Table of Contents

Table Of Figures

Table Of Tables

1 Introduction

This project aims to design and develop a machine learning application which is capable of classifying SMS messages as either malicious messages (**Spam**) or real/legit messages (**Ham**), due to the growth of mobile communication SMS has become one of the primary targets for phishing attacks and unprompted marketing. This project aims to filter these types of messages in order to make user experience secure and solicit.

The project uses **Supervised Machine Learning** in which the model learns to map input data(Features) to output labels(Target Variable) based on the labelled dataset. For the project following concepts will be utilized:

- Natural Language Processing (NLP):
  - It is the field of AI focused on helping computers understand human language.
  - The use of NLP in this project is to clean the data (raw text) and convert it to numerical format for the computer to understand.

Figure 1 fig. NLP

- Vectorization:
  - It is the process of converting non numerical data into numerical vectors for processing.
  - For the project TF-IDF (Term Frequency-Inverse Document Frequency) will be used to transform text into numerical vectors.
  - TF-IDF's main concept is to assign weight based on their importance.

Figure 2 fig. Vectorization

- Classification Algorithm:
    - They are tools in ML which sorts data into categories or classes.
    - The project uses 3 distinct types of learning algorithms for classification model:
        - Naïve Bayes (Probabilistic)
        - Logistic Regression (Linear)
        - KNN K-Nearest Neighbour (Instance based)

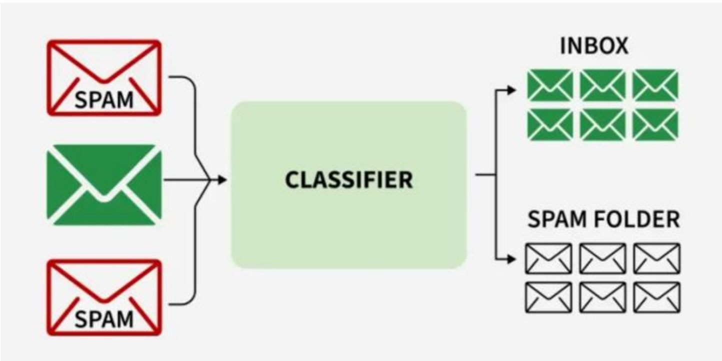

Figure 3 fig. Classification

2 Problem Domain

The **SMS Spam Detection** is a type of binary classification problem. The input is a string of text or sentences (SMS message) and the output is a binary label in our case we assign "0 for Ham" and "1 for Spam". The unstructured structure of the SMS data is a challenge as it contains less characters more often than not messages containing abbreviation, slangs, and misspelling with the addition of lack of metadata.

2.1 Dataset Description

Dataset used: UCI SMS Spam Collection

The project utilizes the UCI SMS Spam Collection which is a public dataset sourced from University of California, Irvine's (UCI) Machine Learning Repository.

Here are some facts that we can get from the dataset: (shown in figure 1)

- Type: .txt (the original dataset is in txt file format)
- No of rows of data: 5572
- Data Type: Unstructured English data (object)
- Class Imbalance: Heavily Imbalanced
- Ham=4824 (86.591276%)
- Spam=747 (13.408724%)
- Implications: Due to the heavy class imbalance the model may have less accuracy so Precision and Recall metrices will be more important than accuracy during the evaluation of the model.

```
print(df['ham'].value_counts())

ham
ham     4824
spam     747
Name: count, dtype: int64

print(df['ham'].value_counts(normalize=True)*100)

ham
ham     86.591276
spam    13.408724
Name: proportion, dtype: float64
```

Figure 4 fig. Ham and Spam class imbalance check

Societal or Business Relevance

- Cybersecurity: SMS phishing is a major threat in current time as it is used to steal banking details or other private information, this classifier is like the first line of defence against these types of threats.
- Business Integrity: Even now almost all top businesses and companies such as X, Meta etc rely on SMS for one time OTP but if due to constant spam email threats users stop trusting or using SMS it compromises security.
- User friendly: the spam filters automatically filters spam SMS saving time for the user and also decluttering mobile storage.

3 Solution

The solution developed for this implements a full machine learning pipeline:

- Text Preprocessing
- Feature Extraction
- ML Algorithms

Text Preprocessing: The raw data (text) is noisy so we apply the following techniques:

- Tokenization: Breaks sentences into words.
- Stop Word Removal: removes words like is, the, that, etc.
- Normalization: lowercases the text.
- Stemming/Lemmatization: convert word to root form.

Feature Extraction: Since the data is in textual form, we need to convert it to numerical form and for that following techniques are to be used:

- TF-IDF Vectorization converts text into numerical vectors. (This method was chosen because it normalizes count of words which prevents longer messages from having unfair weightage.)

