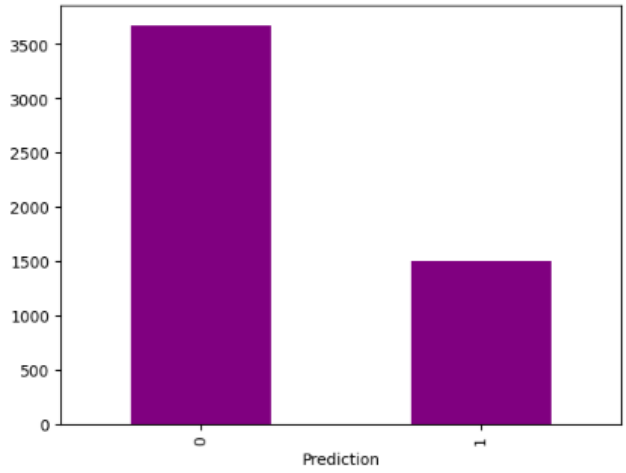
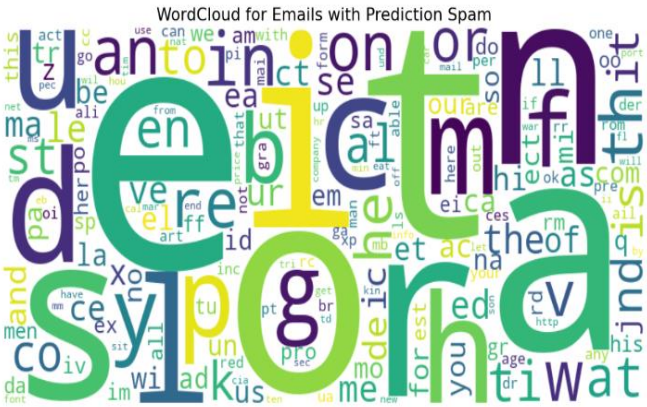
1.Exploratory Data Analysis (EDA)

To begin with, check the balance of spam – ham emails.



From the imbalanced nature of the dataset we conclude that a balancing technique is necessary before the model implementation, so the model can learn to correctly predict a mail as spam.

This WordCloud visualization shows the most frequently occurring words in emails classified as "Spam." The words like "en," "de," "st," "on," "re," "pro," and others are large and bold in the WordCloud, indicating they appear more frequently in the spam emails. The absence of clearly meaningful words might suggest attempts to use generic or filler text in spam messages. The frequent use of words such as "you" or "pro" might indicate spam targeting individuals with offers or promotions.



2.Balancing Techniques

2.1 SMOTE (Synthetic Minority Oversampling Technique)

SMOTE is an oversampling technique used to address class imbalance in datasets by generating synthetic examples for the minority class. It creates new data points along the line segments between existing minority class instances.

How It Works:

1. For each instance in the minority class, SMOTE identifies its k-nearest neighbors (default is 5).
2. A random neighbor is chosen, and a synthetic data point is created by interpolating between the original instance and its neighbor.
3. The process is repeated until the desired level of oversampling is achieved.

2.2 Cluster Centroids (Undersampling Technique)

Cluster Centroids is an undersampling technique used to address class imbalance by reducing the number of majority class samples. It uses clustering to summarize the majority class with a smaller number of representative data points (centroids).

How It Works:

1. Applies a clustering algorithm (e.g., K-Means) to the majority class.
2. The number of clusters is set to match the size of the minority class.
3. The centroids of the clusters become the new representative samples of the majority class.

2.3 Class Weights

Class Weights is a balancing technique used to handle class imbalance in classification tasks, but it works differently from oversampling or undersampling methods. Instead of altering the dataset, it adjusts the learning process by assigning a higher penalty to misclassified examples of the minority class.

How It Works:

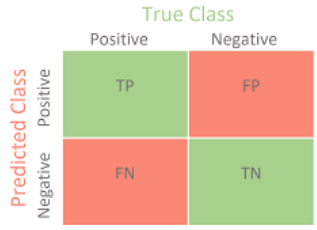
1. Assign higher weights to the minority class and lower weights to the majority class.
2. These weights modify the loss function so that errors on the minority class contribute more to the total loss.

