## Naive Bayes Algorithm

The Naive Bayes Algorithm is a simple method used for classification tasks in machine learning. It works on the principle of Bayes' Theorem, which calculates the probability of a category based on prior knowledge. The algorithm is especially popular for text classification problems such as spam detection, sentiment analysis, and document categorization. It works by determining the probability of each class given a set of features and then predicting the class with the highest probability. By calculating the likelihood of each class and picking the one with the highest probability, the Naive Bayes Algorithm delivers reliable and accurate predictions. Its design, ease of use, and ability to manage both simple and complex classification tasks make it a valuable tool for both beginners and experienced data scientists.

For implementing naive bayes algorithm, we firstly import libraries, and we perform algorithm for IMDB Data dataset.

```
In [1]: import pandas as pd
        import re
        import nltk
        from nltk.corpus import stopwords
        from nltk.stem import WordNetLemmatizer
        from sklearn.model_selection import train_test_split
        from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.naive_bayes import MultinomialNB
        from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
        from sklearn.pipeline import Pipeline
```

```
In [10]: nltk.download('punkt')
         nltk.download('stopwords')
         nltk.download('wordnet')
         import nltk
         nltk.download('omw-1.4')

         [nltk_data] Downloading package punkt to
         [nltk_data]     C:\Users\Rubina\AppData\Roaming\nltk_data...
         [nltk_data]   Package punkt is already up-to-date!
         [nltk_data] Downloading package stopwords to
         [nltk_data]     C:\Users\Rubina\AppData\Roaming\nltk_data...
         [nltk_data]   Package stopwords is already up-to-date!
         [nltk_data] Downloading package wordnet to
         [nltk_data]     C:\Users\Rubina\AppData\Roaming\nltk_data...
         [nltk_data]   Package wordnet is already up-to-date!
         [nltk_data] Downloading package omw-1.4 to
         [nltk_data]     C:\Users\Rubina\AppData\Roaming\nltk_data...
Out[10]: True

In [12]: data = pd.read_csv('IMDB Dataset.csv', encoding='utf-8')
```

This code snippet downloads several natural language processing (NLP) resources using the Natural Language Toolkit (nltk) library in Python. After downloading these resources, the code reads a CSV file named "IMDB Dataset.csv" that is encoded in UTF-8 format using the pandas library (pd). This dataset likely contains movie reviews or related data from IMDB.

## Text Preprocessing for Sentiment Analysis

This code snippet prepares text data for sentiment analysis of movie reviews from an IMDB dataset. It sets up a list of common English stopwords, like "the" and "is", which are removed to focus on meaningful words. It also initializes a lemmatizer to standardize words to their base form. The preprocess_text function processes each review by converting text to lowercase, removing non-letter characters, tokenizing into words, lemmatizing, and filtering stopwords. This ensures the text is cleaned and ready for analysis, improving the accuracy of sentiment analysis or other text-based tasks.

```
In [13]: stop_words = set(stopwords.words('english'))
         lemmatizer = WordNetLemmatizer()

In [14]: def preprocess_text(text):
             text = text.lower()
             text = re.sub(r'[^a-zA-Z\s]', '', text)
             tokens = nltk.word_tokenize(text)
             tokens = [lemmatizer.lemmatize(token) for token in tokens if token not in stop_words]
             return ' '.join(tokens)

In [15]: data['processed_review'] = data['review'].apply(preprocess_text)

In [17]: data
```

Out[17]:

| | review | sentiment | processed_review |
|---|---|---|---|
| 0 | One of the other reviewers has mentioned that ... | positive | one reviewer mentioned watching oz episode you... |
| 1 | A wonderful little production. <br /><br />The... | positive | wonderful little production br br filming tech... |
| 2 | I thought this was a wonderful way to spend ti... | positive | thought wonderful way spend time hot summer we... |
| 3 | Basically there's a family where a little boy ... | negative | basically there family little boy jake think t... |
| 4 | Petter Mattei's "Love in the Time of Money" is... | positive | petter matteis love time money visually stunni... |
| ... | ... | ... | ... |
| 49995 | I thought this movie did a down right good job... | positive | thought movie right good job wasnt creative or... |
| 49996 | Bad plot, bad dialogue, bad acting, idiotic di... | negative | bad plot bad dialogue bad acting idiotic direc... |
| 49997 | I am a Catholic taught in parochial elementary... | negative | catholic taught parochial elementary school nu... |
| 49998 | I'm going to have to disagree with the previou... | negative | im going disagree previous comment side maltin... |
| 49999 | No one expects the Star Trek movies to be high... | negative | one expects star trek movie high art fan expec... |

