LONDON METROPOLITAN UNIVERSITY

Islington college
(इस्लिङ्टन कलेज)

Module Code & Module Title

CU6051NI - Artificial Intelligence

Assessment Weightage & Type

75% Individual Coursework

Year and Semester

2022 Autumn

Student Name: Rishabh Singh

London Met ID: 20048981

College ID: NP01CP4S210251

Assignment Due Date: 11th January 2023

Assignment Submission Date:11th January 2023

I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a marks of zero will be awarde

Table of Contents

Table of figures

1. Introduction

AI is a field of computer science that involves creating systems that can perform tasks that typically require human intelligence, such as understanding natural language, recognizing images, and making decisions. AI technologies are being used in a variety of applications, such as self-driving cars, personal assistants, and healthcare applications. According to the World Economic Forum, artificial intelligence (AI) will be a $13 trillion business by 2030. AI is all around us in today's environment. We are constantly exposed to AI, from simple chatbots to voice assistants like Alexa and Siri. Many of these AI systems run in the background without our knowledge. AI can perform tasks normally performed by humans. Because AI replicates human behavior, it has been hailed as the future of many industries like healthcare and cybersecurity. (Dunham, 2022)

Machine learning is a subfield of computer science that deals with the design and implementation of algorithms for learning from data. This article will give you a basic introduction to the field, covering what machine learning is, how it works and some applications of machine learning. What is machine learning? The concept of utilizing computers to automatically learn from experience and get better without explicit programming is known as machine learning. It is a branch of artificial intelligence in which computer algorithms can learn from data and make predictions and decisions without being programmed in. An algorithm is basically a set of rules forsolving a problem. (Selig, 2022)There are different kind of machine learning they are follows:

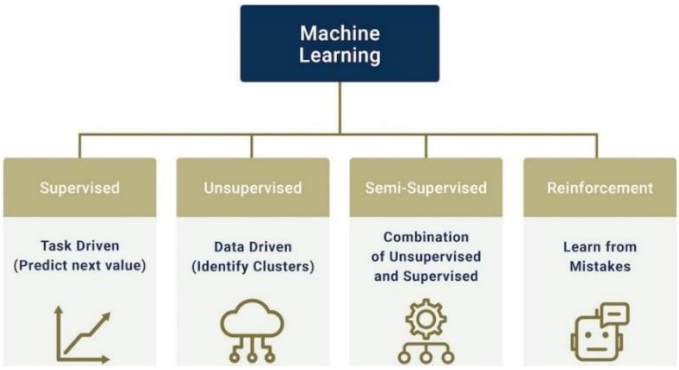


Figure 1: Types of machine learning

- Unsupervised machine learning: In this machine learning technique it does not require supervision and predicts the output using trained unlabeled dataset.
- Supervised machine learning: In this machine learning technique the data is trained using labeled dataset and based on that the machine predicts the output.
- Semi supervised: Semi supervised falls between supervised and unsupervised machine learning techniques it uses both labeled and unlabeled datasets during the training phase. But it mostly consists of unlabeled data (jav, 2022)
- Reinforcement learning: It is a machine learning technique where it learns to behave in an environment with trial and error after every positive outcome it gets feedback and for negative outcome it gets negative feedback.

1.1. Explanation of the topics and concepts

In the context of a spam detection system, AI classifiers are algorithms that use artificial intelligence (AI) technologies to analyze the content of an email and determine whether it is spam or not. These algorithms can be trained on a large dataset of known spam and non-spam emails, allowing them to learn the characteristics that are commonly associated with spam emails. Using the information that it learnt throughout the training process, the AI classifier may then be used to make predictions on new, unseen emails (Brownlee, 2020).

AI classifiers can be useful for detecting spam emails because they can handle a wide range of data types, including text, images, and other types of multimedia. They can also be highly accurate, making them effective at reducing the amount of spam that reaches a user's inbox. Additionally, because AI classifiers are trained on large datasets, they can adapt to changes in spamming techniques and patterns over time, allowing them to maintain their effectiveness even as spamming methods evolve.

There are many different types of AI classifiers that can be used for spam detection, including neural networks, deep learning algorithms, and others. These algorithms each have their own strengths and weaknesses, and the best classifier to use will depend on the specific characteristics of the email data being analyzed. Some of the

mostly used classifiers that are going to be used in this application are Naïve Bayes, Support vector machine, K Nearest Neighbor, Decision Tree.

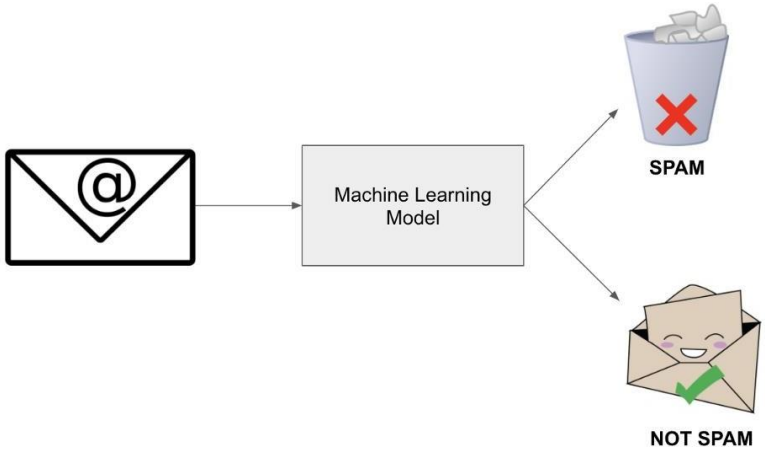


Figure 2: Spam Ham

1.2. Explanation of Problem domain

Spam emails are unwanted and often unsolicited messages that are sent to many recipients to promote a product, service, or idea. These emails can be a nuisance to users and can potentially contain harmful content, such as viruses, malware, or phishing scams.

The problem domain of spam mail refers to the challenges and issues associated with detecting and preventing spam emails. This includes developing and implementing effective spam detection systems, as well as addressing the ever-changing nature of spamming techniques and the large volume of spam emails that are sent every day.

Figure 3: Spam

Some specific challenges and issues in the problem domain of spam mail include:

- Developing spam detection systems that are accurate and effective: This involves identifying the characteristics of spam emails and developing algorithms or other methods that can accurately distinguish them from legitimate emails.
- Keeping up with changes in spamming techniques: Spammers are constantly trying to find new ways to avoid detection, which means that spam detection systems need to be able to adapt and evolve over time.
- Minimizing false positives: It is important to avoid mistakenly identifying legitimate emails as spam, as this can lead to important messages being missed or blocked.
- Dealing with the large volume of spam emails: There are billions of spam emails sent every day, which can make it difficult for spam detection systems to process and analyze all the data in a timely manner.

Overall, the problem domain of spam mail is a complex and constantly evolving area of research and practice, with many challenges and opportunities for developing new and improved spam detection methods.

2. Background

2.1. Research work done on the chosen topic

2.1.1. Spam mail

Unsolicited bulk email, sometimes referred to as spam mail, is a sort of digital communication that is distributed to numerous people in order to advertise a good, service, or concept. Spam emails are typically unwanted and often misleading or deceptive, and they can contain a variety of content, including advertisements, scams, phishing attempts, and other forms of unwanted or harmful material. (ICTEA, 2020)

Spam emails are typically sent in large numbers, often using automated tools and techniques, and they are often difficult to distinguish from legitimate emails. This can make them a nuisance to users and can pose a risk to their security and privacy. Spam emails can also have negative economic and social impacts, such as wasting resources, clogging up networks, and undermining trust in electronic communication.

To address the problem of spam mail, many organizations and individuals use spam detection systems, which are designed to identify and block spam emails before they reach a user's inbox. These systems can use a variety of techniques, including filtering, pattern matching, and machine learning, to distinguish spam from legitimate emails. However, because spamming techniques are constantly evolving, detecting, and preventing spam mail remains a challenging and ongoing problem.

2.1.2. Machine learning and AI in spam mail detection

Machine learning and AI technologies can be used in spam mail detection to develop and improve the accuracy and effectiveness of spam detection systems. With the use of these technologies, algorithms may make predictions or choices based on their learning from data without having to be expressly programmed for doing so.

In the context of spam mail detection, machine learning and AI algorithms can be trained on large datasets of known spam and non-spam emails, to learn the characteristics that are commonly associated with spam emails. This training can involve extracting features from the email data, such as the words and phrases used, the sender's address and other metadata, and the presence of certain patterns or structures. The features that the trained algorithm learnt during the training process may subsequently be utilized to generate predictions on new, unknown emails.

Machine learning and AI algorithms can be particularly useful for spam mail detection because they can handle a wide range of data types and can adapt to changes in spamming techniques over time. They can also be highly accurate, allowing them to reduce the amount of spam that reaches a user's inbox while minimizing false positives (legitimate emails that are mistakenly identified as spam). However, these algorithms can also be complex and require a lot of data and computational resources to train and evaluate, which can be a challenge in the context of spam mail detection (Brownlee, 2020).

2.1.3. Classification

Classification is a type of machine learning algorithm that is used to predict the class or category that a given piece of data belongs to. This can include tasks such as image or facial recognition, speech recognition, anomaly detection, and many others. Classification algorithms are commonly used in a wide range of applications, including security and surveillance, healthcare, transportation, and many others. Classification algorithms use a training dataset of known data points and their corresponding classes or categories to learn how to classify new, unseen data. This training can involve extracting features from the data, such as the pixels in an image or the words in a sentence and using those features to train a model that can predict the class of new data based on its similarity to the training data. Once trained, the classification algorithm can be applied to new data to make predictions about its class or category.

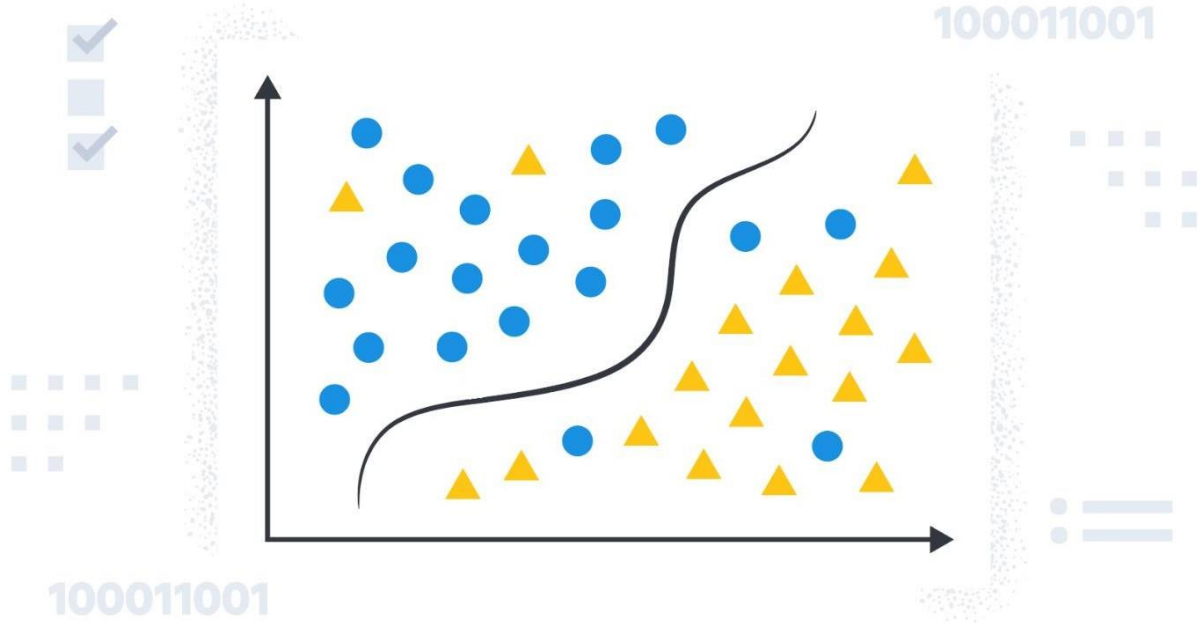


Figure 4: Classification

2.1.4. Advantages of spam mail detection

There are several advantages to using spam mail detection systems, including the following:

- Reduced spam: Spam mail detection systems can help to reduce the amount of spam that reaches a user's inbox, which can make it easier for them to find and access the messages that they want to read. This can also reduce the riskof users being exposed to unwanted or harmful content through spam emails.
- Improved productivity: By reducing the amount of spam that users must deal with, spam mail detection systems can help to improve their productivity and efficiency. This can be particularly beneficial for users who receive many emailsevery day, as it can save them time and effort that would otherwise be spent sortingthrough and deleting spam messages.
- Enhanced security: Spam emails can sometimes contain viruses, malware, or other types of harmful content, which can pose a risk to users' security and privacy. By blocking spam emails, spam mail detection systems can help to protect users from these threats and reduce the likelihood of them falling victim to scams or other types of online fraud.
- Reduced workload: By automatically sorting through emails, machine learning algorithms can reduce the workload of individuals or teams who would otherwise have to do this manually.
- Scalability: Spam mail detection systems can be designed to be scalable, which means that they can be easily deployed and used across many users and devices. This can be particularly beneficial for organizations with many email users, as it can allow them to protect all their users efficiently and effectively from spam.

