

CMPSC 497 MIDTERM PROJECT 1

News Classification Using CNN Model

Ali AlNaseeb

Aba6173@psu.edu

Spring 2025



1. Problem Definition and Dataset Curation

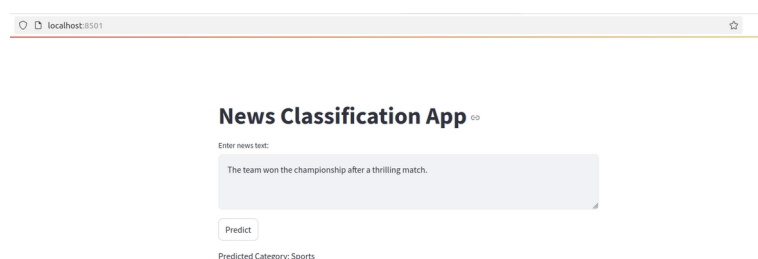
1.1 Problem Definition

The goal of this project is to **automatically classify news articles into one of four categories: World, Sports, Business, or Sci/Tech** . To solve the problem A **Convolutional Neural Network (CNN)** with pre-trained GloVe word embeddings is used for its ability to capture local patterns in text data such as n-grams .

In this project we develop a local **web-based user interface (UI) application** that allows users to input news text and automatically classify it into one of four categories: **World, Sports, Business, or Sci/Tech**. The application is built using the **Streamlit library**, a powerful tool for creating interactive web applications with Python. When a user enters a news article and clicks the **Predict** button, the application leverages a pre-trained **Convolutional Neural Network (CNN) model** to analyze the text and predict the most likely category to which the news belongs. This solution aims to provide an intuitive and efficient way for users to categorize news articles in real time, demonstrating the practical application of machine learning in natural language processing (NLP).

Key Components

- a) **User Input:** Users can input news text into a text box provided by the web application.
- b) **Prediction:** Upon clicking the **Predict** button, the application processes the input text using a pre-trained CNN model.
- c) **Output:** The application displays the predicted news category (World, Sports, Business, or Sci/Tech) to the user.
- d) **Technology Stack:**
 - **Streamlit:** For building the web-based UI.
 - **TensorFlow/Keras:** For the CNN model.
 - **Pre-trained GloVe Embeddings:** For word representations.
 - **NLP Preprocessing:** Tokenization, padding, combining title and description of news artical in train and test data set.



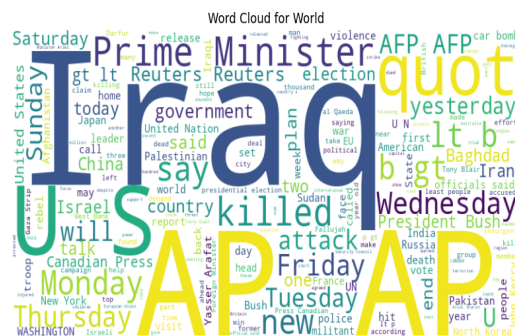
Screenshot : Local Web application for new classification

1.2 Dataset Curation

The dataset used is the **AG News dataset**, which consists of:

- **Training Set:** 120,000 news articles in train.csv where 30,000 for each class.
- **Test Set:** 7,600 news articles in test.csv where 1900 for each class.
- **Features:** Each article has a `title` and `description`.
- **Labels:** Four classes (World, Sports, Business, Sci/Tech).
- **GloVe Embedding** : The `glove.6B` dataset contains 400,000 unique words (vocabulary size) with four different kind of dimesion embeddings 50, 100, 200, and 300-dimensional Vectors.
 - a) `glove.6B.50d.txt`: Every word is represented as vector of 50 dimnention.
 - b) `glove.6B.100d.txt`: Every word is represented as vector of 100 dimnention.
 - c) `glove.6B.200d.txt`: Every word is represented as vector of 200 dimnention.
 - d) `glove.6B.300d.txt`: Every word is represented as vector of 300 dimnention.

Each glove dimension embedding have same vocabulary size (400,000 unique words)



1. Word cloud image of *world* news class



3. Word cloud image of *Business* news class



2. Word cloud image of *Sports* news class



4. Word cloud image of Sci/Tech news class

2. Word Embeddings, Algorithm, and Training

2.1 Word Embeddings

- **GloVe Embeddings:** Pre-trained GloVe embeddings (glove.6B.100d.txt) is used to initialize the embedding layer. These embeddings capture semantic relationships between words, improving the model's understanding of context.
- **Embedding Matrix:** A matrix of size (vocab_size, embedding_dim) is created, where vocab_size = 70,338 and embedding_dim = 100. Words in the training vocabulary were mapped to their corresponding GloVe vectors. This matrix is given as input to embedding layer of CNN.

2.2 Algorithm: Step-by-Step Explanation

2.2.1. Data Loading

- **What:** We are loading dataset from train.csv and test.csv.
- **How:** Using pandas.read_csv() command.
- **Why:** To train and test our news classification model.