If you combine SMOTE with class weights, the synthetic samples from SMOTE provide more data for the minority class, while the class weights ensure that misclassifications on the minority class are penalized more.

Using both undersampling and class weights helps to strike a balance between reducing majority class dominance and emphasizing the importance of the minority class.

3. Model Evaluation

I made a function to measure the scores of each model. It prints the confusion matrix,



the accuracy, the recall, the precision and the F-measure of the model.

$$(10.1)\ Accuracy = \frac{T_p + T_n}{T_p + T_n + F_p + F_n}$$

$$(10.2)\ Precision = \frac{T_p}{T_p + F_p}$$

$$(10.3)\ Recall = \frac{T_p}{T_p + T_n}$$

$$(10.4)\ F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Last, the Roc_Curve visualizes the true and the false positive rate of the model and is a common method for model evaluation in binary classifiers. The precision-recall curve is also a tool for assessing the performance of binary classification algorithms, particularly in scenarios with significant class imbalance. Similar to ROC curves, precision-recall curves offer a graphical representation of a classifier's performance across various thresholds, rather than summarizing it with a single value. A single measure of its own is not enough to come to a conclusion for the model's performance.

4.Methodology

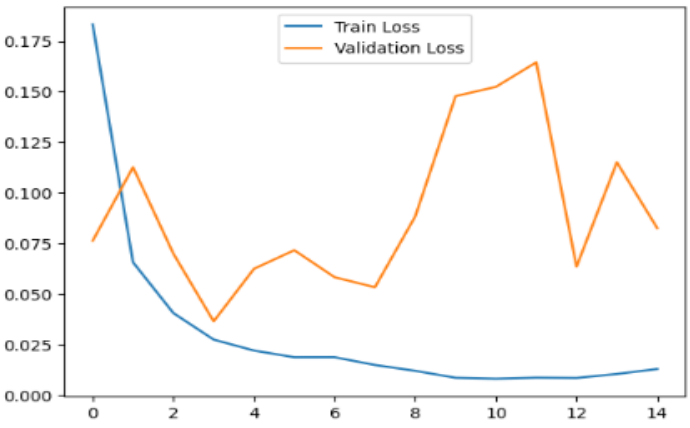
I split the dataset 20% for test set and 80% for train set, and I further split the train set to 10% validation set and 90% training set.

4.1 Oversampling Balancing Technique

I started by using SMOTE and class weights (the weights are {0: 0.704248366013072, 1: 1.724}) for data balancing, and standar scaler to normalize the data. I used both accuracy and F1-score to compile three different models.

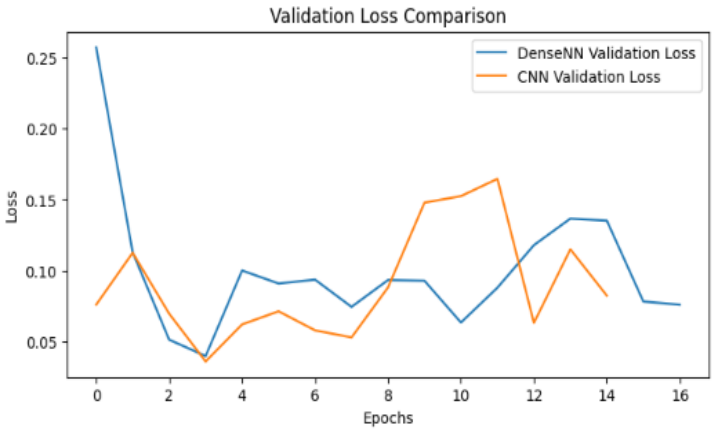
A Dense NN with 4 hidden layers with ReLU activation and output layer with sigmoid activation for binary classification. The model contains Batch Normalization which stabilizes training by normalizing inputs to each layer and Dropout Layers to prevent overfitting. The model was trained on 17 epochs.

A CNN (Convolutional Neural Network) with 1D convolutions to process data, one pooling layer and one dense layer to classify. Also, Dropout (0.3) and BatchNormalization layer. The output layer used sigmoid activation and the training happened on 15 epochs.

