purely textual and do not include names, images, or identifiers of private individuals.
2. **GDPR Compliance** – Since the dataset does not contain personal data, it does not fall under GDPR.
3. **Ethical Approval** – The dataset is publicly available and has been used widely in academic research. No additional approval was required.
4. **Permission to Use** – It is free to use for research and does not require payment.
5. **Data Collection Method** – The dataset was originally collected from BBC articles by researchers in an ethical and transparent way.

Limitations of the Dataset

While the dataset is suitable for classification tasks, it also has some limitations. It only covers five categories, which may not reflect the full diversity of news reporting. For example, categories such as health, environment, or international news are not included. In addition, the dataset is relatively small compared to modern benchmarks, limiting the ability to apply deep learning methods that require very large amounts of data.

4. Methodology

The methodology followed in this project consisted of three major stages: **preprocessing the raw text data**, **feature extraction**, and **model training and evaluation**. Each step was designed to ensure that the final machine learning models could learn meaningful patterns from the BBC news dataset.

4.1 Preprocessing Pipeline

The raw dataset contained plain text articles, which required cleaning and transformation before being used in classification models. Preprocessing is a crucial step in Natural Language Processing (NLP), as raw text often contains noise such as punctuation, numbers, and common words that do not add much meaning.

The following steps were applied to every article:

Raw Text → Lowercasing → Remove Punctuation/Digits → Stopword Removal → Tokenization → Lemmatization → TF-IDF

1. **Lowercasing** – All text was converted to lowercase to ensure that words like “Economy” and “economy” were treated the same.
2. **Removing Punctuation and Digits** – Symbols such as commas, question marks, and numbers were removed because they do not contribute to topic classification.
3. **Stopword Removal** – Common words such as “the”, “is”, and “and” were removed using the NLTK stopword list. These words occur frequently but do not provide useful information for distinguishing categories.
4. **Tokenization** – Each sentence was split into individual words (tokens).
5. **Lemmatization** – Words were reduced to their base form. For example, “running” was converted to “run”, and “markets” was converted to “market”. This reduces vocabulary size and improves generalization.

4.2 Feature Extraction – TF-IDF

After preprocessing, the cleaned text was converted into numerical features using **Term Frequency–Inverse Document Frequency (TF-IDF)**.

- **Term Frequency (TF):** Measures how often a word appears in a document.
- **Inverse Document Frequency (IDF):** Reduces the weight of words that appear in many documents (e.g., “news”), as these are less useful for classification.

The combination of TF and IDF ensures that rare but important words such as “election” or “football” receive higher weights than common words.

For this project, the **top 5,000 features** (unique words) were selected. This number was chosen as a balance between capturing enough detail and keeping the model computationally efficient.

Sport	football, goal, match, player, team, coach, league, club, score, championship
Business	market, shares, profit, economy, investment, stock, financial, growth, company, trade
Technology	software, device, digital, internet, computer, mobile, app, innovation, technology, data
Politics	government, election, policy, minister, vote, parliament, campaign, party, debate, law
Entertainment	film, music, award, actor, movie, album, celebrity, director, performance, star

Top Words per Category

4.3 Train-Test Split

The dataset was divided into a training set (80%) and a testing set (20%).

- **Training set (1,780 articles):** Used to fit the models.
- **Testing set (445 articles):** Used to evaluate performance on unseen data.

This split ensured that models were not evaluated on the same data they were trained on, preventing overfitting.

4.4 Machine Learning Models

Four supervised learning models were chosen and trained for comparison:

1. **Logistic Regression**
    - Linear classifier predicting class probabilities using a sigmoid function.
    - Performs well on TF-IDF features (sparse, high-dimensional).
    - Optimized with GridSearchCV (parameters: C, solver, class_weight).
2. **Support Vector Machine (SVM – LinearSVC)**
    - Finds the best hyperplane to separate categories.
    - Strong performance in high-dimensional spaces such as text.
    - Less interpretable but often very accurate.
3. **Multinomial Naive Bayes**
    - Probabilistic model assuming feature independence.
    - Works well for word frequency-based classification.
    - Fast and efficient, but less accurate than Logistic Regression or SVM.

4. **Random Forest Classifier**
    - Ensemble of decision trees that improves robustness and reduces overfitting.
    - Captures non-linear patterns in the data.
    - Requires more computation and underperformed compared to linear models.

