

**MINI PROJECT**

**COURSE:** DEEP LEARNING (CSE3189)

**TOPIC:** FAKE NEWS DETECTION

**ACADEMIC YEAR:** 2024-2025

**INSTRUCTOR:** SELVARAJ POORNIMA

**BATCH:** 14

**TEAM MEMBERS:**

VARSHINI P D -20221CSE0362

FIDHA BASHEER – 20221CSE0018

PRATIKSHA – 20221CSE0703

**Introduction:**

This deep learning project focuses on classifying news articles based on their content. The overarching goal is to develop a text classification model capable of accurately identifying the category or label associated with a given news item. Text classification is a foundational problem in natural language processing (NLP), and solving it effectively enables various applications such as spam detection, sentiment analysis, and news filtering. The project utilizes Python’s deep learning ecosystem, specifically TensorFlow and Keras, to build and train a neural network. Essential libraries like NumPy and pandas are used for data manipulation and preparation, while scikit-learn is employed for preprocessing tasks such as label encoding. The approach leverages both raw text (titles and articles) and pre-trained word embeddings (GloVe) to enhance semantic understanding. The workflow includes loading and cleaning the dataset, tokenizing text data, constructing embedding matrices, building a neural network, training the model, and finally evaluating its performance. This notebook-style project encapsulates the typical end-to-end lifecycle of a text classification pipeline using deep learning techniques.

**Dataset:**

The dataset used in this project is sourced from a CSV file named news.csv. It contains labeled news articles, including fields such as "title," "text," and "label." The initial step involves loading the dataset using pandas.read\_csv and displaying its structure using data.head(). A non-essential column named “Unnamed: 0” is removed to streamline the dataset. Following this, the categorical labels are encoded into numerical format using LabelEncoder from sklearn.preprocessing, which is necessary for training classification models. The dataset is limited to 3,000 samples for training, possibly to manage computational resources or focus on prototyping. Titles and texts are extracted and stored in separate lists alongside their corresponding labels, which facilitates tokenization and model input preparation. The dataset is further split into training and test sets, with 10% reserved for evaluation. Overall, the dataset section effectively sets up a manageable and structured input for the deep learning pipeline, preparing it for tokenization and embedding processes. The balance and quality of labels, though not explored in detail, will critically impact model performance. Understanding the dataset and ensuring it is representative of real-world distributions is key to building a robust classifier.

**Data Preprocessing:**

The data preprocessing stage is vital in preparing raw textual data for effective model training. In this project, the preprocessing pipeline includes tokenization, padding, and label encoding. Initially, LabelEncoder from scikit-learn is used to convert categorical text labels into numerical form, making them suitable for classification. For the textual data (titles and articles), tokenization is applied using Keras’s Tokenizer, which transforms words into unique integer indices. This is followed by sequence padding using pad\_sequences to ensure that all input sequences are of the same length, an essential requirement for batch processing in neural networks. The padding type and truncation method are set to 'post', meaning that shorter sequences are padded at the end, and longer ones are truncated from the end. This process helps in maintaining structural uniformity. Additionally, a vocabulary index is created, mapping each token to an integer, which is crucial for embedding and input to the model. The processed sequences and labels are then split into training and testing sets, with care taken to ensure label balance. This stage ensures that the raw text is transformed into a clean, structured format suitable for deep learning.

**Word Embeddings:**

To capture the semantic relationships between words, the project incorporates pre-trained GloVe (Global Vectors for Word Representation) embeddings. These embeddings transform words into dense vector representations where semantically similar words occupy nearby points in the vector space. The GloVe file glove.6B.50d.txt is loaded and parsed, creating a dictionary that maps words to their respective 50-dimensional vectors. An embedding matrix is then constructed, aligning the vocabulary from the tokenizer with the GloVe vectors. Words not found in the GloVe dataset are represented with zeros or random vectors, depending on the implementation. This embedding matrix is later used to initialize the weights of the Embedding layer in the neural network. The main advantage of using pre-trained embeddings like GloVe is that they have already learned contextual relationships from a large corpus, which improves generalization, especially when training data is limited. Integrating GloVe into the model allows it to understand the meaning and nuance of words beyond simple one-hot encoding or random initialization. Overall, this stage significantly boosts the model’s language comprehension and contributes to better accuracy and robustness in classification.

