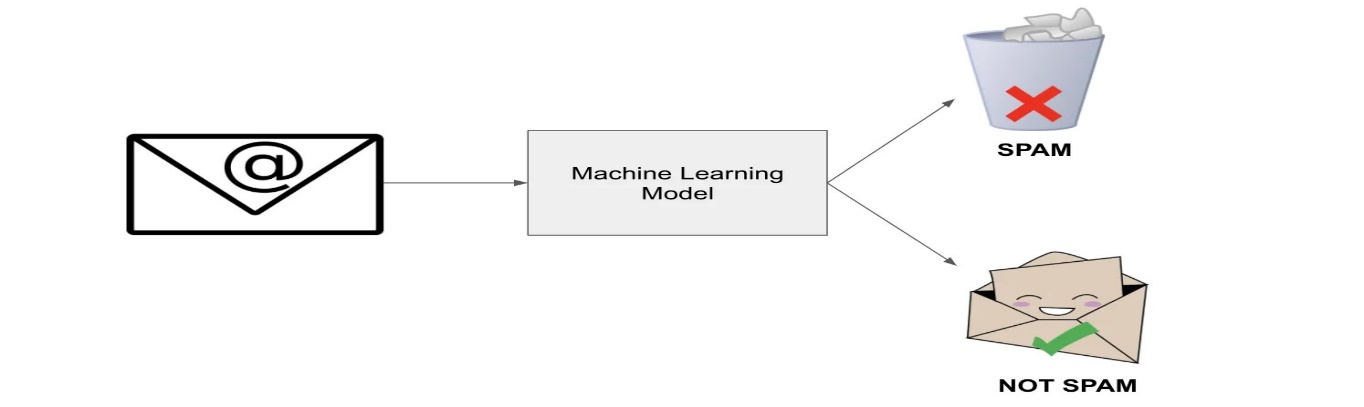
Minor project

Name: G karthik reddy

domain: Artificial intelligence

Topic- Naïve Bayes Classification : Spam Email Detection

Classifier to identify spam emails from legitimate ones



**What is Naive Bayes Classification?**

In 1763, the English statistician and philosopher Thomas Bayes proposed the Bayes Theorem, which serves as the fundamental principle of conditional probability. This theorem states that the likelihood of an event occurring, given the occurrence of another event, is equal to the conditional probability of the second event given the first event, multiplied by the probability of the first event itself.

Naive Bayes is a popular classification approach that is rooted in Bayes' theory. The posterior class probability of a test data point can be calculated using class-conditional density estimation and class prior probability. The test data will then be assigned to the class with the highest posterior class probability.

**Why is it called Naive Bayes ?**

This classification methodology makes a naive assumption that features are independent of each other. For instance, consider the Titanic Dataset. When classifying using naive Bayes, we assume that data labels like age, gender, class, and cabin are all independent of each other.

**Where can we use Naive bayes Classification ?**

A few interesting projects could be :

* Spam Detection
* Character Recognition
* Whether Prediction
* News Article Catagorization
* Face Detection

1. **Problem Statement**

Build a machine learning-based Spam Email Classifier that automatically identifies whether an email is spam or not. The project will focus on feature engineering, model selection, optimization, and evaluation for real-world applications.

1. **Goals and Objectives**

* **Core Goals:**
* Classify emails as spam or non-spam based on textual content.
* Optimize accuracy, precision, and recall to minimize false positives and false negatives.
* **Advanced Goals:**
* Explore multiple algorithms (e.g., Logistic Regression, Naive Bayes, SVM, Random Forest).
* Implement ensemble methods for better performance.
* Visualize the data and results to provide meaningful insights.
* Create a user-friendly interface for deploying the classifier.

1. **Working Process**

**Step 1: Data Collection**

* **Datasets:**
* **Spam Assassin Public Corpus:** A real-world dataset with labeled emails.
* **Kaggle SMS Spam Collection:** A smaller, structured dataset.
* **Custom Emails:** Add manually labeled data for diversity and customization.

**Step 2: Exploratory Data Analysis (EDA)**

* **Analyze the dataset for:**
* Class imbalance (spam vs. non-spam email counts).
* Word frequency distribution (e.g., common words in spam emails).
* Length of emails or SMS messages.
* **Visualizations:**
* Word clouds for spam and non-spam emails.
* Histograms of word counts or email lengths.
* Boxplots showing outliers in spam/non-spam text lengths.

**Step 3: Preprocessing**

* **Defination** : Data preprocessing is the act of cleaning, converting, and organizing raw data such that it may be fed into a machine learning or data analysis algorithm in a more useable and structured shape.