2.2.2 Combining Title and Description

- **What:** The title and description fields are concatenated to form a single text field
- **How:** Using string concatenation.
- **Why:** To provide classification and model context in UI Combining these fields provides richer context for classification, as the title alone may not contain enough information.

2.2.3 Extracting texts and labels

- **What:** Extracting text and class columns from dataframes.
- **How:** Using pandas column access and conversion to lists/numpy arrays.
- **Why:** To separate the input features (text) and target labels (class) for training and testing.

2.2.4 Defining Parameters

- **What:** Defined key parameters like vocab_size, embedding_dim, max_length and num_classes.
- **How:** Based on dataset analysis and experimentation.

```
vocab_size = 70338 # number of unique word in training set
embedding_dim = 100 # Dimension of GloVe embeddings
max_length = 200 # Maximum length of input sequences
num_classes = 4 # AG News has 4 classes
```

- **Why:** We will use these parameters to control the model's input size, vocabulary, and output dimensions.

2.2.5 Tokenization the Text Data

- **What:** Create vocabulary from input text data where unique integer is assigned to each word.
- **How:** Using tokenizer from Keras module. This counts word frequencies and assigns a unique integer value to each word.
- **Why:** It is used for iterating over each word in the given text data to check whether that word is present in GloVe embedding data.

2.2.6 Conversion of Text to Sequences

- **What:** Converting Tokenized texts into sequences of integer values.
- **How:** Using tokenizer.text_to_sequences() function.
- **Why:** Raw text can't be processed directly, so we are tokenizing text into numerical format.

2.2.7 Padding Sequences

- **What:** Padding and truncating sequences to a fixed length (max_length).
- **How:** Using pad_sequences from Keras module.
- **Why:** To batch processing in neural networks, it is required to have all input sequences of the same length.

2.2.8 Conversion of Labels to Numpy Array

- **What:** Converting labels into numpy arrays.
- **How:** Using numpy.array() from numpy module.
- **Why:** TensorFlow/Keras are expecting labels in numpy format.

2.2.9 Loading GloVe Embeddings

- **What:** Loading pre trained GloVe word embeddings from a file.
- **How:** Using file I/O and parsing.
- **Why:** GloVe embeddings provide pre trained word vector that capture semantic relationships, improving model performance.

2.2.10 Creation of Embedding Matrix

- **What:** Creating an embedding matrix that map words in the vocabulary to their GloVe vectors.
- **How:** By iterating over the tokenizer's word index and matching words to GloVe vectors.
- **Why:** To initialize the embedding layer with pre-trained vectors, improving the model's ability to understand word semantics.

2.2.11 Building the CNN Model

- **What:** Building a CNN model using Keras' Sequential API.
- **How:** By stacking layers (Embedding, Conv1D, GlobalMaxPooling1D, Dense, Dropout).
- **Why:** The chosen CNN architecture is designed to balance simplicity, efficiency, and performance.
 - a) **Embedding Layer:** Converts tokenized text into dense vectors using GloVe embeddings.
 - b) **Conv1D Layer:** Applies 128 filters with a kernel size of 5 to extract n-gram features.
 - c) **GlobalMaxPooling1D:** Reduces the dimensionality of the output from the convolutional layer.
 - d) **Dense Layer:** A fully connected layer with 64 units and ReLU activation.
 - e) **Dropout Layer:** We are using 0.5 as dropout rate to prevent model overfitting.
 - f) **Output Layer:** A softmax layer with 4 units (one for each class).

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 200, 100)	7,033,800
conv1d (Conv1D)	(None, 196, 128)	64,128
global_max_pooling1d (GlobalMaxPooling1D)	(None, 128)	0
dense (Dense)	(None, 64)	8,256
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 4)	260

Total params: 7,106,444 (27.11 MB)

Trainable params: 72,644 (283.77 KB)

Non-trainable params: 7,033,800 (26.83 MB)

Screenshot : Model summary

2.2.12 Compiling the Model

- **What:** Compiled model with an optimizer, loss function, and metrics.
- **How:** Using `model.compile()`.
- **Why:** To configure the model for training.

2.2.13 Training the Model

- **What:** Train the model using training data.
- **How:** Using `model.fit()`.
- **Why:** To learn the patterns in the data and optimize the model's parameters.

2.2.14 Evaluation of the Model

- **What:** Evaluating the model performance on the test dataset.
- **How:** Using `model.evaluate()`.
- **Why:** To measure accuracy and generalization ability of the model.

2.2.15 Prediction Using the Model

- **What:** Use the trained model to predict the class of new, unseen news text data.
- **How:** Using `model.predict()`.
- **Why:** To determine model's ability to classify real-world data.