2.1.5. Disadvantages of spam mail detection

There are several disadvantages to using spam mail detection systems, including the following:

- False positives: Spam mail detection systems can sometimes mistakenly identify legitimate emails as spam, which can lead to important messages being missed or blocked. This can be frustrating for users and can cause them to lose trust in the spam detection system.
- Inaccuracy: Spam mail detection systems are not perfect, and they can sometimes fail to identify spam emails or mistakenly classify legitimate emails as spam. This can allow spam emails to reach a user's inbox, potentially exposing them to unwanted or harmful content.
- Resource requirements: Spam mail detection systems can require a significant number of computational resources, including data storage, processing power, and memory, to analyze the large volumes of email data that are generated every day. This can be a challenge for organizations with limited resources or for systems that need to be deployed on a large scale.
- Privacy concerns: Some spam mail detection systems may use techniques that can potentially violate users' privacy, such as scanning the content of their emails or tracking their online activities. This can raise concerns about the ethical and legal implications of using these systems.
- Evolving spamming techniques: Spammers are constantly trying to find new ways to avoid detection, which can make it difficult for spam mail detection systems to keep up and maintain their effectiveness over time. This can require frequent updates and improvements to the spam detection algorithms and systems to maintain their accuracy and effectiveness.

2.1.6. Dataset

The following dataset for the proposed system was derived from Kaggle. The Email Spam Detection Dataset has 2 main attributes category and message there are about 5172 randomly picked email files and their labels for spam or not-spam classification.

Below are the two attributes

- Category: In this attribute there are two respective labels spam or ham. Ham means authentic email and spam means the junk mail. Where there is around 87% ham and 13% spam mail.
- Message: There are total of 5172 randomly selected email.

Link for the dataset https://www.kaggle.com/datasets/shantanudhakadd/email-spam-detection-dataset-classification

2.2. Review and analysis of existing work in the problem domain

2.2.1. Machine Learning based Spam E-Mail Detection using Naïve Bayes and decision tree

Author: Priti Sharma, Uma Bhardwaj

Journal International Journal of Intelligent Engineering and Systems Vol.11 No.3, 2018

To detect spam emails, this work implements the two machine learning algorithms Naive Bayes and J48 (decision tree). It then proposes a machine learning-based hybrid bagging strategy. The dataset is split up into many sets and fed into each algorithm individually during this procedure. The findings of three tests in total are compared in terms of precision, recall, accuracy, f-measure, true negative rate, false positive rate, and false negative rate. Regarding the two trials, separate Naive Bayes & J48 algorithmsare used which gave an accuracy score of 83.5% and 91.5%. The third experiment usesa hybrid bagged technique to build the suggested SMD system. The hybrid bagged method based SMD system obtained a total accuracy of 87.5% (Priri Sharma, 2018).

2.2.2. Email based Spam Detection using Bayes' theorem and Naive Bayes' Classifier

Author: Thashina Sultana, K A Sapnaz, Fathima Sana, Mrs. Jamedar Najath

Journal: International Journal of Engineering Research & Technology (IJERT) Vol. 9 Issue 06, June-2020

Finding these spammers and the spam content might be a challenge. work-intensive activities and a major research issue. Spam email is a procedure to deliver mail-based messages in masse. Since the cost of the receiver bears the majority of the spam; it is essentially postage. due promotion One type of commercial advertising is spam email. This is financially feasible since email may be highly expensive effective sender-to-receiver medium. Using the Bayes theorem, the Naive Bayes classifier, and the IP address of the sender, the suggested model can determine if a given message is spam or not (: Thashina Sultana, 2020).

2.2.3. SMS Spam Detection Based on Long Short-Term Memory and Gated Recurrent Unit

Author: Pumrapee Poomka, Wattana Pongsena, Nittaya Kerdprasop, and Kittisak Kerdprasop

Journal: International Journal of Future Computer and Communication, Vol. 8, No. 1, March 2019

In this article, they suggest a unique method for detecting SMS spam using Deep Learning and Natural Language Processing, based on a case study of English-language SMS spam. They have employed word tokenization, padding, truncating, and word embedding to add extra dimensions to the data and prepare it for the model building process. The model based on the Long Short-Term Memory and Gated Recurrent Unit algorithms is then developed using this data. The suggested models' performance is contrasted with that of models built using machine learning methods like Support Vector Machine and Naive Bayes. The testing findings reveal that the model created using the Long Short-Term Memory approach delivers the best overall accuracy as high as 98.18%. This model demonstrates the capacity to reliably identify spam communications with a 90.96% accuracy rate, whereas misclassifying a legitimate message as spam only results in a 0.74% mistake rate (Pumrapee Poomka, 2019).

2.2.4. A Comparative Analysis of Machine Learning Techniques for Spam Detection

Author: Syed Ishfaq Manzoor, Dr Jimmy Singla

Journal: Shfaq Manzoor et al., International Journal of Advanced Trends in Computer Science and Engineering, Volume 8, No.3, May - June 2019

In this research they have said that Spam, or unsolicited commercial email, accounts for 62% of all internet traffic worldwide, according to estimates. There has been a lot of technology advancement since the first spam mail which was in 1978 but it is still time consuming and expensive in the field of mathematical science. The study in this article compares the performance between K-NN, Decision Tree, Random Forest, Naive Bayes, and SVM for spam identification with other machine learning approaches. After they tested all 962 emails on the given 6 models the most accurate algorithms were Support Vector Machines and Random Forest. The models were not modified at any way (Syed Ishfaq Manzoor, 2019).

2.2.5. Evaluating the Effectiveness of Machine Learning Methods for Spam Detection.

Author: Yuliya Kontsewaya, Evgeniy Antonov, Alexey Artamonov

Journal: Procedia Computer Science Volume 190 – July 2021

In this journal the selected machine learning algorithms are KNN, Support Vector Machin, Naïve Bayes, Logistic regression, decision tree and random forest. The dataset set used for testing these algorithms were ready made. The reason this research was done is because almost every user face spam problem in a daily basis. The research training was done in a ready-made dataset which consisted about 1368 ham and 4360 spam mail. The general flow of the developed algorithm is first the null values was removed, then duplicates, gaps and stop words. Then all the letters were converted to lowercase. After that tokenization was performed and Count Vectorizer was used. Now comes the training stage where all the algorithms were trained using the mentioned machine learning algorithms. Then in the test

stage accuracy, precision, recall and roc area was measured. Among the mentioned algorithms the highest accuracy was of Naïve Bayes and Logistic regression with accuracy percentage up to 99%. A more secure algorithm can be built combining the algorithms or filtering methods (Yuliya Kontsewaya, 2021).

2.2.6. Summarized review

After going through the research papers, it is seen that a lot of work has been done on the field of spam detection using machine learning. A lot of valuable information have been gathered and analyzed to carry on the proposed project. It seems that some of the best algorithms and with the highest accuracy that can be used for spam mail detection are Naïve Byes, Support vector machine and K nearest neighbor.

The reviewed work and the proposed work both has similarities and dissimilarities. The existing projects used different datasets and some different algorithms. But the core architecture was similar with all the projects being importing dataset, preprocessing, visualizing, split, train, and test.

3. Solution

3.1. Approach to solving the problem

The proposed approach to solving the problem is we need to use prebuilt libraries like NumPy, matplotlib, pandas etc. Sklearn was also used then we get the mail data from both spam and ham mail then prepossess the data and convert the text and data into more meaningful model visualize it and split it into train and test data then feed it to our models SVM, KNN, Naïve Byes. After that we will have trained model then we will predict if the mail is spam or ham. Then we will define the hyperparameters grid for all three parameters then initialize the grid search and fit the grid search. After that we check the best combination of hyperparameters and predict. The accuracy score, f1 score, recall and precision are calculated and finally the confusion matrix. After that we test all the three models and test them. When we give the new mail, it can finally predict it.

3.2. Explanation of AI algorithm used

3.2.1. Support Vector Machine

The supervised learning technique Support Vector Machine (SVM) is frequently used for classification problems but may also be utilized for regression challenges. SVM creates a decision boundary to discriminate between classes. The most important aspect of SVM algorithms is the way the decision boundary is drawn or determined. Each observation (or data point) is displayed in n-dimensional space prior to the creation of the decision boundary. The number of features utilized is "n." For example, categorizing different "cells" according to "length" and "width," where observations are shown in a 2-dimensional space and a line serves as the decision boundary. Decision boundary is a plane in three-dimensional space if we employ three features. The decision boundary turns into a hyperplane, which is challenging to see, if we employ more than three features.

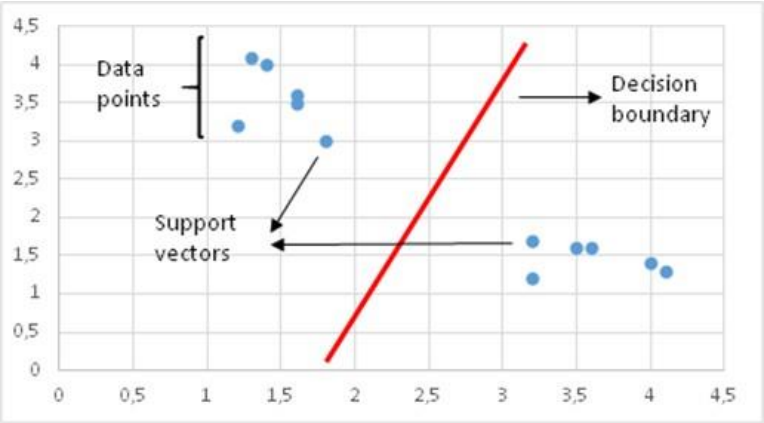


Figure 5: SVM

The support vectors' distance from the decision border is maximized. The decision boundary will be extremely noise-sensitive and difficult to generalize if it is too near to a support vector. A misclassification may result from even very little changes in independent variables. In contrast to the image above, the data points are not always linearly separable. In these situations, SVM employs a kernel method to make the data points linearly separable by comparing their similarity (or proximity) in a higher dimensional space (Yildirim, 2021).

