# Performance Evaluation on Vector-Model Combinations

***Increment 2***

**GitHub link**: *https://github.com/trela47/NLP_Project*

**Team Members:**

Blessy Kuriakose - 11230255

Saisri Teja Pepeti - 11555656

Saikiran Yedulla - 11518301

Yamuna Bollepalli - 11552426

**Introduction:**

In this research, we want to test how well various vectorization-and-model combinations perform when used in the text-based spam detection prediction job. Through this study, we will be able to determine which vectorization will operate with the best model correctness and least amount of time complexity. There are various vectorization methods available, but there is no suitable model to determine which method and model work best for a particular dataset. We will be able to supply the precise model and time complexity for a spam detection dataset in this project. Therefore, since they are already aware of the model that works best for a certain dataset, this saves both businesses and individuals time.

**Background:**

Existing methods implemented in papers include a straightforward absence-presence check on keywords to see if an email has ‘traits’ that qualify it as a spam or not, however this method does not offer the best accuracy in categorization for all emails, especially in cases where keywords found in spam might appear in legit emails. This scenario would create a false positive prediction for spam which is detrimental to the accuracy and, in use case, to the email recipient. There are also direct tokenization methods which produce an accuracy that reaches only 89%. Our aim was to improve the accuracy of existing models, a task which requires us to implement multiple models including Decision tree, SVM, and Random Forest among others, and compare them for the best accuracy with lower time complexity.

**Model and Workflow:**

Several models were undertaken including Naive Bayes, Decision Tree, Support Vector Machines, Random Forest, LSTM/BILSTM, Word2Vec, FastText

For Naive Bayes, Decision Tree, Support Vector Machines, Random Forest, the models were used directly from the 'scikit-learn ' library. The workflow included splitting the shuffled data samples for training and testing, and preprocessing the tf-idf 'features' of the emails to be used for training, testing, and predictions, given the corresponding labels. Preprocessing involved a train_test_split of 0.7 (train) to 0.3 (test), lowercasing, TextBlob, making the text string into a list of lemmatized words, and getting the TF-IDF of each sample.

For BiLSTM, the architecture is as follows:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 2000, 32)          1600000

 bidirectional (Bidirectiona  (None, 128)              49664
 l)

 dense (Dense)               (None, 16)                2064

 dropout (Dropout)           (None, 16)                0

 dense_1 (Dense)             (None, 1)                 17

=================================================================
Total params: 1,651,745
Trainable params: 1,651,745
Non-trainable params: 0
_________________________________________________________________
None
```

In this iteration, a train_test_split of 0.8 to 0.2 was used and a separate, extensive preprocessing was done to the text itself before it is tokenized and padded for input to the model. A LabelEncoder was used on the ham/spam labels before training and testing.

For Word2Vec, gensim's Word2Vec was utilized to create features from the text after other preprocessing steps including getting rid of non-letters, lower-casing all text, stopword removal, and stemming were completed. The following LSTM model was utilized to train and test this data:

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 300, 100)          3417300

 dropout_2 (Dropout)         (None, 300, 100)          0

 lstm_3 (LSTM)               (None, 64)                42240

 dense_4 (Dense)             (None, 1)                 65

=================================================================
Total params: 3,459,605
Trainable params: 42,305
Non-trainable params: 3,417,300
_________________________________________________________________
```

In FastText, we used a 'spamcorpusfile.txt' to do unsupervised training on a ‘skipgram’ model through fasttext. All words in the dataset were manually set to have ‘occurred’ at least once, even if they are not present in the corpus. From there a sequential model with an embedding layer, a Bidirectional-LSTM, and a dense layer utilizing a sigmoidal activation was made. Adam optimizer was also utilized in this model. The model was trained in 20 epochs of batch size 32 with early stopping mechanisms in place.

**Dataset & Analysis of Data:**

We are using a dataset of emails which contain a number of emails in text format and they are coupled in a single folder. Other than the normal alphabet found in words, the text also includes numbers, punctuations, and special characters. When we converted those dataset into an excel sheet we have absorbed nearly 940 emails. It comes pre-split into training and testing sets and contains one line containing the subject heading (not used in our implementation) and another with the body of the email.

