

MODULE - 4 (PRACTICAL)

Dataset

SMSSpamCollection Dataset - <https://archive.ics.uci.edu/ml/datasets/sms+spam+collection>

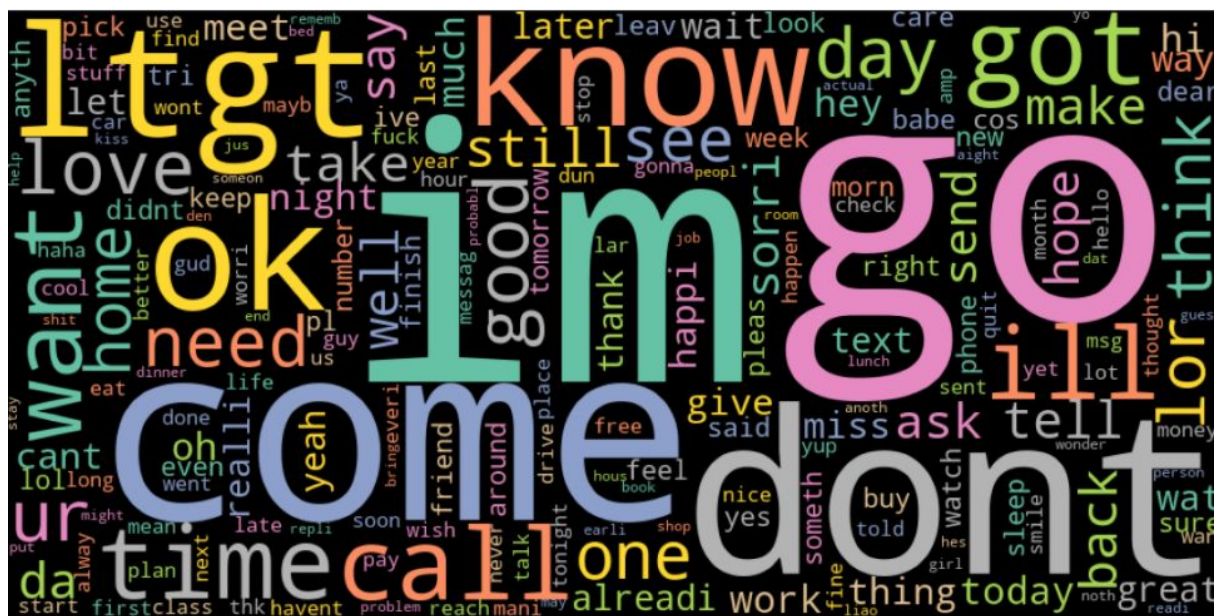
This dataset contains SMS messages divided into 2 categories 'ham' and 'spam'. There are 5574 rows in total.

Cleaning and EDA

The data contains 444 duplicate values mostly from the ham category. While cleaning the text, we remove all the punctuation, stopwords and apply stemming to the data. We do not remove numbers from the text data as it has been observed that the 'Spam' text contains a lot of numbers when compared to the ham data and can be used as a good differentiating factor. The dataset is highly imbalanced even after removing the duplicates, there are 4281 rows in the 'Ham' category and 543 rows in the 'Spam' category. Therefore to increase the data in the minority class, we use SMOTE, an oversampling technique to balance the data.

The number of unique words in the 'ham' are 44 and in 'spam' are 39. The mean length of each sentence in ham is 42.129 and the mean length in the 'spam' is 98.20994475138122 .

Word Count Visualization 'ham' text:

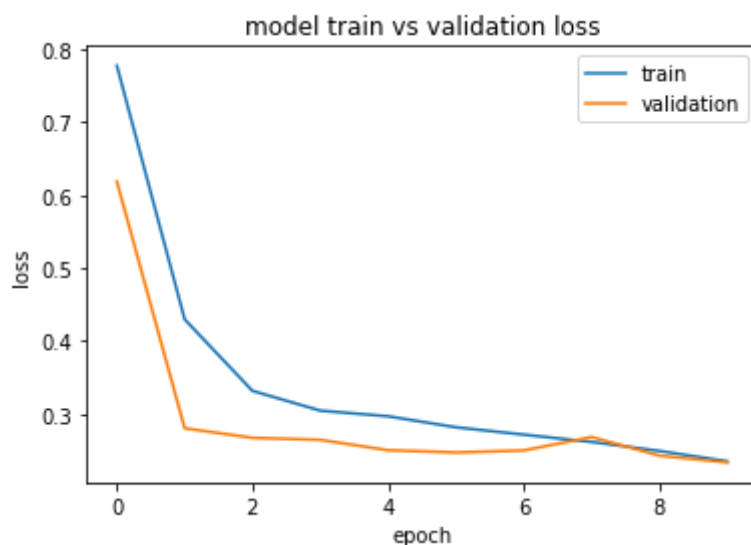


In 'ham' messages, we see that the most common words are 'im', 'come', 'go', 'dont' which signify normal text messages and can be easily differentiated between 'Spam' messages. The larger the size of the word, the more frequent it is in the category and more weightage it must be given for categorizing.

Word Cloud For Spam :

encoded input the same length. Therefore we pad zeros to some vectors and we take 25 as the maximum length. The labels must also be encoded from 'ham', 'spam' to class 1 and class 2. We use the Tokenizer() to encode the labels. Once encoded, we must balance the dataset as explained above. We use SMOTE to equalize the classes. After equalizing, we one-hot encode the labels i.e represent the labels in terms of 3 classes since our labels contain 1 and 2. Next, we build our model.

In our model, we add an Embedding layer where words are encoded as high dimensional vectors and closeness of vectors indicate words with similar meanings. We map each word to 8 dimension vectors and the number of words is 600. Next, we initialize the LSTM layer, we pass the dimension as the input. The last Dense layer we have 3 output dimensions with activation 'softmax'. The loss function is 'categorical cross entropy' and metrics accuracy. The optimizer function is 'adam'.



This graph shows the perfect fit which is not overfitting nor underfitting. The train accuracy is 91.1% and test accuracy is 90.14%.

Naive Bayes Algorithm:

The main assumption made by this algorithm is that it assumes all words are independent of each other and each feature makes an equal contribution. Therefore:

$$P(A,B) = P(A)P(B)$$

This is a fairly strong assumption which does not hold good in most real world applications. In this example Naive Bayes algorithm worked fairly well with 96.21%.