# Sentimental Analysis using BERT Embeddings and LSTM

A Project Report in partial fulfillment of the degree

**Bachelor of Technology**

in

**Computer Science & Engineering/ Electrical & Electronics Engineering**

**By**


| | |
|---|---|
| 19K41A05E4 | S. Sriharsha |
| 19K41A05E7 | T. Ganesh |
| 20K45A0215 | Prabhu Kiran Konda |


**Under the Guidance of**
**D. Ramesh**
**Assistant Professor of CSE**

**Submitted to**


**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**
**S R ENGINEERING COLLEGE(A), ANANTHASAGAR, WARANGAL**
**(Affiliated to JNTUH, Accredited by NBA)**


**Nov-2022**


# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

## CERTIFICATE

This is to certify that the Project Report entitled "Sentimental Analysis using BERT Embeddings and LSTM" is a record of bona fide work carried out by the student(s) S. Sriharsha, T. Ganesh, Prabhu Kiran Konda bearing Roll No(s) 19K41A05E4, 19K41A05E7, 20K45A0215 respectively during the academic year 2021-2022 in partial fulfillment of the award of the degree of **Bachelor of Technology** in **Computer Science Engineering/Electrical & Electronics Engineering** by the Jawaharlal Nehru Technological University, Hyderabad.

**Supervisor** **Head of the Department**

**External Examiner**

# ABSTRACT

Sentiment analysis is a natural language processing (NLP)technique used to determine whether data is positive, negative or neutral. Sentiment analysis is often performed on textual data to help businesses monitor brand and product sentiment in customer feedback, and understand customer needs.

Emotion detection sentiment analysis allows you to go beyond polarity to detect emotions, like happiness, frustration, anger, and sadness.

This project focuses on Classifying emotions based on the given text inputs. It can classify 6 variety of emotions like joy, anger, love, sadness, surprise and fear.

In This Project, we used BERT Embeddings along with Bi-LSTM layers to train the classifier model. Our model is trained on a data with 20000 text samples whereas train data has 16000 samples, test and validation data share the rest of the samples.

Our classifier model scored an accuracy of 89.3% on train data and 75% accuracy on Validation data.

## Table of Contents

# 1. Introduction

Text can be an extremely rich source of information, but extracting insights from it can be hard and time-consuming, due to its unstructured nature. But, thanks to advances in natural language processing and machine learning, which both fall under the vast umbrella of artificial intelligence, sorting text data is getting easier.

## 1.1. **What is Text Classification?**

Text classification is a machine learning technique that assigns a set of predefined categories to open-ended text. Text classifiers can be used to organize, structure, and categorize pretty much any kind of text – from documents, medical studies and files, and all over the web. For example, new articles can be organized by topics; support tickets can be organized by urgency; chat conversations can be organized by language; brand mentions can be organized by sentiment; and so on.

Text classification is one of the fundamental tasks in natural language processing with broad applications such as sentiment analysis, topic labelling, spam detection, and intent detection.

You can perform text classification in two ways:

1. Manual
2. Automatic.

Manual text classification involves a human annotator, who interprets the content of text and categorizes it accordingly. This method can deliver good results but it's time-consuming and expensive.

Automatic text classification applies machine learning, Natural Language Processing (NLP), and other AI-guided techniques to automatically classify text in a faster, more cost-effective, and more accurate manner.

There are many approaches to automatic text classification, but they all fall under three types of systems:

1. Rule-based systems
2. Machine learning-based systems
3. Hybrid systems

## 1.2. **Machine Learning Based Systems**

Instead of relying on manually crafted rules, machine learning text classification learns to make classifications based on past observations. By using pre-labelled examples as training data, machine learning algorithms can learn the different associations between pieces of text, and that a particular output (i.e., tags) is expected for a particular input (i.e., text). A "tag" is the pre-determined classification or category that any given text could fall into.

The first step towards training a machine learning NLP classifier is feature extraction: a method is used to transform each text into a numerical representation in the form of a vector. One of the most frequently used approaches is bag of words, where a vector represents the frequency of a word in a predefined dictionary of words. For example, if we have defined our dictionary to have the following words
{*This, is, the, not, awesome, bad, basketball*}, and we wanted to vectorize the text *"This is awesome,"* we would have the following vector representation of that text: (1, 1, 0, 0, 1, 0, 0).

Then, the machine learning algorithm is fed with training data that consists of pairs of feature sets (vectors for each text example) and tags (e.g. *sports*, *politics*) to produce a classification model

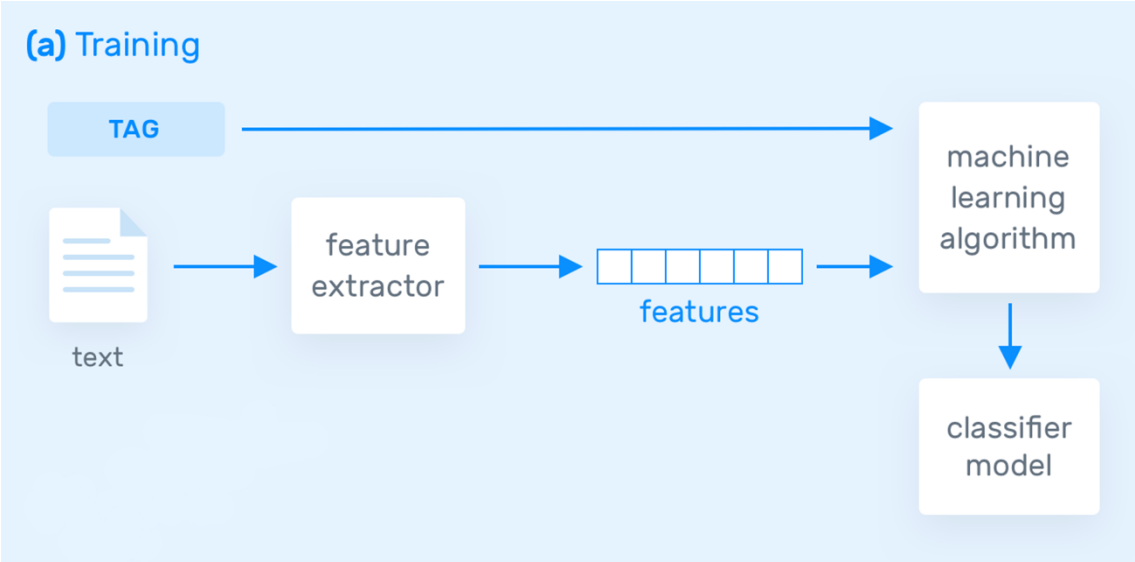



*Fig 1: Training Process of a Text Classification Model*

Once it's trained with enough training samples, the machine learning model can begin to make accurate predictions. The same feature extractor is used to transform unseen text to feature sets, which can be fed into the classification model to get predictions on tags (e.g., *sports*, *politics*):