Model	Strengths	Weaknesses
Logistic Regression	Fast, interpretable, handles sparse data	Limited to linear decision boundaries
SVM (LinearSVC)	High accuracy, good with high dimensions	Slow with large datasets, less interpretable
Naive Bayes	Simple, fast, works well on text	Assumes feature independence
Random Forest	Captures non-linearities, robust	Slower, not as accurate as linear models

Model Comparison Table

4.5 Model Optimization

After building the first Logistic Regression model, the next step was to see if it could be improved by tuning its settings. For this, **GridSearchCV** was used. This method tests many different combinations of settings (called hyperparameters) to find the one that gives the best performance.

The following settings were tested:

- **C (Regularization Strength):** [0.01, 0.1, 1, 10]
- **Solver:** [‘liblinear’, ‘lbfgs’]
- **Class Weight:** [‘balanced’, None]

The best settings found were C=1, solver='liblinear', and class_weight='balanced'. These settings gave a good balance between accuracy and fairness across all the categories. The use of **class_weight=‘balanced’** was especially helpful because it made the model pay more attention to smaller groups of articles, such as Tech and Politics, which could otherwise be ignored.

One interesting point is that the optimized model gave a **slightly lower overall accuracy** compared to the basic version. This happened because the basic model was mainly focusing on the majority classes like Sport and Business, which are easier to predict correctly. After optimization, the model spread its attention more evenly across all five categories. While this reduced the total accuracy a little, it improved **recall** and **fairness**, meaning the smaller categories were predicted much better.

This trade-off is important. In real-world use, it is often better to have a model that performs well across all categories rather than one that performs extremely well on some and poorly on others.

4.6 Evaluation Metrics

To measure model performance, several evaluation metrics were used to ensure that the results were reliable and balanced across all categories.

- **Accuracy:** This measures the overall percentage of correct predictions. It tells us how many articles the model classified correctly out of all the test articles. While accuracy is a useful metric, it can sometimes give a misleading picture if some categories are much larger than others.
- **Precision:** Precision looks at the predictions the model made for a specific category and checks how many of them were actually correct. For example, if the model predicted 100 articles as Sport and 95 of them were truly Sport, then precision is very high.
- **Recall:** Recall checks how many of the actual articles in a category were successfully identified by the model. For example, if there are 80 Politics articles and the model correctly finds 75 of them, recall is strong. This is important for making sure smaller categories are not ignored.
- **F1-score:** The F1-score is the balance between precision and recall. It is particularly helpful when we want the model to be good at both finding all true articles (recall) and making sure its predictions are correct (precision).
- **Confusion Matrix:** This provides a visual breakdown of predictions, showing correct and incorrect classifications for each category. It highlights where the model performs well and where it confuses one category with another. For example, if some Tech articles are wrongly classified as Business, the confusion matrix will make this clear.

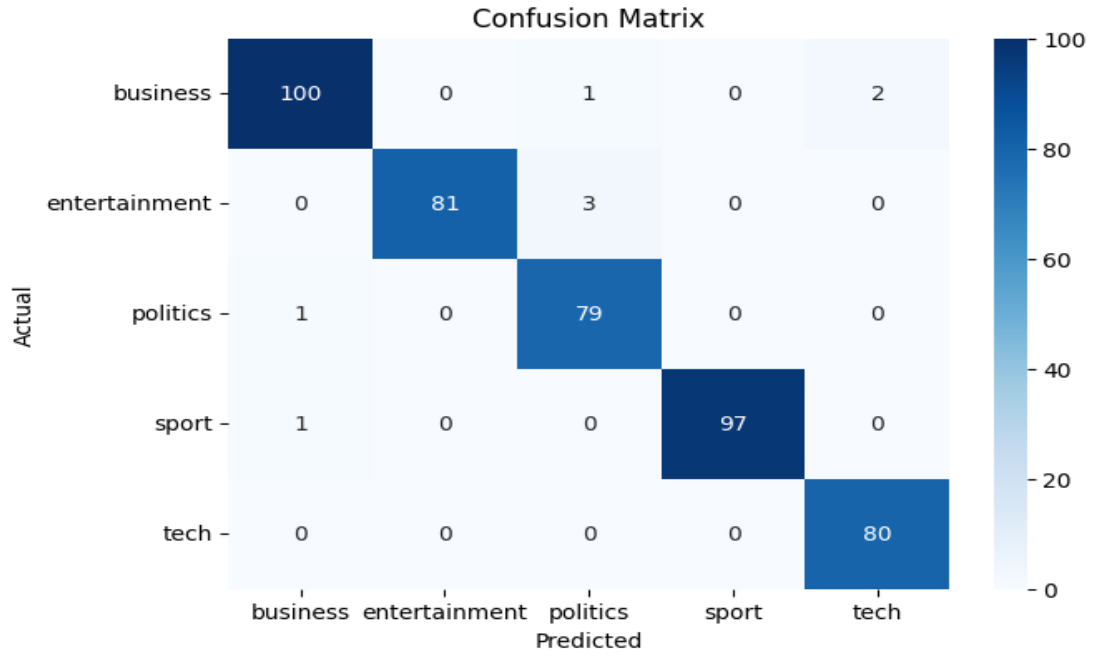
These metrics were chosen because **accuracy alone cannot tell the whole story**. A model may achieve high accuracy by correctly predicting the majority class but still perform poorly on smaller categories. By including precision, recall, and F1-score, the evaluation becomes much more balanced and transparent. In this project, these metrics showed that Logistic Regression was not only accurate overall but also consistent across all five categories, avoiding bias towards larger ones such as Sport or Business.