2.3 Training Details

2.3.1 Optimizer : Adam

- **What :** Adam (Adaptive Moment Estimation) is an adaptive optimization algorithm that combines the benefits of two other popular optimizers:
 - a) **Momentum:** Helps accelerate convergence by accumulating gradients from previous steps.
 - b) **RMSProp:** Based on magnitude of recent gradients, adapts the learning rate for each parameter.
- **Why?**
 - a) **Adaptive Learning Rate:**
 - Adam automatically adjusts the learning rate for each parameter, which is especially useful for text data where word frequencies and importance vary widely.
 - For example, common words like "the" or "and" may require smaller updates, while rare words like "quantum" may require larger updates.
 - b) **Efficient Convergence:**

- Adam converges faster than traditional optimizers like Stochastic Gradient Descent (SGD) because it uses momentum to navigate the loss landscape more effectively.
- This is particularly important for large datasets like AG News, where training time can be a concern.

c) **Robustness to Sparse Gradients:**

- Text data often results in sparse gradients (many zeros) due to the high-dimensional nature of word embeddings. Adam handles sparse gradients well, making it suitable for NLP tasks.

d) **Default Choice for NLP:**

- Adam is widely used in NLP tasks because of its reliability and performance. It requires minimal tuning (e.g., just setting the learning rate) and works well out of the box.

• **Why not other optimizer ?**

a) **SGD (Stochastic Gradient Descent):**

- It requires careful tuning of the learning rate and momentum.
- Slower convergence than Adam.

b) **RMSProp:**

- Lacks momentum, which can slow down convergence.

c) **Adagrad:**

- Aggressively reduces the learning rate, which can lead to premature convergence (stuck in suboptimal solutions).

2.3.2 Loss Function: Sparse Categorical Cross-Entropy

- **What?** : It is a loss function used for multi-class classification task where labels are integer (e.g., 0,1,2,3). It computes the cross-entropy loss between the true labels and the predicted probabilities.

• **Why?**

a) **Integer Labels:**

- In our dataset, the labels are integers (0 for World, 1 for Sports, etc.). Sparse categorical cross-entropy is designed specifically for this type of label encoding.
- It avoids the need to one-hot encode the labels, which saves memory and computation.

b) **Multi-Class Classification:**

- The AG News dataset has 4 classes, make it a multi-class classification problem. Sparse categorical cross-entropy is well-suited for such tasks because it computes the loss value for each class independently.

c) **Probabilistic Output:**

- The output layer of the model uses a **softmax activation**, which produces a probability distribution over 'four' classes. Sparse categorical cross-entropy measures how well the predicted probabilities match the true labels.
- **Why we are not using other loss functions?**
 - a) **Categorical Cross-Entropy:** Requires one-hot encoded labels, which is unnecessary for integer labels and it increases memory use.
 - b) **Binary Cross-Entropy:** It is used for binary classification tasks (only 2 classes) and not suitable for multi class problems.
 - c) **Mean Squared Error (MSE):** Not appropriate for classification tasks because it assumes a continuous output, while classification requires discrete class labels.

2.3.3 Learning Rate: 0.001

- **Why 0.001 ?**
 - The default learning rate for Adam is 0.001, which works well for most tasks, including text classification.
 - It strikes a balance between fast convergence and stable training.
- **Why Not a Higher or Lower Learning Rate ?**
 - a) **Higher Learning Rate (e.g., 0.01):**
 - May cause the model to overshoot the optimal solution, leading to unstable training or divergence.
 - b) **Lower Learning Rate (e.g., 0.0001):**
 - May result in slow convergence, requiring more epochs to achieve good performance.

2.3.4 Batch Size : 128

- **Why 128 ?**
 - **Balance Between Speed and stability :** A batch of 128 is large enough to provide a stable gradient estimate but small enough to fit in memory and allow for faster training.

- **Efficient Use of Resources:** Larger batch sizes(e.g., 256 may require more memory and slow traing, while smaller batch sizes (e.g., 32) may result in noisy gradient.

2.3.4 Epochs : 5

- **Why 5 Epochs ?**
 - a) **Early Stopping:** Training for 5 epochs is often sufficient for the model to converge on text classification tasks, especially with pre-trained embeddings. If the validation loss stops improving, we can use early stopping can be used to prevent model overfitting.
- **Why Not More Epochs?**
 - a) **Overfitting Risk :** If we train for too many epochs, it can cause the model to overfit for the training data and reduces its generalization ability.
 - b) **Diminishing Returns:** After a certain point, additional epochs provide minimal improvement in performance.

