Architecture:

Step 1: Data preprocessing

The extracted information is saved in the relevant database when the material is received. A feature vector was created for each message, with a total of 21700 attributes. The attribute n was either set to 1 if the appropriate word existed in a message or else was assigned to 0 if the term was not present.

Step 2: Description of the retrieved feature

Spam and ham content is removed by the feature extraction modules, after which feature dictionaries and feature vectors are created for the algorithm's input. Feature extraction is utilized to assist in classifier training and verification. In terms of the training component, this module takes note of the number of words that occur in the message text. We use the terms that appear the most frequently as well as terms from the class's core vocabulary. Training messages are represented by feature vectors, as well.

Step 3: Classification of spam

For training, we use standard classification message documents which we preprocess and extract useful information. We save the extracted information as text documents in a prescribed format, which are then broken down into words, after which we extract the spam document's feature vector. Using the feature vector of spam messages, the SVM technique is found to be the most optimal classification method.

Step 4: Evaluation of performance

To measure the efficacy of the above method, Spam Filtering researchers have utilized the most frequently used evaluation tools. Spam Precision (SP), spam recall (SR), and Accuracy (A) are the performance metrics.

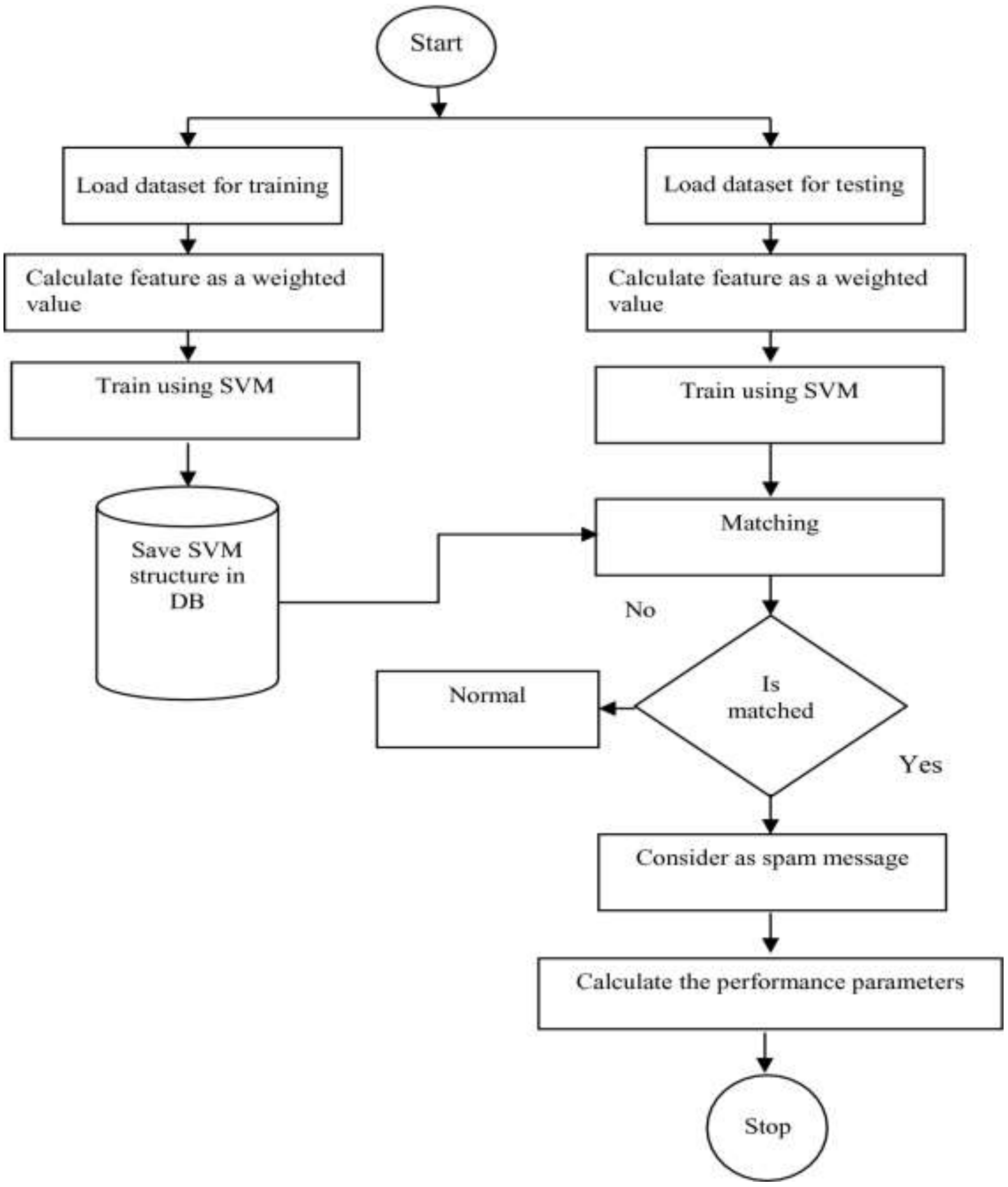


Figure 1. Architecture

3.2. Purification of Data

Because the survey has been made available to the public, the raw data collected from the survey can not be guaranteed to be acceptable for use in such a study. As a consequence, the data needs to be cleaned before further processing. Data cleaning is a procedure that finds inaccurate, incomplete, or infeasible data, such as duplicated, inconsistent data, documentation mistakes, or unqualified data, and then fixes or removes them to improve the quality of the data. Previous research required observing the whole collection of data to determine errors and lack of data.

3.3. Text Pre-processing

