**Data Preprocessing**

In NLP, text preprocessing is the first step in the process of building a model.

Data preprocessing is an essential step in building a Machine Learning model and depending on how well the data has been preprocessed

It transforms text into a more digestible form so that machine learning algorithms can perform better.

**Lowercasing**

Lowercasing ALL your text data, although commonly overlooked, is one of the simplest and most effective form of text preprocessing. It is applicable to most text mining and NLP problems and can help in cases where your dataset is not very large and significantly helps with consistency of expected output.

We found that different variation in input capitalization (e.g. ‘Angry vs. ‘angry) gave us different types of output or no output at all. This was probably happening because the dataset had mixed-case occurrences of the word ‘Angry and there was insufficient evidence for the neural-network to effectively learn the weights for the less common version. This type of issue is bound to happen when your dataset is fairly small, and lowercasing is a great way to deal with sparsity issues.

**Removing Punctuations**:

An important NLP preprocessing step is punctuation marks removal, this marks - used to divide text into sentences, paragraphs and phrases - affects the results of any text processing approach, especially what depends on the occurrence frequencies of words and phrases, since the punctuation marks are used frequently in text.

# **Tokenization:**

Tokenization is a step which splits longer strings of text into smaller pieces, or tokens. Larger chunks of text can be tokenized into sentences, sentences can be tokenized into words, etc. Further processing is generally performed after a piece of text has been appropriately tokenized. Tokenization is also referred to as text segmentation or lexical analysis. Sometimes segmentation is used to refer to the breakdown of a large chunk of text into pieces larger than words (e.g. paragraphs or sentences), while tokenization is reserved for the breakdown process which results exclusively in words.

**Label Encoding**:

refers to converting the labels into numeric form so as to convert it into the machine-readable form. Machine learning algorithms can then decide in a better way on how those labels must be operated. It is an important pre-processing step for the structured dataset in supervised learning.

An attribute having output classes **Angry**, **Sad**, **Joy**. On Label Encoding this column, let **Angry** is replaced with 0 , **Sad** is replaced with 1 and **Joy** is replaced with 2.

[**Outliers**](https://machinelearningmastery.com/how-to-use-statistics-to-identify-outliers-in-data/)**:**

[Outliers](https://machinelearningmastery.com/how-to-use-statistics-to-identify-outliers-in-data/)  are observations in a dataset that don’t fit in some way.

Perhaps the most common or familiar type of outlier is the observations that are far from the rest of the observations or the center of mass of observations.

This is easy to understand when we have one or two variables and we can visualize the data as a histogram or scatter plot, although it becomes very challenging when we have many input variables defining a high-dimensional input feature space.

In our case, Data samples which are extremely long or short are removed, considering them as outliers

**Padding:**

All the neural networks require to have inputs that have the same shape and size. However, when we pre-process and use the texts as inputs for our model e.g. LSTM, not all the sentences have the same length. In other words, naturally, some of the sentences are longer or shorter. We need to have the inputs with the same size, this is where the padding is necessary.

**Data Splitting**

The dataset has been split into training (70%), and testing (30%) dataset

Model Architecture

A sequential model is created using keras. Recurrent Neural Network (RNN) is a deep learning algorithm that is specialized for sequential data. In a RNN the neural network gains information from the previous step in a loop. The output of one unit goes into the next one and the information is passed.

But RNNs are not good for training large datasets. During the training of RNN, the information goes in loop again and again which results in very large updates to neural network model weights which lead to the accumulation of error gradients during the update and the network becomes unstable. At an extreme, the values of weights can become so large as to overflow and result in NaN values. The explosion occurs through exponential growth by repeatedly multiplying gradients through the network layers that have values larger than 1 or vanishing occurs if the values are less than 1.

To overcome this problem Long Short-Term Memory is used. LSTM can capture long-range dependencies. It can have memory about previous inputs for extended time durations. There are 3 gates in an LSTM cell – Forget, Input and Output Gate.

• Forget Gate: Forget gate removes the information that is no longer useful in the cell state.

• Input Gate: Additional useful information to the cell state is added by input gate.

• Output Gate: Additional useful information to the cell state is added by output gate.

Memory manipulations in LSTM are done using these gates. Long short-term memory (LSTM) utilizes gates to control the gradient propagation in the recurrent network’s memory. This gating mechanism of LSTM has allowed the network to learn the conditions for when to forget, ignore, or keep information in the memory cell.

The fist layer of the model is Embedding layer. The embedding layer represents the unique words of the dataset in vector form and is mainly used to compress the input feature space into a smaller one . This compression is usually done to make the computation of the sentences faster. This layer may map the many similar words into the same vector position using their semantic relationship within the context. For example, "No", "Not", "Never" can be mapped together

Its input dimension is 5000 (most commonly used words in the dataset) and output dimension is 64 which will be the size of the output vectors from this layer for each word. The input length of sequence is going to be the maximum length which is 150.

A SpatialDropout Layer is used to prevent the model from Overfitting.

LSTM preserves information from inputs that has already passed through it using the hidden state. Unidirectional LSTM only preserves information of the past because the only inputs it has seen are from the past. Bidirectional LSTM will run the inputs in two ways, one from past to future and one from future to past and what differs this approach from unidirectional is that in the LSTM that runs backwards information from the future is preserved and using the two hidden states combined it is able in any point in time to preserve information from both past and future.

The second is a LSTM layer followed by dropout layer. Its 64 cells (each cell has its own inputs, outputs and memory) are used and return sequence is set to true which means that every time there will be an output which will be fed into another LSTM layer it is sent as a sequence rather than a single value of each input.

The final layer will be a Dense layer with 3 units for the three classes present and the activation is set to softmax which returns a probability distribution over the target classes.

The model is compiled with loss set to ‘categorical\_crossentropy’ as it is used for multi class classification problems as the classes are not one-hot encoded (for binary classes). The optimizer used is ‘adam’ as it is really efficient for working with large datasets. The training metrics used is accuracy which calculates how often the predictions are equal to the actual labels. The model summary is generated.

Training the Model

The model is then trained for 3 epochs. The number of epochs is a hyperparameter of gradient descent that controls the number of complete passes through the training dataset. The model training to be completed takes only about 21 minutes/epoch with the Jupyter notebook service hosted on Google Colab which is using a GPU for accelerated computation.

Evaluating the Model

The model achieves

Accuracy: 0.9861490482847618

F1 Score: 0.9869397463286533

AUC ROC: 0.9861601844657049

Log Loss: 0.42481298574916665

All the predictions are also evaluated against all the ground truths using the test set. A confusion matrix is generated for the test labels in the against the actual classes.