**Text Classification**

**Text Category Classification**

**Mina Samir 212257**

**Steven Hany 210991**

**Introduction:**

Text category classification is the process of automatically classifying text documents into predefined categories or classes based on their content. It involves analyzing the features of a text, such as its vocabulary, syntax, and semantic meaning, and using machine learning algorithms to classify the text into one or more categories.

This technique is widely used in various applications, such as spam filtering, sentiment analysis, topic modeling, and content recommendation. Text category classification can help improve the efficiency and accuracy of many natural language processing tasks, as it can quickly and accurately classify large volumes of text data without human intervention.

**Project Idea:**

In this project, we worked on a model that takes text as input and classifies that text into one of the categories it trained on.

**Dataset: AG News**

The AG News dataset is a collection of news articles with each article belonging to one of four categories: World, Sports, Business, and Sci/Tech. The task associated with this dataset is to classify the news articles into one of these four categories. The articles were collected from more than 2,000 news sources. The articles are preprocessed and stored in a CSV file format, with each row containing the article's label and text. The dataset is commonly used for text classification tasks and has been used to train various machine learning models for natural language processing.

**Code:**

Created an ipynb file naming it Model\_Training.ipynb.

**Libraries:**

**Text

Description automatically generated**

**Pandas:** usedto manipulate and read csv file of the dataset and save it into a pandas dataframe.

**Numpy:** used for numerical computations in Python.

**Gensim.downloader:** is a library for downloading pre-trained word embeddings (word2vec).

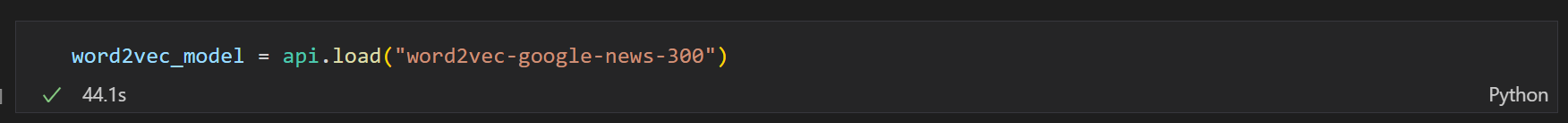
**from sklearn.model\_selection import train\_test\_split:** used for splitting the data into train, valid, and test sets.

**from sklearn.preprocessing import LabelBinarizer:** used to encode topic of the dataset.

**Tensorflow:** is an open-source machine learning framework used to build neural network layers.

**tensorflow.keras.preprocessing:** is a package for text preprocessing.

**Download and Load the Pretrained Model (Word2Vec):**



Here we used genism.downloader to download and load the word2vec pretrained model “word2vec-google-news-300”.

The Google News word2vec model is a pre-trained model that was trained on a large corpus of news articles. It contains 300-dimensional vectors for over 3 million words and phrases. The vectors were trained using the continuous bag-of-words (CBOW) model, which predicts a target word based on its context words, and the skip-gram model, which predicts context words given a target word. The Google News word2vec model is often used as a starting point for natural language processing tasks such as text classification, sentiment analysis, and language generation.

**Read the Dataset:**

AG news comes with two files train.csv and test.csv and topics are presented as numbers. So, we need to merge the two files into one so that we can split the all the dataset to train, valid, and test sets with the amount we need to assign to every set from the data and we need to replace the numbers in this dataset to their corresponding topic. We created a ipynb file (ag\_news\_merge.ipynb) and wrote a code to do all of that.

Text

Description automatically generated

Reading the train.csv file and changing column names.

Displaying the first 5 rows.

Text

Description automatically generated

Done the same with test.csv file.

Text

Description automatically generated

Merged the two dataframes into one dataframe.

A screenshot of a computer

Description automatically generated with medium confidence

Checked that the total number of rows of train and test files are equal to the merge of them to make sure that all rows have been taken.

Text

Description automatically generated

Replaced digits with their corresponding topic.



Created a csv file with the dataset naming it ag\_news\_merged.csv

Text

Description automatically generated

Finally, returned to our Model\_Training.ipynb file and read the dataset from the file we created with only the needed columns which are Topic and Description.