These text messages should be prepared before any are used during training or testing. Many processing steps, including case folding, punctuation removal, tokenization, URL handling, and stemming, are applied to the text before being used. This study doesn't take MWEs (Multiword Expressions) or text relevancy into consideration. While MWE may play a role in document clustering, the brief text currently lacks the relevance that MWE may have. For the sake of this explanation, we'll start by assuming that the text message is already all lower-case. To verify there are no case-sensitive problems, perform this step. A second factor is that the text message is stripped of any punctuation. It is assumed that no punctuation has an impact on the final result. Thirdly, tokenization is required because data training and testing techniques establish each token's probability in the text messages. Other messages you may get contain a URL or phone number, especially those with offers of money in exchange for fraud. Despite similar text messages, these URLs or phone numbers are distinct. Due to this, the findings in this study regard multiple URLs or phone numbers as interchangeable resources. We have replaced all URLs and phone numbers with the terms "URL address" and "phone number." After text pre-processing, the last stage is known as stemming. To retrieve the base word of the token, perform the following steps.

3.4. Data set balancing (Under-sampling or Down Sampling)

To offset the increased data, the abundant class size is reduced. To obtain a more equitable distribution of classes, it limits the number of majority samples. It may exclude valuable information, as it involves removing observations from the original data set. While it helps with the resolution of memory problems.

```
ham.sample = ham.sample (spam.shape[0])

ham.shape,spam.shape

((747, 4), (747, 4))
```

This data will be loaded into a new dataset.

```
OData = ham.append(spam, ignore index=True)
```

3.5. Text Vectorization

We found that the Bag of Words Model, the word vectorizer, was impractical for our Spam or Ham classifier after the examination because it only contains the most basic words. Each vectorizer has its own unique set of advantages, but figuring out which one to use is not always clear. Due to its simplicity of usage and effectiveness in vectorizing documents such as text messages, the TF-IDF vectorizer was utilized in this instance. Vectorization of documents is accomplished by producing a TF-IDF statistic between the document and each phrase in the lexicon. Every statistic in the document vector is applied.