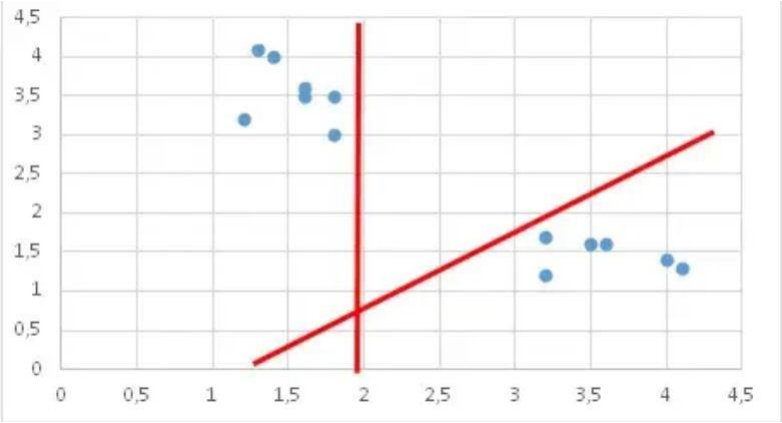


Figure 6: SVM 2

Support vector machines (SVMs) are a type of supervised machine learning algorithm that can be used for classification. In the case of spam mail detection, the algorithm would take in a set of training data that has been labeled as either "spam" or "not spam", and it would use that data to learn the characteristics of each type of email. Then, when presented with a new email, the algorithm would use that learned knowledge to predict whether the new email is likely to be spam or not.

Kernel function may be thought of as a similarity metric. With original features as the inputs, the result is a similarity measure in the new feature space. Here, similarity refers to a degree of proximity. The process of transforming data points into a high-dimensional feature space is expensive. The data points are not really transformed by the algorithm into a brand-new, high-dimensional feature space. Kernelized SVM calculates decision boundaries in a high-dimensional feature space using similarity measures without modifying the data. I believe that's why it's also known as the kernel trick (Yildirim, 2021). When the number of dimensions exceeds the number of samples, SVM is particularly effective. SVM finds the decision boundary by using a selection of training points instead of all of them, which saves memory. On the other hand, training time increases for large datasets which negatively effects the performance. (Yildirim, 2021)

C It is a regularization parameter that controls the trade-off between maximizing the margin and minimizing the misclassification error. A large C value means that the regularization strength is low, which will result in a tighter margin but may also result in fewer misclassifications. A small C value means that the regularization strength is high, which will result in a wider margin but may also result in more misclassifications.

linear It is a kernel function that projects the data into a higher-dimensional space, where the classes can be separated by a linear hyperplane. A linear kernel is appropriate when the data is linearly separable in the original space, which means that it is possible to separate the classes with a straight line or hyperplane.

Gamma It is a coefficient that controls the width of the radial basis function (RBF) kernel, also known as the Gaussian kernel, used in the algorithm. A small gamma value means that the

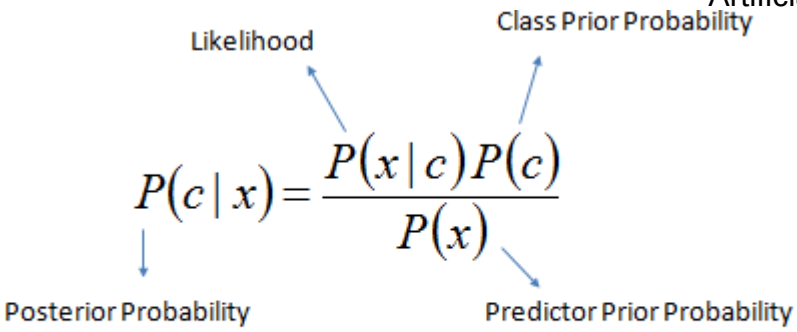
RBF kernel considers points further away from the decision boundary to have more influence. This corresponds to a softer decision boundary and is more likely to result in overfitting the training data. A large gamma value means that the RBF kernel considers points close to the decision boundary to have more influence. This corresponds to a tighter decision boundary and is more likely to result in underfitting the training data.

The hyperparameters that we have usen in the svm model is C:10, gamma: 0.0001 and kernel: linear.

3.2.2. Naive Bayes

Naive Bayes is known as a popular probabilistic machine learning approach for classification problems. It is predicated on the strong simplification assumption that characteristics in a dataset are independent of one another, which is seldom true in real-world data. Despite this, the algorithm often performs well in practice and is computationally efficient, making it a popular choice for many applications.

The Naive Bayes algorithm uses Bayes' theorem, which is a mathematical formula for calculating the probability of an event, to make predictions. Given a dataset with n features and a target variable, the algorithm first calculates the probability of each class of the target variable given each combination of values of the n features. It then uses these probabilities to predict the class of a new data point based on its feature values.



$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Figure 7: Naive bayes

Here,

- P(c | x) is the probability of event c occurring given that event x has occurred (the posterior probability)
- P(x | c) is the probability of event x occurring given that event c has occurred (the likelihood)
- P(c) is the probability of event c occurring (the prior probability)
- P(x) is the probability of event x occurring (the marginal probability)

Naive Bayes is a simple and effective classification technique that is particularly well-suited for dealing with multi-class issues. It is also highly scalable, making it a good choice for large datasets. However, its reliance on the assumption of feature independence can lead to suboptimal predictions in some cases.

In the case of spam mail detection, the algorithm would take in a set of training data that has been labeled as either "spam" or "ham", and it would use that data to learn the characteristics of each type of email. Then, when presented with a new email, the algorithm would use that learned knowledge to predict whether the new email is likely to be spam or not.

Alpha In the Naive Bayes algorithm, the alpha parameter is a smoothing parameter

used to prevent zero probability estimates for any feature. Alpha is used as a additive (Laplace/Lidstone) smoothing to the probability calculations, which is used to smooth the probability estimates. It is a hyperparameter that controls the amount of smoothing applied to the probability estimates.

A small alpha value corresponds to a high level of smoothing, meaning that small count observations will not affect the probability estimates too much. This can help to avoid overfitting the data.

A large alpha value corresponds to a low level of smoothing, meaning that small count observations will greatly affect the probability estimates. This can lead to overfitting the data, especially when the number of observations is small.

The optimal value of alpha can be determined through techniques such as cross-validation or grid search, which are used to tune the hyperparameters of the model.

In Bayesian methodologies like naive bayes, smoothing helps to deal with zero probability which occurs during classification when a feature has never appeared in each class before. It helps to overcome over-fitting in case of small dataset.

The hyper parameter we used in the above model is alpha with value alpha : 0.5

3.2.3. KNN

Lazy learning method K-nearest neighbors (KNN) is non-parametric. It is referred to as lazy since it does not require any training before producing predictions and non-parametric because it does not make any assumptions about the distribution of the underlying data. Instead, the algorithm stores the entire training dataset and uses it as the basis for making predictions.

The KNN method determines the distance between each point in the training dataset and the new data point before producing a prediction for it. It then identifies the K points in the training dataset that are closest to the new data point, based on the distance measure used. The new data point is subsequently classified by the method using the majority class of its K nearest neighbors.

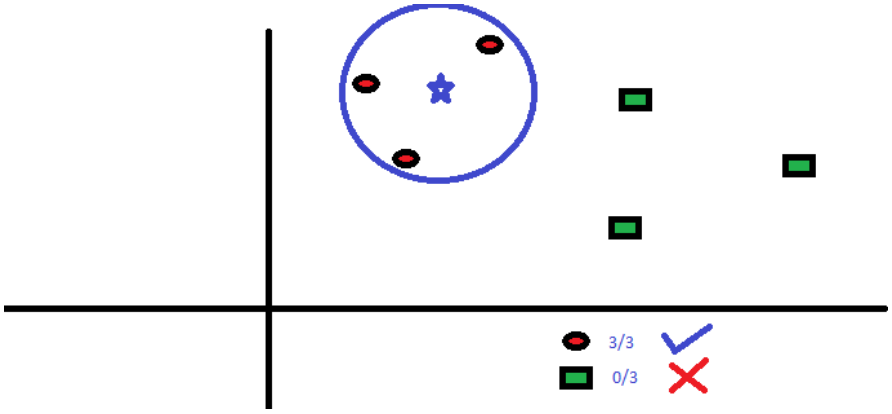


Figure 8: KNN

KNN is a simple and effective algorithm for classification and regression tasks and can be used in a wide range of applications. It is particularly well-suited for dealing with multi-dimensional data. However, it can be computationally expensive, especially for large datasets, and may not be the best choice for real-time applications.

In the case of spam detection, the algorithm would be trained on a dataset of both spam and legitimate emails, with each email represented as a set of features (such as the words used in the email, the sender's address, the email's subject line, etc.). Once the algorithm has been trained, it can then be used to classify new emails as either spam or legitimate by comparing their feature vectors to the feature vectors of the emails in the

training dataset and identifying the closest neighbors. The algorithm would then classify the new email based on the majority class of its closest neighbors.

knn__n_neighbors is a hyperparameter in the k-nearest neighbors (k-NN) algorithm. It determines the number of nearest neighbors to consider when making predictions. Larger values mean more neighbors are considered, resulting in a smoother decision boundary and a higher bias. A smaller value means fewer neighbors, leading to a more complex boundary and a higher variance. Finding the optimal value is usually done by using techniques like cross-validation and grid search.

The hyperparameter used in the above model is knn_n_neighbours: 3.

3.3. Pseudo Code

START

IMPORT libraries

IMPORT algorithm

Import tools

IMPORT dataset

EXPLORE dataset

IF ANY empty columns

Drop columns

ELSE

END IF

Rename the column

VISUALIZE spam and ham in bar

VISUALIZE spam and ham words

CONVERT category column

IF SPAM '1'

ELSE

'0'

Train Test Split dataset

CONVERT the text of emails into numerical representationTrain

the model for dataset using KNN

Fit the algorithm

Prediction on test data

CALCULATE accuracy on the model

DEFINE hyperparameter grid

INITIALIZE the grid search

Fit the grid search

PRINT best parameters

Predict on test set

CALCULATE accuracy

COMPUTE confusion matrix

VISUALIZE confusion matrix

CONVERT the text of emails into numerical representation Train the model for dataset using Support Vector Machine

Prediction on test data

CALCULATE accuracy on the model

DEFINE hyperparameter grid

INITIALIZE the grid search

Fit the grid search

PRINT best parameters

Predict on test set

CALCULATE accuracy

COMPUTE confusion matrix

VISUALIZE confusion matrix

CONVERT the text of emails into numerical representation

Train the model for dataset using Naive Bayes Prediction on test data

CALCULATE accuracy on the model

DEFINE hyperparameter grid

INITIALIZE the grid search

Fit the grid search

PRINT best parameters

Predict on test set

CALCULATE accuracy

COMPUTE confusion matrix

VISUALIZE confusion matrix

COMPARE all three algorithms

Test the models with data from the dataset END

3.4. Flowchart

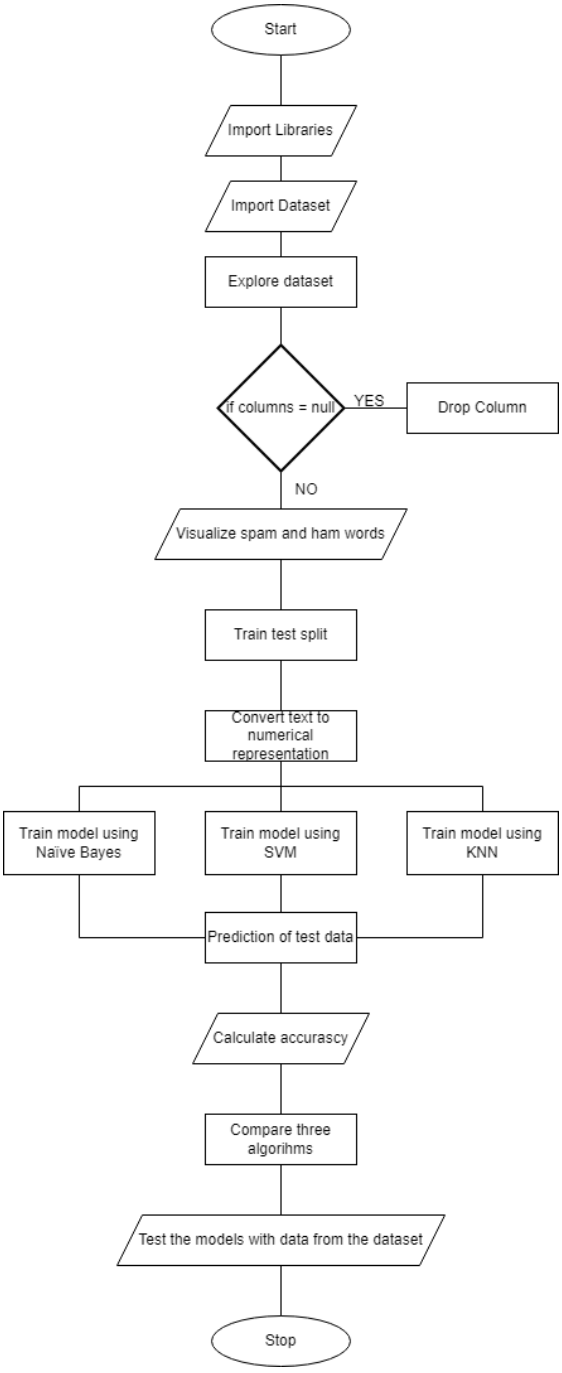


Figure 9: Flow Chart

3.5. Explanation of the development process/Detailed description

Moving forward to the detailed description about development of spam email detection three classification algorithms KNN, Naïve Bayes and SVM will be discussed thoroughly and all the process.

