SENTIMENT ANALYSIS

Master’s Degree in Data Science

Mining Massive Datasets

Nicole Olivetto

VR481171

Introduction to Sentiment Analysis

Sentiment analysis, also known as opinion mining, is a field at the intersection of natural language processing (NLP) and machine learning, dedicated to understanding and extracting subjective information from textual data. In today's digital age, where vast amounts of data are generated daily through social media, product reviews, and online forums, sentiment analysis plays a fundamental role in gaining insights into public opinion, customer feedback, and market trends. By employing computational techniques, sentiment analysis tries to classify text documents into predefined categories such as positive, negative, or neutral sentiments, enabling businesses, governments, and organizations to make informed decisions, enhance customer satisfaction, and tailor their strategies to better meet the needs and expectations of their stakeholders.

Implementation

In this project, my goal was to explore sentiment analysis using a dataset consisting of 50,000 IMDB movie reviews. Drawing inspiration from a research article that applied sentiment analysis techniques to this exact dataset, my objective was to replicate and further investigate their findings through practical implementation. In the article they applied three distinct neural network architectures - CNN, LSTM, and LSTM-CNN - individually to extract features from the sentences. Then, the extracted features were fed into a multilayer perceptron network for the classification of positive and negative sentiments. The primary objective of the paper was to determine the most suitable deep neural network architectures that yield improved classification results on the IMDb movie reviews dataset. These architectures were implemented using the Keras deep learning framework, which operates as a high-level API built on top of TensorFlow.
Despite the primary objective of the article, I aim to take this exploration a step further. Specifically, I am interested in stress-testing the system to assess its robustness in handling various scenarios and edge cases. By introducing noise, creating dataset imbalances, and testing various batch sizes, I also aimed to uncover weaknesses and vulnerabilities in the models.

Data Preparation

Before proceeding with the Convolutional Neural Network (CNN) implementation, the dataset underwent several steps to ensure its suitability for training. Here's a breakdown of each step, along with the parameters chosen:

Data Loading

The initial step involved loading the IMDB dataset from the CSV, which contains textual reviews along with their corresponding sentiment labels (positive or negative).

Text Preprocessing
To ensure the quality and consistency of the textual data, I applied some preprocessing techniques:

- *HTML Tag Removal*: Utilizing the BeautifulSoup library, HTML tags present in the text were removed, ensuring that only the raw textual content remained intact. This step is very important as HTML tags do not contribute to the semantic understanding of the text.
- *Non-Alphabetic Character Removal*: Regular expressions were used to eliminate non-alphabetic characters from the text. This includes punctuation marks, special symbols, and numeric characters, leaving behind only alphabetic characters.
- *Tokenization and Stopword Removal*: The NLTK library was utilized for tokenization, splitting the preprocessed text into individual words or tokens. A set of stopwords from the English language was also employed to filter out common words with little semantic value. This step ensures that the model focuses on relevant words that contribute meaningfully to the sentiment of the reviews.

Tokenization and Padding
The textual data was tokenized and padded to ensure uniformity in sequence length, which is a prerequisite for model training. First, we set the maximum length of sequences to 100 and specified a vocabulary size of 8000. This ensures that our sequences are uniform in length and our vocabulary is manageable in size. Next, we initialize a `Tokenizer` object from the Keras library, which will facilitate the tokenization process. By specifying `num_words=vocab_size`, we ensure that only the top 8000 most frequent words are retained in our vocabulary. Then, we call the `fit_on_texts` method on our `Tokenizer` object, providing it with the preprocessed textual data. This method scans through the text data, tokenizes each text sample, and updates the internal vocabulary accordingly.
Following the tokenization process, I used Padding to make sure that the sequences had all the same length by padding or truncating them. Firstly, I transformed the tokenized sequences into integers using the `texts_to_sequences` method, saving them in the variable `sequences`. These sequences represented the textual data in a numerical format, with each word mapped to a unique integer. Then, I used the `pad_sequences` function to standardize the sequence length. This function either pads sequences shorter than 100 tokens with zeros or truncates longer sequences to match the specified length of 100 tokens.

Encoding Sentiment Labels
This step converts the sentiment labels from words ('positive' and 'negative') into numbers. It assigns the label 'positive' to the number 1 and the label 'negative' to the number 0. This way, we transform the labels into a binary format, making them suitable for use in machine learning algorithms.

Train-Test Split

Finally, the dataset was split into training and testing sets to evaluate the model's performance accurately. The testing set included 30% of the total data, ensuring a significant amount of data for evaluation while allocating the remaining 70% to the training set.