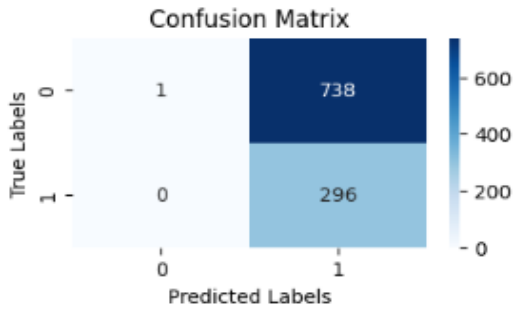


This graph depicts the training and validation loss during the training of the model. The spikes in validation loss suggest possible noise or imbalance in the validation data. Also, validation loss is much higher than training loss in some epochs, which is a sign that the model is overfitting the training data.

The previous model has slightly less fluctuations on the validation loss.



A Long Short-Term Memory (LSTM) model with three layers and dropout layer. The output layer used sigmoid activation and the training happened on 8 epochs. LSTM is a type of recurrent neural network (RNN) architecture specifically designed to handle sequential data and overcome the limitations of traditional RNNs, such as vanishing and exploding gradients.

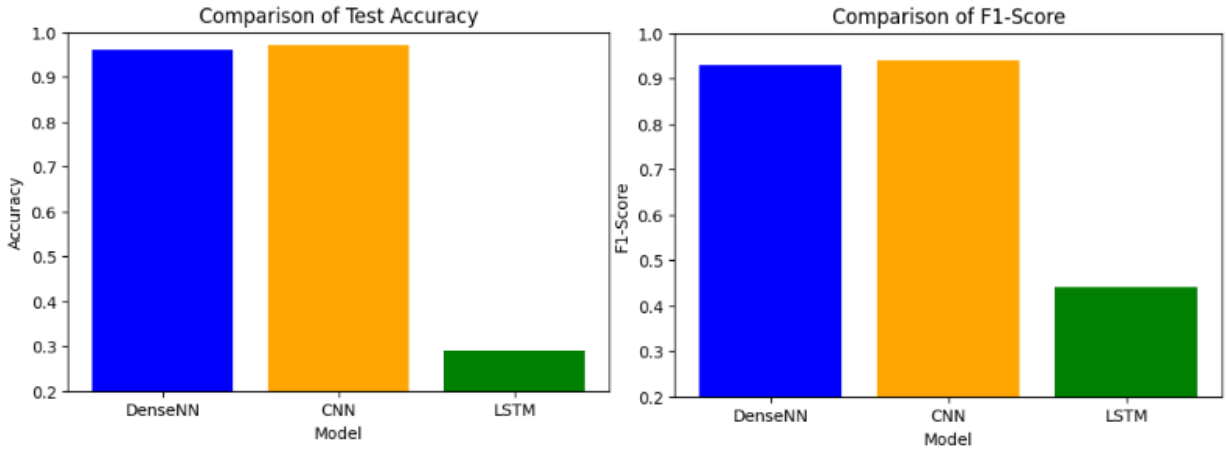


This model predicts as spam almost all emails. Recall is perfect (100%) because the model successfully identified all spam emails without missing any. Accuracy is quite low, meaning the model is not making good predictions overall. Apart from that this model is very time costly.

```
{'Accuracy (%)': '28.70',
 'Precision (%)': '28.63',
 'Recall (%)': '100.00',
 'F1-Score (%)': '44.51'}
```

Comparison of the models:

Model	Train Accuracy	Validation Accuracy	Test Accuracy	Precision	Recall	F1-Score
DenseNN	0.99	0.98	0.96	0.90	0.97	0.93
CNN	1.00	0.97	0.97	0.92	0.97	0.94
LSTM	0.50	0.28	0.29	0.28	1.00	0.44



We spot that from the three models CNN performed better, but it may overfit as we observed before.

4.2 Undersampling Balancing Technique

I continued using cluster centroids undersampling technique together with class weights on two models:

A Dense NN with 4 hidden layers with ReLU activation and output layer with sigmoid activation for binary classification. The model contains three Dropout Layers to prevent overfitting and was trained on 10 epochs.

A CNN (Convolutional Neural Network) with 1D convolutions to process data, one pooling layer and one dense layer to classify. Also, two Dropout Layers (0.3) and BatchNormalization layer. The output layer used sigmoid activation and the training happened on 15 epochs.