From the data set we will be able to check all the different kinds of emails. Then they are zipped into one file so that we will be able to provide this file as an input to our project. With this as a bases, the text data was tokenized and padded before being split. A model was made with neural

**Implementation:**

- All the necessary libraries, such as Pandas, Matplotlib, Numpy, scikit-learn , NLTK, and TextBlob, are imported. Next, PUNKT and wordnet are downloaded from NLTK.
- We will verify our classification models using these measures to see which one performs better by importing the classification report, F1 score, accuracy score, confusion matrix, and roc auc score

metrics from scikit-learn .

● We immediately started data preprocessing; under preprocessing data cleaning, we eliminate all the extraneous data that has an impact on the performance of our models.

●Post loading the data set, we brought up the header information with labels for ham and spam and label numbers for the supplied text.

● Our text and "label" (spam or ham) data are given train and test sets with text size 0.3 (30%) and random state 42.

● After that, conduct lemmatization on each word.

● We were able to determine the polarity of sentences by utilizing Textblob, which contains built-in rules.

● Tf-idf and word2vec are utilized to understand the underlying patterns of these raw data. To construct and test several models, the emails were vectorized using the Tf-idf (Term frequency- Inverse document

● The function-based probabilistic model Tf-idf makes advantage of the nuances of word patterns and usage context in a mathematical formulation to create predictions on category, in this case spam vs. ham, using frequency and the inverse document (not-spam).

● Identifying the line that contains the email's content, segmenting each line into a list of words, eliminating "stopwords," and choosing only words with alphabetic characters from the list of words for further processing.

● Building a word dictionary out of the training set's 2500 most prevalent words (chosen from those prepared in step one).

● We are training our models after applying BOW and TF-Idf vectorization transformer on our data.

● Extraction process using Word2vec

● A variety of supervised methods are used to train the model (Naive Bayes, Random Forest, Support Vector Machine, Decision tree.)

●Select the most precise model, then use it to project the test data (predicting the emails as spam or not spam)

● We created a function that includes all the validation signs in order to analyze our test.

● After achieving validation on all of our models, we additionally plotted confusion matrices to determine the variation between what was expected and what was predicted.

Naive Bayes - 86%

Decision Tree - 94%

Random Forest - 95%

SVM - 98.88%

Fasttext -98.4%

● In order to determine the class to which a multiple word string belongs, the LSTM is then performed. This is quite helpful when working with natural language processing. If the proper layers of embedding and encoding are utilized in the LSTM, the model will be able to discern the true meaning in the input string and will offer the most accurate output class. In the code that follows, the concept of using LSTM for categorizing ham,spam sentences is further discussed.

● The final sequence is created by extracting the features from the vectors after the embedding layer has converted the texts into vectors. The completely linked layer will then use a softmax function to classify the final sequence.

● By setting the LSTM training size to 20% of the test size, we can clean up the text in sentences by replacing newlines and adding numbers, white space, punctuation, special characters, and hyperlinks.

● After cleaning, we divide the data into email train and email test for training and testing. To train and test our model, we are using label encoder to convert the text data into numerical values. This model has a maximum of 50000 features and a maximum of 2000 characters.

● Tokenizing the string next, using the pad sequence function to pad them into sequences

●Dense, Input, LSTM, Embedding, Dropout, Activation, and Bidirectional for Bi-LSTM are imported from Keras.

After training, use embedding to embed all of the words.Relu and Sigmoid activation functions for each sentence's detailed scaling will help us improve our accuracy because they enable our LSTM model to precisely check each point.Binary entropy for optimizer and metrics accuracy, and Adam for loss function validation

● For data validation, we are using X text features and test y, and training our model using a batch size of 32 and an epoch size of 20.

● For a more in-depth understanding, we visualized the graph of model accuracy relative to accuracy; we received a test result of 98.8%..

● With regard to true labels and predicted labels, we plotted a confusion matrix.