To choose TF-IDF, we will have to choose the granularity of our vectorizer. It is a frequent option to assign each word its term instead of using a tokenizer. White space and special characters are used to split documents into tokens. Tokenizers can retrieve data that other types of analyzers may miss. While tokenizers may struggle with colloquial English, they will have problems with URLs and emails. This approach means we'll be employing a word-level analyzer, which groups words into categories. Before training the vectorizer, we divide our data into two sets: a training set and a testing set. We set aside 30% of our data for testing.

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split

(OData['message'], OData['label'], test_size = 0.3, random_state = 0, shuffle = True)

# vectorizer training
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

svm = Pipeline([("tfidf",TfidfVectorizer()) , ("classifier",SVC(C= 100, gamma ='auto'))])
```

3.6. Building and Testing the Classifier

During this step, classifiers will be selected and tested on the training set to see which one performs the best. We could assume that a Support Vector Machine performs well enough to ensure consistency. A higher value of C usually leads to a hyperplane with a smaller margin because it places a larger priority on precision over margin. The purpose is to detect the term C to influence objective function as a regularisation. The grid search can properly adjust these settings.

```
from sklearn import svc

svm.fit(X_train, y_train)
```

3.7.Performance Metrics:

The current framework was evaluated using the following standard metrics. To evaluate accuracy, recall, and precision, a specific dataset is needed. We have to figure out how many instances were correctly identified as belonging to the class, i.e., True Positives, the number of instances that were correctly identified but did not belong to the class, i.e., True Negatives, and the instances that were mistakenly classified as class examples, i.e., False Positives, as well as instances that weren't identified as class examples, i.e., False Negatives.

Accuracy may be defined as the total number of correct predictions to the total number of predictions.

Accuracy = (TP+TN) / (TP+TN+FP+FN).

Where TP = True Positives, TN = True Negatives, FP = False Positives, FN = False Negatives.

Recall (Sensitivity)

A statistical measure known as the recall is used to represent the ratio of successfully grouped positive models to the total number of positive models. The class was acceptable provided the recall rate was high.

Recall = TP / (TP+FN)

Precision

To estimate the accuracy of our classifier, we divide the total number of correctly classified examples by the number of predicted positive cases.

Precision = TP / (TP+FP).

F-Measure

It has an estimator that speaks to both measures (Precision and Recall) since we have two metrics (Precision and Recall). We developed an F-measure based on Harmonic Mean rather than Mean since it is more sensitive to unusual features. The formula for computing the F-measure may be found in the table below. The F-Measure will always be closer to the estimated precision or recall value than the expected precision or recall value.

F-measure = (2*Recall*Precision) / (Recall+Precision).

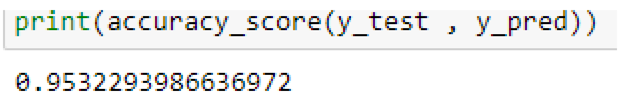
Let's test it.

```
from sklearn.metrics import accuracy_score

from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report

y_pred = svm.predict(x_test)

print(accuracy_score(y_test, y_pred))

print(confusion_matrix(y_test, y_pred))

print(classification_report(y_test, y_pred))
```

IV. Result

In this study, we used the "spam.tsv" file, which contains 5572 emails, 747 of which are spam and 4825 of which are ham. We achieved an accuracy score of 95.32 percent after applying all the techniques of data collecting, preprocessing, balancing, and then applying the Support vector machine algorithm to categorise the messages into spam or ham.

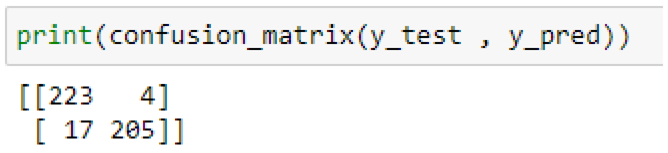
Accuracy Score:



```
print(accuracy_score(y_test , y_pred))