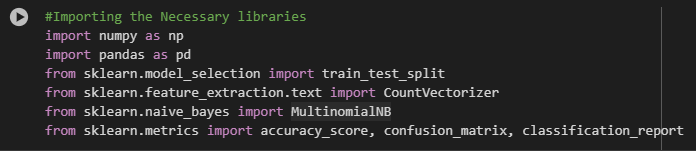
**why is it necessary?**

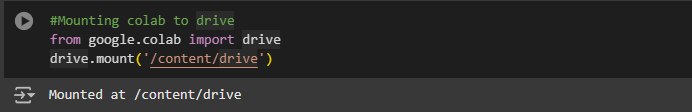
* **Quality assurance:** Raw data may have errors, inconsistencies, or missing numbers. By addressing these challenges, preprocessing ensures data quality.
* **Better Results:** Accurate, dependable insights are generated by good data. Clean, well-organized data helps algorithms function better.
* **Feature Engineering:** By combining existing features, you can construct new, useful ones that improve the model's capacity to grasp the data.
* **Reduced Noise:** Outliers or extreme values might cause results to be distorted. Preprocessing assists in identifying and dealing with them.
* **Standardization:** Different data sources may have varying units or scales. Data is more similar after preprocessing.
* **Missing Values:** Algorithms may struggle to handle missing values. Preprocessing aids in the filling or removal of missing data.
* **Efficiency:** Preparing data correctly saves time and computational resources during analysis.
* **Text Cleaning:**
* Remove HTML tags, URLs, special characters, and stopwords.
* Convert text to lowercase.
* **Tokenization and Lemmatization**:
* Use NLTK or spaCy for splitting text into meaningful words and normalizing them.
* **Feature Extraction:**
* Implement Bag of Words (BoW) and TF-IDF methods.
* Explore word embeddings (e.g., Word2Vec, GloVe) for better contextual understanding.

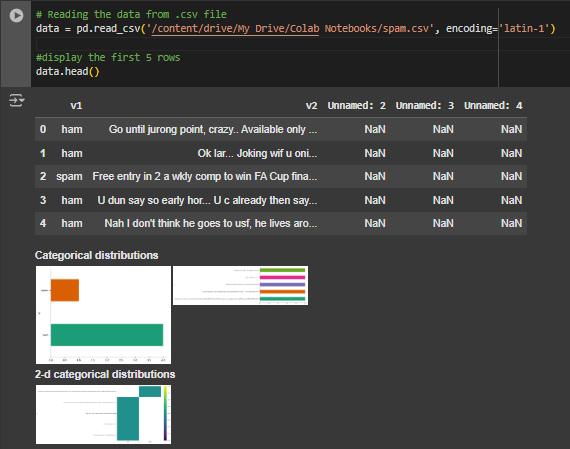
**Step 4: Model Development**

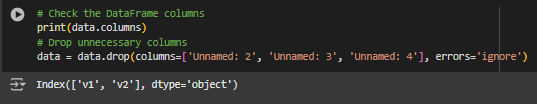
* **Algorithms to Explore:**
* **Naive Bayes:** Baseline model for text classification.
* **Logistic Regression:** For linear separation of spam and non-spam emails.
* **Support Vector Machines (SVM):** For high-dimensional feature spaces.
* **Ensemble Methods:**
* Random Forest
* Gradient Boosting (e.g., XGBoost, LightGBM)
* **Hyperparameter Tuning:**
* Use GridSearchCV or RandomizedSearchCV to find optimal hyperparameters.
* **Training the Model:**
* Split the dataset into training and testing sets (e.g., 80% training, 20% testing).
* Train the model using the training dataset.
* **Model Comparison:**
* Compare models based on metrics like accuracy, precision, recall, F1-score, and ROC-AUC.

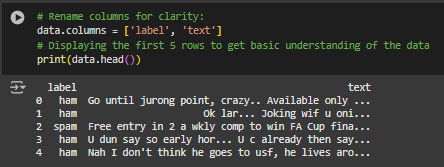
**Step 5: Evaluation and Insights**

* Evaluate the models on test data and generate detailed reports.
* Plot confusion matrices, ROC curves, and feature importance charts.
* **Code**





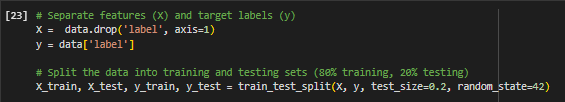




* **Separate Features and Target Labels**

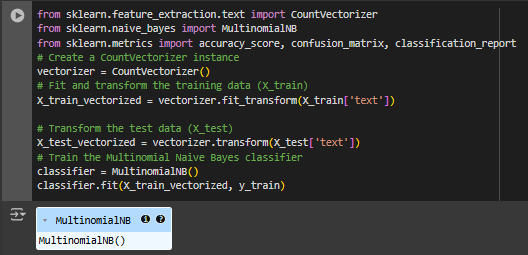
A typical dataset consists of input features and corresponding target labels. The input features are the attributes or variables that are used to make predictions, while the target labels are the values we are trying to predict.