● We label the data separately for spam and given text

● We are transforming our data using label encoder, and we are using porterstemmer to stem all the words and compare them to the words already present in the ham

● With the help of sigmoid activation function, we employed a sequential approach to arrange all the training data, adding word embedding layers and LSTM for the specified dropout and repeated dropout functions. By dividing the data and dividing it into training and testing, this was done as usual.

● We received 3417300 parameters with shapes of 300 and 100 under the embedding layer.

●Consequently, we finally achieved 97% for Word2Vec and plotted a confusion matrix for the same.

● FastText renders each word as an n-gram of characters rather than immediately learning word vectors.

● Another embedding technique that builds on the word2vec paradigm is fastText.

● In this model we obtained an accuracy if *98.46%. Precision: 97.97%, Recall: 96.67%, F1 Score: 97.32%*

**Analysis of implementation:**

The main goal of implementing such models, in contrast to the models that are already accessible on the internet, is that those models are well-trained over millions of records so they could eventually achieve good accuracy, and that the data set plays a major role where, preprocessing the data for each model, is what allows us to measure the accuracies by comparing to each model with respect to accuracies.

**Results:**

Upon examination we find that the second BiLSTM model does the best of all eight models examined in this project in terms of accuracy with the Support Vector Machine (SVM) model and the FastText model coming in a close second and third, respectively . **Table 1**, below, compares the metrics of each of the above eight models completed in this project.

**Table 1: Comparison of results of all models.**

| Model | Accuracy | Precision | Recall | F1-Score | AUC |
|---|---|---|---|---|---|
| Naive Bayes | 0.8840 | 1.0 | 0.5755 | 0.7305 | 0.7877 |
| Decision Tree | 0.9214 | 0.8495 | 0.8656 | 0.8575 | 0.9040 |
| SVM | 0.9865 | 0.9590 | 0.9929 | 0.9757 | 0.9885 |
| Random Forest | 0.9575 | 0.9303 | 0.9127 | 0.9214 | 0.9435 |
| BiLSTM | 0.9585 | 0.9169 | 0.9490 | 0.9327 | -- |
| Word2Vec | 0.8502 | 0.6701 | 0.9135 | 0.7731 | -- |
| FastText | 0.9884 | 0.9865 | 0.9733 | 0.9799 | -- |

**Table 2**, below, gives the confusion matrix values for the scikit-learn models of Naive Bayes, Decision Tree, Support Vector Machines (SVM), and Random Forest.

**Table 2: Confusion matrices of the four scikit-learn models.** *For each matrix, top left is true negative (not spam), top right is false negative, bottom left is false positive, and bottom right is true positive (spam).*

| Naive Bayes | | Decision Tree | | SVM | | Random Forest | |
|---|---|---|---|---|---|---|---|
| 1128 | 0 | 1063 | 65 | 1110 | 18 | 1099 | 29 |
| 180 | 244 | 57 | 367 | 3 | 421 | 37 | 387 |

The figures below (**Figures 1, 2, and 3**) give the results of the BiLSTM, Word2Vec, and FastText models, respectively.

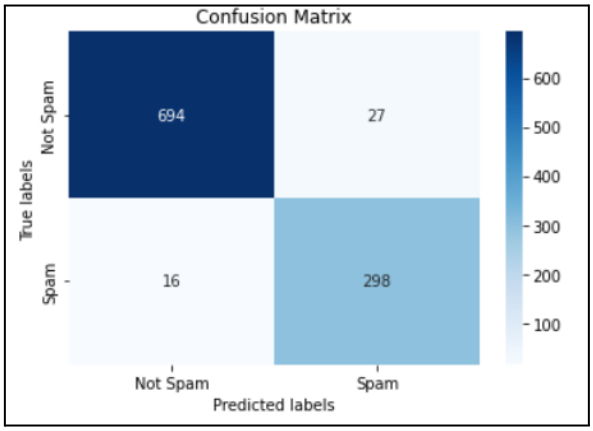


**Figure 1: Confusion matrices of BiLSTM model.** *Accuracy: 95.85%, Precision: 91.69%, Recall: 94.90%, F1 Score: 93.27%.*