3. Results and Presentations

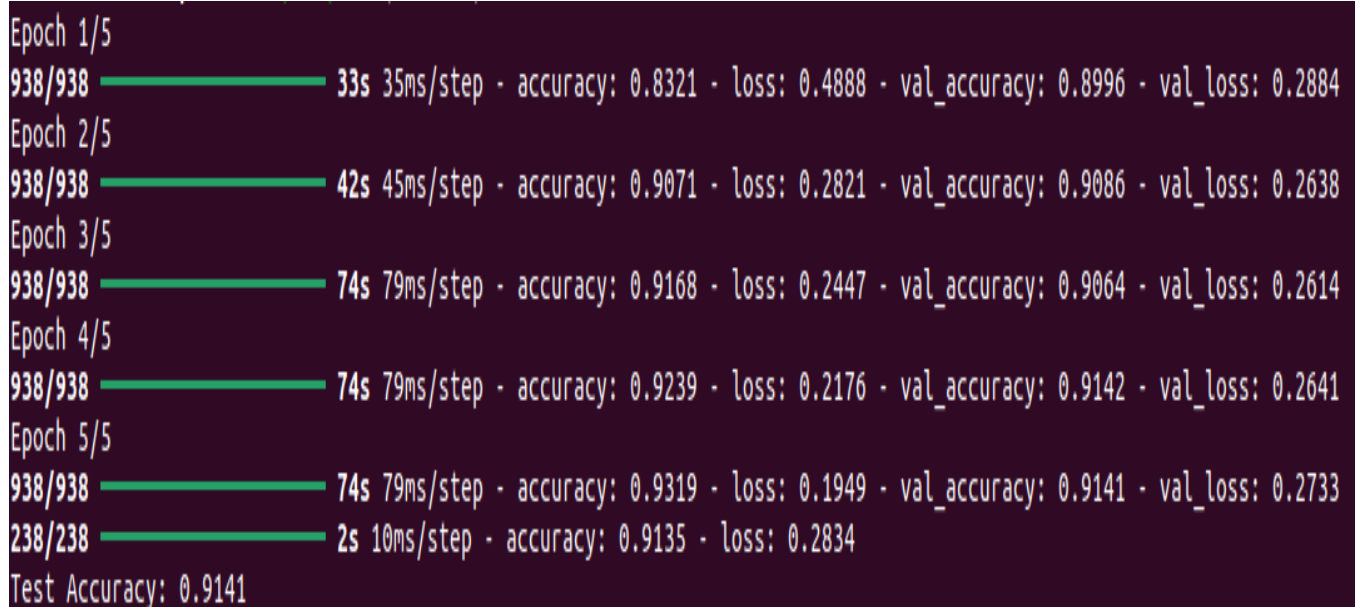
3.1 Accuracy : The model achieved an accuracy of **91%** with deviation of 0.5% on the test data set.

3.1.1 Training Accuracy :

- **Definition:** Training accuracy is the accuracy of the model on the **training data** during training.
- **Purpose:** It measures how well the model is learning to fit the training data.
- **Calculation :** The model is marking predications on training data after completion of each epoch. These predictions are compared to the true labels to calculate accuracy.

3.1.2 Valaidation Accuracy :

- **Definition:** Validation accuracy is the accuracy of the model on a validation set during training. We used test dataset only as validation set.
- **Purpose:** We are using it to determine the performance of the model and detect overfitting.
- **Calculation :** After each epoch, the model makes predictions on the validation data. These predictions are compared to the true labels to calculate accuracy.



```
Epoch 1/5
938/938 ————— 33s 35ms/step - accuracy: 0.8321 - loss: 0.4888 - val_accuracy: 0.8996 - val_loss: 0.2884
Epoch 2/5
938/938 ————— 42s 45ms/step - accuracy: 0.9071 - loss: 0.2821 - val_accuracy: 0.9086 - val_loss: 0.2638
Epoch 3/5
938/938 ————— 74s 79ms/step - accuracy: 0.9168 - loss: 0.2447 - val_accuracy: 0.9064 - val_loss: 0.2614
Epoch 4/5
938/938 ————— 74s 79ms/step - accuracy: 0.9239 - loss: 0.2176 - val_accuracy: 0.9142 - val_loss: 0.2641
Epoch 5/5
938/938 ————— 74s 79ms/step - accuracy: 0.9319 - loss: 0.1949 - val_accuracy: 0.9141 - val_loss: 0.2733
238/238 ————— 2s 10ms/step - accuracy: 0.9135 - loss: 0.2834
Test Accuracy: 0.9141
```

Screenshot : Training Acuracy and Validation Accuracy at every epoch

3.1.3 Observations: Figure1.0 illustrates the training and validation accuracy of the model over 5 epochs. The training accuracy (blue line) increases steadily, reaching 93% by the final epoch, indicating that the model is effectively learning the patterns in the training data. The validation accuracy (orange line) also increases but at a slower rate, reaching 91.4% by the final epoch. Based on the above observation, the model generalizes well to unseen(or new) data, with minimal overfitting.

The small gap between the training and validation accuracy curves indicates that the model is not overfitting significantly. This is likely due to the use of dropout and pre-trained embeddings, which help regularize the model.

The consistent increase in both training and validation accuracy across epochs demonstrates that the model is learning effectively and converging well.