0.9532293986636972
```

Figure 2. Accuracy Score

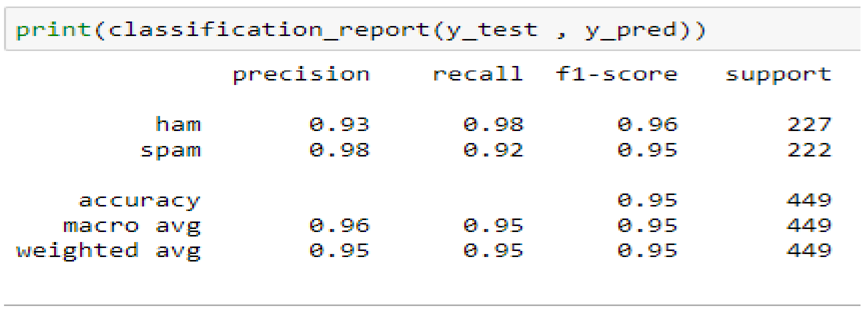
Confusion Matrix:



```
print(confusion_matrix(y_test , y_pred))

[[223   4]
 [ 17 205]]
```

Figure 3. Confusion Matrix

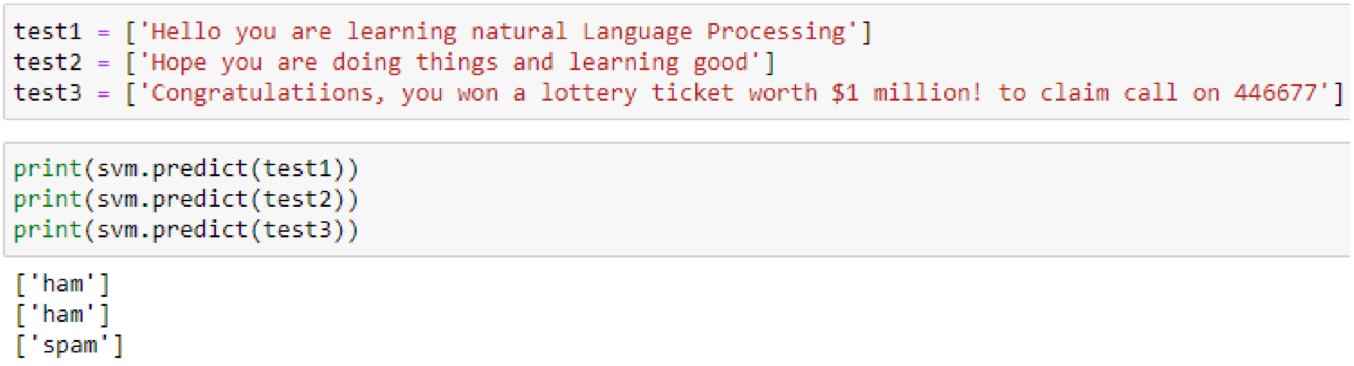
Classification Report:



```
print(classification_report(y_test , y_pred))

              precision    recall  f1-score   support

         ham       0.93      0.98      0.96       227
        spam       0.98      0.92      0.95       222

    accuracy                           0.95       449
   macro avg       0.96      0.95      0.95       449
weighted avg       0.95      0.95      0.95       449
```

Figure 4. Classification Report

Validating our model through the use of real-world examples :



```
test1 = ['Hello you are learning natural Language Processing']
test2 = ['Hope you are doing things and learning good']
test3 = ['Congratulatiions, you won a lottery ticket worth $1 million! to claim call on 446677']
```

```
print(svm.predict(test1))
print(svm.predict(test2))
print(svm.predict(test3))
```