* **Key Terms :**
* **train\_test\_split:** This function from the sklearn.model\_selection module is used to split the data into training and testing sets.
* **X\_train:** This variable holds the subset of input features that will be used for training the model.
* **X\_test:** This variable holds the subset of input features that will be used for testing the model.
* **y\_train:** This variable holds the corresponding target labels for the training set.
* **y\_test:** This variable holds the corresponding target labels for the testing set.
* **test\_size=0.2:** This parameter indicates that 20% of the data will be allocated for testing, and the remaining 80% will be used for training.
* **random\_state=42:** This parameter is used to seed the random number generator, ensuring that the data is split in a reproducible manner. Using the same seed will produce the same split each time you run the code.



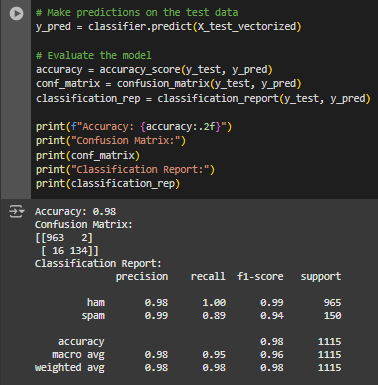
* **Train the Classifier (Multinomial Naive Bayes)**

**Why are we performing Count vectorization?**

****We're utilizing the MultinomialNB() classifier for this project, which exclusively accepts numeric values. However, our X\_train and X\_test datasets comprise text data (email messages). This is where CountVectorizer() comes in. It is being used here to convert the provided text into a vector, considering the frequency (count) of each word appearing throughout the entire text. This transformation is essential to enable the classifier to work with the text data effectively.

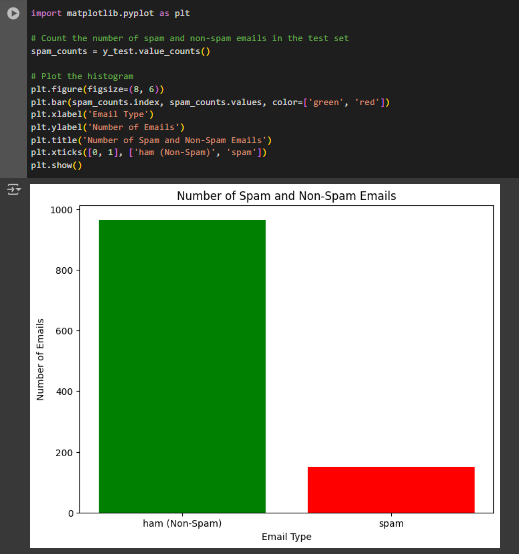
* **Make Predictions on the Test Data**

In this step, we are predicting the accuracy of our model by evaluating how precisely it can predict outcomes on new, unseen data.



* **Visualizing the Data**

Understanding raw numbers or datasets can often be challenging. Therefore, it is crucial to visually represent our data. By generating visual representations of data, complex patterns, trends, and relationships become easier to comprehend than when dealing with raw numbers alone. Visualization also aids in identifying anomalies within the data. In the code snippet below, we have visualized the data using a histogram that displays the distribution of spam and non-spam emails.



* **Conclusion**

The Spam Email Detection Project successfully demonstrated the application of text classification techniques to identify spam emails with high accuracy. By preprocessing the data using techniques such as text cleaning, tokenization, and feature extraction (TF-IDF), and implementing machine learning algorithms like Naive Bayes and SVM, the classifier effectively distinguished between spam and non-spam emails.

The performance of the model was evaluated using metrics like accuracy, precision, recall, and F1-score, ensuring its reliability in real-world scenarios. The project highlights the importance of handling imbalanced datasets and optimizing feature extraction methods for better results.

This work lays the foundation for deploying a scalable spam detection system, showcasing the potential of machine learning to enhance email security and improve user experience.

* **References**
* Google
* chatgpt