# Task 3.3: Spam Detection/Classification

**Preprocessing Steps:**

The data may contain a lot of noise and unwanted character such as punctuation, white space, numbers, hyperlink and etc. so Data Cleaning is necessary in any ML model. We can use:

1. Lowercasing: Convert all text to lowercase to ensure consistent tokenization and easier access to checks.
2. Removing Non-ASCII Characters: Remove emojis and any non-ASCII characters that might not contribute to the meaning of the text.
3. Removing numbers: Keywords are the ones that will play a major role in spam detection, unlike numbers which will not, so we can remove numbers to reduce the noise.
4. Removing Hyperlink
5. Tokenization: Split the text into individual words or tokens. This step is crucial as it breaks down the text into smaller units for further analysis.
6. Removing Punctuation: Remove punctuation marks, as they usually do not carry significant meaning in the context of spam detection.
7. Stop word Removal: Words like “the”, “and”, “is”, etc. have a very high frequency in messages that do not carry much information. These words can be found in language-specific stop word and should be removed due to their irrelevance to our goal.
8. Stemming/Lemmatization: Apply stemming or lemmatization to reduce words to their base or root form. This helps in treating different forms of the same word as equivalent. NLTK Library allows for easy application of these algorithms, so we would not need to write that complicated code ourselves every time.

**Feature Extraction:**

After preprocessing, we need to convert the text into numerical representations that machine learning algorithms can work with.

Count Vectorizer the classic way to count word frequency in a text, but is It is a good idea to use the Term Frequency-Inverse Document Frequency (TF-IDF) vectorization technique. This technique calculates a value for each word in each text block, based on how often the word appears in the block and how unique it is across the entire dataset, and it will downscale words appearing across multiple documents very frequently such as “the”, “and” etc.

Word embedding is converting a word to a vectorized format and this vector represents the position of this word in a higher dimensional space. For words that have similar meaning, the cosine distance of those two word-vectors will be shorter and they will be closer to each other, which will allow us to perform operations on them.

**Classification:**

The most favoured algorithm for text classification that includes a high-dimensional training dataset is the Naïve-Bayes classifier, which we will apply on the TFIDF Frequency Vectorizer.

The dataset will be split into training and testing sets to evaluate the model's performance accurately. The training set is used to train the model, and the testing set helps assess its ability to classify new, unseen data. We must make sure the training data and testing data have a similar ratio of distribution of spam vs ham messages when compared to the overall dataset.

Naïve Bayes algorithm is a supervised learning classification algorithm, called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features.

It is based on Bayes theorem:



P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

(For my understanding:

P(Yes|Sunny)= P(Sunny|Yes)\*P(Yes)/P(Sunny)

P(Yes|Sunny) means probability of saying yes to going out if it is sunny

P(Sunny|Yes) means the probability of it being sunny when yes is said to going out)

This algorithm will check if it is more probable for a message to be in the spam class or in the ham class, and will classify the message accordingly.

**Challenges and Considerations:**

1. Imbalanced Data: If there is a severe imbalance between the number of spam and non-spam instances, the model might favour the majority class. Techniques like oversampling the minority class will help with this.
2. Language and Context: The effectiveness of the model can vary based on the language used in the text and the specific context of the messages. Slang, cultural references, and specific language use might affect the model's accuracy.
3. Evolution of Spam: Spam tactics constantly evolve. Regular retraining with updated data is essential to keeping the performance of the model strong.
4. False Positives/Negatives: No model is perfect, and there will be cases where spam is misclassified as legitimate (false negatives) or legitimate messages are flagged as spam (false positives). Hard Ham messages, which contain certain keywords like limited time offer and are legitimate messages are easy to misclassify.
5. Complex Spam Techniques: Some spammers use sophisticated techniques to bypass spam filters, including intentional misspellings of the keywords, or image-based spam. These techniques will challenge a model's ability to classify spam.