Machine Learning Algorithm: Following machine learning algorithm are to be used:

- **Multinomial Naïve Bayes:**
  - assumes independence between features
  - Speed and high performance on high dimensionality data.
  - Uses techniques like Laplace smoothing to handle unseen words preventing zero probabilities

$$\underbrace{P(c \mid x)}_{\text{Posterior Probability}} = \frac{\overbrace{P(x \mid c)}^{\text{Likelihood}}\,\overbrace{P(c)}^{\text{Class Prior Probability}}}{\underbrace{P(x)}_{\text{Predictor Prior Probability}}}$$

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

Figure 5 fig. Multinomial Naïve Bayes

- **Logistic Regression**:
    - Learns by creating linear decision boundary between classes.
    - Uses sigmoid function.
    - Provides probabilities for prediction of class.

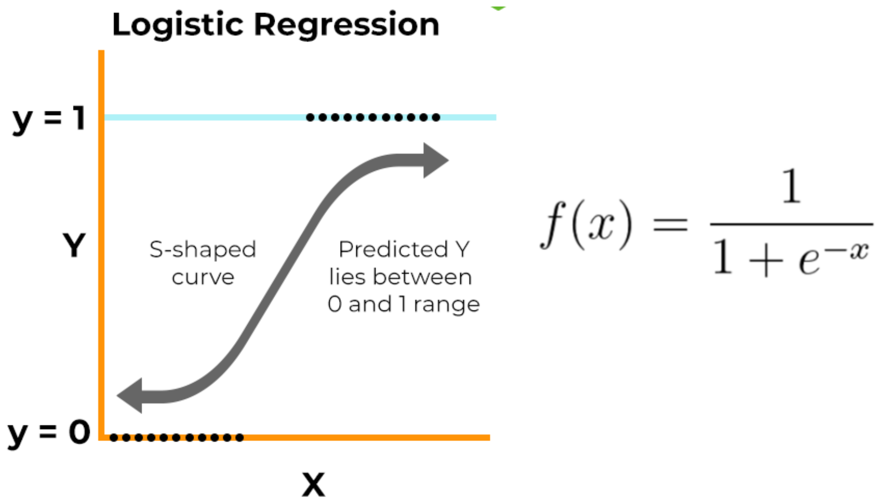

Figure 6 fig. Logistic Regression

- **KNN (K Nearest Neighbour):**
    - Classifies based on distance similarity.
    - Instance based supervised machine learning algorithm.

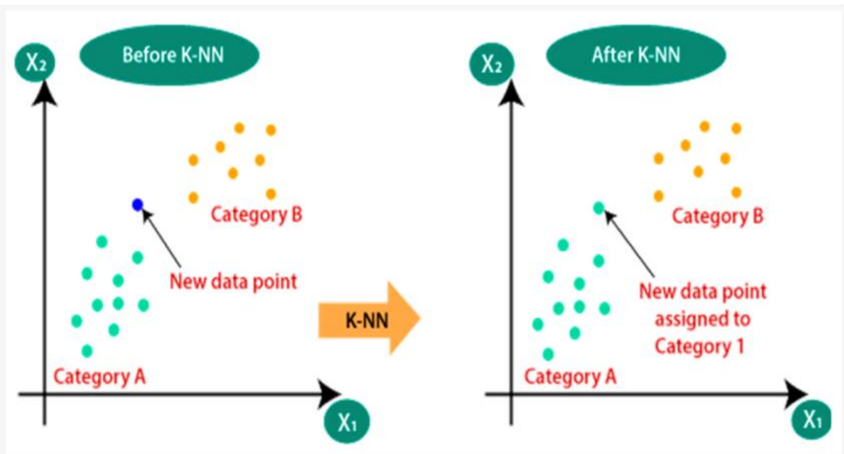

Figure 7 fig. K Nearest Neighbour

3.1 Pseudocodes

3.1.1 Multinomial Naïve Bayes

START

```
PROCESS NaiveBayes
    READ Training_Data


    COMPUTE Prior_Probabilities
        Total_Count = Total number of messages
        Spam_Prior = Count of Spam messages / Total_Count
        Ham_Prior = Count of Ham messages / Total_Count
    BUILD Vocabulary
        Collect all unique words from all messages
        Set Vocabulary_Size to count of unique word
    COMPUTE Word_Likelihoods (with Laplace Smoothing)
        FOR each word in Vocabulary
            Spam_Word_Prob = (Count of word in Spam + 1) / (Total words in Spam + Vocabulary Size)
            Ham_Word_Prob = (Count of word in Ham + 1) / (Total words in Ham + Vocabulary_Size)
    PREDICT New_Message
        Tokenize New_Message into words
        Initialize Spam_Score = Log(Spam_Prior)
        Initialize Ham_Score = Log(Ham_Prior)


        FOR each word in New_Message
```