3.5.1. Importing Libraries/Algorithms

First, we import all the required libraries algorithms and tools required for the development of this application.

```
In [169]: import pandas as pd

          import matplotlib.pyplot as plt

          import numpy as np

          # Algorithms
          from sklearn.naive_bayes import GaussianNB,MultinomialNB,BernoulliNB
          from sklearn.svm import SVC
          from sklearn.neighbors import KNeighborsClassifier
          from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc
          from sklearn.model_selection import train_test_split
          from sklearn.feature_extraction.text import CountVectorizer
          from sklearn.pipeline import Pipeline
          from sklearn.metrics import confusion_matrix
          from sklearn.model_selection import GridSearchCV
          import seaborn as sns
```

Figure 10: Import

Import Libraries

- **pandas**, for data manipulation and analysis
- **matplotlib**, for data visualization
- **numpy**, for numerical computation
- **sklearn**, which is a machine learning library in Python that provides many efficient tools for machine learning and statistical modeling
- **seaborn**, for data visualization

Then we import the algorithms

- GaussianNB, MultinomialNB, and BernoulliNB from the naive_bayes module, which are three different types of Naive Bayes classifiers
- **SVC**, which stands for Support Vector Classification, a type of support vector machine

- KNeighborsClassifier, which is a type of k-nearest neighbors' classifier

Other tools being imported include:

- **train_test_split** and **confusion_matrix** from the **model_selection** module and **metrics** module, which can be used for model evaluation and hyperparameter tuning
- **CountVectorizer** from the **feature_extraction.text** module, which can be used to extract features from text data
- **Accuracy scores** this function calculates the proportion of correct prediction among all positive prediction
- **Precision scores** this function calculates the proportion of true positive predictions out of all positive predictions.
- **Recall scores** this function calculates the proportion of true positive predictions out of all actual positive instances.
- **F1 scores** this function calculates a balance of precision and recall. It is the harmonic mean of precision and recall, and ranges between 0 and 1, with 1 being the best possible score.
- **Pipeline**, which can be used to chain together multiple transformer objects and an estimator into a single pipeline

3.5.2. Data Preprocessing

After importing all the required packages, algorithms and tools. Then a csv file called spam is read using the read_csv from the panda's library. This function is used to reading a csv file and create a pandas data frame object, which is two-dimensional tabular data structure with rows and columns. Then the encoding parameter is used to specify the encoding of the file. The value "ISO-8858-1" is a character encoding that represents most of the printable ASCII characters as single 8-bit bytes. Upon running the code, the df variable will be a pandas DataFrame containing the contents of the spam.csv file.

```
In [2]: df = pd.read_csv("spam.csv", encoding="ISO-8859-1")
```

Figure 11: Read spam csv

After reading the CSV file we then use the df.head(10) the head method returns the first n rows of the DataFrame, where n is the parameter passed to the method. In this case, df.head(10) will return the first 10 rows of the dataframe df.

```
In [3]: df.head(10)
```

Out[3]:

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN
5	spam	FreeMsg Hey there darling it's been 3 week's n...	NaN	NaN	NaN
6	ham	Even my brother is not like to speak with me. ...	NaN	NaN	NaN
7	ham	As per your request 'Melle Melle (Oru Minnamin...	NaN	NaN	NaN
8	spam	WINNER!! As a valued network customer you have...	NaN	NaN	NaN
9	spam	Had your mobile 11 months or more? U R entitle...	NaN	NaN	NaN

Figure 12: head

As the above table has null values, we need to drop the null column. The code used helps in dropping empty columns from the DataFrame df. The empty columns are specified by their indices in the cols list, which in this case is [2, 3, 4]. The drop method is used to romove the 2, 3 and 4 columns from the dataframe and the inplace parameter is set to true, meaning the changes will be made inplace and the modified dataframe will be stored in the same variable df. The axis parameter is specifying that we are dropping columns (axis=1) rather than rows (axis=0).

After running the code, the dataframe will no longer have the columns with the indices 2, 3 and 4. The modified DataFrame is then displayed using the head method, which shows the first 10 rows of the modified DataFrame.

In [4]:
```
#Drop empty columns
cols = [2,3,4]
df.drop(df.columns[cols],axis=1,inplace=True)
df.head(10)
```

Out[4]:

	v1	v2
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
5	spam	FreeMsg Hey there darling it's been 3 week's n...
6	ham	Even my brother is not like to speak with me. ...
7	ham	As per your request 'Melle Melle (Oru Minnamin...
8	spam	WINNER!! As a valued network customer you have...
9	spam	Had your mobile 11 months or more? U R entitle...

Figure 13: Drop column

Now as we have dropped the empty columns, we rename the remaining columns of the DataFrame df. The rename method is used to change the names of the columns, and the columns parameter is a dictionary that maps the old column names to the new column names. In this case, the column named ‘v1’ is being renamed as ‘Category’ and ‘v2’ as ‘Message’. The inplace parameter is set to true, meaning the changes will be made inplace and the modified dataframe will be stored in the same variable df.

After running this code, DataFrame df will consist columns names ‘Category’ and ‘Message’ instead of v1 and v2. Then the dataframe is displayed using head method showing 10 rows of the modified dataframe.

```
In [5]: #Rename columns as category and message
        df.rename(columns = {'v1':'Category', 'v2':'Message'}, inplace = True)
        df.head(10)
```

Out[5]:

	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
5	spam	FreeMsg Hey there darling it's been 3 week's n...
6	ham	Even my brother is not like to speak with me. ...
7	ham	As per your request 'Melle Melle (Oru Minnamin...
8	spam	WINNER!! As a valued network customer you have...
9	spam	Had your mobile 11 months or more? U R entitle...

Figure 14: Rename column

After renaming the columns we will then print the number of rows in the DataFrame df, which corresponds to the number of e-mails in the dataset. The shape attribute of a DataFrame returns a tuple containing the number of rows and columns, and the [0] index is used to access the number of rows. The f'{}' string format is used to include the value of df.shape[0] in the printed string. The Dataset consist of 5572 E-Mails.

```
In [6]: print(f'Dataset consist of {df.shape[0]} E-Mails.')
        Dataset consist of 5572 E-Mails.
```

Figure 15: Print total emails

Now we count the number of occurrences of each unique value in the 'Category' column of the DataFrame df. The value counts method returns a series containing the counts of unique values in each Panda's series.

```
In [7]: df['Category'].value_counts()
Out[7]: ham     4825
        spam     747
        Name: Category, dtype: int64
```

Figure 16: Display value

Now we create a bar chart and plot using matolotlib to visualize the counts of the unique values in the 'Category' column of the DataFrame df. First a bew figure is created using

plt.figure() size being set with figzize. The value counts method is used to count the occurrence of unique values in category column and the result is passed to the plot.bar() method to create bar plot. The color is set to green for ham and red for spam. And the title method sets the title to total num of ham and spam in the dataset.

```
plt.figure(figsize=(8,6))

df['Category'].value_counts().plot.bar(color = ["green","red"])
plt.title('Total number of ham and spam in the dataset')
plt.show()
```

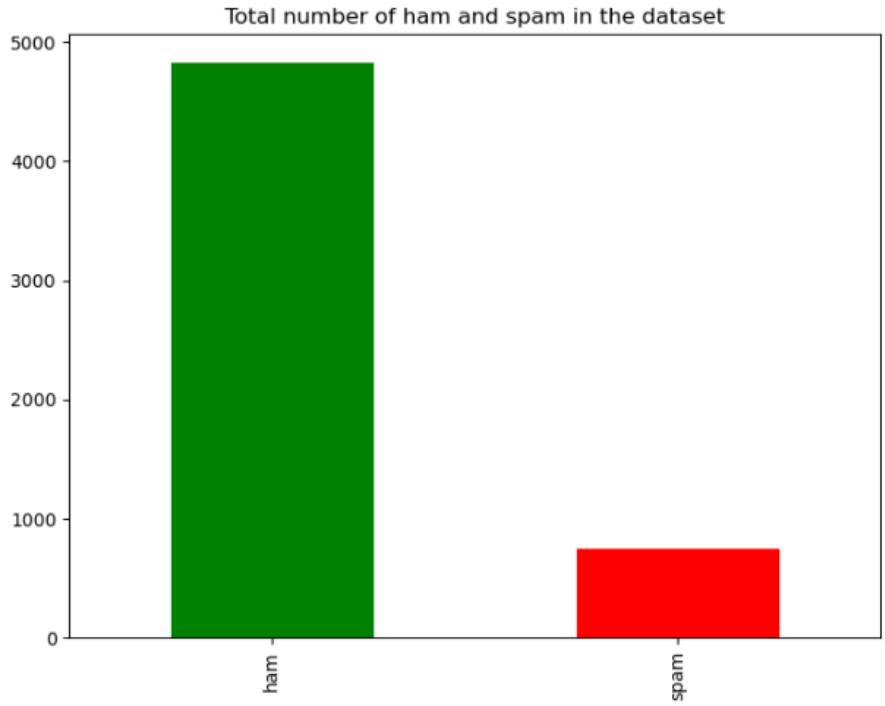


Figure 17: Bar graph

After the bar chart we create a word cloud using the wordcloud library to visualize the most common words in the 'Message' column of the DataFrame df for rows where the 'Category' column is 'ham'. First, a new figure is created using plt.figure() and the size of the figure is set using the figsize parameter. Next, a WordCloud object is created using the WordCloud class from the wordcloud library. The max_words parameter specifies the maximum number of words to include in the word cloud, and the width and height parameters specify the size of the word cloud image. The generate method is used to generate the word cloud using the 'Message' column for the 'ham' category. The word cloud is then displayed using the imshow method from matplotlib and the interpolation parameter is set to 'bilinear', which specifies the interpolation method to use when resizing the image. Finally, a title for the

word cloud is added using the title method. This will create a word cloud showing the most common words in the 'Message' column for rows where the 'Category' column is 'ham', with the size of the words in the cloud representing their frequency. The title of the word cloud will be 'Ham Word Cloud'. It will be same for ham mail also.

```
In [9]: from wordcloud import WordCloud

        plt.figure(figsize = (15,15))
        wc = WordCloud(max_words = 2000 , width = 1000 , height = 500).generate(" ".j
        plt.imshow(wc , interpolation = 'bilinear')
        plt.title("Ham Word Cloud")

Out[9]: Text(0.5, 1.0, 'Ham Word Cloud')
```

