AIML428 – ASSIGNMENT 1 REPORT

We were provided with two datasets from the AG news dataset, which includes titles and descriptions categorized into four major classes: "World", "Sports", "Business", and "Sci/Tech". Since the dataset was divided into separate training and test sets, I loaded each into its respective dataset.

Both the training and test datasets contained three columns: ['Title'], ['Description'], and ['Class Index']. Given that the class index was already represented as integer labels, there was no need to perform label encoding on the classes. I combined the 'Title' and 'Description' columns for both datasets, creating a unified text input for all models.

Step 1 – Term Frequency (TF) Classifier

Following the code examples from the lecture resources, I utilized CountVectorizer to vectorize the text input, excluding common stop words such as 'a', 'an', and 'the'. This process formed the basis for the Term Frequency classifier. For this assignment, I employed Multinomial Naive Bayes (NB), which yielded the highest accuracy scores when compared to Decision Trees. I did not attempt to use the SVM model due to its long execution time. The accuracy score for this step is 85%.

Step 2 – TF-IDF Classifier

Also following the code examples from the lecture resources, I utilized TfidfVectorizer to vectorize the text input with similar basis to the TF Classifier. The classification performance on the testing data is 90.26% and per-class metrics is:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Classes** | **Precision** | **Recall** | **F1-score** | **support** |
| 1 | 0.91 | 0.90 | 0.90 | 1900 |
| 2 | 0.95 | 0.98 | 0.96 | 1900 |
| 3 | 0.87 | 0.86 | 0.87 | 1900 |
| 4 | 0.88 | 0.88 | 0.88 | 1900 |

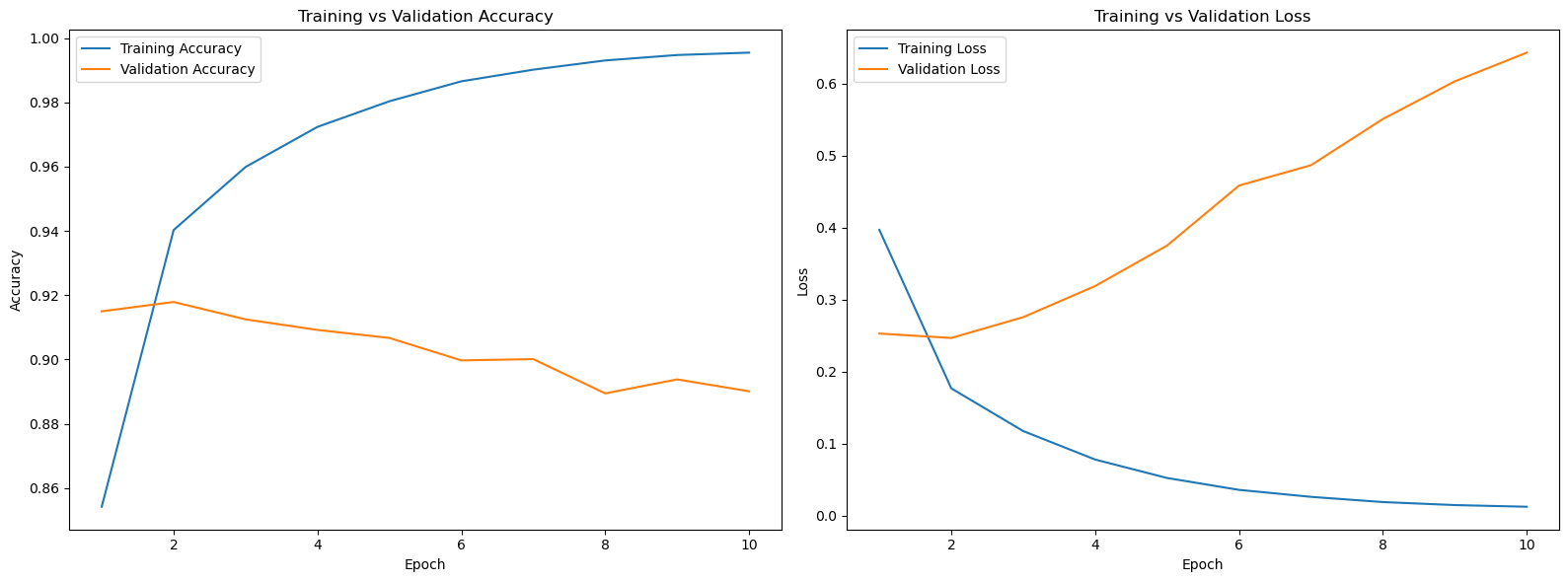
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Precision** | **Recall** | **F1-score** | Support |
| **Accuracy** |  |  | 0.90 | 7600 |
| **Macro avg** | 0.90 | 0.90 | 0.90 | 7600 |
| **Weighted avg** | 0.90 | 0.90 | 0.90 | 7600 |