50000 rows × 3 columns

## Text Classification Pipeline Overview

This code snippet sets up a pipeline for sentiment analysis of IMDB movie reviews using machine learning. It first prepares the data by splitting the processed review texts (X) and their corresponding sentiment labels (y) into training and testing sets. The pipeline consists of two main stages: the TfidfVectorizer, which transforms text into numerical features using TF-IDF values, capturing both single words and pairs of consecutive words to better understand context (ngram_range=(1, 2)). It limits features to 5000 with minimum document frequency of 5 and maximum document frequency of 70% to filter out less informative terms. The second stage involves a Multinomial Naive Bayes classifier (MultinomialNB()) that learns from these transformed features to predict whether reviews convey positive or negative sentiment. This structured approach enables automated sentiment analysis by leveraging machine learning techniques to process and classify textual data effectively.

```
In [18]: X = data['processed_review']
         y = data['sentiment']
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

In [19]: pipeline = Pipeline([
             ('tfidf', TfidfVectorizer(ngram_range=(1, 2), max_features=5000, min_df=5, max_df=0.7)),
             ('classifier', MultinomialNB())
         ])

In [20]: pipeline.fit(X_train, y_train)
Out[20]: Pipeline(steps=[('tfidf',
                          TfidfVectorizer(max_df=0.7, max_features=5000, min_df=5,
                                          ngram_range=(1, 2))),
                         ('classifier', MultinomialNB())])
```

After training the sentiment analysis pipeline on IMDB movie reviews, this code snippet evaluates its performance on a test subset (X_test, y_test). First, it predicts sentiment labels (positive or negative) for the test reviews using pipeline.predict(X_test). The accuracy_score(y_test, y_pred) function computes and prints the accuracy of these predictions compared to the actual sentiment labels (y_test), indicating how often the model's predictions match the true labels.

Next, the code generates a confusion matrix using confusion_matrix(y_test, y_pred). This matrix provides a detailed breakdown of the model's predictions versus the actual labels, showing the number of true positives, true negatives, false positives, and false negatives. It helps assess the model's performance in correctly classifying positive and negative reviews.

Lastly, the classification_report(y_test, y_pred) function generates a report that includes precision, recall, and F1-score metrics for each sentiment class (positive and negative). Precision measures the accuracy of positive predictions, recall indicates the proportion of actual positives correctly identified, and the F1-score is the harmonic mean of precision and recall, offering a balanced assessment of the model's performance for each sentiment category. Together, these outputs provide comprehensive insights into how well the sentiment analysis model performs on unseen data, guiding further refinement or deployment decisions.

```
In [21]: y_pred = pipeline.predict(X_test)

In [22]: print("Accuracy:", accuracy_score(y_test, y_pred))
         print("\nConfusion Matrix:")
         print(confusion_matrix(y_test, y_pred))
         print("\nClassification Report:")
         print(classification_report(y_test, y_pred))

         Accuracy: 0.8564

         Confusion Matrix:
         [[4188  773]
          [ 663 4376]]

         Classification Report:
                       precision    recall  f1-score   support

             negative       0.86      0.84      0.85      4961
             positive       0.85      0.87      0.86      5039

             accuracy                           0.86     10000
            macro avg       0.86      0.86      0.86     10000
         weighted avg       0.86      0.86      0.86     10000
```

After generating confusion matrix, this code predicts whether three new movie reviews are positive or negative using a model trained on IMDB reviews. It first prepares the reviews by making them lowercase, removing unnecessary characters, and transforming them into a format the model can understand. Then, it uses the model to make predictions about the sentiment of each review. Finally, it displays each review alongside its predicted sentiment. This shows how the model can automatically analyze new reviews based on what it learned from past data, providing insights into whether the reviews are positive or negative.

In [23]:
```
new_reviews = [
    "This movie was excellent! I loved it.",
    "Terrible film. I hated every minute of it.",
    "An average movie, nothing special."
]
```

In [24]:
```
processed_new_reviews = [preprocess_text(review) for review in new_reviews]

new_predictions = pipeline.predict(processed_new_reviews)

for review, sentiment in zip(new_reviews, new_predictions):
    print(f"Review: {review}")
    print(f"Predicted Sentiment: {sentiment}\n")
```

```
Review: This movie was excellent! I loved it.
Predicted Sentiment: positive

Review: Terrible film. I hated every minute of it.
Predicted Sentiment: negative

Review: An average movie, nothing special.
Predicted Sentiment: negative
```

## Conclusion

Therefore, we developed and applied a sentiment analysis system for IMDB movie reviews. Beginning with text preprocessing to clean and standardize the data, we trained a model using TF-IDF vectorization and a Multinomial Naive Bayes classifier to predict whether reviews were positive or negative. Evaluating the model's performance involved assessing accuracy, constructing a confusion matrix to visualize errors, and generating a classification report for detailed metrics. Finally, we demonstrated the model's ability to predict sentiments for new reviews, highlighting its effectiveness in automating sentiment analysis tasks and extracting insights from textual data. it helped in analyzing and understanding sentiment in text-based content.