Sentiment Analysis on Movie Reviews

Objective: The goal of this project is to perform sentiment analysis on a dataset of movie reviews. We aim to build and evaluate several machine learning models to classify a review as either **positive** or **negative**.

---

1. Data Loading and Exploration

The first step involves loading the dataset and understanding its structure and content. The data is in a CSV file and is loaded into a pandas DataFrame.

1.1. Dataset Overview

The dataset consists of two columns: `review` (the text of the movie review) and `sentiment` (the corresponding label, either 'positive' or 'negative').

A quick look at the first five rows:

	review	sentiment
0	A complete flop with no redeeming qualities.	negative
1	A masterpiece that I will watch again and again.	positive
2	The acting was terrible and the story was nons...	negative
3	What a great film! I would definitely recommen...	positive
4.	I absolutely loved this movie! The plot was fa...	positive

1.2. Data Information

The DataFrame contains **1000 entries** with no missing values. Both the `review` and `sentiment` columns are of the `object` data type.

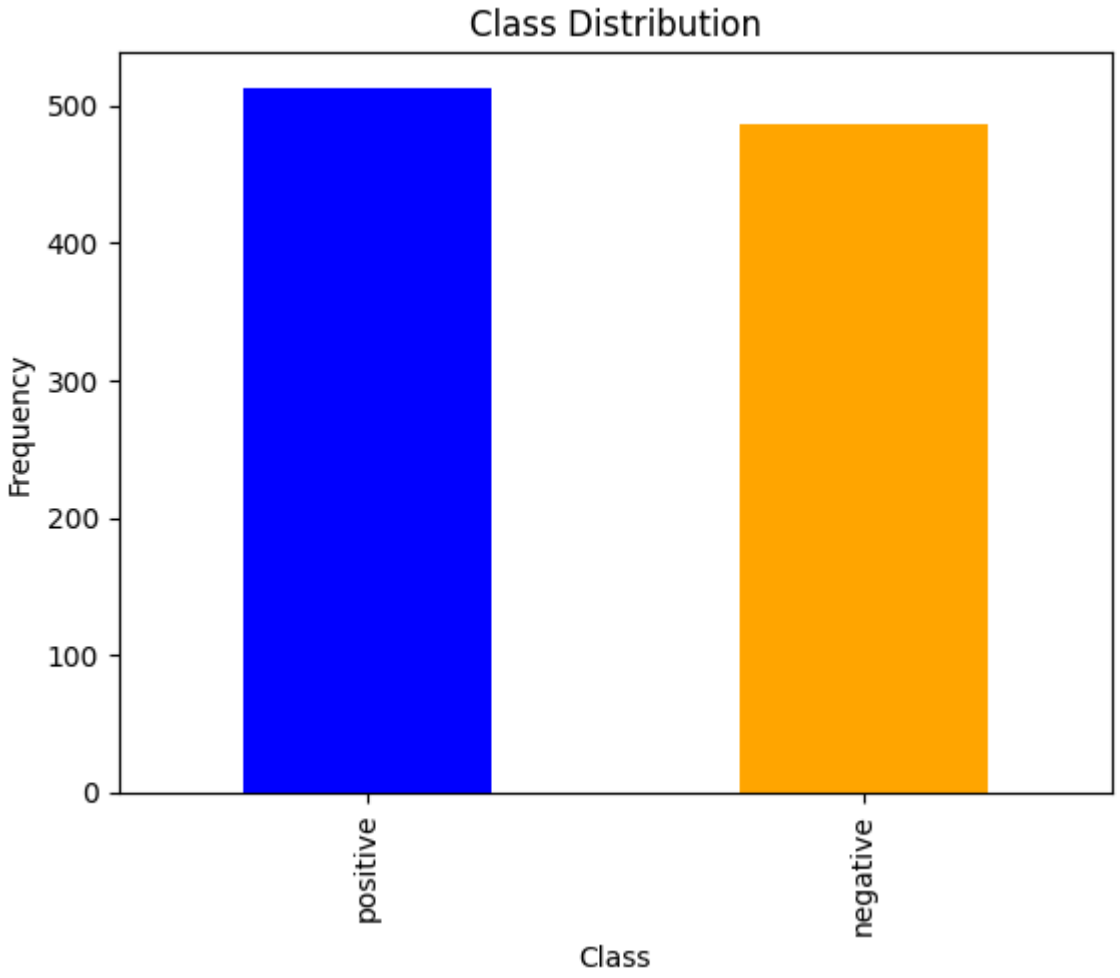
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   review     1000 non-null   object
 1   sentiment  1000 non-null   object
memory usage: 15.8+ KB
```

1.3. Class Distribution

The dataset is well-balanced, with a nearly equal number of positive and negative reviews. This balance is crucial for training an unbiased classifier.

- **Positive:** 513 reviews
- **Negative:** 487 reviews

The class distribution is visualized in the bar chart below:



---

2. Data Preprocessing and Feature Extraction

2.1. Text Cleaning

Before feeding the text data into our models, it needs to be cleaned and converted into a numerical format. The following cleaning steps were applied:

1. **Remove Non-alphanumeric Characters:** All characters that are not words were removed.
2. **Convert to Lowercase:** The entire text was converted to lowercase to ensure uniformity.
3. **Tokenization & Stopword Removal:** The text was split into individual words (tokens), and common English words that don't add much meaning (e.g., "the", "a", "is"), known as stopwords, were removed using the NLTK library.

Here is a sample of the reviews after cleaning:

```
0       complete flop redeeming qualities
1                 masterpiece watch
2       acting terrible story nonsensical
3    great film would definitely recommend
4    absolutely loved movie plot fantastic
Name: cleaned_review, dtype: object
```

2.2. Feature Extraction using TF-IDF

The cleaned text was transformed into numerical feature vectors using the **Term Frequency-Inverse Document Frequency (TF-IDF)** technique. `TfidfVectorizer` from scikit-learn was used for this, configured to consider the top 5000 most frequent words (`max_features=5000`).

The target variable `y` was also converted into a numerical format, mapping 'positive' to `1` and 'negative' to `0`.

2.3. Data Splitting

The dataset was split into training and testing sets, with **80% of the data used for training** the models and the remaining **20% for testing** their performance.

---

3. Model Building and Evaluation

Several classification models were trained on the preprocessed data. Their performance was evaluated on the test set using accuracy, a classification report (including precision, recall, and F1-score), and a confusion matrix.

3.1. Naive Bayes Classifiers

3.1.1. Gaussian Naive Bayes

This classifier is typically used for continuous data, but it was applied here to the TF-IDF features.

- **Accuracy:** 1.0000
- **Classification Report:**

	precision	recall	f1-score	support
0	1.00	1.00	1.00	94
1	1.00	1.00	1.00	106
accuracy			1.00	200
macro avg	1.00	1.00	1.00	200

weighted avg 1.00 1.00 1.00 200

- **Confusion Matrix:**
  [[94 0]

  [0 106]]

3.1.2. Multinomial Naive Bayes

This variant is well-suited for features with discrete counts, like word counts in text.

- **Accuracy:** 1.0000
- **Classification Report:**

	precision	recall	f1-score	support
0	1.00	1.00	1.00	94
1	1.00	1.00	1.00	106
accuracy			1.00	200
macro avg	1.00	1.00	1.00	200
weighted avg	1.00	1.00	1.00	200

- **Confusion Matrix:**
  [[94 0]

  [0 106]]

3.1.3. Bernoulli Naive Bayes

This model is effective for binary/boolean features. When used with TF-IDF, it considers whether a word is present or not.

- **Accuracy:** 1.0000