3.1.4 Conclusion: The high validation accuracy (91%) suggests that the model is well-suited for the task of news article classification and can be deployed in real-world applications with confidence.

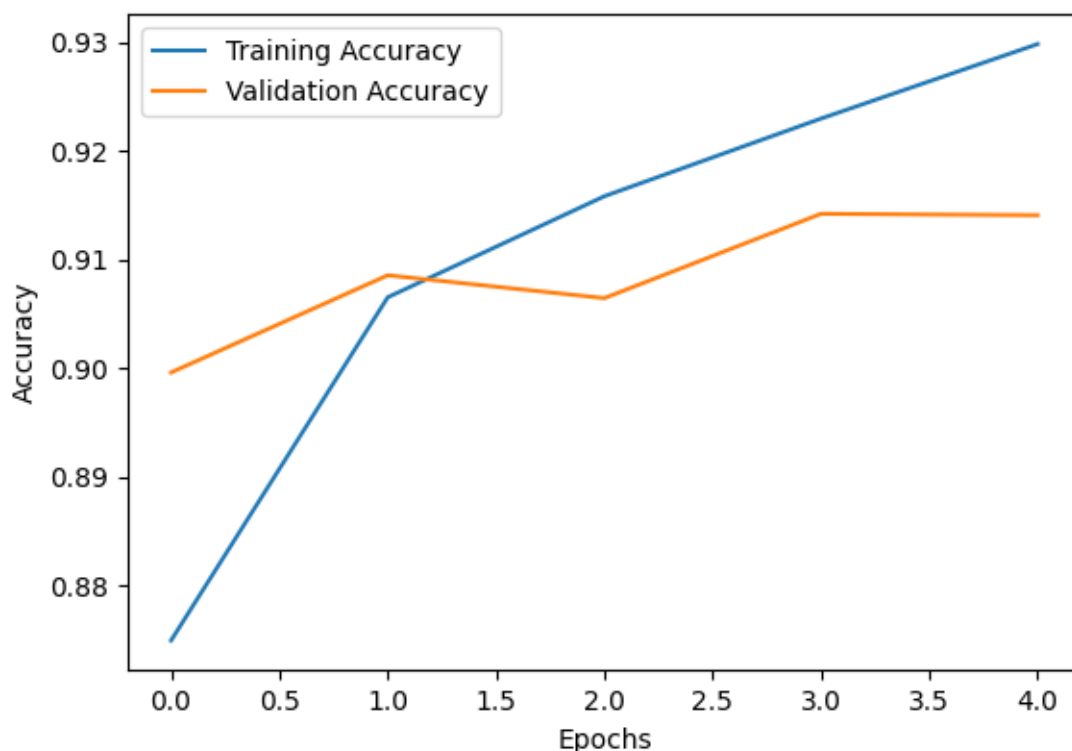


Figure1.0 : Training Accuracy and Validation Accuracy over 5 epoch

3.2 Confusion Matrix:

- **Definition:** Confusion matrix is a $N \times N$ matrix. By definition a confusion matrix C is such that C_{ij} is equal to the number of observations known to be in class i and predicted to be in class j .
- **Structure :** Rows represent the actual classes and Columns represent the predicted classes. Each cell shows the number of instances that belong to a specific actual class and were predicted as a specific class.
- **Purpose :** The confusion matrix provides a detailed breakdown of the model's predictions, showing how well it performs for each class and where it makes mistakes.
- **Observations :** In the confusion matrix, diagonal elements are representing correct predictions. For example, 1707 World articles were correctly classified as World, and 1866 Sports articles were correctly classified as Sports.

The off-diagonal elements represent misclassifications. For instance, 83 World articles were misclassified as Business, and 125 Sci/Tech articles were misclassified as Business.

The model performs exceptionally well for the Sports category, with 1859 correct predictions and only 41 misclassifications.

The Business category shows some confusion with Sci/Tech, with 157 Business articles misclassified as Sci/Tech and 125 Sci/Tech articles misclassified as Business.

- **Conclusion :** The confusion matrix highlights areas for improvement, such as reducing misclassifications between Business and Sci/Tech articles. This could be achieved by incorporating additional features or fine-tuning the model.