Convolutional Neural Network

Convolutional Neural Networks (CNNs) are a class of deep learning models widely used for processing structured grid data, including but not limited to visual imagery. These networks are designed to automatically learn hierarchical patterns and features from input data, capturing spatial and temporal dependencies effectively. While CNNs are commonly associated with image processing tasks, such as image classification, object detection, and segmentation, they have also found applications in various domains, including natural language processing (NLP), speech recognition, and time series analysis. CNNs are commonly used for sentiment analysis due to their ability to effectively capture local patterns and features within text data. In sentiment analysis, understanding the sentiment expressed in a piece of text often relies on identifying specific patterns or combinations of words that indicate positive, negative, or neutral sentiment. CNNs excel at this task because they can automatically learn and extract relevant features from text data through convolutional operations.

Implementation

In my implementation, the process begins with embedding words into dense vectors, using the embedding layer. This layer converts input sequences of integer-encoded words into dense vectors of fixed size. The parameters `input_dim`, `output_dim`, and `input_length` determine the vocabulary size, the dimensionality of the embedding space, and the length of input sequences, respectively.

Following embedding, convolutional operations are employed through two convolutional layers. These layers function as feature extractors, using filters to scan input data and identify local patterns. The parameters `filters`, `kernel_size`, and `activation` determine the number of filters, the size of the convolutional window, and the activation function used, respectively. The ReLU activation function (`activation='relu'`) introduces non-linearity, allowing the model to capture complex relationships within the data.

Next, I used some pooling layers to downsample extracted features while retaining essential information. The `pool_size` parameter dictates the size of the pooling window, influencing the degree of downsampling.

The output of convolutional layers is then flattened and is passed through a dense layer, serving as a feature learner. The `units` parameter determines the dimensionality of the output space, with `units=64` signifying the complexity of learned features.

To mitigate overfitting, a dropout layer is incorporated. The `dropout` parameter, set to 0.5, determines the fraction of neurons to drop during training. This regularization technique

encourages the model to learn robust and generalized representations of input data, enhancing its ability to generalize to unseen examples.
Finally, the output layer produces binary classification predictions using a sigmoid activation function. The `units` parameter, set to 1, signifies the single neuron responsible for predicting the probability of input belonging to the positive class.

After defining the model's architecture, the next step involves compiling it, which essentially prepares it for training. This compilation entails selecting suitable optimization algorithms, loss functions, and evaluation metrics. In this case, the Adam optimizer, known for its adaptive learning rates and efficient convergence properties, is used for optimization, as indicated by setting `optimizer='adam'`.
For the loss function, binary cross-entropy is selected. Binary cross-entropy is a common choice for binary classification tasks, where the model aims to predict one of two classes. It quantifies the difference between the predicted probabilities and the actual labels. This selection is indicated by setting `loss=‘binary_crossentropy'`.

Following the training of the CNN model, its performance is evaluated using the test dataset. The model's loss and accuracy on the test data are computed and printed, providing insights into its generalization capabilities. Additionally, the model generates predictions for the test data, which are then thresholded at 0.5 to obtain binary predictions. Various evaluation metrics are calculated to assess the model's performance. These metrics include accuracy, providing an overall measure of correct classifications, as well as a classification report detailing precision, recall, and F1-score for each class. Furthermore, a confusion matrix is generated, visually summarizing the model's predictions against the true labels.

LSTM

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture designed to address the limitations of traditional RNNs in capturing long-range dependencies in sequential data. Introduced by Hochreiter and Schmidhuber in 1997, LSTM networks utilize a unique gating mechanism to selectively store and access information over extended time periods. This allows LSTMs to effectively learn and remember patterns in sequential data, making them well-suited for tasks such as time series prediction, natural language processing, and speech recognition. LSTMs work well for sentiment analysis because they can effectively capture long-range dependencies and sequential patterns in text data. Sentiment analysis often involves understanding the sentiment expressed in a piece of text, which can be influenced by the context and order of words. LSTMs, with their ability to retain and utilize information over extended sequences, are well-suited for this task. They can learn to recognize nuanced patterns in language, such as negations, modifiers, and context shifts, which are crucial for accurately determining sentiment.