Graphical user interface, text

Description automatically generated

In this code:

* The first line prints the total number of rows in the DataFrame.
* The second line calculates the number of unique topics in the "Topic" column and assigns it to the variable N\_Topics which is equal to 4.
* The third line prints the names and counts of each unique topic in the "Topic" column using the value\_counts() method.

This code is useful for getting a quick summary of the distribution of topics in the DataFrame and verifying that the data is properly labeled.

Text

Description automatically generated

Saved description and its topic in separate variables.

Graphical user interface, application

Description automatically generated with medium confidence

In this code convert to One-hot encoding:

* a binary label matrix from a list of string labels using the LabelBinarizer class from the scikit-learn library. Here's how it works:
* The first line creates an instance of the LabelBinarizer class and assigns it to the variable Str2Bin.
* The second line fits the Str2Bin object to the list of string labels called Topics. This step essentially creates a mapping between each unique label in Topics and a unique binary value.
* The third line uses the transform() method of the Str2Bin object to convert the list of string labels into a binary label matrix called Topic\_Bin.

The resulting Topic\_Bin matrix has a shape of (n\_samples, n\_classes), where n\_samples is the number of text documents and n\_classes is the number of unique topics. Each row of the matrix represents a single text document, and each column represents a binary indicator for the presence or absence of a particular topic. For example, if there are 3 unique topics, and a particular document is labeled as belonging to the first and third topics, then the corresponding row of Topic\_Bin would be [1, 0, 1].

Graphical user interface

Description automatically generated with medium confidence

In this code we splitting data using using the train\_test\_split() function from the scikit-learn library

* The first and second lines splits the original Descriptions and Topic\_Bin arrays into training and test sets with a test size of 20% (specified by test\_size = 0.2). The stratify parameter ensures that the same proportion of each unique topic label is represented in both the training and test sets.

After we run this code we will have:

* Desc\_train, Desc\_valid, Desc\_test, Topic\_train, Topic\_valid, and Topic\_test. The Desc\_\* arrays contain the text data, while the Topic\_\* arrays contain the corresponding binary label matrices for the training, validation, and test sets.

Graphical user interface, text, application

Description automatically generated

In this code we tokenize the text data in the Desc\_train array using the Tokenizer class from the Keras preprocessing module:

* The first line creates an instance of the Tokenizer class and assigns it to the variable tokenizer. The parameters passed to the constructor specify how the tokenizer should process the text data. lower=True converts all text to lowercase, split=' ' specifies that words should be tokenized based on spaces, oov\_token='<pad>' specifies a special out-of-vocabulary token to be used for words that are not in the vocabulary, and filters specifies a list of characters to be removed from the text data.
* The second line fits the tokenizer object to the text data in the Desc\_train array. This step essentially builds the vocabulary of the tokenizer based on the words in the training set.
* The third line assigns the vocabulary of the tokenizer to the variable desc\_vocab.

At the end of this code we will have:

* tokenizer will have learned the vocabulary of the text data in Desc\_train, and desc\_vocab will be a dictionary mapping each unique word in the training set to a unique integer index.

Text

Description automatically generated

In this code we convert the text data Desc\_train, Desc\_valid, and Desc\_test into sequences of integers using the vocabulary learned by the Tokenizer object, and pads the sequences to a fixed length specified by max\_text\_length using the pad\_sequences() function from the Keras preprocessing module:

* The first, second and third lines converts the text data in Desc\_train, Desc\_valid, and Desc\_test into sequences of integers using the texts\_to\_sequences() method of the tokenizer object. The resulting sequences are padded with zeros to a fixed length of max\_text\_length using the pad\_sequences() function, and the padded sequences are assigned to the variable Desc\_train\_seq, Desc\_valid\_seq, Desc\_test\_seq.
* At the end of this code:
* Desc\_train\_seq, Desc\_valid\_seq, and Desc\_test\_seq will be arrays of shape (n\_samples, max\_text\_length) containing sequences of integers representing the padded text data. The integer values in each sequence correspond to the indices of the words in the vocabulary learned by the Tokenizer object.