IF word exists in Vocabulary

Spam_Score = Spam_Score + Log(Spam_Word_Prob)

Ham_Score = Ham_Score + Log(Ham_Word_Prob)

IF Spam_Score > Ham_Score

RETURN "Spam"

ELSE

RETURN "Ham"

END

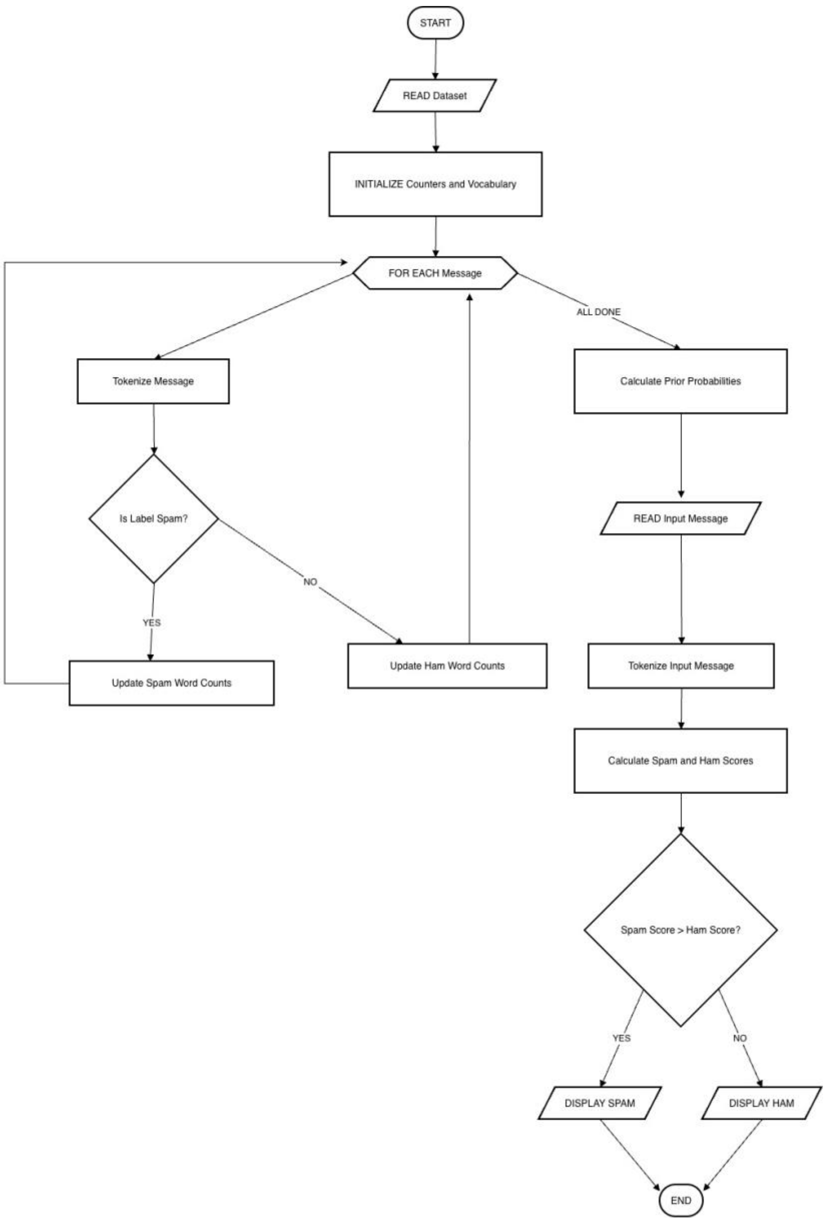

Figure 8 fig. Multinomial Naïve Bayes Flowchart

3.1.2 Logistic Regression

```
START

PROCESS LogisticRegression
  INITIALIZE Parameters
    Weights = Set all to 0
    Bias = 0
    Learning_Rate = Set alpha (e.g., 0.01)
    Iterations = Set total loops
  TRAIN Model
    FOR each iteration
      FOR each record in Dataset
        Calculate Linear_Output = (Weights * Input_Features) + Bias
        Calculate Probability = 1 / (1 + exp(-Linear_Output))
        Calculate Error = Probability - Actual_Label
        Update_Gradients
          Weight_Gradient = Input_Features * Error
          Bias_Gradient = Error
        Update_Parameters
          Weights = Weights - (Learning_Rate * Weight_Gradient)
          Bias = Bias - (Learning_Rate * Bias_Gradient)
  PREDICT New_Message
    Calculate Linear_Output = (Weights * New_Vector) + Bias
    Calculate Final_Prob = 1 / (1 + exp(-Linear_Output))
```