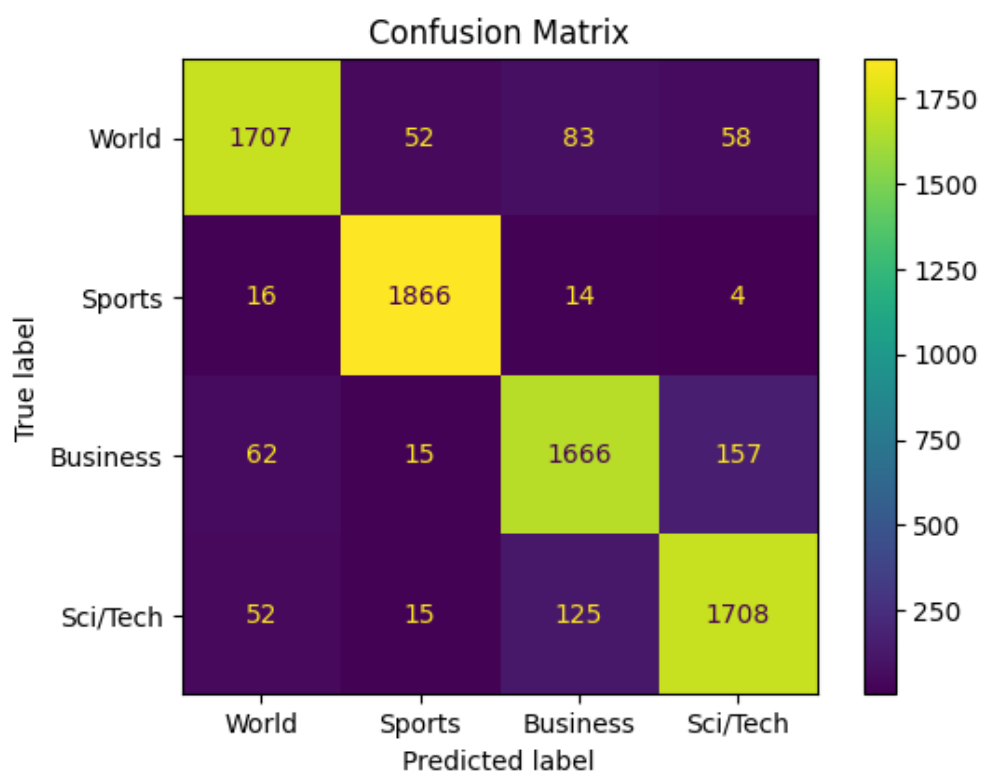


Figure2.0 : Confusion Matrix for given four classes

3.3 Classification Metrics:

- **True Positive (TP):** Instance belong to class i and predicted as class i.
- **False Positive(FP) :** Instance belonging to some other class predicted as class i.
- **False Negative(FN) :** Instance belongs to class i but predicted as some other class.
- **True Negative(TN) :** Instance belongs to class some other class and not predicted as class i.

3.3.1 Classification Accuracy : In general, accuracy metric measures the ratio of correct prediction over the total number of instances evaluated. The best value is 1 and the worst value is 0.

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

3.3.2 Precision : The precision is the ratio true positives over total predicted positives. The best value of precision is 1 and 0 is worst value. When the costs of False Positive is high, it is a good measure

$$\text{Precision} = \frac{TP}{TP+FP}$$

3.3.3 Recall : Recall calculates how many of the Actual Positives our model capture through labeling it as Positive (True Positive). When there is a high cost associated with False Negative, it is a good measure

$$\text{Recall} = \frac{TP}{TP+FN}$$

3.3.4 F-1 Score : The harmonic mean of precision and recall. F1 Score seek a balance between precision and recall.

$$\text{F-1 Score} = \frac{2 * (Precision * Recall)}{Precision + Recall}$$

3.3.5 Support : Number of occurrences of each class in the dataset is called support.

3.4 Classification Reports :

Class	Accuracy	Precision	Recall	F-1 Score	Support
World	0.90	0.93	0.90	0.91	1900
Sports	0.98	0.96	0.98	0.97	1900
Business	0.88	0.88	0.88	0.88	1900
Sci/Tech	0.90	0.89	0.90	0.89	1900

Table1.0 : Classification Report

- **Purpose** : The classification report provides a detailed evaluation of the model's performance for each class, including precision, recall, F1-score, and support.
- **Observations** : The model achieves high precision, recall, and F1-scores for all classes, indicating strong performance across the board. The Sports category has the highest F1-score (0.97), demonstrating the model's ability to accurately identify sports-related articles.

The Business and Sci/Tech categories show slightly lower F1-scores (0.88-0.89), suggesting some confusion between these classes.

The balanced support indicates that the dataset is evenly distributed across the four classes.

- **Class wise Performance** :