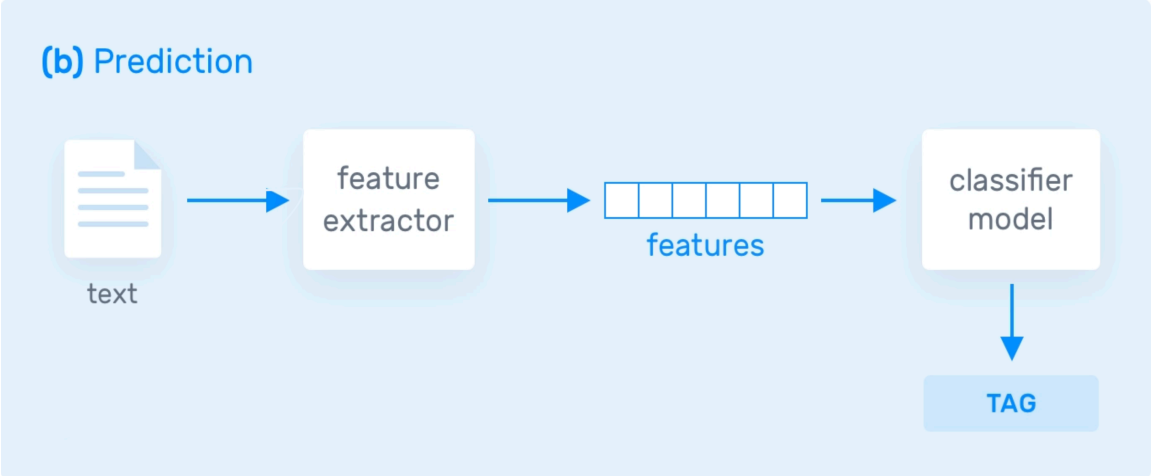



*Fig 2: Prediction Process of a Classification Model*

Both approaches have their pros and cons, concretely, linguistic approaches have great precision but their recall is low as the context where the features are useful is not as big as the 2 Santiago Gonz´alez-Carvajal, Eduardo C. Garrido-Merch 'an one processed by machine learning algorithms. Although, the precision of classical NLP systems was, until recently, generally better as the one delivered by machine learning. Nevertheless, recently, thanks to the rise of computation, machine learning text classification dominates in scenarios where huge sizes of texts are processed. Generally, linguistic approaches consist in applying a series of rules, which are designed by linguistic experts . An example of linguistic approach can be found at . The advantage of these type of approaches over ML based approaches is that they do not need large amounts of data. Regarding ML based approaches, they usually have a statistical base . We can find many examples of these type of approaches: BERT, Transformers, etc. Another issue with traditional NLP approaches is multilingualism . We can design rules for a given language, but sentence structure, and even the alphabet, may change from one language to another, resulting in the need to design new rules. Some approaches such as the Universal Networking Language (UNL) standard try to circumvent this issue, but the multilingual resource is hard to build and requires experts on the platform. Another problem with UNL approaches and related ones, would be that, given a specific language, the different forms of expression, i.e. the way we write in, for example, Twitter, is very different from the way we write a more formal document, such as a research paper.

## 1.3. **Bidirectional Encoder Representations for Transformers (BERT) Model**

Bidirectional Encoder Representations from Transformers) is a recent paper published by researchers at Google AI Language. It has caused a stir in the Machine Learning community by presenting state-of-the-art results in a wide variety of NLP tasks, including Question Answering (SQuADv1.1), Natural Language Inference (MNLI), and others. BERT's key technical innovation is applying the bidirectional training of Transformer, a popular attention model, to language modelling. This is in contrast to previous efforts which looked at a text sequence either from left to right or combined left-to-right and right-to-left training. The paper's results show that a language model which is bidirectionally trained can have a deeper sense of language context and flow than single-direction language models. In the paper, the researchers detail a novel technique named Masked LM (MLM) which allows bidirectional training in models in which it was previously impossible.

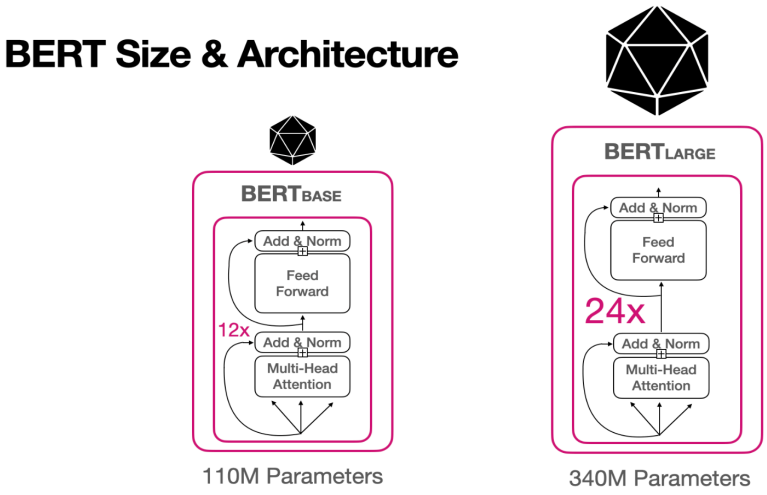
BERT makes use of Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text. In its vanilla form, Transformer includes two separate mechanisms — an encoder that reads the text input and a decoder that produces a prediction for the task. Since BERT's goal is to generate a language model, only the encoder mechanism is necessary. The detailed workings of Transformer are described in a paper by Google.

As opposed to directional models, which read the text input sequentially (left-to-right or right-to-left), the Transformer encoder reads the entire sequence of words at once. Therefore it is considered bidirectional, though it would be more accurate to say that it's non-directional.

This characteristic allows the model to learn the context of a word based on all of its surroundings (left and right of the word).