IF Final_Prob >= 0.5

RETURN "Spam"

ELSE

RETURN "Ham"

END

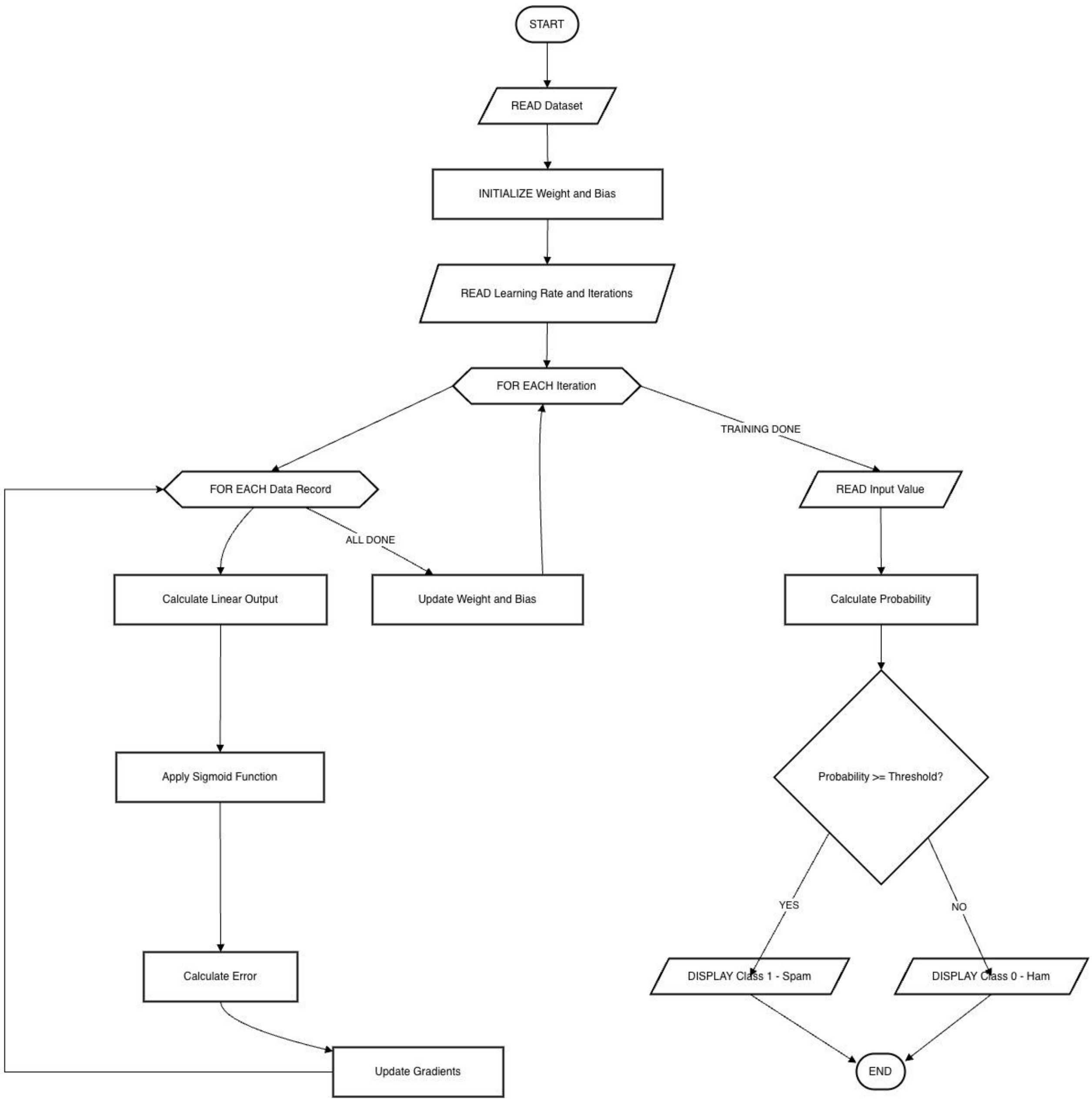

Figure 9 fig. Logistic Regression Flowchart

3.1.3 K-Nearest Neighbour

```
START

PROCESS KNN

   STORE Training_Data

      Load all message vectors and their labels

      Set K = number of neighbors (e.g., 5)

   PREDICT New_Message

      Initialize Distance_List = Empty list

      Vector_New = Convert new message to TF-IDF vector

      FOR each message in Training_Data

         Calculate Distance = Euclidean distance between Vector_New and
Training_Vector

         Add (Distance, Label) to Distance_List

      SORT Distance_List by Distance in ascending order

      SELECT K_Neighbors

         Identify the first K items in Distance_List

      MAJORITY_VOTE

         Count occurrences of "Spam" in K_Neighbors

         Count occurrences of "Ham" in K_Neighbors


         IF Spam_Count > Ham_Count

            RETURN "Spam"

         ELSE

            RETURN "Ham"

         END
```