```
['ham']
['ham']
['spam']
```

Figure 5. Random Input

V. CONCLUSION

Unfortunately, the need for total accuracy in spam classifications has not yet been fully satisfied, according to this research. To put it bluntly, spam is one of the biggest nuisances to the global PC environment. We propose a novel spam detection method in this study that is effective at telling spam apart from its contents. With sparse data structures and suitable recall and precision values, SVM can handle data with a huge number of features. Also, the SVM is often viewed as a critical kernel method, and this is very important in machine learning. The user can either filter spam or continue to receive real emails. During classifying a collection of datasets, the recommended classifier gets a classification accuracy of 95.32 percent. Next, a mobile application will be developed using the model that was built in this study to detect spam text messages. To be able to better identify spam text messages, it is also required to construct a server that will be used to generate a better model. It is necessary to introduce fresh spam messages to the model regularly to improve the model. It may be used as a tool to provide the most current spam text messages, delivering the most recent spam text messages that have been flagged by the user. The resulting server model creates a new form of spam text message detector.

Acknowledgment

I would like to express my deep gratitude to my professor, Dr. B. Indira Reddy, for guiding me through the process of writing this research paper.

REFERENCES

[1] D. Puniškis, R. Laurutis, R. Dirmeikis, An Artificial Neural Nets for Spam e-mail Recognition, electronics and electrical engineering ISSN 1392 – 1215 2006. Nr. 5(69)

[2] H. Najadat, N. Abdulla, R. Abooraig, and S. Nawasrah, ''Mobile SMS spam filtering based on mixing classifiers,'' Int. J. Adv. Comput. Res., vol. 1, no. 1, pp. 1–7, 2014.

[3] Q. Xu, E. W. Xiang, Q. Yang, J. Du, and J. Zhong, ''SMS spam detection using noncontent features,'' IEEE Intell. Syst., vol. 27, no. 6, pp. 44–51, Nov./Dec. 2012.

[4] A. Modupe, O. O. Olugbara, and S. O. Ojo, ''Filtering of mobile short messaging service communication using latent Dirichlet allocation with social network analysis,'' in Transactions on Engineering Technologies. Springer, 2014, pp. 671–686.

[5] A. Skudlark, ''Characterizing SMS spam in a large cellular network via mining victim spam reports,'' in Proc. 20th ITS Biennial Conf., Rio de Janeiro, Brazil Nov./Dec. 2014, pp. 1–23.

[6] C. F. M. Foozy, R. Ahmad, and M. F. Abdollah, ''A framework for SMS spam and phishing detection in malay language: A case study,'' Int. Rev. Comput. Softw., vol. 9, no. 7, pp. 1248–1254, 2014.

[7] Youn and Dennis McLeod, " A Comparative Study for Email Classification, Seongwook Los Angeles" , CA 90089, USA, 2006.

[8] L. Chen, Z. Yan, W. Zhang, and R. Kantola, ''TruSMS: A trustworthy SMS spam control system based on trust management,'' Future Generat. Comput. Syst., vol. 49, pp. 77–93, Aug. 2015.

[9] H. Saeed and W. Waheeb, ''The performance of soft computing techniques on content-based SMS spam filtering,'' M.S. thesis, Dept. Elect. Eng., Univ. Tun Hussein Onn Malaysia, Johor, Malaysia, 2015.

[10] E.-S. M. El-Alfy and A. A. AlHasan, ''Spam filtering framework for multimodal mobile communication based on dendritic cell algorithm,'' Future Generat. Comput. Syst., vol. 64, pp. 98–107, Nov. 2016.

[11] E. Ezpeleta, U. Zurutuza, and J. M. G. Hidalgo, ''Short messages spam filtering using personality recognition,'' in Proc. 4th Spanish Conf. Inf. Retr., 2016, p. 7.

[12] M. Z. Rafique and M. Abulaish, ''Graph-based learning model for detection of SMS spam on smartphones,'' in Proc. 8th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC), Aug. 2012, pp. 1046–1051.

[13] T. A. Almeida, T. P. Silva, I. Santos, and J. M. G. Hidalgo, ''Text normalization and semantic indexing to enhance instant messaging and SMS spam filtering,'' Knowl.-Based Syst., vol. 108, pp. 25–32, Sep. 2016.