Implementation
Just like for the CNN I started with the embedding laye, responsible for converting input sequences of words into dense vectors of fixed size.
Following the embedding layer is a Long Short-Term Memory (LSTM) layer, configured with a certain number of units (`units`), determining the dimensionality of the output space.
Two dense (fully connected) layers are then added to the model architecture. The first dense layer consists of 64 units and employs the Rectified Linear Unit (ReLU) activation function. A dropout layer is incorporated after the first dense layer, with a dropout rate of 0.5, to mitigate overfitting by randomly dropping a fraction of input units during training.
The final dense layer consists of a single unit with a sigmoid activation function, which is, as I previously said, suitable for binary classification tasks. This layer outputs a probability score between 0 and 1, representing the likelihood of the input belonging to the positive class.
The model is compiled using the Adam optimizer and binary cross-entropy is chosen as the loss function. Accuracy is selected as the evaluation metric to assess the model's performance during training.
In the training phase, the `fit()` function is used to train the LSTM model on the provided training dataset. The model is trained over 5 epochs, with a batch size of 128 samples per batch. Additionally, a validation split of 20% (`validation_split=0.2`) is specified, allowing a portion of the training data to be reserved for validation during training. The `callbacks` parameter includes the `early_stopping` callback, which monitors the validation loss and stops training if it fails to improve for a specified number of epochs, thus preventing overfitting.

The evaluation of the LSTM model mirrors that of the CNN model, starting with the computation of loss and accuracy on the test dataset to gauge performance. Next, the model generates binary predictions by thresholding probabilities at 0.5. Accuracy, precision, recall, and F1-score, are computed to assess the model's efficacy and a confusion matrix visually summarizes model predictions against true labels.

LSTM-CNN

The LSTM-CNN architecture represents a significant advancement in deep learning models tailored for natural language processing tasks. It integrates the strengths of two prominent neural network architectures, Long Short-Term Memory (LSTM) and Convolutional Neural Networks (CNN), to provide a robust approach to processing sequential data. LSTM, known for its ability to retain and utilize information over extended sequences, is combined with CNN, which excels in extracting local features and patterns. This amalgamation results in a versatile model capable of capturing both short-term nuances and long-range dependencies within textual data. The LSTM-CNN architecture has garnered attention for its effectiveness across various applications, including sentiment analysis tasks.

Implementation

The input data first passes through an embedding layer and the parameters includes the size of the vocabulary, the dimensionality of the embedding space, and the length of input sequences.
Following the embedding layer, a Bidirectional LSTM layer is added to the model. Bidirectional LSTMs process input sequences in both forward and backward directions, capturing information from past and future contexts. The LSTM layer's units parameter determines the dimensionality of the output space.
Two 1D convolutional layers are then added to the model, which extract local patterns and features from the input sequences using filters of specified sizes and activation functions.
Max-pooling layers are incorporated within the model architecture to downsample the feature maps derived from the convolutional layers.
Following the convolutional layers, the output undergoes a transformation as it is flattened into a one-dimensional array, which makes sure that the convolutional features are arranged in a way that's good for further analysis and understanding.
To facilitate additional feature extraction and transformation, a dense (fully connected) layer with 64 units and Rectified Linear Unit (ReLU) activation is added into the model.
A dropout layer is then introduced in order to systematically deactivate a proportion of input units at random during the training process, which prevents the model from excessively relying on specific features.
The model concludes with a final dense layer, consisting of a single unit and utilizing a sigmoid activation function. This functions as the output layer, supplying a probability score that indicates the likelihood of the input being associated with the positive class.

In the following step the model is compiled using the Adam optimizer, binary cross-entropy loss function, and accuracy as the optimization metric. Moving on to the training phase, the model is trained on the training dataset using the fit method, specifying parameters such as the number of epochs, batch size, and validation split. An early stopping callback is utilized one again to monitor the validation loss and prevent overfitting.

Once training is complete, the model's performance is evaluated on the test dataset. This involves computing the loss and accuracy metrics to quantify the model's predictive performance and generating predictions for the test data, which are then thresholded to obtain binary predictions. Finally, a comprehensive evaluation is conducted, including accuracy, precision, recall, F1-score, classification report, and confusion matrix.

Results

The metrics utilized to evaluate the performance of the sentiment analysis models are the following:

- *Accuracy*: measures the overall correctness of the model's predictions, indicating the proportion of correctly classified instances.

- *Precision*: assesses the model's ability to correctly identify positive and negative instances, representing the ratio of correctly predicted instances to the total predicted instances for a specific class.

- *Recall*: evaluates the model's ability to capture all positive and negative instances, representing the ratio of correctly predicted instances to the total actual instances for a specific class.

- *F1-score*: is the harmonic mean of precision and recall, providing a balance between the two metrics and offering a comprehensive assessment of the model's performance.

Additionally, the *confusion matrix* is a tabular representation that provides a comprehensive breakdown of the model's predictions compared to the actual labels in a classification problem. It consists of rows and columns corresponding to the predicted and actual classes, respectively. Each cell in the matrix represents the count or proportion of instances classified by the model.