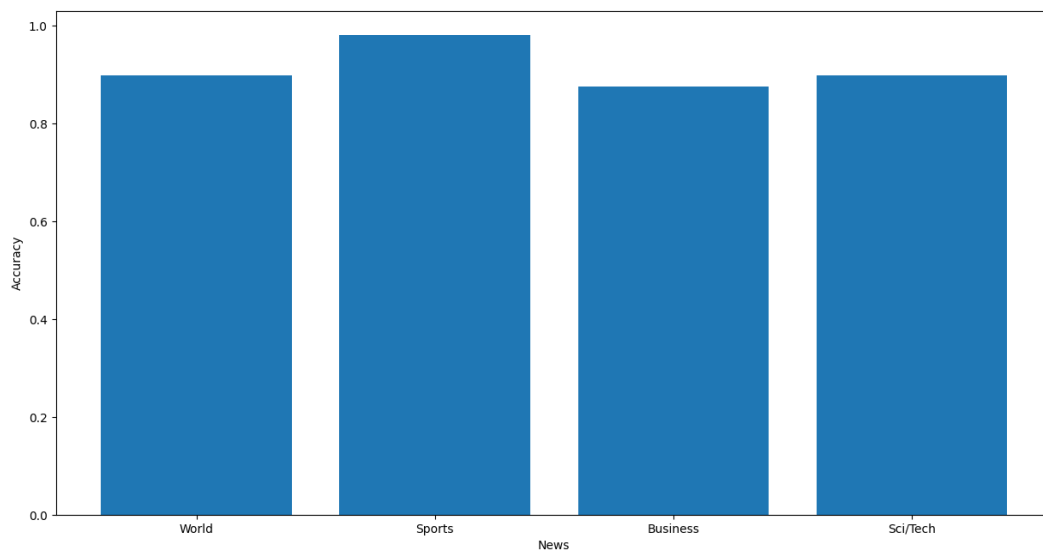


Figure3.0 : Accuracy Score of each individual class

- **Conclusion** : The classification report summarizes the model's performance for each class, including precision, recall, F1-score, and support. The model achieves high precision, recall, and F1-scores for all classes, with the highest performance for the Sports category. It highlights areas for improvement, such as reducing confusion between Business and Sci/Tech articles.

4. Analyses and Experiments

4.1 Computational Complexity :

- **Training Time** : 173.37 seconds
- **Inference Time** : Average 100 mili second
- **Memory Usage During Model Training** : 54.63 MB
- **Memory Usage During Prediction** : 14MB

4.2 Training Data :

- In the training dataset total 70337 unique word are present out of which 56959 words are matching with GloVe dataset and 13378 are not matching. Most of the words which are not matching are words having punctuation marks for example : i'd, don't, they're, etc. Some alpha numeric words of training dataset does not match with GloVe vocabulary for example : p2pnet, power5, inq7, etc.
- By Default training data is shuffled before every epoch to prevent the learning from order of the data. This shuffling may reslt slight variation in the classification reports as every time we run the code. We can make shuffling off by giving parameter `shuffle = False` to reproduce same result but making shuffling off is bad idea as the model can learn from the order of training data and result in less test accuracy and other classification metrics. However the validation data is not shuffled by default. This ensures the evaluation is consistent across epochs.

4.2 Experiment with Different GloVe Embedding vector dimention :

In the project we have used glove.6B.100d.txt as baseline embedding vector but as mentioned earlier glove.6B contains four different kind of embedding vector dimension data i) glove.6B.50d.txt, ii) glove.6B.100d.txt , iii) glove.6B.200d.txt, iv) glove.6B.300d.txt

In the experiment part we will train the model with each GloVe embedding data which is used as embedding layer of the CNN model and we will analysis the complexity of the models and accuracy achieved by them one by one.

In analyses we see time taken in each traing of model, runtime memory usage while training the model with each kind of embedding vector dimension, total parameters including tainable and non trainable, the size of learned model when we save it in *.h5 format and the accuracy score. We should take care of fact that while training the models **no other application** should run as other application may affect the performance analyses.

GloVe Embedding Vector	Training Time (seconds)	Run Time Memory Usage (MB)	Total Training Parameters	Size of Learned Model (MB)	Accuracy Score
50 Dimension	123.57	99.77	3,557,544	14.6	91.14%
100 Dimension	173.37	54.63	7,106,444	29.0	91.57%
200 Dimention	291.36	59.40	14,204,244	57.9	91.67%
300 Dimension	404.25	28.32	21,302,044	86.8	91.57%

Table2.0 : Model Training complexity with four different embedding vector dimensions

- **Observations from Table2.0:**

- As the dimension size of embedding vctor increases the trainning time also drastically increases.
- It is very surprising that run time memory usage for 50 Dimensional data is very high as compared to other vector dimensional data. But it is what it is.
- Total training parameter increases by huge amount as the vector dimension increases.
- The size of saved model after the learning also directly proportional to the size of embedding vector dimension.
- Accuracy score achieved by 100-D, 200-D, 300-D is almost same while 50 Dimensional vector achieved slightly less accuracy score as compared to other three.

- **Inference time and memory usage in prediction :**

As inference time and memory usage can vary based on the input string. So we will use same news text input to see what is the inference time and memeory usage for all four saved model which contains corresponding one of four embedding vector matrix.

News text : “Warren Buffett Amasses more cash and sells more stock, but does not explain why in Annual letter.”