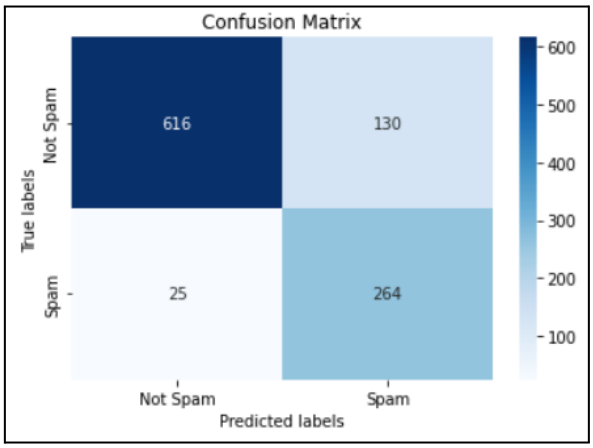


**Figure 2: Confusion matrix of Word2Vec model.** *Accuracy: 85.02%, Precision: 67.01%, Recall: 91.35%, F1 Score: 77.31%.*

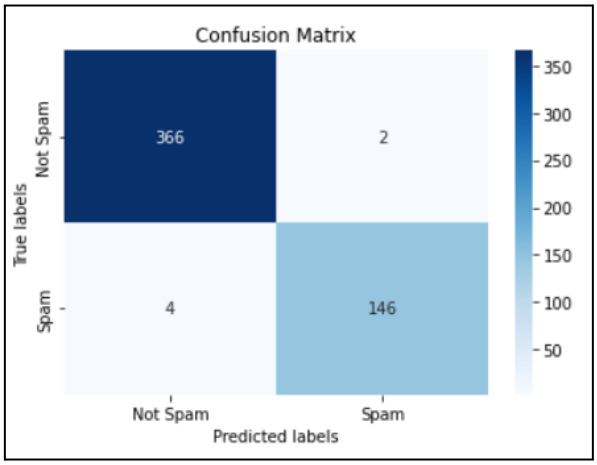


**Figure 3: Confusion matrix of FastText model.** *Accuracy: 98.84%. Precision: 98.65%, Recall: 97.33%, F1*

*Score: 97.99%.*

As we can see, FastText does the best in terms of accuracy, however it should be noted that the number of samples used for training in each model varies, as can be seen by the number of samples that remain to be used for testing.

**Responsibility (Task, Person):**

Data Preprocessing: Saikiran Yedulla, Blessy Kuriakose

TF-IDF: Saikiran Yedulla, Blessy Kuriakose

Bag-Words : Yamuna Bollepalli, Saisri Teja Pepeti,

Decision Tree: Saisri Teja Pepeti,

Random Forest: Saikiran Yedulla,

Naive bayes: Blessy Kuriakose

SVM:Yamuna Bollepalli,

Data Preprocessing: Saisri Teja Pepeti, Yamuna Bollepalli

LSTM & BILSTM: Saisri Teja Pepeti, Yamuna Bollepalli

Word2Vec: Saisri Teja Pepeti, Yamuna Bollepalli, Saikiran Yedulla, Blessy Kuriakose

FastText: Saisri Teja Pepeti, Yamuna Bollepalli, Saikiran Yedulla, Blessy Kuriakose

**Contributions (members/percentage) :**

Blessy Kuriakose - 25%

Saisri Teja Pepeti - 25%

Saikiran Yedulla - 25%

Yamuna Bollepalli - 25%

**References**

https://medium.com/analytics-vidhya/magic-of-tf-idf-202649d39c2f

https://neptune.ai/blog/vectorization-techniques-in-nlp-guide

https://www.sketchbubble.com/en/presentation-project-life-cycle.html

https://machine-learning.paperspace.com/wiki/machine-learning-models-explained

https://www.kaggle.com/datasets/benros0305/spamdetect

**Recording Link :** https://youtu.be/zwTZqv-x-SU