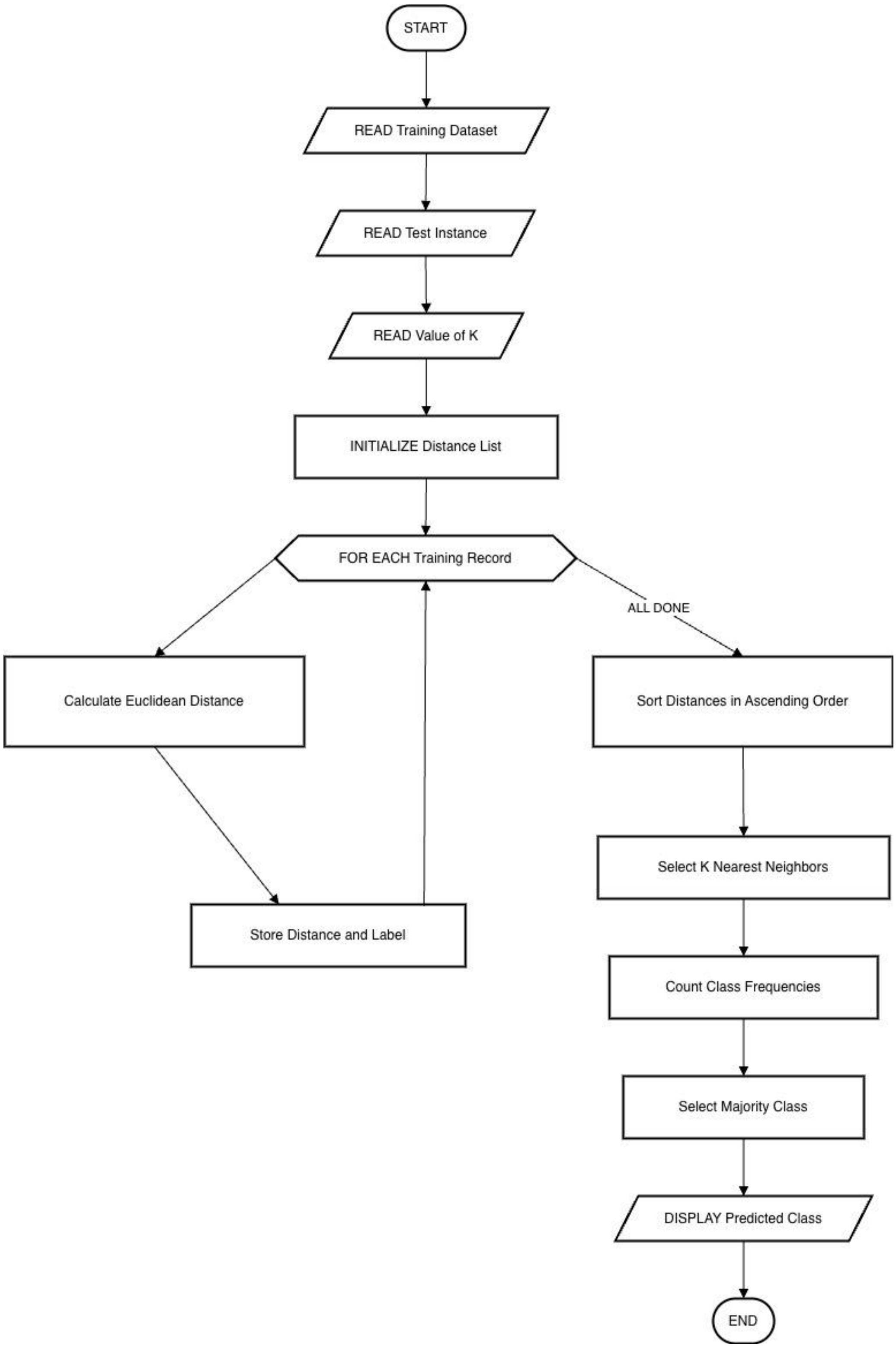

Figure 10 fig. KNN flowchart

3.2 State Transition Diagram

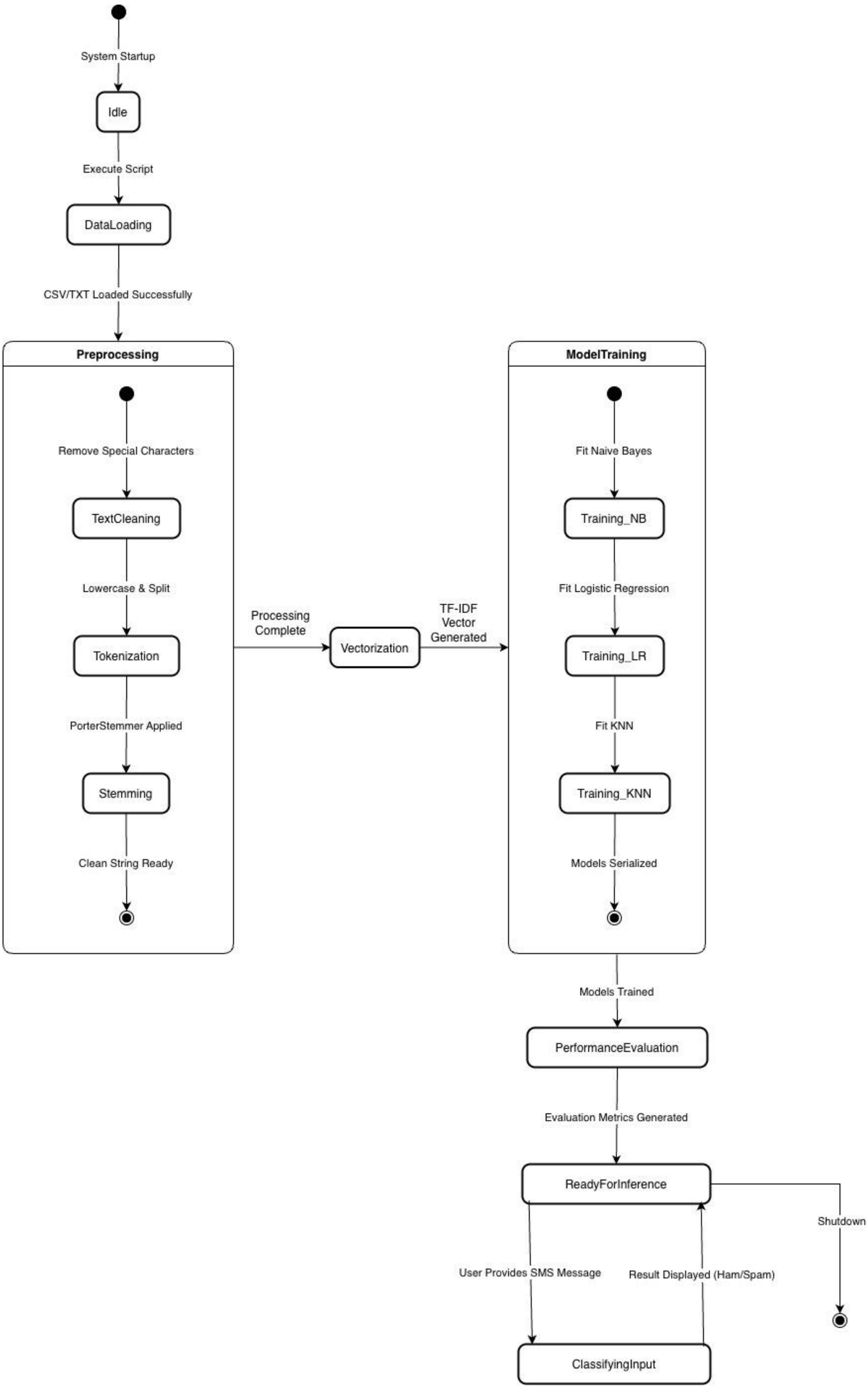

Figure 11 fig. Overall System State Transition Diagram

3.3 Development Process and Technologies

3.3.1 Development Phases

- **Phase 1: Data Loading and exploration**

  This process began with loading the dataset in the notebook. During exploration many problems were found, and solutions were created such as the need of Latin-1 encoding for the dataset to load, the raw data being in comma separated text file form and other details such as the no of data, features, class imbalance etc.

Project Initiation

```
[208]: df = pd.read_csv("Spam Dataset/SMSSpamCollection",sep="\t",header=None,names=["label", "message"],encoding="latin-1")
       # This Step is the most important since the data in the dataset in in txt format seperated by commas
       # If the dataset is already converted into a csv File then the upper commented code works too.

[210]: df.head() # ---- Looking if the data is in needed format or not
```

[210]:

	label	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...