Step 3 – Convolutional Neural Network (CNN)

A significant portion of the code was adapted from the lecture resources' examples. The embedding layer was configured with an embedding dimension of 50 and set as trainable (trainable=True). Most model settings followed the examples, except for the final dense layer. In this layer, the number of units was set to 5 to accommodate the classification problem, although the dataset actually contains four major classes. The activation function was set to 'softmax' to ensure that the output probabilities sum to 1, which is suitable for multi-class classification tasks like this assignment.

For the loss function, 'sparse\_categorical\_crossentropy' was chosen as the baseline. This was appropriate because the class labels are represented as integers (not one-hot encoded), which is common in multi-class classification problems.

Graphs to show accuracy changes during the training process for both the training and testing data



Step 4 – CNN-based classifier using pre-trained word embeddings

Building on the baseline CNN model from Step 3, I utilized the GloVe embedding file ('glove.6B.100d.txt') after downloading it. The code was largely adapted from the resource provided on the assignment page: <https://realpython.com/python-keras-text-classification/>. Instead of using random weights, the embedding layer was initialized with a pre-trained embedding matrix, and its trainable parameter was set to False. This ensured that the pre-trained embeddings remained unchanged during the training process.

Accuracy on the testing data is 90% and confusion matrix for this model:

A screenshot of a graph

AI-generated content may be incorrect.

Step 5 – CNN-based classifier by applying techniques

Building on the models from Steps 3 and 4, I decided to maintain the classifiers while focusing on enhancing data preprocessing and hyperparameter tuning. With references and utilized from text classification preprocessing techniques, such as those used in a similar problem on Kaggle (<https://www.kaggle.com/code/mdnurnabirana/toxic-comment-classification-using-rnn-lstm>), I implemented several preprocessing steps. These included normalizing text to lowercase, removing punctuation, and eliminating stopwords using functions like remove\_stopwords, normalize\_text, and remove\_punctuation. These functions were applied to both the training (X\_train) and test (X\_test) datasets.

In details:

* The function “remove\_stopwords” is to split the text into words, checks each word against a list of stopwords, and joins the remaining words back into a sentence. This is to removes common words like “the”, “and”, etc. that do not add much value to the meaning of the text.
* The function “normalize\_text” converts text to lowercase and remove special characters and normalizes spacing.
* The function “remove\_punctuation” is to remove punctuation marks from the text

After preprocessing, the datasets were tokenized and padded to prepare them for the CNN model. The CNN model architecture was based on code from Real Python (<https://realpython.com/python-keras-text-classification/>), with settings similar to those in Steps 3 and 4. To optimize performance, a grid search was conducted over hyperparameters such as the number of filters, kernel size, vocabulary size, embedding dimension, and sequence length. A randomized search with cross-validation identified the best hyperparameter combination. The best model was then evaluated on the test data, and a final model was trained with these optimal parameters. The final model achieved a test accuracy of 90%.

Step 6 – Summary that reports the test accuracy of all models

**Comparison results:**

|  |  |
| --- | --- |
| **Steps Description/Model** | **Test Accuracy** |
| Step 1 - Multinomial Naive Bayes (Term Frequency) | 0.847894737 |
| Step 2 - Multinomial Naive Bayes (TF-IDF) | 0.902631579 |
| Step 3 - CNN with Random Embeddings | 0.890131593 |
| Step 4 - CNN with Pre-trained Embeddings | 0.90407896 |
| Step 5 - CNN with Techniques | 0.900526345 |

**Brief analysis:**

Overall, all the models performed really well, with only minor differences between them,, with test results ranging from 84% to 90%. The models that give highest accuracy scores were the ones using TF-IDF features and pre-trained embeddings, which shows how important it is to tap into the meanings and importance of words in text classification.

The CNN models also saw a boost in accuracy when we added some extra preprocessing steps and tweaked the hyperparameters. However, the final model didn't quite match the performance of the CNN with pre-trained embeddings. This suggests that while these extra techniques can help, they might not always be enough to beat the power of pre-trained embeddings.