- **Classification Report:**

	precision	recall	f1-score	support
0	1.00	1.00	1.00	94
1	1.00	1.00	1.00	106
accuracy			1.00	200
macro avg	1.00	1.00	1.00	200
weighted avg	1.00	1.00	1.00	200

- **Confusion Matrix:**
  [[94 0]

  [0 106]]

3.2. Support Vector Machine (SVM)

3.2.1. Linear SVM

A standard SVM with a linear kernel was trained. The regularization parameter `C` was set to 1.

- **Accuracy:** 1.0000
- **Classification Report & Confusion Matrix:** The results were identical to the Naive Bayes models, showing perfect classification.

3.2.2. Soft vs. Hard Margin SVM

The effect of the regularization parameter `C` was explored. A small `C` (0.1) creates a "soft margin" SVM, while a large `C` (1000) creates a "hard margin" SVM. Both models achieved a perfect accuracy of 1.0000, indicating the data is linearly separable.

3.2.3. Hinge Loss

Hinge loss is the loss function used by SVMs. A value of 0 indicates that all samples were classified correctly.

- **Hinge Loss:** 0.0000

3.2.4. Kernelization (RBF vs. Polynomial)

To handle non-linear data, SVMs use kernels. The performance of the Radial Basis Function (RBF) and Polynomial (Poly) kernels was tested.

- **RBF SVM Accuracy:** 1.0000
- **Poly SVM Accuracy:** 1.0000

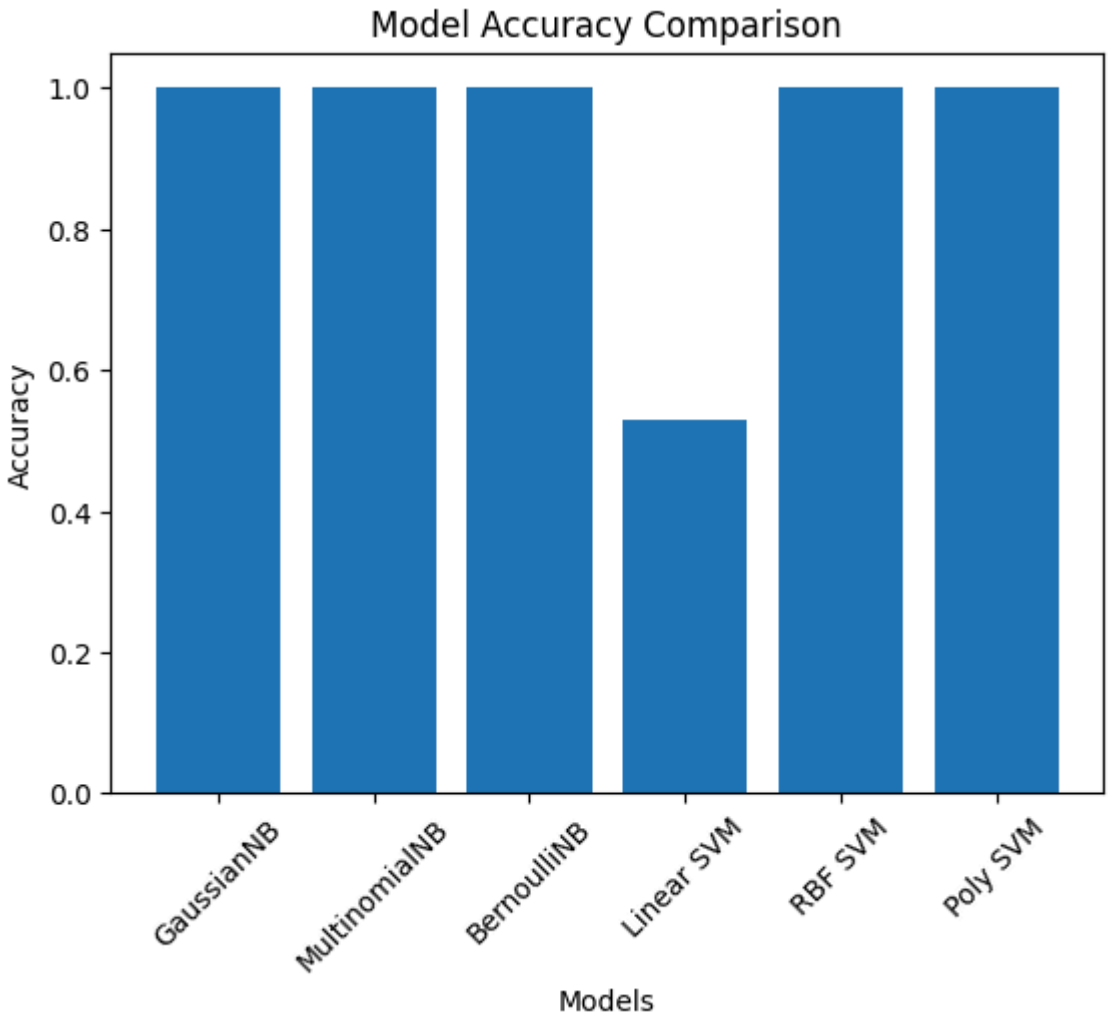
Both non-linear kernels also achieved perfect accuracy on the test set.

---

4. Model Comparison and Conclusion

4.1. Accuracy Comparison

A comparison of all models shows that every classifier achieved a perfect accuracy score of 100% on the test data.



4.2. Conclusion

In this sentiment analysis task, all the implemented models—Gaussian, Multinomial, and Bernoulli Naive Bayes, as well as Linear, RBF, and Polynomial SVMs—performed exceptionally well, achieving **100% accuracy**.

This perfect performance suggests that the dataset, after preprocessing, is linearly separable and relatively straightforward to classify. For this specific task, any of the tested models would be an excellent choice. The simplicity and speed of **Multinomial Naive Bayes** often make it a strong baseline and a preferred choice for text classification tasks like this one.