Comparison of the models:

Model	Train Accuracy	Validation Accuracy	Test Accuracy	Precision	Recall	F1-Score
DenseNN	1	0.98	0.99	0.98	1.00	0.9899
CNN	1	0.97	0.99	0.99	0.99	0.9881

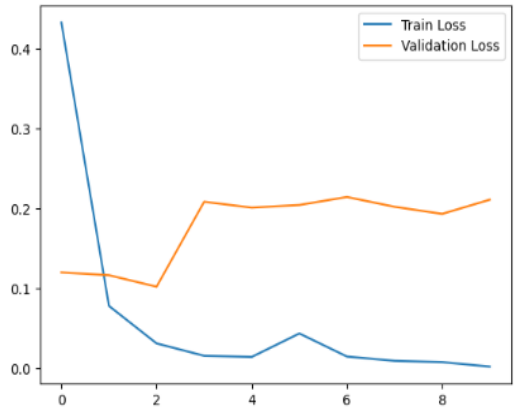


Figure 1: Train and Validation Loss of DNN

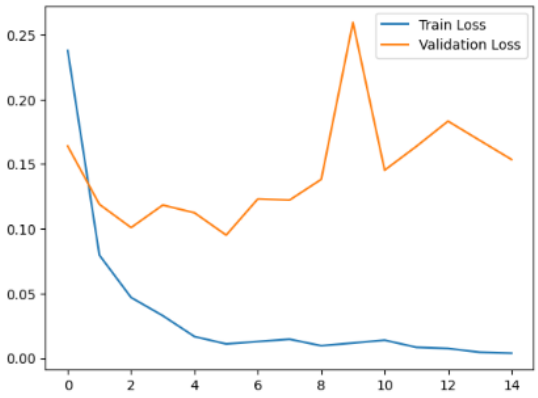
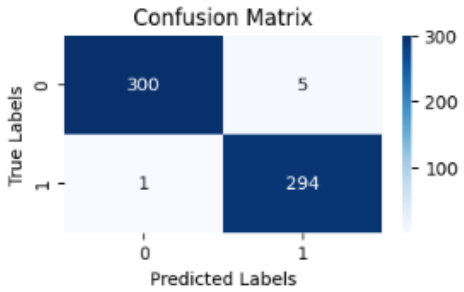


Figure 2: Train and Validation Loss of CNN

From the above we conclude that the **Dense Neural Network** is suitable for the problem of automatically distinguishing emails to spam or not spam classes.



This model misclassified only 6 emails out of 600, achieving the highest values on the evaluation metrics used.

```
{'Accuracy (%)': '99.00',
 'Precision (%)': '98.33',
 'Recall (%)': '99.66',
 'F1-Score (%)': '98.99'}
```

Generally, the use of undersampling technique for data balancing resulted in better performance of the models than the use of SMOTE.

5. Selected Model Architecture

Here is the summary of the chosen model:

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_35 (Dense)            (None, 32)                96032

 dropout_28 (Dropout)        (None, 32)                0

 dense_36 (Dense)            (None, 64)                2112

 dropout_29 (Dropout)        (None, 64)                0

 dense_37 (Dense)            (None, 128)               8320

 dropout_30 (Dropout)        (None, 128)               0

 dense_38 (Dense)            (None, 32)                4128

 dropout_31 (Dropout)        (None, 32)                0

 dense_39 (Dense)            (None, 1)                 33

=================================================================
Total params: 110,625
Trainable params: 110,625
Non-trainable params: 0
_________________________________________________________________
```

Layer Breakdown:

dense_35 (Dense): Fully connected layer with 32 neurons. Param #: 96,032 parameters: Params=(input features+1)×units= (3000+1)×32=96,032

dropout_28 (Dropout): Adds dropout regularization to the previous layer with no trainable parameters.

dense_36 (Dense): A dense layer with 64 units and 2,112 parameters.

dropout_29 (Dropout): Applies dropout to the previous dense layer.

Etc.

Dropout layers (e.g., dropout_28, dropout_29, etc.) are used to reduce overfitting by randomly deactivating a fraction of neurons during training.

All 110,625 parameters are trainable, meaning they will be updated during backpropagation.

The last layer produces a single output between 0 and 1 (sigmoid activation), representing the probability of belonging to a particular class.