Classification Report and Confusion Matrix

Logistic Regression

Classification Report:

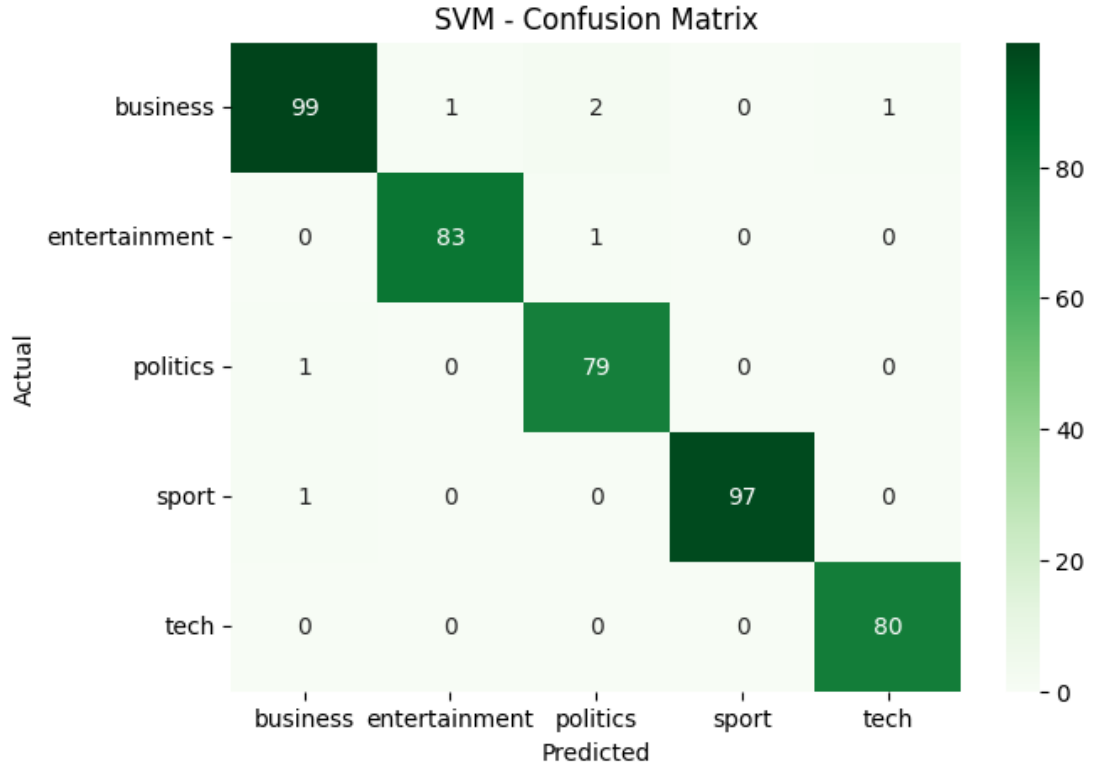
	precision	recall	f1-score	support
business	0.98	0.97	0.98	103
entertainment	1.00	0.96	0.98	84
politics	0.95	0.99	0.97	80
sport	1.00	0.99	0.99	98
tech	0.98	1.00	0.99	80
accuracy			0.98	445
macro avg	0.98	0.98	0.98	445
weighted avg	0.98	0.98	0.98	445



Supply Vector Machine (LinearSVC)

SVM (LinearSVC) Classification Report:

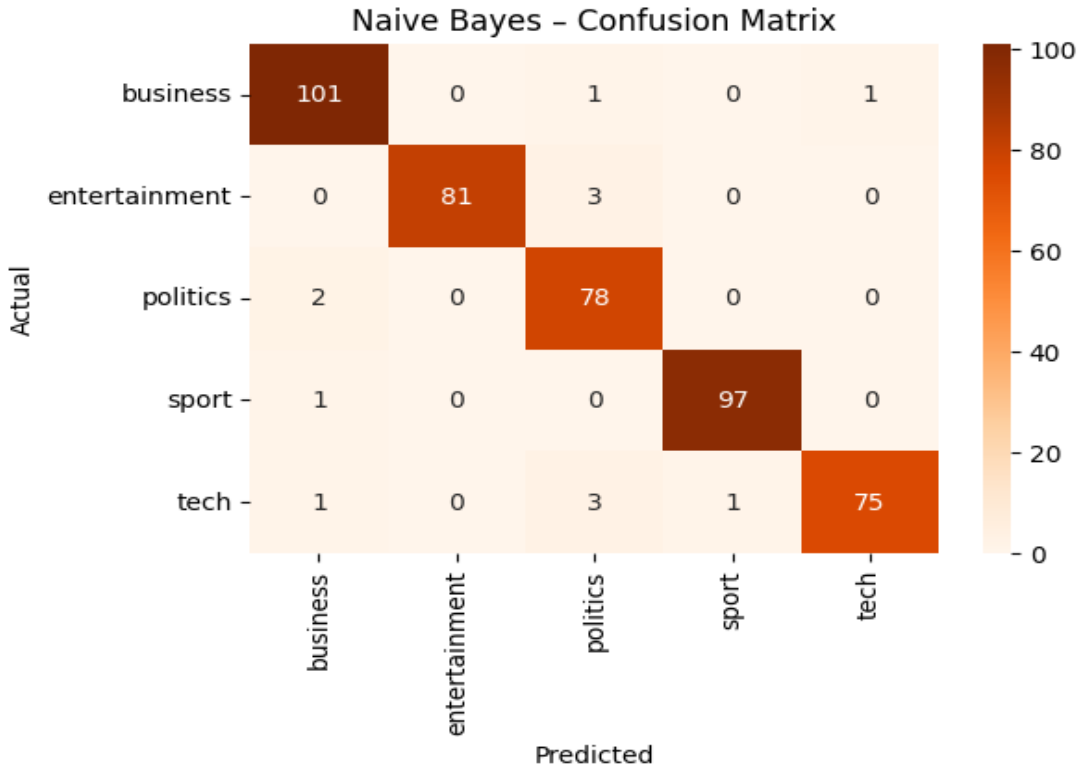
	precision	recall	f1-score	support
business	0.98	0.96	0.97	103
entertainment	0.99	0.99	0.99	84
politics	0.96	0.99	0.98	80
sport	1.00	0.99	0.99	98
tech	0.99	1.00	0.99	80
accuracy			0.98	445
macro avg	0.98	0.99	0.98	445
weighted avg	0.98	0.98	0.98	445



Multinomial Naive Bayes

Naive Bayes Report:

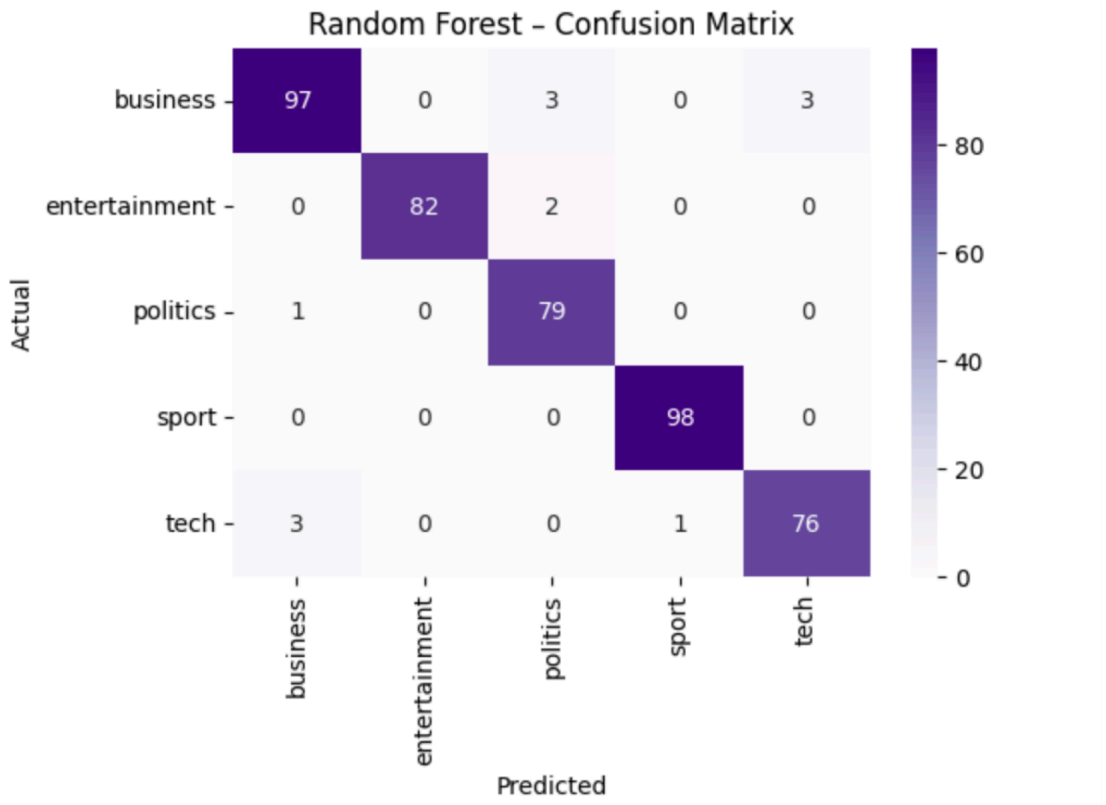
	precision	recall	f1-score	support
business	0.96	0.98	0.97	103
entertainment	1.00	0.96	0.98	84
politics	0.92	0.97	0.95	80
sport	0.99	0.99	0.99	98
tech	0.99	0.94	0.96	80
accuracy			0.97	445
macro avg	0.97	0.97	0.97	445
weighted avg	0.97	0.97	0.97	445



