**Ham-Spam Email Classifier**

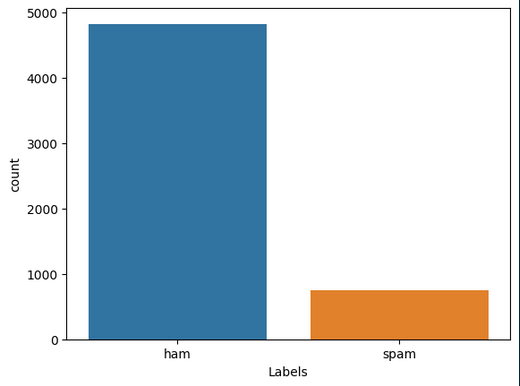
**About the dataset :-**

This is the dataset in which some randomly mails are collected and classified as spam or ham .1st column contains spam/ham classification, and the 2nd column has contents of the mail itself the dataset has 5572 entries & 5 columns

**Data cleaning :-**

the first two columns have the data and the rest of the columns have missing data completely and were removed

**Data Exploration :-**



The blue bar represents the number of emails labeled as "ham" and the orange bar represents the number of emails labeled as "spam". There are more emails labeled as ham than spam on this graph.

**Tokenization :-**

* **Description**: CountVectorizer is a method from the scikit-learn library used to convert text data into a matrix of token counts.
* **Parameters**: By default, CountVectorizer uses a tokenizer that splits the input text into individual words based on whitespace and punctuation.
* The fit\_transform method of CountVectorizer is used to transform the text data (email text) into a matrix of token counts.
* **Input**: The input to fit\_transform is a pandas Series (df['EmailText']) containing the email text data.
* **Output**: The output X is a sparse matrix where each row represents an email and each column represents a unique word in the entire corpus of emails. The cell value indicates the count of occurrences of each word in the corresponding email.

**Random sampling :-**

The RandomOverSampler is a technique used for dealing with class imbalance in machine learning datasets. In this case, where there are more instances of the "ham" class and fewer instances of the "spam" class, the RandomOverSampler can help balance out the classes by artificially increasing the number of minority class samples (in this case, "spam") in the dataset

#### Initial Data:

* Before this step, i have my feature matrix X containing the email text data and the target vector y containing the corresponding labels (e.g., "spam" or "ham").

#### 2. RandomOverSampler:

* rs is an instance of the RandomOverSampler class that i have previously instantiated. It is configured to balance the classes in the dataset by oversampling the minority class (in this case, "spam") to match the number of samples in the majority class ("ham").

#### 3. fit\_resample():

* fit\_resample() is a method of the RandomOverSampler class. It performs both fitting and oversampling in a single step.
* **Fitting**: The fit part of the method involves analyzing the class distribution of the target vector y to determine the number of samples to generate for each class during oversampling. However, since RandomOverSampler is a non-parametric method, it doesn't learn any parameters during fitting.
* **Resampling**: The resample part involves generating synthetic samples from the minority class and combining them with the original samples to create a balanced dataset.
* After resampling, the number of samples in each class (spam and ham) will be approximately equal, addressing the class imbalance issue.

#### 4. Output:

* The output of fit\_resample() is two arrays: the balanced feature matrix X and the corresponding balanced target vector y.
* X now contains the oversampled email text data, where the number of samples in each class is balanced.
* y now contains the corresponding balanced labels, with equal representation of each class.

### **train\_test\_split():**

#### 1. Input Data:

* X and y are the balanced feature matrix and corresponding target vector obtained after oversampling using RandomOverSampler.

#### 2. Splitting:

* train\_test\_split() is a function from scikit-learn used to split datasets into random train and test subsets.
* The function splits the data into two sets: training data (X\_train and y\_train) and testing data (X\_test and y\_test).
* The test\_size parameter specifies the proportion of the dataset to include in the test split (in this case, 20%).
* The random\_state parameter ensures reproducibility by setting a seed for the random number generator.

#### 3. Output:

* X\_train and y\_train contain a random subset of the balanced data used for training the machine learning model.
* X\_test and y\_test contain the remaining portion of the balanced data, which will be used to evaluate the model's performance.