When training language models, there is a challenge of defining a prediction goal. Many models predict the next word in a sequence (e.g. "The child came home from ___"), a directional approach which inherently limits context learning. To overcome this challenge, BERT uses two training strategies:

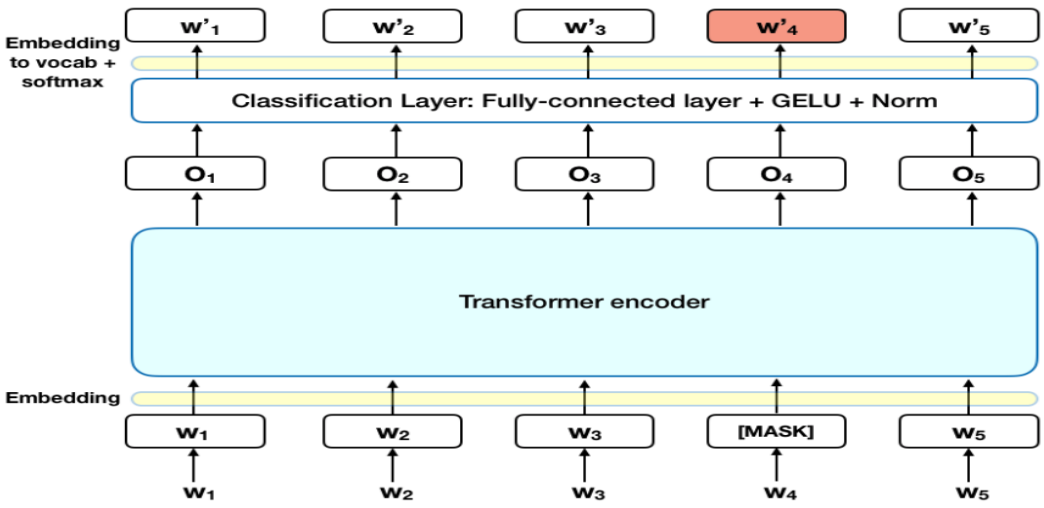
1. Masked LM (MLM)
2. Next Sentence Prediction (NSP)




**1. Masked LM (MLM):**

Before feeding word sequences into BERT, 15% of the words in each sequence are replaced with a [MASK] token. The model then attempts to predict the original value of the masked words, based on the context provided by the other, non-masked, words in the sequence. In technical terms, the prediction of the output words requires:

1. Adding a classification layer on top of the encoder output.
2. Multiplying the output vectors by the embedding matrix, transforming them into the vocabulary dimension.
3. Calculating the probability of each word in the vocabulary with SoftMax.

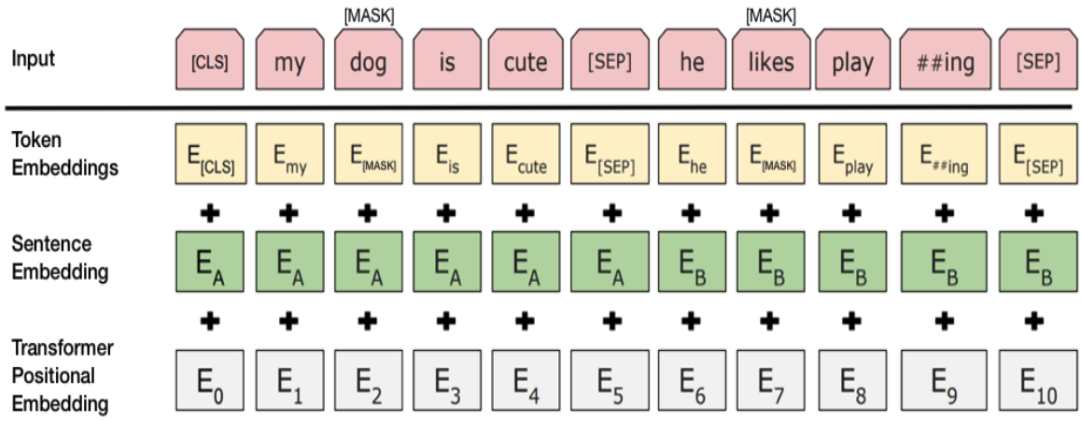


The BERT loss function takes into consideration only the prediction of the masked values and ignores the prediction of the non-masked words. As a consequence, the model converges slower than directional models, a characteristic which is offset by its increased context awareness

**2. Next Sentence Prediction (NSP):**

In the BERT training process, the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original document. During training, 50% of the inputs are a pair in which the second sentence is the subsequent sentence in the original document, while in the other 50% a random sentence from the corpus is chosen as the second sentence. The assumption is that the random sentence will be disconnected from the first sentence. To help the model distinguish between the two sentences in training, the input is processed in the following way before entering the model:

1. A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
2. A sentence embedding indicating Sentence A or Sentence B is added to each token. Sentence embeddings are similar in concept to token embeddings with a vocabulary of 2.
3. A positional embedding is added to each token to indicate its position in the sequence. The concept and implementation of positional embedding are presented in the Transformer paper.




To predict if the second sentence is indeed connected to the first, the following steps are performed:

1. The entire input sequence goes through the Transformer model.
2. The output of the [CLS] token is transformed into a 2×1 shaped vector, using a simple classification layer (learned matrices of weights and biases).
3. Calculating the probability of IsNextSequence with SoftMax.

When training the BERT model, Masked LM and Next Sentence Prediction are trained together, with the goal of minimizing the combined loss function of the two strategies

# 2. Literature Survey

## [1] Political Text Classification using Word Embeddings and LSTM

**Accuracy: 87.5% | Model: LSTM**

In this work, we apply word embeddings and neural networks with Long Short-Term Memory (LSTM) to text classification problems, where the classification criteria are decided by the context of the application. We examine two applications in particular. The first is that of Actionability, where we build models to classify social media messages from customers of service providers as Actionable or Non-Actionable. We build models for over 30 different languages for actionability, and most of the models achieve accuracy around 85%, with some reaching over 90% accuracy. We also show that using LSTM neural networks with word embeddings vastly outperform traditional techniques. Second, we explore classification of messages with respect to political leaning, where social media messages are classified as Democratic or Republican. The model is able to classify messages with a high accuracy of 87.57%. As part of our experiments, we vary different hyperparameters of the neural networks, and report the effect of such variation on the accuracy. These actionability models have been deployed to production and help company agents provide customer support by prioritizing which messages to respond to. The model for political leaning has been opened and made available for wider use.