Specifically, a confusion matrix for binary classification typically includes four terms:

1. True Positives (TP): Instances correctly predicted as positive by the model.
2. False Positives (FP): Instances incorrectly predicted as positive by the model.
3. True Negatives (TN): Instances correctly predicted as negative by the model.
4. False Negatives (FN): Instances incorrectly predicted as negative by the model.

CNN

Metric	Class 0	Class 1	Total
Precision	0,86	0,88	-
Recall	0,88	0,86	-
F1-score	0,87	0,87	-
Support	7411	7589	-
Accuray	-	-	0,8728

Confusion matrix	Predicted 0	Predicted 1
Actual 0	6531	880
Actual 1	1028	6561

Accuracy: The CNN model achieves an accuracy of 0.8728, indicating that approximately 87.28% of the test instances are correctly classified.

Precision: For class 0 (negative sentiment), the precision is 0.86, indicating that when the model predicts a review as negative, it is correct 86% of the time. For class 1 (positive sentiment), the precision is 0.88, suggesting that when the model predicts a review as positive, it is correct 88% of the time.

Recall: The recall for class 0 is 0.88, indicating that the model correctly identifies 88% of all actual negative reviews. For class 1, the recall is also 0.86, indicating that the model correctly identifies 86% of all actual positive reviews.

F1-score: The F1-score, which balances precision and recall, is approximately 0.87 for both classes.

Confusion Matrix: The CNN model correctly predicts a large number of instances (true positives and true negatives), but there are also instances of misclassifications (false positives and false negatives). Specifically, there are 6531 true negatives, 6561 true positives, 880 false positives, and 1028 false negatives.

Overall, the CNN model demonstrates strong performance in sentiment analysis tasks, with high accuracy and balanced precision and recall.

LSTM

Metric	Class 0	Class 1	Total
Precision	0,90	0,86	-
Recall	0,84	0,91	-
F1-score	0,87	0,88	-
Support	7411	7589	-
Accuray	-	-	0,8773

Confusion matrix	Predicted 0	Predicted 1
Actual 0	6257	1154
Actual 1	686	6903

Accuracy: The LSTM model achieves an accuracy of 0.8773, indicating that approximately 87.73% of the test instances are correctly classified.

Precision: For class 0 (negative sentiment), the precision is 0.90, indicating that when the model predicts a review as negative, it is correct 90% of the time. For class 1 (positive sentiment), the precision is 0.86, suggesting that when the model predicts a review as positive, it is correct 86% of the time.

Recall: The recall for class 0 is 0.84, indicating that the model correctly identifies 84% of all actual negative reviews. For class 1, the recall is 0.91, indicating that the model correctly identifies 91% of all actual positive reviews.

F1-score: The F1-score, is approximately 0.87 for class 0 and 0.88 for class 1.

Confusion Matrix: The LSTM model correctly predicts a large number of instances (true positives and true negatives). Specifically, there are 6257 true negatives, 6903 true positives, 1154 false positives, and 686 false negatives.

The LSTM model shows robust performance in sentiment analysis tasks, with high accuracy and balanced precision and recall, like the CNN model.

LSTM-CNN

Metric	Class 0	Class 1	Total
Precision	0,89	0,87	-
Recall	0,86	0,89	-
F1-score	0,87	0,88	-
Support	7411	7589	-
Accuray	-	-	0,8761

Confusion matrix	Predicted 0	Predicted 1
Actual 0	6379	1032
Actual 1	826	6763

Accuracy: The LSTM-CNN model achieves an accuracy of 0.8761, indicating that approximately 87.61% of the test instances are correctly classified.

Precision: For class 0 (negative sentiment), the precision is 0.89, indicating that when the model predicts a review as negative, it is correct 89% of the time. For class 1 (positive sentiment), the precision is 0.87, suggesting that when the model predicts a review as positive, it is correct 87% of the time.

Recall: The recall for class 0 is 0.86, indicating that the model correctly identifies 86% of all actual negative reviews. For class 1, the recall is 0.89, indicating that the model correctly identifies 89% of all actual positive reviews.

F1-score: The F1-score is approximately 0.87 for class 0 and 0.88 for class 1.

Confusion Matrix: The confusion matrix shows that the LSTM-CNN model correctly predicts a large number of instances. Specifically, there are 6379 true negatives, 6763 true positives, 1032 false positives, and 826 false negatives.

Overall, the LSTM-CNN model demonstrates strong performance in sentiment analysis tasks, similar to the LSTM and CNN models.

Comparison of the results

Comparing the results of the three models we observe:

- Accuracy: The LSTM model performs slightly better than the CNN and LSTM-CNN models, with an accuracy of 0.8773, followed closely by the LSTM-CNN model with an accuracy of 0.8761. The CNN model achieves the lowest accuracy at 0.8728.

