📃 Email Spam Detection using Naive Bayes (from Scratch)

1. Project Overview

This project implements a **Naive Bayes Classifier** from scratch using Python to distinguish between **spam** and **ham (non-spam)** emails. It demonstrates the use of probabilistic modeling for text classification and emphasizes hands-on implementation without relying on scikit-learn's classifiers.

2. Dataset

- **Source**: Spam Email Dataset from Kaggle
- **Columns**:
    - Category (spam or ham)
    - Message (content of the email)

The dataset is **imbalanced**, with many more ham messages than spam. To address this:

- The number of ham messages was **randomly downsampled** to match the number of spam messages.
- This created a **balanced dataset**, reducing bias during training.

3. Preprocessing

To prepare the data for classification:

1. **Lowercase Conversion**: All characters in messages are converted to lowercase.
2. **Punctuation Removal**: Punctuation is removed using str.translate().
3. **Tokenization**: Messages are split into tokens (words) using split().

4. Exploratory Data Analysis

Basic information and statistics were displayed using:

- `df.head()`
- `df.info()`
- `df.describe()`
- `df['Category'].value_counts()`

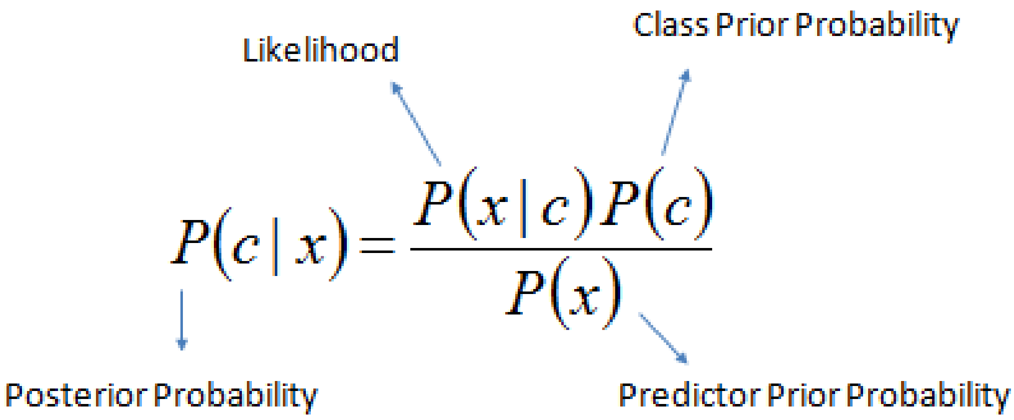
The number of ham and spam messages was determined.

After downsampling, the dataset was shuffled and reset using:

```
df_balanced = pd.concat([ham_sample, spam_df]).sample(frac=1,
random_state=42).reset_index(drop=True)
```

5. Naive Bayes Algorithm

Naive Bayes is a probabilistic classifier based on **Bayes' Theorem**:



Where:

- *C* is the class (spam or ham)
- *X* is a given email (a set of words)

Assuming **feature independence** (hence the "naive"), the model calculates the likelihood of an email being spam or ham by multiplying the conditional probabilities of each word given the class.

5.1. Training Phase

- For each class (ham, spam), the following were computed:
  - Word frequency dictionaries
  - Message count
  - Total word count

```
word_counts = {'ham': defaultdict(int), 'spam': defaultdict(int)}

message_counts = {'ham': 0, 'spam': 0}

total_words = {'ham': 0, 'spam': 0}
```

Each message's tokens were iterated over, and the respective counters were updated.

5.2. Probability Estimation with Laplace Smoothing

- Laplace smoothing is applied:

$$\hat{P}(w_i \mid c) = \frac{count(w_i, c) + 1}{\sum_{w \in V} \left(count(w, c) + 1\right)}$$

$$= \frac{count(w_i, c) + 1}{\left(\sum_{w \in V} count(w, c)\right) + |V|}$$

This avoids assigning zero probability to unknown words.

6. Prediction Function

The function predict() takes a new email message and:

1. Preprocesses and tokenizes it
2. Computes log-prior for both classes
3. Adds the log-likelihood of each word
4. If a word was not seen in training, a small smoothed probability is used
5. Returns the class with the higher total log-probability

```
return 'ham' if log_prob_ham > log_prob_spam else 'spam'
```

Also prints intermediate values:

Log P(Ham): -45.40406366418128

Log P(Spam): -38.771398106968206

Since log P(Spam) is higher (less negative), the model predicts:

Predicted label: **spam**