## [2] Bidirectional LSTM with attention mechanism and convolutional layer for text classification

**Accuracy: 83.2% | Model: Bi-LSTM + CNN**

Neural network models have been widely used in the field of natural language processing (NLP). Recurrent neural networks (RNNs), which have the ability to process sequences of arbitrary length, are common methods for sequence modelling tasks. Long short-term memory (LSTM) is one kind of RNNs and has achieved remarkable performance in text classification. However, due to the high dimensionality and sparsity of text data, and to the complex semantics of the natural language, text classification presents difficult challenges. In order to solve the above problems, a novel and unified architecture which contains a bidirectional LSTM (BiLSTM), attention mechanism and the convolutional layer is proposed in this paper. The proposed architecture is called attention-based bidirectional long short-term memory with convolution layer (AC-BiLSTM). In AC-BiLSTM, the convolutional layer extracts the higher-level phrase representations from the word embedding vectors and BiLSTM is used to access both the preceding and succeeding context representations. Attention mechanism is employed to give different focus to the information outputted from the hidden layers of BiLSTM. Finally, the SoftMax classifier is used to classify the processed context information. AC-BiLSTM is able to capture both the local feature of phrases as well as global sentence semantics. Experimental verifications are conducted on six sentiment classification datasets and a question classification dataset, including detailed analysis for AC-BiLSTM. The results clearly show that AC-BiLSTM outperforms other state-of-the-art text classification methods in terms of the classification accuracy.

## [3] Multi-class Text Classification using BERT-based Active Learning.

**Accuracy: 83% | Model: BERT**

Text Classification finds interesting applications in the pickup and delivery services industry where customers require one or more items to be picked up from a location and delivered to a certain destination. Classifying these customer transactions into multiple categories helps understand the market needs for different customer segments. Each transaction is accompanied by a text description provided by the customer to describe the products being picked up and delivered which can be used to classify the transaction. BERT-based models have proven to perform well in Natural Language Understanding. However, the product descriptions provided by the customers tend to be short, incoherent and code-mixed (Hindi-English) text which demands fine-tuning of such models with manually labelled data to achieve high accuracy. Collecting this labelled data can prove to be expensive. In this paper, we explore Active Learning strategies to label transaction descriptions cost effectively while using BERT to train a transaction classification model. On TREC-6, AG's News Corpus and an internal dataset, we benchmark the performance of BERT across different Active Learning strategies in Multi-Class Text Classification.

## [4] Fine-grained Sentiment Classification using BERT

**Accuracy: 93.2% | Model: BERT**

Sentiment classification is an important process in understanding people's perception towards a product, service, or topic. Many natural language processing models have been proposed to solve the sentiment classification problem. However, most of them have focused on binary sentiment classification. In this paper, we use a promising deep learning model called BERT to solve the fine-grained sentiment classification task. Experiments show that our model outperforms other popular models for this task without sophisticated architecture. We also demonstrate the effectiveness of transfer learning in natural language processing in the process.

## [5] News Text Classification Based on Improved Bi-LSTM-CNN

**Accuracy: 96.4% | Model: Bi-LSTM + CNN**

The traditional text classification methods are based on machine learning. It requires a large amount of artificially labelled training data as well as human participation. However, it is common that ignoring the contextual information and the word order information in such a way, and often exist some problems such as data sparseness and latitudinal explosion. With the development of deep learning, many researchers have also been using deep learning in text classification. This paper investigates the application issue of NLP in text classification by using the Bi-LSTM-CNN method. For the purpose of improving the accuracy of text classification, a kind of comprehensive expression is employed to accurately express semantics. The experiment shows that the model in this paper has great advantages in the classification of news texts.

## [6] Hate Speech Detection Using Static BERT Embedding

**Accuracy: 93.2% | Model: BERT**

With increasing popularity of social media platforms hate speech is emerging as a major concern, where it expresses abusive speech that targets specific group characteristics, such as gender, religion or ethnicity to spread violence. Earlier people use to verbally deliver hate speeches

but now with the expansion of technology, some people are deliberately using social media platforms to spread hate by posting, sharing, commenting, etc. Whether it is Christchurch mosque shootings or hate crimes against Asians in west, it has been observed that the convicts are very much influenced from hate text present online. Even though AI systems are in place to flag such text but one of the key challenges is to reduce the false positive rate (marking non hate as hate), so that these systems can detect hate speech without undermining the freedom of expression. In this paper, we use ETHOS hate speech detection dataset and analyze the performance of hate speech detection classifier by replacing or integrating the word embeddings (fastText (FT), GloVe (GV) or FT + GV) with static BERT embeddings (BE). With the extensive experimental trails it is observed that the neural network performed better with static BE compared to using FT, GV or FT + GV as word embeddings. In comparison to fine-tuned BERT, one metric that significantly improved is specificity.

## [7] Text Classification Improved by Integrating Bidirectional LSTM with Two-dimensional Max Pooling

**Accuracy: 96% | Model: Bi-LSTM + 2D Max Pooling**

Recurrent Neural Network (RNN) is one of the most popular architectures used in Natural Language Processing (NLP) tasks because its recurrent structure is very suitable to process variable-length text. RNN can utilize distributed representations of words by first converting the tokens comprising each text into vectors, which form a matrix. And this matrix includes two dimensions: the time-step dimension and the feature vector dimension. Then most existing models usually utilize one-dimensional (1D) max pooling operation or attention-based operation only on the time-step dimension to obtain a fixed-length vector. However, the features on the feature vector dimension are not mutually independent, and simply applying 1D pooling operation over the time-step dimension independently may destroy the structure of the feature representation. On the other hand, applying two-dimensional (2D) pooling operation over the two dimensions may sample more meaningful features for sequence modelling tasks. To integrate the features on both dimensions of the matrix, this paper explores applying 2D max pooling operation to obtain a fixed-length representation of the text. This paper also utilizes 2D convolution to sample more meaningful information of the matrix. Experiments are conducted on six text classification tasks, including sentiment analysis, question classification, subjectivity classification and newsgroup classification. Compared with the state-of-the-art models, the proposed models achieve excellent performance on 4 out of 6 tasks. Specifically, one of the proposed models achieves highest accuracy on Stanford Sentiment Treebank binary classification and fine-grained classification tasks.