**Model Architecture:**

The model architecture is built using Keras’s Sequential API. It begins with an Embedding layer initialized with the previously constructed GloVe embedding matrix. This layer maps each word in the input sequence to a dense vector. Following the embedding layer, the architecture includes a Global Average Pooling layer, which reduces the dimensionality of the output by computing the average of all word embeddings in a sequence. This is a simple yet effective method for condensing information from entire sequences. It is followed by Dense (fully connected) layers that add non-linearity and learn complex patterns in the embedded space. The final output layer uses a softmax activation function to predict the probabilities of each class label. This makes the model suitable for multi-class classification problems. The architecture is deliberately kept shallow and efficient to prevent overfitting and to allow for quick experimentation. Each component of the model serves a distinct purpose, from embedding semantic relationships to making final label predictions. The design choices reflect a balance between simplicity and effectiveness, making it ideal for a text classification task where interpretability and speed are valuable.

**Model Training:**

Model training involves fitting the constructed neural network to the preprocessed data. The training data includes padded token sequences and their corresponding labels. The model is compiled using the Adam optimizer, which adjusts learning rates dynamically to improve convergence. The loss function used is categorical crossentropy, appropriate for multi-class classification. Training occurs over multiple epochs, with each epoch iterating through the entire dataset. The model’s performance is monitored using accuracy as a metric, and training history can be logged to observe how loss and accuracy evolve. Although not explicitly used here, callbacks such as EarlyStopping can be integrated to halt training when the validation loss stops improving, thus avoiding overfitting. Batch size and epoch count are parameters that influence the speed and stability of training. By using pre-trained embeddings and well-structured input data, the model converges effectively to a state where it can make accurate predictions. This training phase is crucial as it directly determines how well the model generalizes to unseen data. It is during this step that the neural network fine-tunes its weights based on input patterns, learning to associate specific text features with output labels.

**Evaluation:**

The evaluation phase assesses how well the trained model performs on unseen data. The test set, previously set aside from the training process, serves this purpose. The model’s predictions on the test sequences are compared against the actual labels to calculate performance metrics such as accuracy. Depending on the classification task, more detailed metrics like precision, recall, and F1-score may also be considered (though not explicitly shown in the notebook). The test accuracy provides a measure of the model’s ability to generalize beyond the training set. It is essential to ensure that the model performs consistently across different subsets of data, especially if it is to be deployed in real-world applications. Poor performance on the test set may indicate overfitting, underfitting, or data imbalance. Visualization of performance metrics or a confusion matrix could offer deeper insights, though this is not covered in the current implementation. Nonetheless, the evaluation step confirms the practical utility of the model and validates the overall approach taken in preprocessing, embedding, and architecture design.

**Conclusion:**

In conclusion, this project successfully demonstrates a deep learning pipeline for text classification using real-world news data. It covers the full workflow from data loading and preprocessing to model training and evaluation. The use of pre-trained GloVe embeddings enhances the semantic understanding of the model, and a straightforward yet effective neural architecture ensures robust classification. The model achieves acceptable accuracy on test data, indicating it has learned meaningful representations from the input. This project also highlights the importance of preprocessing, especially for text data, where tokenization and padding directly affect performance. Moreover, it showcases how transfer learning through word embeddings can significantly boost model capability, even when the dataset is relatively small. While the model is relatively simple, it serves as a strong baseline and can be expanded with more sophisticated architectures like LSTMs or transformers. Future improvements might also include hyperparameter tuning, additional evaluation metrics, or the incorporation of more diverse textual features. Overall, the project exemplifies the practical steps involved in building and validating a deep learning model for NLP tasks.