GloVe Embedding Vector	Inference Time (Mili Second)	Memory Usage during prediction (MB)	Prediction class
50 Dimension	114.18	13.38	Business
100 Dimension	111.71	13.25	Business
200 Dimention	109.22	13.38	Business
300 Dimension	112.28	13.25	Business

Table3.0 : Inference Time and memory usage with four different embedding vector dimensions

- **Conclusion :** All model predicted the correct class '*Business*'. If we compare all the models then we can clearly see that we have some benefit in resource consumption with model trained using glove.6B.50d.txt while some benefit in accuracy score and resource consumption with model trained using glove.6B.300d.txt. The model trained with glove.6B.100d.txt is balanced approach which cover benefit of both higher dimension and lower dimension.

News Classification App 50-D

Enter news text:

Warren Buffett Amasses more cash and sells more stock, but does not explain why in Annual letter.

Predict

Predicted Category: Business

Inference Time: **114.18 ms**

Memory Usage: **13.38 MB**

Screenshot : Inference time and memory usage with 50 Dimensional embedding vectors

News Classification App 100-D

Enter news text:

Warren Buffett Amasses more cash and sells more stock, but does not explain why in Annual letter.

Predict

Predicted Category: Business

Inference Time: **111.71 ms**

Memory Usage: **13.25 MB**

Screenshot : Inference time and memory usage with 100 Dimensional embedding vectors

News Classification App 200-D

Enter news text:

Warren Buffett Amasses more cash and sells more stock, but does not explain why in Annual letter.

Predict

Predicted Category: Business

Inference Time: **109.22 ms**

Memory Usage: **13.38 MB**

Screenshot : Inference time and memory usage with 200 Dimensional embedding vectors

News Classification App 300-D

Enter news text:

Warren Buffett Amasses more cash and sells more stock, but does not explain why in Annual letter.

Predict

Predicted Category: Business

Inference Time: **112.28 ms**

Memory Usage: **13.25 MB**

Screenshot : Inference time and memory usage with 300 Dimensional embedding vectors

5. Lessons&Experience

5.1 Lesson Learned

i) CNN is effective tool for text classification : As CNN is popular for image data classification. In this project we learned that CNN can also be used for text classification where instead of image which is array of pixel values here text is array of words and each word itself can be represented as vector of integer values. Generally, for image classification CNN model uses 2D convolution layer but in text classification 1D convolution layer is effective.

ii) Overfitting issue : We have to take care of overfitting issue by adding drop out layer and early stopping so that model does not stick too much to training data and so could perform well on the new text which is out of the vocabulary of training data.

iii) Misclassification : Most of the misclassification occurs due to mixed vocabulary in news text for example : '*climate change*' this vocabulary can be present in 'world' news or can be present in sci/tech news. We learned that how important domain-specific vocabulary is, in text classification tasks. The confusion matrix revealed specific patterns in misclassifications, such as confusion between the Business and Sci/Tech categories.

iv) Slight Deviation in classification metrics : As weights are initially initialized with random values it may result in slight variation in evaluation metrics every time we try to train the model. This variability can be fixed by using random seed value. However this alone does not contribute to variability another factor is shuffling of training data to prevent model learning from order of training data. Making shuffling of training data off is not a good idea.

v) Scope of improvement : Although this project achieved good results, there is still room for improvements. We can use hybrid model like combination of CNN and LSTM(RNN) to get better classification of new text, by capturing both local pattern(CNN) and long term dependency(LSTM) between words in the given text.

We can add more convolution layer to get better performance but there is tradeoff between accuracy and computational complexity.

There were so many words in training which did not match with GloVe embedding vocabulary. For unmatched words we filled them with zero vector. There are better approaches in which unmatched words are filled with specific probability distribution methods.

5.2 Experience

i) Simple and Effective : While working on this project I experienced that simple model can also give good result. Initially I was trying to build LSTM (RNN) based solution but I found that model like LSTM need huge computational resources to train the model. I kept running the model for one day but still got very poor result. Whereas CNN gave 91% accuracy by just training time of approximately 5 minutes.

ii) Pre trained embedding (GloVe) : Thanks to GloVe embedding dataset, by using that we easily got good accuracy other wise building matrix for embedding layer by our own would have been hectic task and with unsurity of good satisfactory results.

iii) Feasibility of iterative experiments : Since the model training time was very less I did so many iterative experiment like

(a.) Making pretrained embedding layer parameter `Tunable = False/True` and see the performance difference between the `Tunable = False` and `Tunable = True`. There were no significant difference in the performance when `Tunable` is on and off.

(b.) Study of complexity of model with each of four different GloVe embedding dimensional vectors

(c.) Adding of multiple layers in CNN and see how the model perform.

If the model training would have been time consuming then this much iterative experiment may not be possible.

iv) Time Consuming Debuging : Sometimes debugging the error like '*key value error*' or '*Attribute error*' was very time consuming. Sometimes I took help from internet to see the solution for the error and have to patiently apply the given solutions one by one till the error get resolved.

v) Web based application : I feel very happy that I made the local web based news prediction application which can be used for personal experiments and can demonstrate the real world application of the project.