## [8] Research on Patent Text Classification Based on Word2Vec and LSTM

**Accuracy: 93% | Model: LSTM + Word2Vec**

In order to efficiently classify patent texts in the security field, a patent text classification model based on Word2Vec and Long-short term memory (LSTM) was established. Combined with the features of the patent text, first of all, in the text pre-processing process, words frequently appearing in patent documents such as “the invention”, “involvement”, and “utility model” were added to the stop word list to save storage space and improve efficiency; Secondly, the pre-trained word2vec model was introduced to solve the dimensional disaster caused by the traditional methods. Finally, by training the LSTM classification model, text features were extracted and patent

text classification in the security field was performed. 50,000 patent documents were divided into the training set and the test set according to the ratio of 4:1, and the accuracy and ROC curve evaluation model were used to analyse and evaluate the classification results. The results showed that the classification accuracy rate of this method is 93.48%. At the same time, the LSTM classification model, K Nearest Neighbour (KNN) classification model, Convolutional Neural Network (CNN) classification model, and models based on CNN and Word2Vec were further compared. The experimental results showed that this method can better classify the patent texts in the security field, laying the foundation for further research and effective use of patents.

## [9] Offensive Text in Social Media - A Text Classification Approach using LSTM-BOOST

**Accuracy: 92.6% | Model: LSTM**

Recently, offensive content has become increasingly popular for harassing and criticizing people on numerous social media platforms. This paper proposes an offensive text classification algorithm named LSTM-BOOST employing Long Short-Term Memory(LSTM) model with ensemble learning to recognize offensive Bengali texts in various social media platforms. The proposed LSTM-BOOST model uses the modified AdaBoost algorithm employing principal component analysis(PCA) along with LSTM networks. In the LSTM-Boost model, the dataset is divided into three categories, and PCA and LSTM networks are applied to each part of the dataset to obtain the most significant variance and reduce the weighted error of the weak hypothesis of the model. Furthermore, different classifiers are used for baseline experiment and the model is evaluated on various word embedding vector methods. Our investigation found that the LSTM-BOOST algorithms outperform most of the baseline architecture, leading F1-score of 92.61% on the Bengali offensive text from Social Platforms(BHSSP) dataset.

## [10] LSTM-CNN Deep Learning Model for French Online Product Reviews Classification

**Accuracy: 93.7% | Model: LSTM**

Sentiment analysis (SA) is one of the most popular areas for analysing and discovering insights from text data from various sources, including Facebook, Twitter, and Amazon. It plays a pivotal role in helping enterprises actively improve their business strategies and better understand customers' feedback on products. In this work, the dataset has been taken from Amazon, which contains French reviews. After pre-processing and extracting the features using contextualized word embedding for French-language including ELMO, ULMFiT, and CamemBERT, we applied deep learning algorithms including CNN, LSTM, and combined CNN and LSTM to classify reviews as positive or negative. The results show that the combined model (LSTM+CNN) using CamemBERT achieved the best performance to classify the French reviews with an accuracy of 93.7%.

## 3. Dataset Insights

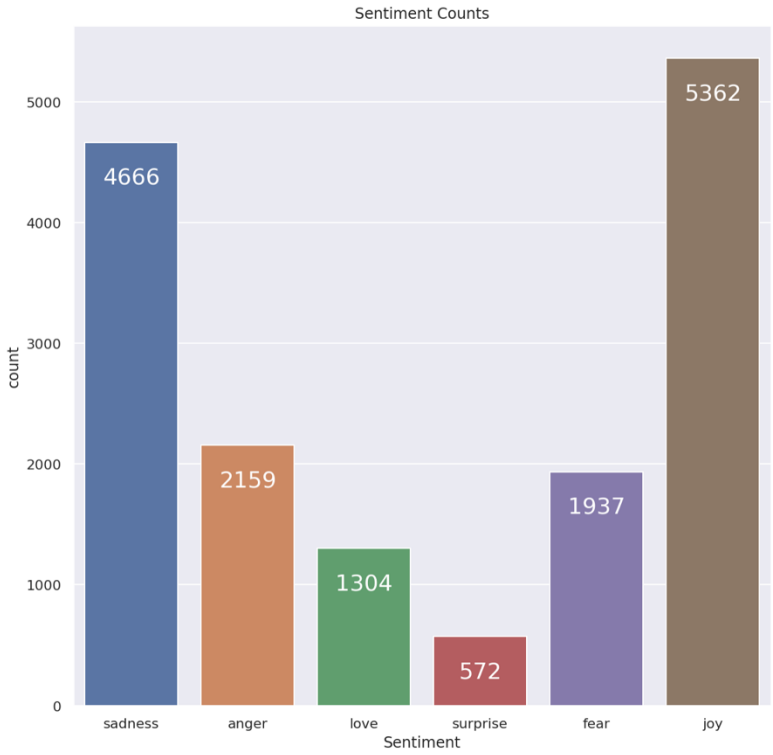
Dataset contains mainly two columns.

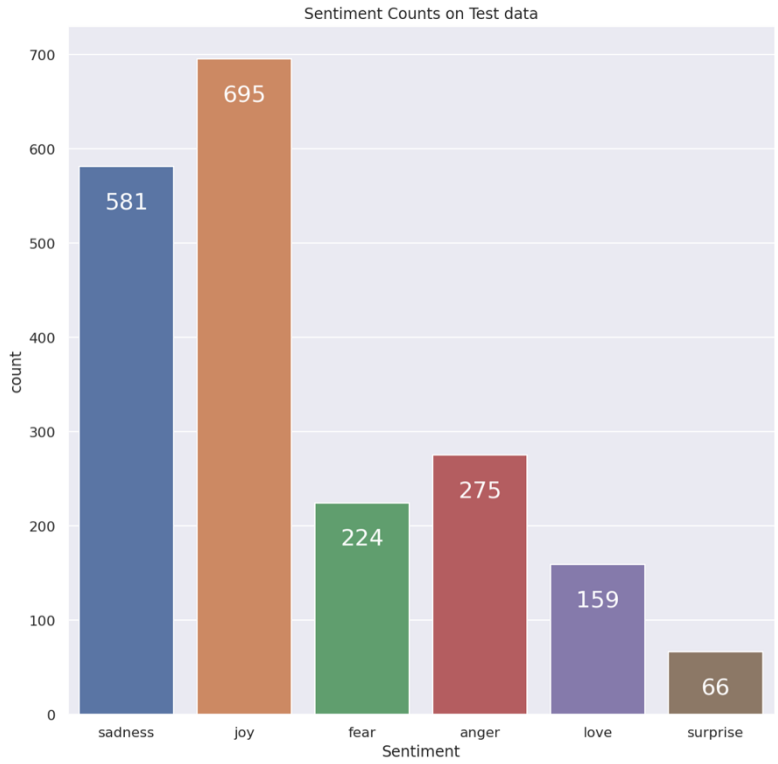
1. Input – text column
2. Sentiment or Emotion Column.

