Sentiment Analysis Report

1. Data Pre-processing

The movie review dataset was preprocessed by applying text cleaning techniques. Firstly, I defined a function to clean the text data, which removed non-alphabetic characters and 'br' tags, converted the text to lowercase, tokenized the text, removed stop words, and lemmatized the tokens. This function was then applied to both the training and testing data. In addition, the cleaned data was encoded such that positive reviews were labeled as 1 and negative reviews as 0 for both training and testing dataset.

Furthermore, visualizations were created using WordClouds to gain insights into the most frequent words in both positive and negative reviews. As the results showed in the notebook, positive reviews tend to include words like 'good' and 'great' while negative reviews tend to include words like 'bad' and 'worst'.

The text data was then tokenized using the Keras Tokenizer class and the texts were converted to sequences. The sequences were also padded to a maximum length of 100 using the Keras 'pad_sequences' function. The label data were encoded using the LabelEncoder class from the scikit-learn library.

2. Methodology

2.1. Method 1 – Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) is a type of neural network that is specially designed for tasks related to image and text classification. CNN operates by convolving a set of filters over the input data and learning a set of features that are relevant for the particular task. In the context of movie review text classification, the input data consists of sequences of words, and the filter slides over these sequences to those features that are essential for sentiment analysis.

The CNN model in the provided code uses an embedding layer to transform the input sequences of words into a dense vector representation, Then the convolutional layer applies a set of filters to the output of the embedding layer, and the resulting feature maps are down sampled using max pooling. The flattened output then goes through a dropout layer to prevent overfitting, and a fully connected layer is employed to categorize the input data into either positive or negative sentiment.

2.2. Method 2 – Long Short-Term Memory (LSTM)

The Long Short-Term Memory (LSTM) mode is a kind of Recurrent Neural Network (RNN) architecture that is designed to better capture long-term dependencies in sequential data, In an LSTM mode, each time step of the input sequence is processed by a cell that includes a memory block and three gates: the input gate, output gate, and forget gate. The input gate regulates the flow of new information into the memory block, the output gate controls the output from the memory block, and the forget gate controls the retention of previous information in the memory block.

The LSTM model is extensively employed in various applications, such as speech recognition, natural language processing, and time series analysis, where accurate predictions depend on long-term dependencies.

2.3. Method 3 – K-means Algorithm

The K-means algorithm is a clustering technique that divides a dataset into K clusters with the goal of making the points within each cluster as similar as possible while also keeping them as different as possible from the points in other clusters. To achieve this, the model assigns each data point to the nearest cluster centroid in an iterative process and then recalculates the centroids based on the new assignment.

K-means is a commonly used clustering algorithm that operates in the following manner:

- First, k centroids are chosen at random by selecting k distinct examples from the dataset, and the centroids are positioned at their respective locations.
- Next, the algorithm repeats the following process until convergence is reached, meaning that the centroids stop moving:
    1. Assign each example to the closest centroid.
    2. Recalculate the centroids to be the mean of the instances assigned to them.

3. Performance, Evaluation & Comparison

3.1. Method 1 – Convolutional Neural Network (CNN)

I trained a CNN model on the preprocessed text data using Keras with TensorFlow backend. To find the optimal hyperparameters for the CNN model, I used a randomized search with cross-validation over a range of possible values. The best hyperparameters were found to be:

- Optimizer: Rmsprop
- Number of filters: 64
- Kernel size: 3
- Dropout rate: 0.5
- Activation function: tanh

This CNN model was trained for 3 epochs with a batch size of 128. Table 1-1 shows the overall summary of the model's performance, including metrics like precision, recall, and F1 score:

Table 1-1:

```
782/782 [==============================] - 3s 4ms/step
              precision    recall  f1-score   support

           0       0.88      0.86      0.87     12500
           1       0.87      0.88      0.87     12500

    accuracy                           0.87     25000
   macro avg       0.87      0.87      0.87     25000
weighted avg       0.87      0.87      0.87     25000
```

Table 1-2 is the confusion matrix of the CNN model; there are 10804 true negatives (i.e., correctly predicted negatives) and 10976 true positives (i.e., correctly predicted positives) in the classification results.

Table 1-2:

	Predicted 0	Predicted 1
Actual 0	10804	1696
Actual 1	1524	10976

In summary, the overall accuracy of the model was 0.87. The F1-score for both classes was high and balanced, indicating that the model performs well for both positive and negative review.

3.2. Method 2 – Long Short-Term Memory (LSTM)

The LSTM model used in this experiment consists of an input layer, followed by an LSTM layer with 64 units, a dropout rate of 0.2 and a dense output layer with a sigmoid activation function. I used a random search approach to perform hyperparameter tuning for this LSTM model and trained the model using the training set and validated it using a 3-fold cross validation. After running the random search, I obtained the following best parameters for the LSTM model:

- Optimizer: Rmsprop
- Number of filters: 64
- Kernel size: 3
- Recurrent dropout: 0.4
- Activation function: relu

I evaluated the best LSTM model on the test set and obtained the following classification report in Table 2:

Table 2

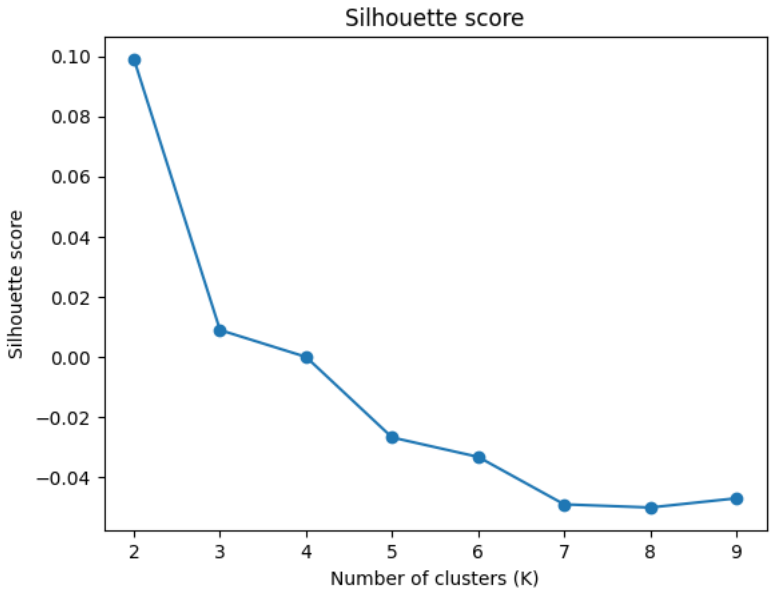
```
782/782 [==============================] - 20s 25ms/step
              precision    recall  f1-score   support

           0       0.87      0.86      0.86     12500
           1       0.86      0.88      0.87     12500

    accuracy                           0.87     25000
   macro avg       0.87      0.87      0.87     25000
weighted avg       0.87      0.87      0.87     25000

LSTM Best Parameters: {'recurrent_dropout': 0.4, 'optimizer': '
```

In summary, the classification report shows that the LSTM model achieved an overall accuracy of 87% on the test set, which seems to be a good performance. The precision and recall values for both classes are above 0.85, indicating that the model is able to predict both positive and negative classes accurately. The weighted average F1 score of 0.87 also indicated well overall performance of the LSTM model.

3.3. Method 3 – K-means Algorithm

For the K-means algorithm, firstly I build a K-means model on a range of different values of K (number of clusters) and select the value of K that results in the highest Silhouette score. The Silhouette score is a measure of how well-defined the clusters are and is used to evaluate the quality of the clustering. The Graph 3-1 shows that when K equals 2, it results in the highest Silhouette score, and the K-means model is fine-tuned to achieve the best clustering performance.

Graph 3-1



Finally, I used Silhouette score to evaluate the best K-means model on making predictions on the test data. Table 3-2 shows the evaluation results:

Table 3-2

Best K	2
Inertia score of best K	2458448209519.679
Silhouette score of best K	0.10738757530348989

- The inertia score indicates the sum of squared distances of all the points within the clusters to their respective centroids. A lower inertia score generally means that the clusters are more compact and better defined.
- Meanwhile, the silhouette score of 0.107 indicates that the clustering is not very well-defined, with some overlapping or scattered points. It is still a positive score, but it is relatively low.

In summary, based on these metrics, the clustering result may not be considered as optimal, and may require further improvement.

3.4. Comparison

Table 4 summarized the results of each method and their respective accuracy scores:

Table 4

Models	Accuracy Score	Silhouette score
CNN	87.12 %	
LSTM	86.62 %	
K-means		0.10738757530348989

Based on the Accuracy Scores and Silhouette Score, it can be seen that the CNN model has the highest accuracy on the test set with an accuracy score of 87.12%, followed closely by the LSTM

model with an accuracy score of 86.62%. The K-means algorithm has the lowest accuracy score with a silhouette score of 0.107, which is significantly lower than the other two deep learning models.

4. Conclusion

Based on the analysis presented in the report, it can be concluded deep learning models, such as CNN and RNN/LSTM, outperform clustering models such as K-means, in terms of text classification and sentiment analysis tasks on movie review datasets.

The CNN mode achieved an accuracy score of 87.12%, which is higher than the LSTM model's score of 86.62%. This indicates that the CNN model was able to better distinguish between positive and negative movie reviews. Additionally, the silhouette score of the K-means model was only 0.107, which indicates that the clustering of the data was not well-performed.

Overall, the results of these experiments suggest that deep learning models are a powerful tool for text classification and sentiment analysis, particularly on large and complex datasets. Further research can be done to explore the potential of these models on other text or image based datasets and real-world applications.

Appendix

Appendix A: Running the Code

To run the code in Google Colab, please follow the steps below:

1. Make sure you have access to Jupyter Notebook and all necessary libraries and packages. You can install them by running ‘pip install -r requirements.txt’ in your terminal or command prompt.
2. Download the Jupyter Notebook containing the full code of this assignment.
3. Open Google Colab
4. Click on File > Upload notebook.
5. Upload the Jupyter Notebook you downloaded in step 2.
6. Follow the instructions within the notebook to run the code.
7. Make sure to save any changes you make to the notebook before closing.

Appendix B: Hardware and Software Environment

The code was developed and tested on a Google Colab notebook using the following hardware and software environment:

- CPU: Intel(R) Xeon(R) CPU @ 2.30GHz
- RAM: 12 GB
- Operating System: Ubuntu 18.04.5 LTS (Bionic Beaver)
- Python version: 3.9.16