Figure 12 fig. Dataset Loading

```
print(df['ham'].value_counts())

ham
ham      4824
spam      747
Name: count, dtype: int64

print(df['ham'].value_counts(normalize=True)*100)

ham
ham      86.591276
spam     13.408724
Name: proportion, dtype: float64
```

Figure 13 fig. Dataset Exploration

- Phase 2: Text Cleaning

  The next part is cleaning the text which includes:

  - Lowercasing
  - Removing signs, numbers and punctuations

```
[ ]: import re

[ ]: # re (regular expressions) handles all the extra characters like @, #, $ and such signs and symbols.

[ ]:

[276]: def data_preprocessing(message):
           # lowercasing all the messages.
           message = message.lower()
           # removing anything that is not a lower cased alphabet with blank space.
           message = re.sub(r'[^a-z]', ' ', message)
           # creating boundary between words while removing the double white spaces if there are any.
           message = " ".join(message.split())
           return message
```

Figure 14 fig. Text Cleaning

  - Stop words Removal
  - Stemming

```
[91]: from nltk.corpus import stopwords
      from nltk.stem import PorterStemmer

[70]: nltk.download('stopwords')

      [nltk_data] Downloading package stopwords to
      [nltk_data]     /Users/prashantrijal/nltk_data...
      [nltk_data]   Package stopwords is already up-to-date!
[70]: True

[72]: stemmer = PorterStemmer()
      stop_words = set(stopwords.words('english'))

[115]: def data_preprocessing_1(message):
           word = message.split()
           refined_words = [stemmer.stem(word) for word in message if word not in stop_words]
           return " ".join(refined_words)

[ ]:

[119]: data['final_processed_message'] = data['processed_message'].apply(data_preprocessing_1)

[113]: data.head()
```

[113]:

	label	message	processed_message	final_processed_message
0	1	Go until jurong point, crazy.. Available only ...	go until jurong point crazy available only in ...	go jurong point crazi avail bugi n great world...
1	1	Ok lar... Joking wif u oni...	ok lar joking wif u oni	ok lar joke wif u oni
2	0	Free entry in 2 a wkly comp to win FA Cup fina...	free entry in a wkly comp to win fa cup final ...	free entri wkli comp win fa cup final tkt st m...
3	1	U dun say so early hor... U c already then say...	u dun say so early hor u c already then say	u dun say earli hor u c alreadi say
4	1	Nah I don't think he goes to usf, he lives aro...	nah i don t think he goes to usf he lives arou...	nah think goe usf live around though

Figure 15 fig. Stopwrod removal and Stemming

- **Phase 3: Tokenization:**

Though this phase we will be tokenizing the cleaned messages.

Tokenization

```
[91]: from nltk.tokenize import word_tokenize

      # Downloading necessary resources
      nltk.download('punkt')

[nltk_data] Downloading package punkt to
[nltk_data]     /Users/prashantrijal/nltk_data...
[nltk_data]   Package punkt is already up-to-date!

[91]: True

[93]: def tokenize_message(message):
          # This function takes the cleaned string and breaks it into a list of words
          return word_tokenize(message)

[95]: data['tokens'] = data['final_processed_message'].apply(tokenize_message)

[97]: # Review the result
      print(data[['final_processed_message', 'tokens']].head())

                             final_processed_message  \
0  go jurong point crazi avail bugi n great world...
1                              ok lar joke wif u oni
2  free entri wkli comp win fa cup final tkt st m...
3                u dun say earli hor u c alreadi say
4               nah think goe usf live around though

                                              tokens
0  [go, jurong, point, crazi, avail, bugi, n, gre...
1                     [ok, lar, joke, wif, u, oni]
2  [free, entri, wkli, comp, win, fa, cup, final,...
3      [u, dun, say, earli, hor, u, c, alreadi, say]
4     [nah, think, goe, usf, live, around, though]
```

Figure 16 fig. Tokenization

- Phase 4: Vectorization and Train Test Split

Vectorization

```
[114]: from sklearn.feature_extraction.text import TfidfVectorizer
       from sklearn.model_selection import train_test_split

[116]: tfidf = TfidfVectorizer(max_features=3000)
       # We limit to 3000 features to keep the model efficient and remove rare noise

[118]: X = tfidf.fit_transform(data['final_processed_message']).toarray()
       # Transform the text into a numerical matrix (X)
       # We use the final cleaned column

[120]: y = data['label'].values
       # Extract the target labels (y)
       # Assuming your label column is named 'label'
```