Random Forest

Random Forest Report:

	precision	recall	f1-score	support
business	0.96	0.94	0.95	103
entertainment	1.00	0.98	0.99	84
politics	0.94	0.99	0.96	80
sport	0.99	1.00	0.99	98
tech	0.96	0.95	0.96	80
accuracy			0.97	445
macro avg	0.97	0.97	0.97	445
weighted avg	0.97	0.97	0.97	445



4.7 Summary of Methodology

In summary, this project combined Natural Language Processing (NLP) preprocessing, TF-IDF feature extraction, and supervised machine learning models to address the research question. Four different models Logistic Regression, Support Vector Machine, Naive Bayes, and Random Forest were trained and evaluated systematically. Among them, Logistic Regression emerged as the best-performing model, offering the highest accuracy and balanced results across all categories.

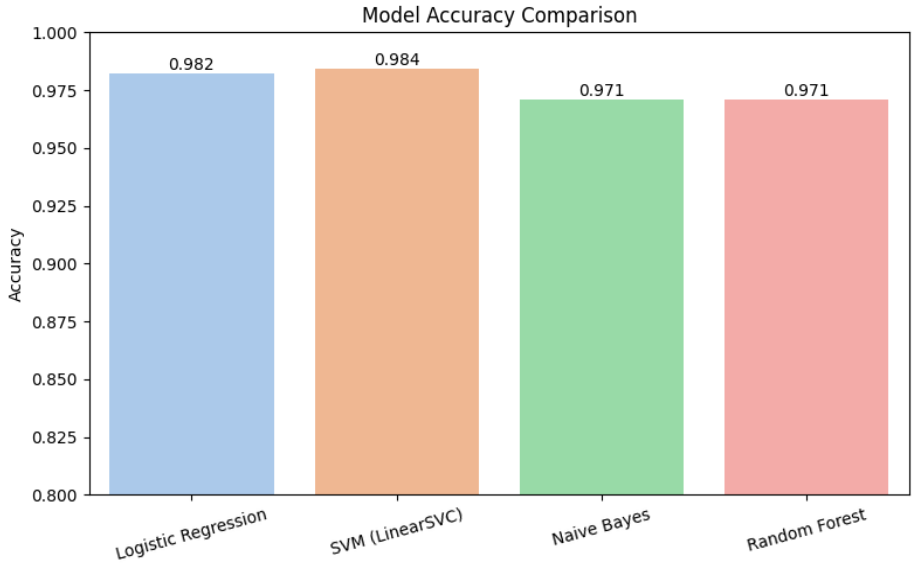
5. Results and Analysis

The models were trained and evaluated using the BBC News dataset. The performance of each model was assessed using accuracy, precision, recall, and F1-score.

The **Logistic Regression** model achieved the best overall performance with an accuracy of **98%** on the test set. The **Support Vector Machine (SVM)** followed closely with approximately **97.7% accuracy**, while **Random Forest** achieved around **96.5%**, and **Naive Bayes** recorded about **95%**. Although the differences in accuracy appear small, Logistic Regression consistently outperformed the others in terms of balanced precision and recall across all categories.