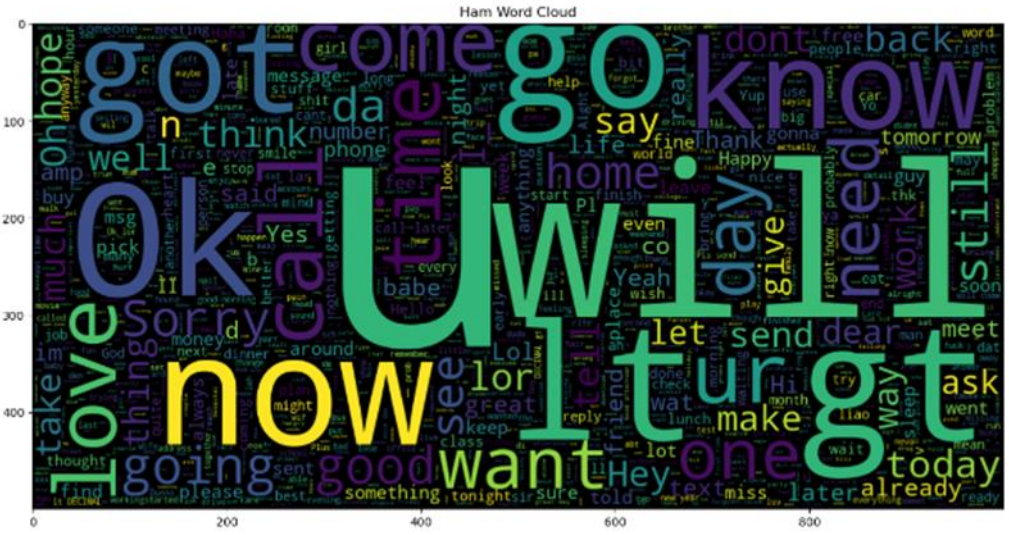


Figure 18: Word cloud

Now that we have visualized the data, we move forward to convert the ‘Category’ column of the DataFrame df from string to integers. To be more specific, it is replacing the string ‘spam’ with the integer 1 and the string ‘ham’ with 0. Then the apply method is used to apply a function to each element in the category column. The function that is being applied is defined using lambda function. The lambda function takes a value x as input and returns 1 if x is equal to spam and 0 of its ham. After running the code, the category column will contain values 0 and 1, The modified DataFrame is then displayed using the head method, which shows the first 5 rows of the dataframe.

```
In [11]: #0: Ham, 1: Spam
         df['Category']=df['Category'].apply(lambda x: 1 if x=='spam' else 0)
         df.head()
```

Out[11]:

	Category	Message
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf, he lives aro...

Figure 19: Convert category column to integer

3.5.3. Train Test Split

Now we create two separate variables X and Y from the DataFrame df. X is assigned with the values in the 'Message' column of the DataFrame, which represents the feature data. Y is assigned the values in the category column of the DataFrame, which represents the target labels. It wil be used as the input to our machine learning models. X will be used as the input features and Y will be used as the corresponding target labels. The goal of the model will be to learn a function that maps the input features to the target labels.

```
In [12]: X=df['Message']
         Y=df['Category']
```

Figure 20: Create 2 variable

Now we train test and split the data we import the train_test_split function from the sklearn.model_selection module to split the feature data X and target labels Y into training and test sets. The train_test_split function randomly splits the data into a training set and a test set, with the test set being a fixed size and the training set taking up the remaining portion of the data. By default, the function splits the data into 75% training data and 25% test data. The function returns four objects: X_train, X_test, y_train, and y_test. X_train and y_train are the feature data and target labels for the training set, and X_test and y_test are the feature data and target labels for the test set. These objects can then be used to train a machine learning model on the training data and evaluate the model on the test data.

```
In [13]: X_train, X_test, y_train, y_test = train_test_split(X,Y)
```

Figure 21: train test split

3.5.4. Model Fit

After we have done train, test and split we move forward to using the models now for the svm model we create a pipeline object that chains together a CountVectorizer transformer and a SVC estimator. The pipeline object is stored in the clf_svm variable. The Pipeline class from the sklearn.pipeline module is used to create the pipeline. The pipeline object takes a list of tuples as input, where each tuple contains a string (the name of the transformer or estimator) and an object (the transformer or estimator itself). In this case, the pipeline has two steps: The CountVectorizer transformer is applied to the data. This transformer converts the input data into a matrix of token counts.

The SVC estimator is applied to the transformed data. This estimator trains a support vector machine (SVM) model to predict the target labels. The pipeline object can then be used like any other estimator in scikit-learn, with the fit method being used to fit the model to the training data and the predict method being used to predict labels for new data.

```
In [127]: # Define the pipeline
clf_svm = Pipeline([
    ('vectorizer', CountVectorizer()),
    ('svc', SVC())
])
```

Figure 22: Pipeline SVM

```
In [138]: #Defineing Naive Baised
clf_NaiveBaised= Pipeline([
    ('vectorizer', CountVectorizer()),
    ('nb', MultinomialNB())
])
```

Figure 23: Pipeline Naive biased

```
In [150]: clf_knn= Pipeline([
    ('vectorizer', CountVectorizer()),
    ('knn', KNeighborsClassifier())
])
```

Figure 24: Pipeline KNN

After that we will fit, the models we fit the pipeline object which is stored in the clf_svm to the training data. The fit method is used to fit the model to the training data. The method takes two arguments: X train, which is the feature data for the training set, and y_train, which is the target

labels for the training set. After running the code, the clf_svm pipeline will be trained on the training data and will be able to make predictions on new data using the predict method.

```
In [128]: #Fiting the algorithm
          clf_svm.fit(X_train,y_train)

Out[128]: Pipeline(steps=[('vectorizer', CountVectorizer()), ('svc', SVC())])
```

Figure 25: Fit pipeline svm

```
In [139]: #Fiting the algorithm
          clf_NaiveBaised.fit(X_train,y_train)

Out[139]: Pipeline(steps=[('vectorizer', CountVectorizer()), ('nb', MultinomialNB())])
```

Figure 26: Fit pipeline naive baised

```
In [151]: clf_knn.fit(X_train,y_train)

Out[151]: Pipeline(steps=[('vectorizer', CountVectorizer()),
                          ('knn', KNeighborsClassifier())])
```

Figure 27: Fit pipeline KNN

After fitting the model, we use the predict method of the clf_svm pipeline to make predictions on the test data. The predict method takes a single argument: X_test, which is the feature data for the test set. The method returns an array of predicted labels for the test data. The predicted labels are stored in the y_pred_SVM variable. These predicted labels can then be compared to the true labels (stored in y_test) to evaluate the performance of the model.

```
In [129]: #Make prediction on X_test
          y_pred_SVM=clf_svm.predict(X_test)
```

Figure 28: Make prediction test data svm

```
In [140]: #Make prediction on X_test
          y_pred_NB=clf_NaiveBaised.predict(X_test)
```

Figure 29: Make prediction test data naive baised

```
In [152]: y_pred_KNN=clf_knn.predict(X_test)
```

Figure 30: Make prediction test data knn

3.5.5. Model/Hyperparameter Tuning

Now moving forward to tuning we define a grid of hyperparameter values to search over when tuning the SVM model.

The param_grid dictionary contains three key-value pairs, where the keys are the names of the hyperparameters, and the values are lists of possible values for each hyperparameter. The hyperparameters being searched are:

- 'svc__C': The regularization parameter of the SVM model, with possible values of 0.1, 1, 10, 100, and 1000.
- 'svc__kernel': The kernel function to use, with possible values of 'linear' and 'rbf'.
- 'svc__gamma': The kernel coefficient for the 'rbf' kernel, with possible values of 0.0001, 0.001, 0.01, and 0.1.

The grid search will try all possible combinations of these hyperparameter values and evaluate the model's performance on the validation set for each combination. The best set of hyperparameters will be chosen based on the model's performance.

```
In [18]: param_grid = {'svc__C': [0.1, 1, 10, 100, 1000],
                      'svc__kernel': ['linear', 'rbf'],
                      'svc__gamma': [0.0001, 0.001, 0.01, 0.1]}
```

Figure 31: Hyper parameter svc

```
In [142]: param_grid = {'nb__alpha': [0.5, 1.0, 1.5]}
```

Figure 32: Hyper parameter nb

```
In [154]: param_grid = {'knn__n_neighbors': [3, 5, 7, 9]}
```

Figure 33: Hyper parameter knn

Now we will create a GridSearchCV object that will perform grid search to tune the hyperparameters of the clf_svm pipeline. The GridSearchCV class is a meta-estimator that takes an estimator (in this case, the clf_svm pipeline) and a dictionary of hyperparameters to search over. The class performs cross-validation on the training data to evaluate the performance of the estimator for each combination of hyperparameters. The param_grid argument specifies the grid of hyperparameter values to search over, and the scoring

argument specifies the metric to use to evaluate the model's performance. The cv argument specifies the number of folds to use for cross-validation. After creating the GridSearchCV object, the model can be fit to the training data using the fit method, just like any other estimator in scikit-learn. The best set of hyperparameters can then be accessed using the best_params_ attribute of the GridSearchCV object.

```
In [19]: grid_search = GridSearchCV(clf_svm, param_grid, scoring='accuracy', cv=5)
```

Figure 34: GridSearchCV object svm

```
In [143]: grid_search = GridSearchCV(clf_NaiveBaised, param_grid,scoring='accuracy', cv=5)
```

Figure 35: GridSearchCV object NaiveBaised

```
In [155]: grid_search = GridSearchCV(clf_knn, param_grid, cv=5)
```

Figure 36: GridSearchCV object knn

Now we fit the grid_search object to the training data. The fit method trains the model on the training data and performs grid search to tune the hyperparameters of the clf_svm pipeline. The method takes two arguments: X_train, which is the feature data for the training set, and y_train, which is the target labels for the training set. After running this code, the grid_search object will be trained on the training data and will have determined the best set of hyperparameters for the clf_svm pipeline based on the performance on the validation set. You can access the best set of hyperparameters using the best_params_ attribute of the grid_search object, and you can access the best score (according to the scoring metric specified in the constructor) using the best_score_ attribute.

```
In [20]: grid_search.fit(X_train, y_train)
Out[20]: GridSearchCV(cv=5,
                      estimator=Pipeline(steps=[('vectorizer', CountVectorizer()),
                                                ('svc', SVC())]),
                      param_grid={'svc__C': [0.1, 1, 10, 100, 1000],
                                  'svc__gamma': [0.0001, 0.001, 0.01, 0.1],
                                  'svc__kernel': ['linear', 'rbf']},
                      scoring='accuracy')
```

Figure 37: Fit grid search object to training data svm

```
In [144]: grid_search.fit(X_train, y_train)

Out[144]: GridSearchCV(cv=5,
                       estimator=Pipeline(steps=[('vectorizer', CountVectorizer()),
                                                 ('nb', MultinomialNB())]),
                       param_grid={'nb__alpha': [0.5, 1.0, 1.5]}, scoring='accuracy')
```

Figure 38: Fit grid search object to training data

```
In [156]: grid_search.fit(X_train, y_train)

Out[156]: GridSearchCV(cv=5,
                       estimator=Pipeline(steps=[('vectorizer', CountVectorizer()),
                                                 ('knn', KNeighborsClassifier())]),
                       param_grid={'knn__n_neighbors': [3, 5, 7, 9]})
```

Figure 39: Fit grid search object to training data KNN

The below code is used for printing the best set of hyperparameters found by the grid search object during the grid search process. The pentagrams_ attribute is a dictionary containing the key-value pairs of the best set of hyperparameters found by the grid search. The keys are the names of the hyperparameters, and the values are the optimal values of the hyperparameters.

```
In [21]: print(grid_search.best_params_)
         {'svc__C': 1000, 'svc__gamma': 0.0001, 'svc__kernel': 'rbf'}
```

Figure 40: Print best parameter svm

```
In [145]: print("Best hyperparameters: {}".format(grid_search.best_params_))
          Best hyperparameters: {'nb__alpha': 0.5}
```

Figure 41: print best parameter

```
In [157]: print("Best hyperparameters: {}".format(grid_search.best_params_))
          Best hyperparameters: {'knn__n_neighbors': 3}
```

Figure 42: Print best parameter KNN

Now we use the predict method of the grid_search object to make predictions on the test data. The predict method takes a single argument: X_test, which is the feature data for the test set. The method returns an array of predicted labels for the test data. The predicted labels are stored in the y_pred_SVM variable. These predicted labels can then be compared to the true labels (stored in y_test) to evaluate the performance of the model. Note that the

grid_search object is actually an estimator itself (it is a GridSearchCV object that wraps the clf_svm pipeline), so it has a predict method just like any other estimator in scikit-learn. When you call the predict method on the grid_search object, it will use the best set of hyperparameters found during the grid search process to make the predictions.

```
In [22]: y_pred_SVM = grid_search.predict(X_test)

In [146]: y_pred_NB = grid_search.predict(X_test)

In [158]: y_pred_KNN = grid_search.predict(X_test)
```

Figure 43: make prediction

3.5.6. Performance Matrices

Moving forward to performance matrix we have calculated multiple evaluation metrics to assess the performance of the classifier. In addition to the accuracy score, we have calculated the F1-score, recall, and precision.