Train Test Split

```
[122]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
       # Split the data into Training and Testing sets (80% Train, 20% Test)

[124]: print(f"Feature matrix shape: {X.shape}")
       print("Vectorization and Split Complete.")

Feature matrix shape: (5572, 3000)
Vectorization and Split Complete.

[ ]:
```

Figure 17 fig. Vectorization & Test Train Split

- **Phase 5: Model Implementation:**

  Now that the data has been split into train and test data we go into training the data using the following models:

  - **Multinomial Naive Bayes**:

Multinomial Naive Bayes

```
[134]: from sklearn.naive_bayes import MultinomialNB
       from sklearn.metrics import classification_report, accuracy_score

[132]: nb_model = MultinomialNB(alpha=1.0)
       # Configured with Laplace smoothing (alpha=1.0) for the starting
       nb_model.fit(X_train, y_train)

[132]: ▾ MultinomialNB
       MultinomialNB()

[136]: nb_pred = nb_model.predict(X_test)

[156]: accuracy_score(y_test, nb_pred)

[156]: 0.9811659192825112

[160]: print(classification_report(y_test, nb_pred))
              precision    recall  f1-score   support

           0       0.99      0.87      0.92       149
           1       0.98      1.00      0.99       966

    accuracy                           0.98      1115
   macro avg       0.99      0.93      0.96      1115
weighted avg       0.98      0.98      0.98      1115
```

Figure 18 Multinomial Naive Bayes

  - **Logistic Regression:**

Logistic Regression

```
[163]: from sklearn.linear_model import LogisticRegression

[165]: lr_model = LogisticRegression(solver='liblinear')
       lr_model.fit(X_train, y_train)

[165]: ▾ LogisticRegression
       LogisticRegression(solver='liblinear')

[167]: lr_pred = lr_model.predict(X_test)

[169]: accuracy_score(y_test, lr_pred)

[169]: 0.9713004484304932

[171]: print(classification_report(y_test, lr_pred))
              precision    recall  f1-score   support

           0       1.00      0.79      0.88       149
           1       0.97      1.00      0.98       966

    accuracy                           0.97      1115
   macro avg       0.98      0.89      0.93      1115
weighted avg       0.97      0.97      0.97      1115
```

3.3.2 Tools and Technologies Used

For the completion of the project following tools and technologies were used:

- **Programming Language and Environment**
  - **Python 3.0:** This is primary language chosen for the project due to its easiness and vast libraries.
  - **Jupyter Notebook:** This is the IDE (Integrated Development Environment) chosen for the project which is due to its cell-based structure and other auxiliary functions.
- **Data Manipulation and Analysis:**
  - Pandas: used to load and transform the data
- **Machine Learning and NLP**
  - **Scikit-learn:** The most critical library for the project this library holds all the core logic of the models that we ran e.g.
    - TFidfVectorizer
    - Train Test Split
    - Multinomial Naive Bayes, Logistic Regression
    - Evaluation Matrices
  - **Nltk:** The primary library mostly used for text preprocessing tasks such as:
    - Stopword removal
    - Stemming
  - **re (Regular Expression):** The library used to handle removal of punctuations, numbers and special charaters.

In summary:

Table 1 tbl. Technology Table

Category	Technology
Language	Python
Environment	Jupyter Notebook
Data Cleaning	Pandas, Re (Regex)
NLP	NLTK (Stemming, Stopwords)
Vectorization	TF-IDF (Scikit-learn)
ML Algorithms	Naive Bayes, Logistic Regression, KNN

4 Conclusion

The project was able to deploy a machine learning pipeline to recognize SMS messages by either classifying it as Ham or Spam extremely accurately. After comparing three algorithms Multinomial Naive Bayes, Logistic Regression, and K-Nearest Neighbors, the results showed that Naive Bayes and Logistic Regression were more applicable in this field of problem. The scores on recalls, especially the ones in the recognition of malicious spam, validate the fact that text preprocessing and TLT-IDF word vectors steps played an important role in converting unstructured text to meaningful features. Such a strict method enabled the models to solve the major issue of class imbalance in the dataset, both the genuine messages and the malicious ones will be filtered accordingly.

This application would be useful in the actual world as a first-level defense mechanism against both Smishing and automated phishing attacks and helps the users to avoid potential financial fraud and identity theft. Automation of the process of identifying unsolicited marketing and phishing links by the tool contributes to the safety of mobile communication to a large extent. The system may be extended with Deep Learning architectures, e.g. LSTMs or Transformers, to learn more about the semantics of messages. Also, the procedure of implementing the trained models as a real-time API would enable them to directly integrate it into messaging platforms to offer protection against any emerging cyber threats remotely.

References

GitHub: https://github.com/Prashant-Rijal-dev/SMS_Spam_Detection