The **confusion matrix** for Logistic Regression confirmed that the model was able to classify articles correctly across all five categories with minimal misclassifications. For example, “Sport” and “Tech” articles achieved near-perfect recall, while “Business” and “Politics” showed slightly more overlap but still maintained very high accuracy.

The classification report further demonstrated that precision and recall values for all categories were above 0.95. This indicates that the model not only predicted most articles correctly but also avoided bias towards any single class. These results suggest that traditional linear models, when combined with effective preprocessing and TF-IDF feature extraction, remain extremely powerful for text classification tasks.



Bar Chart of Model Accuracy

6. Discussion

The findings of this project align closely with prior literature. Logistic Regression and SVM confirmed their reputation as strong performers in text classification tasks. The results are consistent with Joachims (1998), who demonstrated the strength of SVMs on sparse text data, and with Sebastiani (2002), who emphasized the effectiveness of linear models with bag-of-words or TF-IDF representations.

Naive Bayes, while simple, produced competitive results with 95% accuracy. This is in line with McCallum and Nigam (1998), who showed that Naive Bayes can be highly effective for text classification despite its independence assumption. Random Forest, on the other hand, was less effective compared to linear models, confirming that tree-based models are not always optimal in high-dimensional sparse spaces.

One of the strengths of this project was the use of multiple evaluation metrics rather than relying solely on accuracy. Precision, recall, and F1-score provided deeper insights into model performance across categories, ensuring that no single category was disproportionately advantaged. The confusion matrix also made it possible to identify specific misclassifications, which can be valuable for refining preprocessing steps in future work.

7. Future Work

While the results of this project are strong, there are several directions for improvement and extension. Future work could explore the use of **deep learning methods**, such as LSTMs or transformer-based models like BERT, which are capable of capturing contextual meaning beyond TF-IDF. With a larger dataset, these models may outperform traditional approaches.

Another possible extension is to experiment with **topic modelling** techniques such as Latent Dirichlet Allocation (LDA), which could uncover hidden thematic structures in the articles. Additionally, expanding the dataset to include more categories (e.g., health, environment, or world news) would provide a broader evaluation of model performance.

Finally, the trained model could be deployed as a **web-based application**, where users upload articles and receive instant categorization. This would demonstrate the practical value of the work and open possibilities for real-time applications.

8. Conclusion

This project set out to investigate whether machine learning models can automatically classify BBC news articles into predefined categories using TF-IDF features. The results clearly show that this is possible. With proper preprocessing and feature extraction, classical machine learning models can achieve very high accuracy on text classification tasks.

Among the four models compared, **Logistic Regression emerged as the best-performing classifier**, achieving **98% accuracy** and showing strong precision and recall across all categories. This finding reinforces the idea that linear models remain highly effective for text classification and can be used as reliable baselines even in the current era of deep learning. Models like SVM, Naive Bayes, and Random Forest also performed well, but Logistic Regression provided the most consistent balance between accuracy and fairness across the dataset.

The findings have practical implications for real-world applications, such as automated news categorization for online publishers, search engines, and content aggregation platforms. By reducing the need for manual tagging, such systems can improve efficiency, scalability, and consistency in handling large collections of articles. This demonstrates how data science solutions can directly support industries that deal with high volumes of unstructured text.

Another important outcome of this study is the confirmation that **data quality and preprocessing are just as important as the choice of model**. Simple techniques such as tokenization, stopword removal, and lemmatization, combined with TF-IDF, allowed relatively simple models to achieve excellent results. This shows that in many cases, well-prepared data and robust classical methods can compete with, or even outperform, more complex approaches.

This project also highlights the value of **systematic evaluation** using multiple performance metrics rather than relying on accuracy alone. By examining precision, recall, and F1-scores alongside the confusion matrix, it was possible to ensure that the model performed well across all categories, not just the majority ones. This balanced performance is crucial in real-world applications, where misclassifying smaller or less frequent categories could lead to biased or misleading results.

In addition, the study demonstrates the importance of finding the right balance between model simplicity and performance. Logistic Regression is a relatively lightweight model, yet it was able to produce results that are highly competitive. This means that organisations with limited computational resources can still achieve reliable results without having to adopt very complex or resource-intensive deep learning models.

Finally, this project highlights the wider value of data science in solving practical, real-world problems. Even with a relatively small dataset, careful design choices and systematic evaluation produced reliable results. This gives confidence that similar methods can be applied to other areas where large amounts of text need to be organised, such as healthcare, law, or customer service. By showing what is possible with clear objectives and structured methodology, this work lays a strong foundation for future projects that aim to combine machine learning with natural language processing.