Text

Description automatically generated

In this code we made a matrix called vector\_matrix of shape (n\_words\_in\_vocab + 1, 300) containing word embeddings for each word in the vocabulary learned by the Tokenizer object:

* The first line creates a list list\_of\_keys containing the keys (words) of the desc\_vocab dictionary.
* The second line initializes the vector\_matrix with zeros of shape (n\_words\_in\_vocab + 1, 300). The +1 is added to the number of words in the vocabulary to account for the special out-of-vocabulary token <pad> that was specified when creating the tokenizer object.

In the for loop:

* gets the word at index i in the list\_of\_keys list and assigns it to the variable word.
* In the next line we check if the word is in the word2vec\_model dictionary (which contains pre-trained word embeddings).
* If the word is in the word2vec\_model dictionary, the fifth line assigns the corresponding word embedding to the vector\_matrix at row i+1 [reason for adding 1 to the index i in the fifth line is to account for the special out-of-vocabulary token <pad> at index 0 of the vector\_matrix]

After the end of this code:

* vector\_matrix will be a matrix of shape (n\_words\_in\_vocab + 1, 300) containing word embeddings for each word in the vocabulary learned by the Tokenizer object. If a word is not in the word2vec\_model dictionary, its corresponding row in the vector\_matrix will be all zeros.

A screenshot of a computer

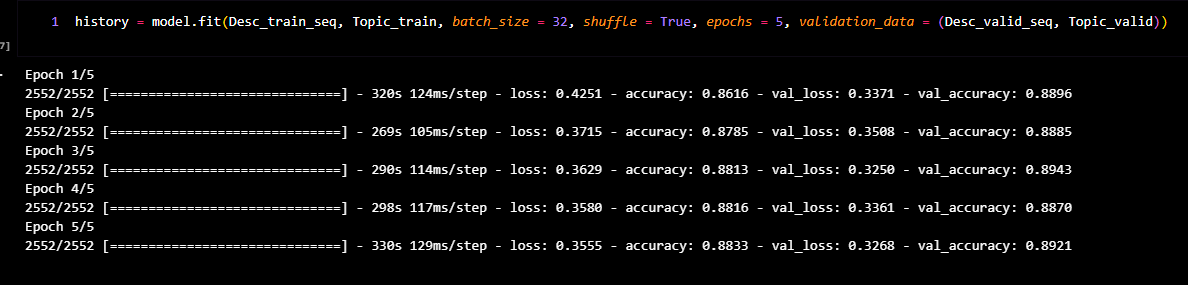
Description automatically generated

In this code we define a Sequential model using the Keras API in TensorFlow for text classification with LSTM layers:

* In the first line initializes a sequential model object.
* The second line adds an input layer to the model that takes input of shape Desc\_train\_seq[0,:].shape (which is (max\_text\_length,)).
* The third line adds an Embedding layer to the model that maps the words in the input sequence to their corresponding pre-trained word embeddings. The input\_dim argument is set to len(desc\_vocab) + 1 to account for the special out-of-vocabulary token <pad>. The output\_dim argument is set to vector\_matrix.shape[1] to match the dimension of the pre-trained word embeddings. e weights argument is set to the vector\_matrix created in the previous code block to use pre-trained word embeddings. Finally, the trainable argument is set to False to freeze the pre-trained word embeddings during training.
* The fourth line adds a Bidirectional LSTM[A bidirectional LSTM is a type of recurrent neural network that is able to process input sequences in both forward and backward directions, which can help capture information from past and future time steps in the sequence] layer to the model with 32 units and a dropout rate of 0.4.
* The fifth line adds a Dropout layer to the model with a dropout rate of 0.4.
* The sixth line adds a Dense layer to the model with 32 units and a ReLU activation function.
* The seventh line[output layer] adds a final Dense layer to the model with N\_Topics units and a softmax activation function to output the predicted class probabilities for each topic.
* The eighth line compiles the model by specifying the optimizer (Adam with a learning rate of 0.01), the loss function (categorical\_crossentropy), and the metrics to track during training (accuracy).

After running of this code we will have compiled model that can be trained on the training data using the fit method.

**Training:**



In this code fitting the compiled Keras model using the training data and validating it using the validation data.

model.fit() method takes in the following arguments:

* Desc\_train\_seq: the preprocessed training data
* Topic\_train: the corresponding labels for the training data
* batch\_size: the number of samples per gradient update
* shuffle: whether to shuffle the training data before each epoch
* epochs: the number of times the model will iterate over the entire training dataset
* validation\_data: the preprocessed validation data to evaluate the model after each epoch.