The F1-score is the harmonic mean of precision and recall and is a measure of a test's accuracy. The best value is 1 and the worst value is 0.

The recall is the ratio of the total number of correctly classified positive observations to the total number of observations in actual class. The recall score is also called Sensitivity, Hit Rate or True Positive Rate (TPR).

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The precision score is also called Positive Predictive Value (PPV).

In this case we have used the f1_score, recall_score, and precision_score functions from the sklearn.metrics module to calculate the F1-score, recall, and precision, respectively. The function takes two arguments: the true labels of the data (y_test), and the predicted labels generated by the classifier (y_pred_SVM).

It is important to note that all the scores are based on the number of correctly classified examples, so even if the recall, precision or f1-score are high, if the overall accuracy is low, it could be indication of a problem. The result we got is 0.99 with accuracy, 0.97 with F1, 0.95 with recall and 1.0 with precision.

```
In [173]: # Calculate the accuracy score
svm_acc = accuracy_score(y_test, y_pred_SVM)
svm_f1 = f1_score(y_test, y_pred_SVM)
svm_recall = recall_score(y_test, y_pred_SVM)
svm_precision = precision_score(y_test, y_pred_SVM)
print("SVM Model Metrics:")
print("Accuracy: ", svm_acc)
print("F1 Score: ", svm_f1)
print("Recall: ", svm_recall)
print("Precision: ", svm_precision)

SVM Model Metrics:
Accuracy:  0.9935391241923905
F1 Score:  0.9765013054830287
Recall:  0.9540816326530612
Precision:  1.0
```

Figure 44: Performance matrices SVM

```
In [175]: # Calculate the accuracy score
naive_acc = accuracy_score(y_test, y_pred_NB)
naive_f1 = f1_score(y_test, y_pred_NB)
naive_recall = recall_score(y_test, y_pred_NB)
naive_precision = precision_score(y_test, y_pred_NB)
print("Naive Baised Model Metrics:")
print("Accuracy: ", naive_acc)
print("F1 Score: ", naive_f1)
print("Recall: ", naive_recall)
print("Precision: ", naive_precision)

Naive Baised Model Metrics:
Accuracy:  0.9877961234745154
F1 Score:  0.9565217391304348
Recall:  0.9540816326530612
Precision:  0.958974358974359
```

Figure 45: Performance matrices

```
In [176]: # Calculate the accuracy score
knn_acc = accuracy_score(y_test, y_pred_KNN)
knn_f1 = f1_score(y_test, y_pred_KNN)
knn_recall = recall_score(y_test, y_pred_KNN)
knn_precision = precision_score(y_test, y_pred_KNN)
print("KNN Model Metrics:")
print("Accuracy: ", knn_acc)
print("F1 Score: ", knn_f1)
print("Recall: ", knn_recall)
print("Precision: ", knn_precision)

KNN Model Metrics:
Accuracy:  0.9246231155778895
F1 Score:  0.6341463414634146
Recall:  0.4642857142857143
Precision:  1.0
```

Figure 46: Performance matrices knn

3.5.7. Confusion Matrix

The code below is using the confusion_matrix function from the sklearn.metrics module to compute the confusion matrix for the predictions made by the grid_search object on the test data. The confusion matrix is a 2x2 matrix that contains the number of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions made by the model. The rows of the matrix represent the actual class labels, and the columns represent the predicted class labels. The confusion matrix can be computed using the confusion_matrix function, which takes two arguments: y_test, which is the true labels for the test set, and y_pred_SVM, which is the predicted labels for the test set. The function returns a 2x2 matrix where the first row represents the true negatives and false negatives, and the second row represents the false positives and true positives. The SVM model made 1223 true negatives, 152 true positives, 0 false negatives, and 18 false positives.

For visually representing the matrix we use the heatmap function from the seaborn library to visualize the confusion matrix. The heatmap function takes a single argument: the confusion matrix (stored in the cm variable). The annot argument specifies that the values in the matrix should be annotated on the plot, and the fmt argument specifies that the values should be formatted as integers ('d' stands for decimal). The xlabel and ylabel functions are used to set the x-axis and y-axis labels of the plot, respectively. The show function is then used to display the plot. The resulting plot will show the number of true negatives and false negatives on the y-axis and the number of false positives and true positives on the x-axis. The color of each cell in the plot corresponds to the value in the cell, with darker colors indicating higher values. This

plot can be used to visually inspect the performance of the model and identify any areas of weakness.

Confusion matrix for SVM

```
In [24]: # Compute the confusion matrix
         cm = confusion_matrix(y_test, y_pred_SVM)

In [25]: # Visualize the confusion matrix
         sns.heatmap(cm, annot=True, fmt='d')
         plt.xlabel('Predicted')
         plt.ylabel('True')
         plt.show()
```

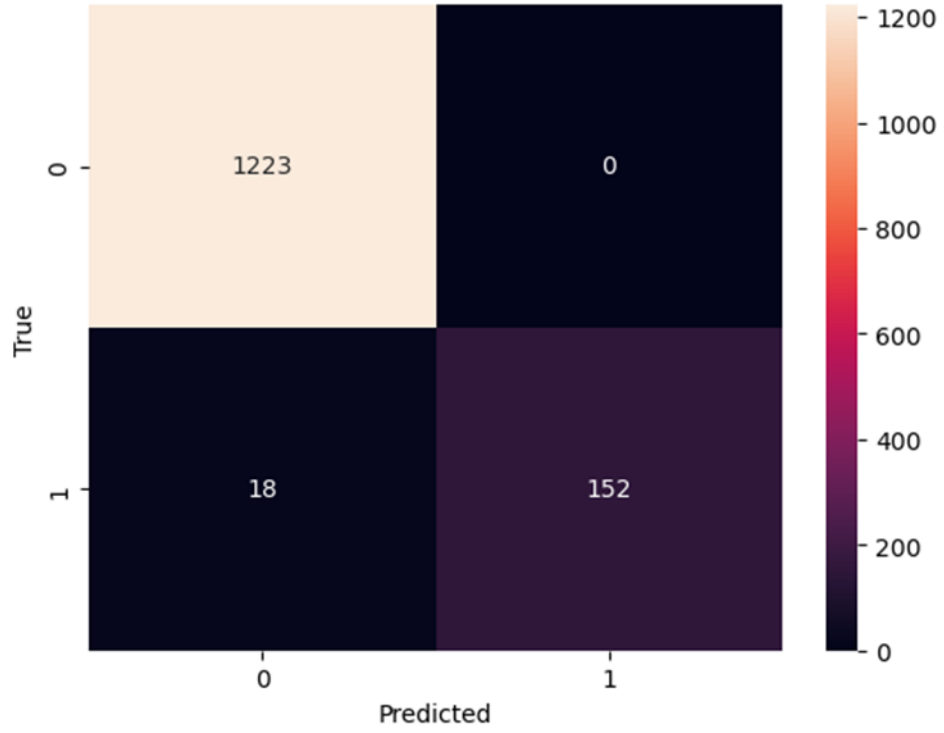


Figure 47: Confusion matrix svm

Confusion matrix for Naïve Byes

```
In [148]: cm = confusion_matrix(y_test, y_pred_NB)

In [149]: # Visualize the confusion matrix
          sns.heatmap(cm, annot=True, fmt='d')
          plt.xlabel('Predicted')
          plt.ylabel('True')
          plt.show()
```

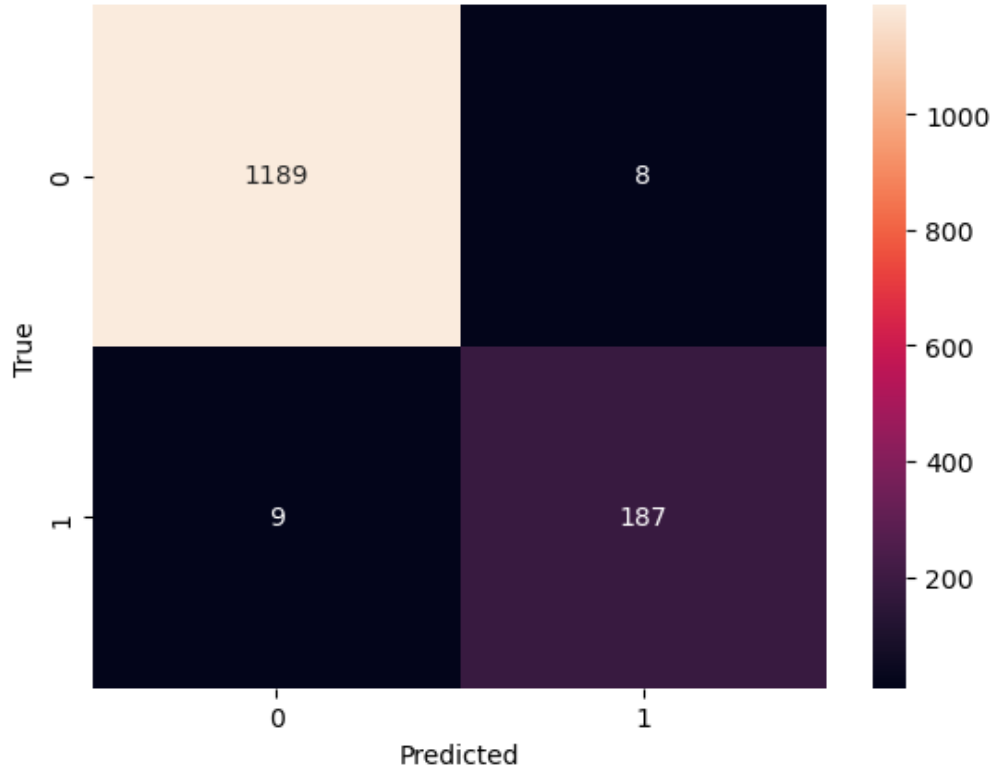


Figure 48: Confusion matrix nb

Confusion matrix for KNN

```
In [160]: cm = confusion_matrix(y_test, y_pred_KNN)

In [161]: # Visualize the confusion matrix
          sns.heatmap(cm, annot=True, fmt='d')
          plt.xlabel('Predicted')
          plt.ylabel('True')
          plt.show()
```