9. References

- Joachims, T. (1998) 'Text categorization with Support Vector Machines: Learning with many relevant features', Proceedings of the 10th European Conference on Machine Learning (ECML), pp. 137–142.
  Available at: https://link.springer.com/chapter/10.1007/BFb0026683

- Kim, Y. (2014) 'Convolutional Neural Networks for Sentence Classification', Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1746–1751.
  Available at: https://aclanthology.org/D14-1181

- Kowsari, K., Meimandi, K.J., Heidarysafa, M., Mendu, S., Barnes, L. and Brown, D. (2019) 'Text classification algorithms: A survey', Information, 10(4), pp. 1–41.
  Available at: https://doi.org/10.3390/info10040150

- McCallum, A. and Nigam, K. (1998) 'A comparison of event models for Naive Bayes text classification', Proceedings of the AAAI Workshop on Learning for Text Categorization, pp. 41–48.
  Available at: https://www.aaai.org/Papers/Workshops/1998/WS-98-05/WS98-05-006.pdf

- Rennie, J.D., Shih, L., Teevan, J. and Karger, D.R. (2003) 'Tackling the poor assumptions of Naive Bayes text classifiers', Proceedings of the 20th International Conference on Machine Learning (ICML), pp. 616–623.
  Available at: https://www.cs.cmu.edu/~roni/11761/renn03a.pdf

- Sebastiani, F. (2002) 'Machine learning in automated text categorization', ACM Computing Surveys, 34(1), pp. 1–47.
  Available at: https://doi.org/10.1145/505282.505283

- Wang, S. and Manning, C. (2012) 'Baselines and bigrams: Simple, good sentiment and topic classification', Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL), pp. 90–94.
  Available at: https://aclanthology.org/P12-2018

- Yang, Y. and Liu, X. (1999) 'A re-examination of text categorization methods', Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 42–49.
  Available at: https://dl.acm.org/doi/10.1145/312624.312647

10. Appendices

Importing all required libraries for text preprocessing, feature extraction, model training, and evaluation.

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# NLP tools

import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import string
import re
import warnings
warnings.filterwarnings('ignore')

# Machine Learning models

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# Evaluation metrics

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

nltk.data.path.append('./nltk_data')# Importing all required libraries for text preprocessing, feature extraction, model training, and evaluation.

import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# NLP tools

import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import string
```

```
import re
import warnings
warnings.filterwarnings('ignore')

# Machine Learning models

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# Evaluation metrics

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

nltk.data.path.append('./nltk_data')

# Loading the BBC News dataset into a DataFrame for analysis.

data = []
labels = []

base_dir = "bbc"
categories = os.listdir(base_dir)

for category in categories:
folder_path = os.path.join(base_dir, category)
if os.path.isdir(folder_path):
for file_name in os.listdir(folder_path):
with open(os.path.join(folder_path, file_name), 'r', encoding='latin1') as f:
content = f.read()
data.append(content)
labels.append(category)

# Displaying the first few rows to understand the structure of the dataset.

df = pd.DataFrame({'text': data, 'label': labels})
df

# Defining a function to clean text by removing punctuation, stopwords, and applying
lemmatization.

stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def clean_text(text):
```

```
text = text.lower()
text = re.sub(r'\d+', '', text)
text = text.translate(str.maketrans('', '', string.punctuation))
tokens = text.split()
tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
return ' '.join(tokens)

df['clean_text'] = df['text'].apply(clean_text)
df.head()

# Shape of the data
df.shape

# Missing values
print(df.isnull().sum())

empty_texts = df[df['clean_text'].str.strip() == '']
print(f"Empty clean_text entries: {len(empty_texts)}")

# Bar plot for Distribution of new categories
label_counts = df['label'].value_counts()
print(label_counts)

sns.barplot(x=label_counts.index, y=label_counts.values)
plt.title("Distribution of News Categories")
plt.xlabel("Category")
plt.ylabel("Number of Articles")
plt.xticks(rotation=45)
plt.show()

# Displaying each content for each category after preprocessing
for category in df['label'].unique():
print(f"\nCategory: {category}")
print(df[df['label'] == category]['clean_text'].iloc[0][:300], "...")