Furthermore, dataset is divided into 3 sets.

1. Train
2. Test
3. Validation

Train set has 16000 samples while Test and Validation has 2000 samples in each.




Emotion column is used as the output and text column is used as input column.

There are 6 different types of emotions available in the dataset, they are anger, sadness, joy, surprise, love and fear.

As Emotion comes under category, each emotion is label encoded into unique integer values. The values are

| Emotion | Label |
|---|---|
| **Joy** | 0 |
| **Anger** | 1 |
| **Love** | 2 |
| **Sadness** | 3 |
| **Fear** | 4 |
| **Surprise** | 5 |

```
Shape of Train Dataset: (16000, 2)
```

| | Input | Sentiment |
|---|---|---|
| 0 | i didnt feel humiliated | sadness |
| 1 | i can go from feeling so hopeless to so damned hopeful just from being around someone who cares and is awake | sadness |
| 2 | im grabbing a minute to post i feel greedy wrong | anger |
| 3 | i am ever feeling nostalgic about the fireplace i will know that it is still on the property | love |
| 4 | i am feeling grouchy | anger |
| ... | ... | ... |
| 15995 | i just had a very brief time in the beanbag and i said to anna that i feel like i have been beaten up | sadness |
| 15996 | i am now turning and i feel pathetic that i am still waiting tables and subbing with a teaching degree | sadness |
| 15997 | i feel strong and good overall | joy |
| 15998 | i feel like this was such a rude comment and im glad that t | anger |
| 15999 | i know a lot but i feel so stupid because i can not portray it | sadness |

### 3.1 Data Preprocessing

For any NLP tasks, preparing the data is the very first step which we need to do and that includes data preprocessing. Data preprocessing includes removing any unnecessary information from the dataset, null values and any information that might not be very useful to the Machine Learning model.

For NLP related data, we usually remove the unnecessary texts from the data and changing its form. For example, if a text sample has a word "*you're*" converting it to "*you are*" will make it more meaning full and easy to extract features.

Also, removing emails, special characters and stop words, etc. will make the data more efficient and model can be trained accurately.

## 4. Model

### 4.1. Word Embedding

Embedding is a process of converting the text into numerical vectors. Embeddings translate large sparse vectors into a lower-dimensional space that preserves semantic relationships.

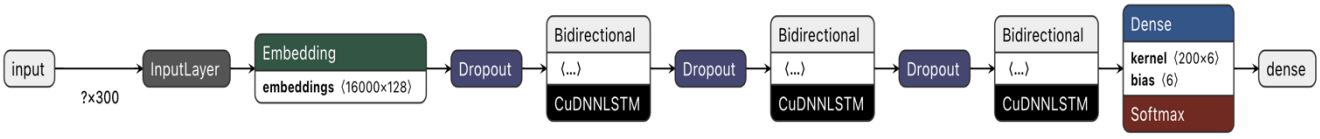
Word embeddings is a technique where individual words of a domain or language are represented as real-valued vectors in a lower dimensional space. ***TF-IDF***, **Word2Vec**, ***FastText*** are frequently used Word Embedding methods. In this project, we're using BERT embeddings rather than using Word2Vev Embeddings. BERT Embedding process is explained in the Introduction section.

```
1  embeddings[0]
```

✓ 0.3s

```
Output exceeds the size limit. Open the full output data in a text editor
<tf.Tensor: shape=(128,), dtype=float32, numpy=
array([ 1.94015086e-01,  1.17504276e-01,  4.75768633e-02,  3.03060971e-02,
        1.49399057e-01, -1.74436439e-02,  6.10478548e-03, -1.63987681e-01,
       -2.47460410e-01, -2.53385082e-02,  3.17302011e-02, -1.66262209e-01,
        7.16791349e-03, -1.66237831e-01,  4.67780530e-02,  7.87274763e-02,
        8.54017287e-02, -4.04364690e-02, -6.84206635e-02, -1.77727062e-02,
        3.14794667e-02, -9.54064652e-02, -5.49799651e-02, -1.46893412e-01,
       -9.19090584e-02,  2.05793213e-02,  2.02135354e-01,  1.74860077e-04,
       -1.37475962e-02,  7.42460601e-03,  1.13660805e-01,  1.14404280e-02,
        8.34165886e-02, -1.67289209e-02,  5.22099845e-02,  5.51486462e-02,
       -1.06314912e-01, -1.72274443e-03,  1.19523987e-01,  1.60723910e-01,
       -1.40672952e-01, -9.15073305e-02, -4.68659401e-03, -3.38490218e-01,
        8.98956433e-02,  2.52285469e-02,  1.54300570e-01,  1.84254676e-01,
       -2.70993225e-02,  3.82548273e-02,  1.35485153e-03,  9.36541185e-02,
       -1.13191336e-01,  6.53649643e-02, -1.38775051e-01, -1.89261176e-02,
       -9.81955696e-03, -6.37699012e-03, -9.83554050e-02,  3.68331149e-02,
       -3.05639029e-01,  1.06813483e-01,  4.59266677e-02,  2.47224569e-02,
        5.89316338e-02, -4.65162247e-02,  3.66874561e-02,  1.01371586e-01,
        9.25871059e-02, -8.94232169e-02, -1.88781515e-01, -4.00711372e-02,
       -6.56137592e-04, -9.33553949e-02, -9.54112504e-03,  6.69819936e-02,
        4.50966656e-02, -1.76144898e-01,  2.18259841e-02,  9.43342522e-02,
        1.56322077e-01,  6.65141717e-02,  9.52373445e-03, -9.97099429e-02,
       -6.44072294e-02,  2.17686351e-02, -6.62123635e-02,  2.73634046e-02,
        2.05104634e-01,  8.44343081e-02,  5.57318181e-02,  2.79472135e-02,
        3.77110280e-02, -7.00062215e-02, -1.54579524e-03,  1.05243817e-01,
...
       -1.27652436e-01,  2.30749720e-03, -8.39290395e-03, -6.07798137e-02,
        1.43255964e-02, -1.24460056e-01, -9.06638727e-02, -4.93461937e-02,
        1.20305650e-01, -5.15336469e-02,  4.75794040e-02,  1.14922069e-01,
       -8.60543773e-02, -3.00288443e-02, -8.51421505e-02,  1.17604785e-01],
      dtype=float32)>
```

## 4.2. Model Architecture




Our model is a classifier model which is based on Bidirectional LSTM. We have 3 Bi-LSTM layers along with dropout layers. On top of the model, there's an embedding layer which takes BERT Embeddings as its weights.