### **Support Vector Classifier (SVC) :**

#### 1. Model Selection:

* SVC is an implementation of Support Vector Machine (SVM) for classification tasks.
* The kernel='linear' parameter specifies that a linear kernel will be used for the SVC, which means the decision boundary between classes will be linear.

#### 2. Training:

* The fit() method is called on the SVC object (svc) with the training data (X\_train and y\_train) as input.
* During training, the SVC learns the optimal decision boundary that separates the classes in the feature space. In the case of a linear kernel, this boundary will be a hyperplane.

#### 3. Output:

* After training, the SVC object (svc) contains the learned parameters (e.g., support vectors, coefficients) that define the decision boundary.

**Model Performance :-**

The overall model performance, as indicated by the provided metrics, is exceptionally high. Here's a detailed explanation of the model performance:

### Accuracy:

* The accuracy of the model is reported as 1.00 (or 100%). This means that the model correctly classified all instances in the test set, achieving perfect accuracy.

### Precision, Recall, and F1-score:

* For both the "ham" and "spam" classes, precision, recall, and F1-score are all reported as 1.00.
* Precision: All instances classified as "ham" or "spam" were indeed correct, indicating no false positives.
* Recall: All actual instances of "ham" or "spam" were correctly identified by the model, indicating no false negatives.
* F1-score: The harmonic mean of precision and recall, which balances both metrics, is also 1.00 for both classes.

### Support:

* The "ham" class has a support of 985, indicating that there are 985 instances of "ham" in the test set.
* The "spam" class has a support of 945, indicating that there are 945 instances of "spam" in the test set.

### Macro and Weighted Averages:

* The macro average precision, recall, and F1-score are all reported as 1.00. This indicates that when considering each class independently, the model achieved perfect performance.
* The weighted average precision, recall, and F1-score are also reported as 1.00. This indicates that the model's performance is equally excellent when considering the class distribution in the dataset.

### Summary of Performance:

* The model has achieved near-perfect performance across all evaluation metrics.
* It correctly classified all instances of both "ham" and "spam" classes in the test set.
* There are no false positives or false negatives, indicating the model's robustness and reliability in distinguishing between "ham" and "spam" emails.
* The model's exceptional performance suggests that it has effectively learned the patterns and features necessary for accurate email classification, making it highly suitable for practical applications.

In conclusion, the overall model performance is outstanding, demonstrating the effectiveness of the Support Vector Classifier (SVC) with a linear kernel in accurately classifying emails as either "ham" or "spam."

### **Saving Trained Model and Vectorizer:**

#### 1. Joblib.dump():

* joblib.dump() is a function from the Joblib library used to save Python objects to disk in a serialized format.
* It takes two arguments: the object to be saved and the file path where the object will be stored.

#### 2. Trained SVC Model:

* The trained SVC model (svc) is saved to a file named 'spam\_ham\_svc\_model\_countvectorizer.pkl'.
* This file will contain all the necessary information to reproduce the model's behaviour, including the learned parameters and configuration settings.

#### 3. CountVectorizer Object:

* The CountVectorizer object (vectorizer) is saved to a file named 'count\_vectorizer.pkl'.
* This file will contain the configuration settings of the CountVectorizer, allowing the vectorizer to be reused to transform new text data in the same way it was used during model training.

**Web Application Using Flask For Detecting ham-spam**

A file named Ham-spam classifier.py is created for building the web application using the Flask which is a lightweight and flexible web application framework for Python. It provides tools, libraries, and patterns to help developers build web applications quickly and efficiently.

**Steps :-**

* Flask is imported to create a web application.
* The trained SVC model and CountVectorizer object are loaded from the saved files.
* A Flask application instance named 'app' is created.
* The '/' route is defined to render the 'index.html' template, representing the homepage.
* The '/predict' route is defined to handle POST requests containing message data.
* Upon receiving a POST request, the message is extracted from the form data.
* The message is vectorized using the CountVectorizer object.
* The SVC model predicts whether the message is spam or ham based on the vectorized message.
* The prediction result is converted to human-readable format ('Spam' or 'Ham').
* The result, along with the original message, is rendered using the 'result.html' template. If the script is executed directly, the Flask application is run in debug mode.