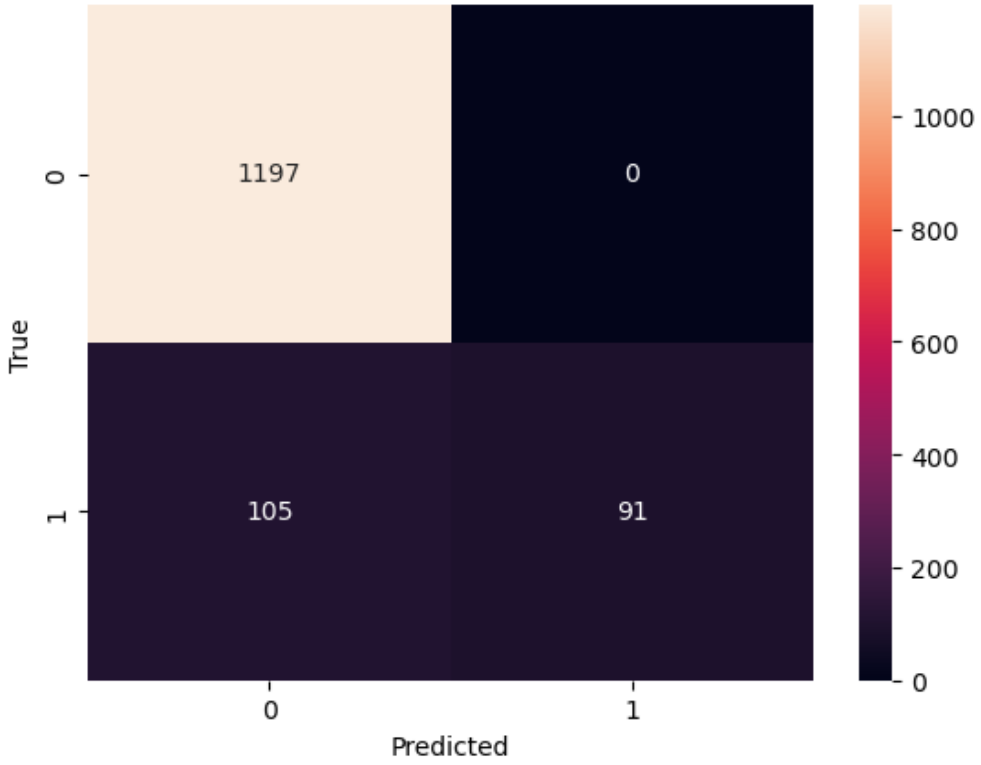


Figure 49: Confusion matrix knn

3.5.8. Model Comparison

Moving forward to model comparison we create a NumPy array menMeans which will contain the accuracy value using with the help of variables svm_acc, naïve_acc and knn_acc. As the accuracy values are shown using percentage. The accuracy score is multiplied by 100 which will convert the accuracy values. E.g., 0.98 to 98%.

```
In [162]: menMeans = np.array([naive_acc,svm_acc,knn_acc])*100
```

Figure 50: menMeans

After converting the values we then create a new variable called ind which will store a list containing the names of three machine learning models named Support vector machine, Naïve Bayes and KNN. The list will be used to plot x-axis label.

```
In [163]: ind = ['Naive Bayes','SVM','KNN']
```

Figure 51: New variable ind

Now we create a bar plot of the menMeans array, with the names of the machine learning models listed in the ind variable as the x-axis labels. fig, ax = plt.subplots(figsize = (11,7)) creates a new figure and axis using the pyplot module from matplotlib library, and sets the size of the figure to be 11 inches in width and 7 inches in height. ax.bar(ind,menMeans,width=0.3,color ='lightblue') creates a bar chart using the ax object, passing in the ind variable for the x-axis labels and the menMeans array for the y-axis values. The width parameter is set to 0.3 and the color of the bars is set to lightblue. Then, it uses a for loop with the enumerate() function to iterate over the menMeans array. enumerate() adds a counter to an iterable and returns it in a form of enumerate object. Which has the elements in the form (index, element). With the index variable you can reference the index of the current bar and with data the value of the bar. plt.text(x=index , y =data+1 , s="{:.2f}".format(data) , fontdict=dict(fontsize=20)) adds text labels to each bar, displaying the accuracy value of each machine learning model, with a format of 2 decimal places using the format() function. The text is placed at the top of each bar with a y-coordinate of data+1 and the x-coordinate corresponding to the index of the current bar. The font size of the text is set to 20. plt.tight_layout() is used to automatically adjust subplot params so that the subplot(s) fits in to the figure area. plt.show() is used to display the plot. This will create a bar chart showing the

accuracy for each of the three machine learning models with the names of the models on the x-axis and the accuracy values on the y-axis. The accuracy values are also displayed as text labels on top of each bar.

```
fig, ax = plt.subplots(figsize = (11,7))
ax.bar(ind,menMeans,width=0.3,color ='lightblue')
for index,data in enumerate(menMeans):
    plt.text(x=index , y =data+1 , s="{:.2f}".format(data) , fontdict=dict(fonts
plt.tight_layout()
plt.show()
```

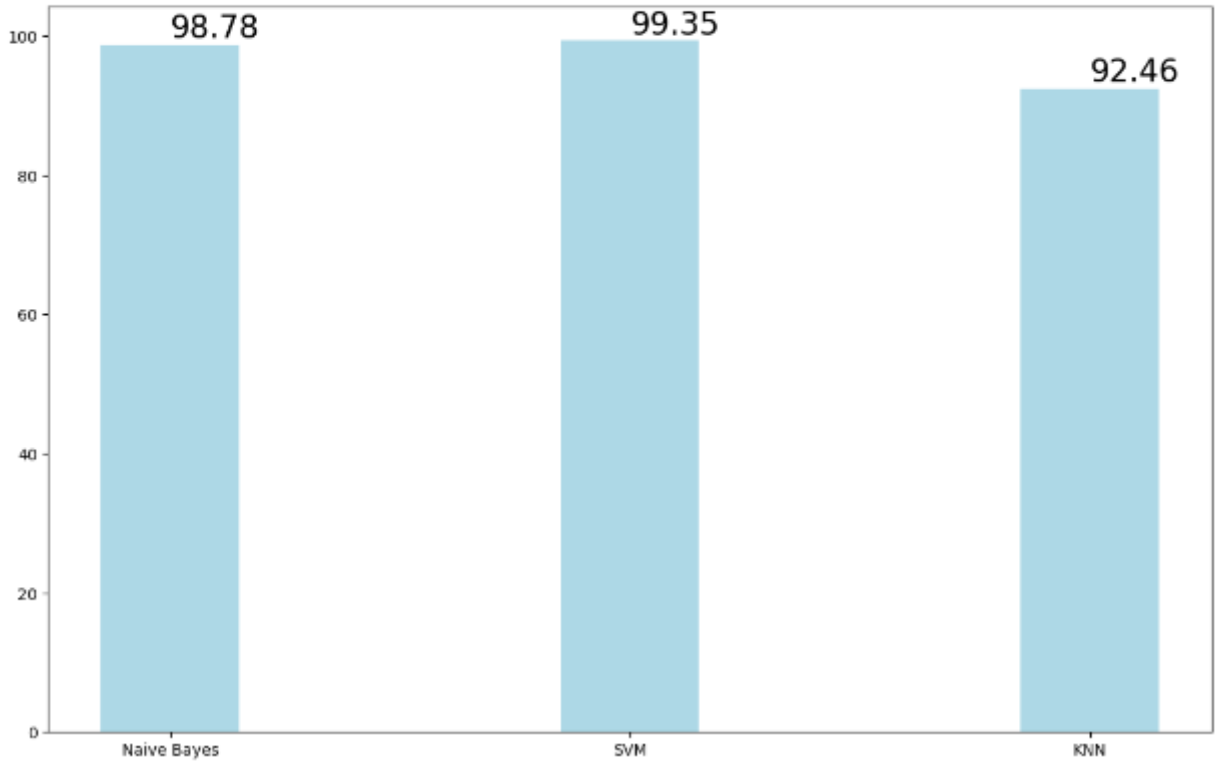


Figure 52: Plot accuracy bar chart

After looking after the above results we can come to conclusion that the SVM provides the highest accuracy about 99% which is very good and KNN with the lowest accuracy of 92%.

3.5.9. Test

For testing the algorithms, we define a function spam_detect that takes two arguments: clf, which is a tuple containing a trained classifier and a string label, and txt, which is a string containing the message to classify. The function first uses the predict method of the classifier (stored in the clf tuple) to make a prediction on the input message (txt). If the prediction is 1 (i.e., spam), the function prints a message indicating that the message is spam. If the

prediction is 0 (i.e., not spam), the function prints a message indicating that the message is real. The function also includes the label of the classifier (stored in the second element of the clf tuple) in the output message. This can be used to identify which classifier made the prediction. You can call this function with a trained classifier and a message as arguments to classify the message as spam or not spam.

```
In [165]: def spam_dect(clf,txt):
              a=clf.predict([txt])
              if a==1:
                  print(f"{clf[1]} This is a Spam email \n")
              else:
                  print(f"{clf[1]} This is a Real email \n")
```

Figure 53: Spam detect function

The code below defines a variable message and assigns it the value of the user input. Then the input() function waits for the user to enter a string and press the enter key. Then the string is returned as output. Then the string which was entered by user is stored in message variable. The variable is then passed to the spam_detect function above to classify the message as spam or ham.

```
In [27]: message= input()
         i know you where in trouble Jason
```

Figure 54: Take input

The code below is using a for loop to iterate over a list of classifiers, which are clf_NaiveBaised, clf_svm and clf_knn. It passes each classifier and the input message to the spam_dect function. It creates an empty list i then appends the classifiers to it, then iterates over the classifiers in i and in each iteration, it passes the classifier and the message to the spam_dect function. This way, the function will use each classifier to predict the label of the message and prints out the result. This is useful if you want to compare the performance of multiple classifiers on the same input and observe how their outputs differ. The classifier names that are passed in the spam_det function, should match the actual classifier objects for the function to work as intended, otherwise it would raise an error.

```
clf_1 = clf_NaiveBaised
clf_2 = clf_svm
clf_3 = clf_knn
i = [clf_1, clf_2,clf_3]
for x in i:
    spam_dect(x,message)
```

```
MultinomialNB() This is a Spam email

SVC() This is a Spam email

KNeighborsClassifier() This is a Real email
```

Figure 55:loop toiterate over list of classifier

Below we call spam detect function with the clf_svm classifier and the message variable as arguments. The spam detect function will use the clf svm classifier to make a prediction on the variable message, after that we will print a message indicating if the message is spam or ham. The classifier label will also be included in the output.

```
#Let's make a predict with SVM
spam_dect(clf_svm, message)
```

```
SVC() This is a Spam email
```

Figure 56: Output for single model

3.6. Achieved Results

Moving forward we can look forward to our achieved results we can see the screenshots of the application and detailed description about the project in the above topic. Now let's see if we got our results or not.

3.6.1. Accuracy of models

By looking at the performance metrices we can come to conclusion that svm model has the highest accuracy with accuracy score around 0.99, f1 score of 0.97, recall of 0.95 and precision of 1.0. By these results we can confidently use this model. The lowest accuracy was of KNN with accuracy score of 0.92.