## Model Summary

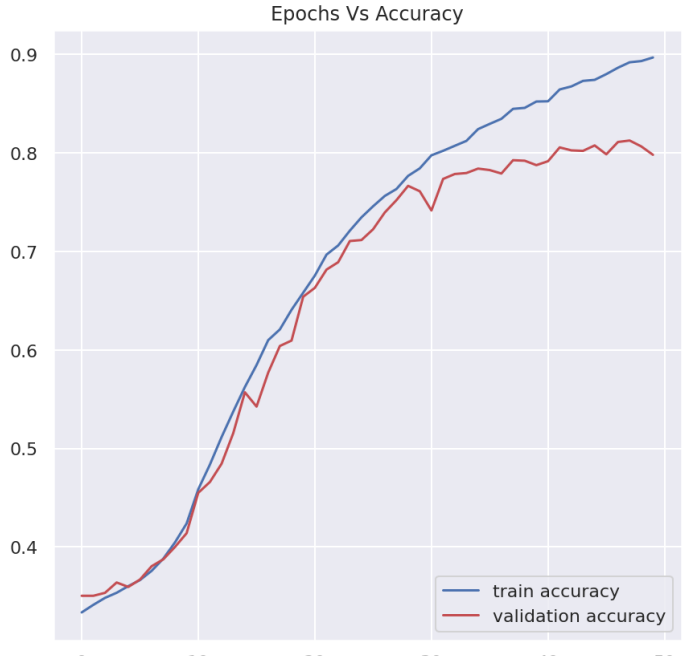
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 300, 128)          2048000

 dropout (Dropout)           (None, 300, 128)          0

 bidirectional (Bidirectiona  (None, 300, 200)         184000
 l)

 dropout_1 (Dropout)         (None, 300, 200)          0

 bidirectional_1 (Bidirectio  (None, 300, 400)         643200
 nal)

 dropout_2 (Dropout)         (None, 300, 400)          0

 bidirectional_2 (Bidirectio  (None, 200)              401600
 nal)

 dense (Dense)               (None, 6)                 1206

=================================================================
Total params: 3,278,006
Trainable params: 1,230,006
Non-trainable params: 2,048,000
_________________________________________________________________
```

## Model Training Progress

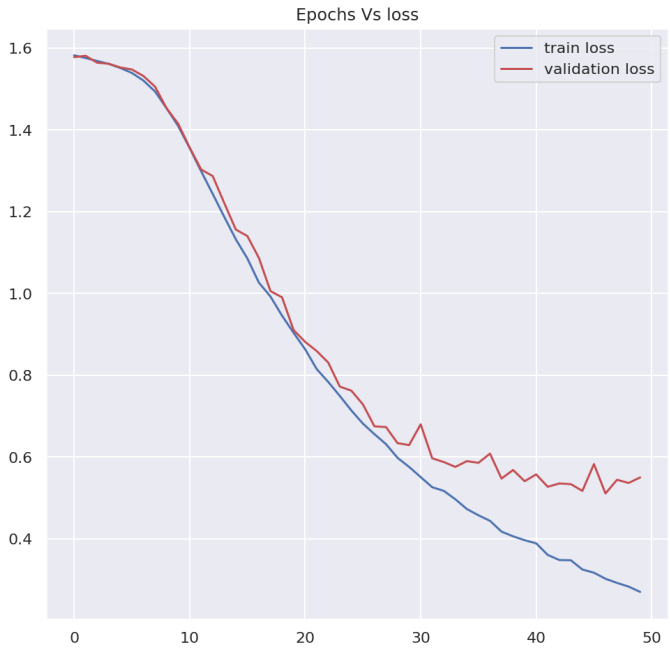
```
Output exceeds the size limit. Open the full output data in a text editor
Epoch 1/50
125/125 [==============================] - ETA: 0s - loss: 1.5820 - accuracy: 0.3336
Epoch 1: val_accuracy improved from -inf to 0.35050, saving model to ./model.h5
125/125 [==============================] - 36s 204ms/step - loss: 1.5820 - accuracy: 0.3336 - val_loss: 1.5773 - val_accuracy: 0.3505
Epoch 2/50
125/125 [==============================] - ETA: 0s - loss: 1.5752 - accuracy: 0.3413
Epoch 2: val_accuracy did not improve from 0.35050
125/125 [==============================] - 24s 189ms/step - loss: 1.5752 - accuracy: 0.3413 - val_loss: 1.5807 - val_accuracy: 0.3505
Epoch 3/50
125/125 [==============================] - ETA: 0s - loss: 1.5680 - accuracy: 0.3484
Epoch 3: val_accuracy improved from 0.35050 to 0.35350, saving model to ./model.h5
125/125 [==============================] - 23s 187ms/step - loss: 1.5680 - accuracy: 0.3484 - val_loss: 1.5637 - val_accuracy: 0.3535
Epoch 4/50
125/125 [==============================] - ETA: 0s - loss: 1.5610 - accuracy: 0.3536
Epoch 4: val_accuracy improved from 0.35350 to 0.36400, saving model to ./model.h5
125/125 [==============================] - 24s 190ms/step - loss: 1.5610 - accuracy: 0.3536 - val_loss: 1.5613 - val_accuracy: 0.3640
Epoch 5/50
125/125 [==============================] - ETA: 0s - loss: 1.5509 - accuracy: 0.3603
Epoch 5: val_accuracy did not improve from 0.36400
125/125 [==============================] - 23s 184ms/step - loss: 1.5509 - accuracy: 0.3603 - val_loss: 1.5522 - val_accuracy: 0.3595
Epoch 6/50
125/125 [==============================] - ETA: 0s - loss: 1.5388 - accuracy: 0.3664
Epoch 6: val_accuracy improved from 0.36400 to 0.36700, saving model to ./model.h5
125/125 [==============================] - 23s 187ms/step - loss: 1.5388 - accuracy: 0.3664 - val_loss: 1.5473 - val_accuracy: 0.3670
Epoch 7/50
...
Epoch 50/50
125/125 [==============================] - ETA: 0s - loss: 0.2704 - accuracy: 0.8967
Epoch 50: val_accuracy did not improve from 0.81250
125/125 [==============================] - 23s 186ms/step - loss: 0.2704 - accuracy: 0.8967 - val_loss: 0.5502 - val_accuracy: 0.7980
```

## 5. Result Analysis

After running the training process for 50 epochs, the model is able to get 89.3% accuracy on training data. Below are the different visualizations of the model performance.

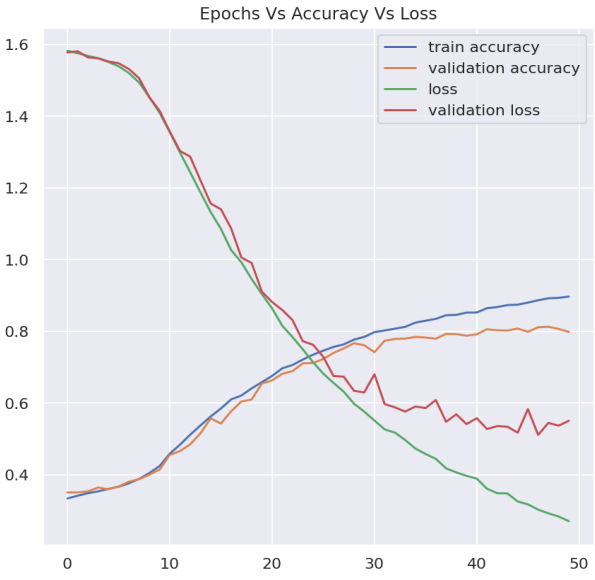



The adjacent plot shows the graph between Epochs Vs Accuracy. We can see that the accuracy almost reached 90% while validation accuracy stopped at 80%.




The adjacent plot shows the graph between Epochs Vs Loss. We can see that the training loss is at 0.2 and validation loss at 0.5 which is not bad.

Below graph shows the complete accuracy and loss graphs



### 5.1 Predictions