# Histogram for Article Distribution by category
df['text_length'] = df['clean_text'].apply(lambda x: len(x.split()))

plt.figure(figsize=(8,5))
sns.histplot(df['text_length'], bins=30, kde=True)
plt.title("Distribution of Article Word Counts")
plt.xlabel("Number of Words")
plt.ylabel("Frequency")
plt.show()

# Statistical description
df['text_length'].describe()

# Box plot for Word count Distribution
plt.figure(figsize=(10,6))
```

```
sns.boxplot(x='label', y='text_length', data=df, palette='pastel')
plt.title("Word Count Distribution by Category")
plt.xlabel("Category")
plt.ylabel("Word Count")
plt.xticks(rotation=45)
plt.show()

# Pie-chart for Category Distribution
plt.figure(figsize=(6,6))
df['label'].value_counts().plot.pie(autopct='%1.1f%%',
colors=sns.color_palette('Set2'))
plt.title("BBC News Category Distribution")
plt.ylabel("")
plt.show()

# Converting cleaned text into numerical vectors using TF-IDF representation.
vectorizer = TfidfVectorizer(max_features=5000)

X = vectorizer.fit_transform(df['clean_text'])
y = df['label']

# Splitting the dataset into training and test sets for model evaluation.

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

print("Train size:", X_train.shape)
print("Test size:", X_test.shape)

# Training and evaluating Logistic Regression on the TF-IDF features.

logreg = LogisticRegression(max_iter=1000, class_weight='balanced',
solver='liblinear')

logreg.fit(X_train, y_train)

# Classification report and Confusion matrix for Logistic Regression
y_pred = logreg.predict(X_test)

print("Classification Report:\n")
print(classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(7,5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=logreg.classes_,
yticklabels=logreg.classes_)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

```
# Training and evaluating Support Vector Machine (LinearSVC) for classification.

svm_model = LinearSVC(class_weight='balanced')
svm_model.fit(X_train, y_train)

# Classification report and Confusion matrix for Support Vector Machine (LinearSVC)

y_pred_svm = svm_model.predict(X_test)

print("SVM (LinearSVC) Classification Report:\n")
print(classification_report(y_test, y_pred_svm))

cm_svm = confusion_matrix(y_test, y_pred_svm)

plt.figure(figsize=(7,5))
sns.heatmap(cm_svm, annot=True, fmt='d', cmap='Greens',
xticklabels=svm_model.classes_, yticklabels=svm_model.classes_)
plt.title("SVM - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Training and evaluating Naive Bayes classifier for text classification.

nb = MultinomialNB()
nb.fit(X_train, y_train)
y_pred_nb = nb.predict(X_test)

# Classification report and Confusion matrix for Naive Bayes classifier

print("Naive Bayes Report:")
print(classification_report(y_test, y_pred_nb))

cm_nb = confusion_matrix(y_test, y_pred_nb)
plt.figure(figsize=(6,4))
sns.heatmap(cm_nb, annot=True, fmt='d', cmap='Oranges', xticklabels=nb.classes_,
yticklabels=nb.classes_)
plt.title("Naive Bayes – Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Training and evaluating Random Forest classifier for text classification.

rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

# Classification report and Confusion matrix for Random Forest classifier
print("Random Forest Report:")
```

```
print(classification_report(y_test, y_pred_rf))

cm_rf = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize=(6,4))
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Purples', xticklabels=rf.classes_,
yticklabels=rf.classes_)
plt.title("Random Forest – Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Comparing the accuracy scores of all trained models using a bar chart.

model_scores = {
"Logistic Regression": accuracy_score(y_test, y_pred),
"SVM (LinearSVC)": accuracy_score(y_test, y_pred_svm),
"Naive Bayes": accuracy_score(y_test, y_pred_nb),
"Random Forest": accuracy_score(y_test, y_pred_rf),
}

# Bar chart comparison
plt.figure(figsize=(8,5))
ax = sns.barplot(x=list(model_scores.keys()), y=list(model_scores.values()),
palette='pastel')
plt.ylabel("Accuracy")
plt.ylim(0.8, 1.0)
plt.title("Model Accuracy Comparison")
for p in ax.patches:
height = p.get_height()
ax.annotate(f'{height:.3f}', (p.get_x() + p.get_width() / 2., height),
ha='center', va='bottom', fontsize=10, color='black')
plt.xticks(rotation=15)
plt.tight_layout()
plt.show()

# Using GridSearchCV to tune Logistic Regression hyperparameters such as C,
solver, and class weight to improve performance.

param_grid = {
'C': [0.01, 0.1, 1, 10],
'solver': ['liblinear', 'lbfgs'],
'class_weight': ['balanced', None]
}

grid = GridSearchCV(LogisticRegression(max_iter=1000), param_grid, cv=5,
scoring='accuracy')
grid.fit(X_train, y_train)

print("Best Parameters:", grid.best_params_)
print("Best CV Accuracy:", grid.best_score_)
```

```
y_pred_grid = grid.predict(X_test)
print("Test Accuracy:", accuracy_score(y_test, y_pred_grid))
```