Accuracy score for SVM

```
In [173]: # Calculate the accuracy score
svm_acc = accuracy_score(y_test, y_pred_SVM)
svm_f1 = f1_score(y_test, y_pred_SVM)
svm_recall = recall_score(y_test, y_pred_SVM)
svm_precision = precision_score(y_test, y_pred_SVM)
print("SVM Model Metrics:")
print("Accuracy: ", svm_acc)
print("F1 Score: ", svm_f1)
print("Recall: ", svm_recall)
print("Precision: ", svm_precision)

SVM Model Metrics:
Accuracy:  0.9935391241923905
F1 Score:  0.9765013054830287
Recall:  0.9540816326530612
Precision:  1.0
```

Figure 57: Accuracy score svm

Accuracy score for Naïve Bayes

```
In [175]: # Calculate the accuracy score
naive_acc = accuracy_score(y_test, y_pred_NB)
naive_f1 = f1_score(y_test, y_pred_NB)
naive_recall = recall_score(y_test, y_pred_NB)
naive_precision = precision_score(y_test, y_pred_NB)
print("Naive Baised Model Metrics:")
print("Accuracy: ", naive_acc)
print("F1 Score: ", naive_f1)
print("Recall: ", naive_recall)
print("Precision: ", naive_precision)

Naive Baised Model Metrics:
Accuracy:  0.9877961234745154
F1 Score:  0.9565217391304348
Recall:  0.9540816326530612
Precision:  0.958974358974359
```

Figure 58: Accuracy score nb

Accuracy score for KNN

```
In [176]: # Calculate the accuracy score
knn_acc = accuracy_score(y_test, y_pred_KNN)
knn_f1 = f1_score(y_test, y_pred_KNN)
knn_recall = recall_score(y_test, y_pred_KNN)
knn_precision = precision_score(y_test, y_pred_KNN)
print("KNN Model Metrics:")
print("Accuracy: ", knn_acc)
print("F1 Score: ", knn_f1)
print("Recall: ", knn_recall)
print("Precision: ", knn_precision)

KNN Model Metrics:
Accuracy:  0.9246231155778895
F1 Score:  0.6341463414634146
Recall:  0.4642857142857143
Precision:  1.0
```

Figure 59: Accuracy score knn

Bar chart SVM, KNN, Naïve bayes

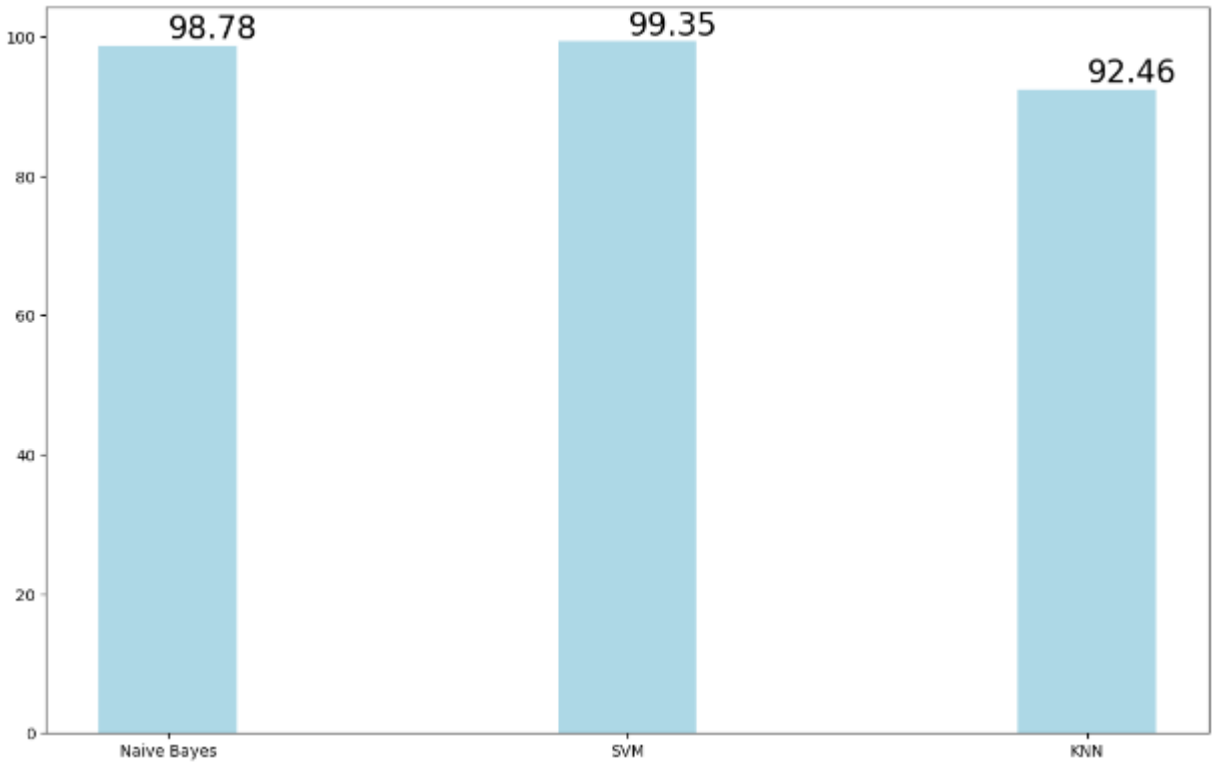


Figure 60: Bar accuracy score

3.6.2. Result Attained

Now we can come to the last part where we enter a string and know if it's a spam or ham email. For that we have made a function called spam dect for testing the algorithms, we define a function spam_detect that takes two arguments: clf, which is a tuple containing a trained classifier and a string label, and txt, which is a string containing the message to classify. The function first uses the predict method of the classifier (stored in the clf tuple) to make a prediction on the input message (txt). If the prediction is 1 (i.e., spam), the function prints a message indicating that the message is spam. If the prediction is 0 (i.e., not spam), the function prints a message indicating that the message is real. The function also includes the label of the classifier (stored in the second element of the clf tuple) in the output message. This can be used to identify which classifier made the prediction. You can call this function with a trained classifier and a message as arguments to classify the message as spam or not spam.

```
In [165]: def spam_dect(clf,txt):
              a=clf.predict([txt])
              if a==1:
                  print(f"{clf[1]} This is a Spam email \n")
              else:
                  print(f"{clf[1]} This is a Real email \n")
```

Figure 61: Function spam detect

After that we have taken a text input "I know you were in trouble Jason" which is then stored in message variable.

```
In [70]: message= input()
         K..k:)where are you?how did you performed?
```

Figure 62: Text input

For the final part we have used a for loop to iterate over a list of classifiers, which are clf_NaiveBaised, clf_svm and clf_knn. It passes each classifier and the input message to the spam_dect function. It creates an empty list i then appends the classifiers to it, then iterates over the classifiers in i and in each iteration, it passes the classifier and the message to the spam_dect function. This way, the function will use each classifier to predict the label of the message and prints out the result. This is useful if you want to compare the performance of multiple classifiers on the same input and observe how their outputs differ. The classifier names that are passed in the spam_det function, should match the actual classifier objects for

the function to work as intended, otherwise it would raise an error

```
In [71]: clf_1 = clf_NaiveBaised
         clf_2 = clf_svm
         clf_3 = clf_knn
         i = [clf_1, clf_2,clf_3]
         for x in i:
             spam_dect(x,message)

         MultinomialNB() This is a Real email

         SVC() This is a Real email

         KNeighborsClassifier() This is a Real email
```

Figure 63: Models test

Now taking another string which is a spam mail and testing it o the models.

```
In [65]: message= input()

         Congrats! 1 year special cinema pass for 2 is yours. call 09061209465 now! C
         Suprman V, Matrix3, StarWars3, etc all 4 FREE! bx420-ip4-5we. 150pm. Dont mi
         ss out!
```

As shown in the results SVC and NB has predicted the correst result where as knn has predicted wrong.

```
In [68]: clf_1 = clf_NaiveBaised
         clf_2 = clf_svm
         clf_3 = clf_knn
         i = [clf_1, clf_2,clf_3]
         for x in i:
             spam_dect(x,message)

         MultinomialNB() This is a Spam email

         SVC() This is a Spam email

         KNeighborsClassifier() This is a Real email
```

4. Conclusion

4.1 Analysis of the work done:

In conclusion, this project has given an opportunity to research on the problem domain. Various problem domain and machine learning model was researched which gave in depth understanding of ai and machine learning. After researching a lot of problem domain Spam mail detection was picked.

After the problem domain selection, a lot of research was carried out on the problem domain and the existing woks related to the topic. A lot of research paper, journal and articles were studied. It provided with the common flow of approach to solve the problem. After reviewing the journals and research papers there was a clear vision and understanding of the workflow and which models to choose for the improvement in the problem domain. After the basic understanding of the existing work SVM, Naïve bayes and KNN algorithm was chosen as the model for the project. Finally, the approach to solving the problem was documented in the form of pseudo code and flowchart. Then upon looking at the pseudo code and flow chart the coding process started.

The coding part was divided into different parts data preprocessing, train test split, model fit, model tune, precision metrices and confusion matrix. After that all the three models was compared and tested using spam and ham strings. Overall, the project was a success, and a proper spam detection application was developed. This project provided a in depth understanding of conducting research, documenting the project, designing pseudo code and flowchart, machine learning algorithms and in depth about ai and its real-world usage.

4.2 How the solution addresses real world problems:

Spam emails are typically sent in large numbers, often using automated tools and techniques, and they are often difficult to distinguish from legitimate emails. This can make them a nuisance to users and can pose a risk to their security and privacy. Spam emails can also have negative economic and social impacts, such as wasting resources, clogging up networks, and undermining trust in electronic communication. To address the problem of spam mail, this spam detection systems, was built which is designed to identify spam mail. These systems can use a variety of techniques, including filtering, pattern matching, and machine learning, to distinguish spam from legitimate emails. However, because spamming techniques are constantly evolving, detecting, and preventing spam mail remains a challenging and ongoing problem.

4.3 Further Work

Now onto further work as we have completed the coding process as required by the course work and tested the system. Looking further into the project a web interface will be designed that will make this project easier to use and add and research more algorithms which may have higher accuracy and make the prediction more trustable. Overall, it has given a very high accuracy of 99% using the svm algorithm and by tuning the hyperparameters. Hopefully soon this project will be fully functional and be used I the real world to detect spam emails which has been a problem. And may the designed project help encounter the problem.

5. References

Thashina Sultana, K. A. S. F. S. J. N., 2020. Email based Spam Detection. *International Journal of Engineering Research & Technology,* 9(6), p. 5.

Anon., 2022. *Tutotials point.* [Online]
Available at: https://www.tutorialspoint.com/scikit_learn/index.htm
[Accessed 9 1 2023].

Brownlee, J., 2020. *Machine Learning Mastery.* [Online]
Available at: https://machinelearningmastery.com/different-results-each-time-in-machine-learning/
[Accessed 9 1 2023].

Dunham, 2022. *dunhamweb.* [Online]
Available at: https://dunhamweb.com/services/artificial-intelligence-machine-learning
[Accessed 2 1 2022].

ICTEA, 2020. *ICTEA.* [Online]
Available at: https://www.ictea.com/cs/index.php?rp=%2Fknowledgebase%2F2063%2FiQue-el-correo-no-deseado-spam.html&language=english
[Accessed 10 1 2023].

jav, 2022. *javatpoint.* [Online]
Available at: https://www.javatpoint.com/types-of-machine-learning
[Accessed 11 1 2023].

Mahto, P., 2020. *medium.* [Online]
Available at: https://medium.com/mlpoint/pandas-for-machine-learning-53846bc9a98b
[Accessed 11 1 2023].

Priri Sharma, U. B., 2018. Machine Learning based Spam E-Mail Detection. *International Journal of Intellegent Engineering and Syatems,* 11(3), p. 10.

Pumrapee Poomka, W. P. N. K. K. K., 2019. SMS Spam Detection Based on Long Short-Term. *International Journal of Future Computer and Communication,* 8(1), p. 5.

Selig, J., 2022. *expert.ai.* [Online]
Available at: https://www.expert.ai/blog/machine-learning-definition/
[Accessed 10 1 2023].

Syed Ishfaq Manzoor, J. S., 2019. A Comparative Analysis of Machine Learning Techniques.

International Journal of Advanced Trends in Computer Science and , 8(3), p. 5.

Yildirim, S., 2021. *Towards Data Science.* [Online]

Available at: https://towardsdatascience.com/15-must-know-machine-learning-algorithms-44faf6bc758e

[Accessed 10 1 2023].

Yuliya Kontsewaya, E. A. A. A., 2021. Evaluating the Effectiveness of Machine Learning Methods for. *Procedia Computer Science ,* Volume 190, p. 486.