```
tests = [
    'i am so lucky to get placed in my favourite company',
    'i hate that person. he is so rude to me for no reason',
    'i feel down. i could not cover the whole syllabus',
    'i feel so threatened',
    'to my utter surprise, he actually cleared the exam'
]

for e in tests:
    predict(e, model)
```

```
1/1 [==============================] - 0s 18ms/step
i am so lucky to get placed in my favourite company : joy
1/1 [==============================] - 0s 16ms/step
i hate that person. he is so rude to me for no reason : anger
1/1 [==============================] - 0s 17ms/step
i feel down. i could not cover the whole syllabus : sadness
1/1 [==============================] - 0s 20ms/step
i feel so threatened : fear
1/1 [==============================] - 0s 17ms/step
to my utter surprise, he actually cleared the exam : surprise
```

### 5.2 Classification Report and Prediction Mismatches

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.80 | 0.88 | 0.84 | 695 |
| 1 | 0.73 | 0.83 | 0.77 | 275 |
| 2 | 0.76 | 0.59 | 0.66 | 159 |
| 3 | 0.88 | 0.79 | 0.83 | 581 |
| 4 | 0.85 | 0.79 | 0.82 | 224 |
| 5 | 0.70 | 0.73 | 0.71 | 66 |
| accuracy | | | 0.81 | 2000 |
| macro avg | 0.79 | 0.77 | 0.77 | 2000 |
| weighted avg | 0.81 | 0.81 | 0.81 | 2000 |

| | Inputs | Actual | Predicted |
|---|---|---|---|
| 0 | i felt anger when at the end of a telephone call | anger | fear |
| 1 | i don t feel particularly agitated | fear | anger |
| 2 | i pay attention it deepens into a feeling of b... | fear | joy |
| 3 | i was feeling as heartbroken as im sure katnis... | sadness | joy |
| 4 | i feel like my only role now would be to tear ... | sadness | anger |
| ... | ... | ... | ... |
| 378 | i am now and i still feel the aching lonelines... | sadness | joy |
| 379 | i dont want to always be judgmental of particu... | sadness | joy |
| 380 | im feeling cooped up and impatient and annoyin... | anger | fear |
| 381 | i have found myself fighting back as he wakes ... | sadness | love |
| 382 | i feel all weird when i have to meet w people ... | fear | surprise |

383 rows × 3 columns

Out of 2000 test samples, while testing our model predicted 383 samples falsely which means it has around 19.5% error.

# 6. Conclusion

To conclude the project “Emotion Classification using BERT Embeddings + LSTM”, we used BERT to get embeddings and Bi-LSTM to build the model architecture. our model performed well on some emotions while performed slightly less on other emotions due to unbalanced data and our Model able to achieve an overall 89.3% of accuracy which is not bad.

There might be better results if try with different types of embeddings or increasing model layers but it is left as the task to the reader.

## 7. References


[1] Political Text Classification using Word Embeddings and LSTM
https://arxiv.org/abs/1607.02501

[2] Using pre-trained word embeddings in a Keras model
https://keras.io/examples/nlp/pretrained_word_embeddings/

[3] News Text Classification Based on Improved Bi-LSTM-CNN
https://ieeexplore.ieee.org/abstract/document/8589431

[4] Hate Speech Detection Using Static BERT Embeddings
https://link.springer.com/chapter/10.1007/978-3-030-93620-4_6

[5] Research on Patent Text Classification Based on Word2Vec and LSTM
https://ieeexplore.ieee.org/abstract/document/8695493

[6] Bidirectional LSTM with attention mechanism and convolutional layer for text classification
https://www.sciencedirect.com/science/article/abs/pii/S0925231219301067

[7] Multi-class Text Classification using BERT-based Active Learning
https://arxiv.org/abs/2104.14289