The output of model.fit() is a History object, which contains information about the training and validation loss and accuracy at each epoch. This information can be used to plot the learning curves and analyze the performance of the model.

In the epochs:

* We check each loss and accuracy of each epoch [loss: The value of the loss function, which is a measure of how well the model is performing. The goal is to minimize this value. Accuracy: The proportion of correct predictions made by the model. The goal is to maximize this value.]

Text

Description automatically generated

In this code:

* We print the accuracy of the last epoch for both training and validation data and the accuracy of the training data and valid data :
  + - Accuracy Training data: 88.33%
    - Accuracy Valid data: 89.21%

**Final Evaluation:**

Text

Description automatically generated

In this code we will print the accuracy and loss for the testing data:

* Accuracy=89.38%
* Loss= 0.3185

Graphical user interface, text

Description automatically generated

In this code we used the trained model to predict the topic label for the set of input descriptions, and saved the probabilities in 2D-array called pred

In the first dimension:

* There are the whole probabilities for the 4 topics, and we will select the greater probability.

In the second dimension:

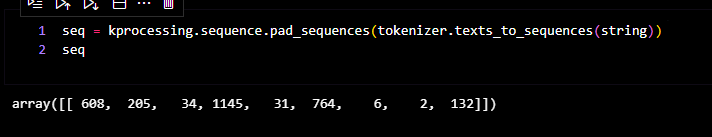
* After getting the greatest probability we will have the topic for each sentence in the test data

Graphical user interface, text, website

Description automatically generated

In this code we will convert the probabilities of pred and one-hot encoding of topic\_test to string that declare the correct class.

**Input handling code**



In this code:

* We will tokenize the input string using the same tokenizer as we used before in the dealing we training, validating and testing data
* We will also put padding in the sequence with zeros in case it is less than the max\_len and in case of it is greater than the max\_len we will truncate it.

After this code we will have:

* We will preprocess the input from the user.

Text

Description automatically generated

In this code:

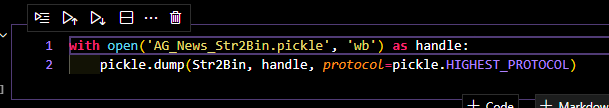
* After taking the sequence from the tokenizer we will pass it to the model to predict the array of probabilities, and confidence\_Score will take the greatest score of this array.

Text

Description automatically generated

In this code:

* + We will take the 2D-array of predictions and convert it to string as the same we did before.



In this code:

* the Str2Bin object has been saved as a pickle file named AG\_News\_Str2Bin.pickle.

Text

Description automatically generated with low confidence

In this code:

* We also save tokenizer object save as pickle file as before named AG\_News\_Tokenizer.pkl

A screenshot of a video game

Description automatically generated with medium confidence

In this code

* We saved the coded as h5 file and we will not want to run the whole project again.

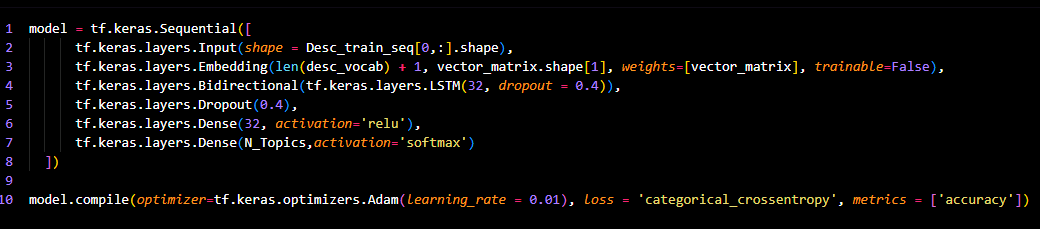
Text

Description automatically generated

In this code:

* At the end of this code we print the model summary with details like layer type and output shape and parameters.

**Assumptions**



In this code:

* We change the value of dropout many times like 0.4 to make sure that there is no overfitting in our model .
* We put number of neurons to 32 in the many layers like output layer to make it balanced neither overfitting nor underfitting.

**Input entry**

**Text

Description automatically generated**

in the above photo user will enter the news in the text area and the topic will reveal after the user press over the submit button with the confidence score.