- Precision: The LSTM-CNN model achieves the highest precision for class 0 (negative sentiment) at 0.89, while the CNN model achieves the highest precision for class 1 (positive sentiment) at 0.88.

- Recall: The LSTM model achieves the highest recall for class 1 (positive sentiment) at 0.91, while the CNN model achieves the highest recall for class 0 (negative sentiment) at 0.88.

- F1-score: The F1-scores for both classes are similar across all models, with the LSTM and LSTM-CNN models achieving slightly higher scores compared to the CNN model.

In summary, all three models demonstrate competitive performance in sentiment analysis tasks, with slight variations in their strengths and weaknesses across different evaluation metrics. The choice between these models may depend on specific requirements, such as computational efficiency, interpretability, and the importance of precision or recall in the application domain.

Stress-testing

In the field of sentiment analysis, stress-testing assumes a fundamental role in evaluating the reliability and robustness of predictive models under varying conditions. As sentiment analysis models are used to process vast amounts of textual data and make predictions on sentiment, it becomes essential to assess their performance across different scenarios and potential challenges. In this analysis, stress-testing allows us to scrutinize the models' effectiveness in accurately classifying sentiments under extreme circumstances. By subjecting the models to rigorous stress tests, we can identify their limitations, vulnerabilities, and areas for improvement, thereby enhancing their reliability and applicability in real-world scenarios.

Noise

First of all, I tried to stress-test the models by adding noise to the input data during training. By intentionally introducing noise, it's possible to identify potential weaknesses or vulnerabilities in the models. For example, if the performance drops significantly on noisy data compared to clean data, it may indicate that the models are overly sensitive to noise or that they have not learned robust features. By evaluating the models' performance on such data, it becomes possible to see how well they generalize to real-world scenarios where the input may contain noise or variations. If the models demonstrate consistent performance on both clean and noisy data, it indicates that they have learned meaningful and strong representations, which can be indicative of their overall reliability.

In my study I added Gaussian noise to the word embeddings of the input data. The noise addition function `add_noise_to_embeddings` is defined to generate Gaussian noise with a specified factor, which is then added to the word embeddings. The noise factor can be adjusted to control the amount of noise added. After splitting the data into training and testing sets, the noise is applied only to the training data (`noisy_X_train`), while the testing data remains unchanged. Finally, the models (LSTM, CNN, and LSTM-CNN) are defined and trained using the noisy training data, and their performance is evaluated on the original testing data.

I used three different noise factors: 0.01, 0.1, and 1. The reason was to observe how varying levels of noise influence the accuracy and stability of the models.
Each noise factor represents a different level of perturbation applied to the input data. A noise factor of 0.01 introduces minimal noise, while a noise factor of 1 introduces significant noise to the data.

Results

- Noise Factor = 0,01

	Accuracy	Precision (Class 0)	Recall (Class 0)	F1-score (Class 0)	Precision (Class 1)	Recall (Class 1)	F1-score (Class 1)
LSTM	0,8207	0,82	0,82	0,82	0,82	0,83	0,82
CNN	0,8115	0,85	0,75	0,80	0,78	0,87	0,82
LSTM-CNN	0,8178	0,82	0,81	0,81	0,81	0,83	0,82

- Noise Factor = 0,1

	Accuracy	Precision (Class 0)	Recall (Class 0)	F1-score (Class 0)	Precision (Class 1)	Recall (Class 1)	F1-score (Class 1)
LSTM	0,8142	0,83	0,78	0,81	0,80	0,85	0,82
CNN	0,8087	0,78	0,85	0,81	0,84	0,77	0,80
LSTM-CNN	0,8193	0,83	0,80	0,81	0,81	0,84	0,82

- Noise Factor = 1

	Accuracy	Precision (Class 0)	Recall (Class 0)	F1-score (Class 0)	Precision (Class 1)	Recall (Class 1)	F1-score (Class 1)
LSTM	0,7987	0,82	0,76	0,79	0,78	0,83	0,81
CNN	0,7975	0,79	0,81	0,80	0,81	0,79	0,80
LSTM-CNN	0,8045	0,83	0,77	0,79	0,79	0,84	0,81

The three different levels of noise (0.01, 0.1, and 1) affected the models in different ways. For the LSTM model, precision, recall, and F1-score metrics remained relatively stable at a noise factor of 0.01, indicating its resilience to low levels of noise. However, as the noise factor increased to 0.1, there was a slight decline in performance, with a more pronounced degradation observed at a noise factor of 1. Similarly, the CNN model showed consistent performance at a noise factor of 0.01, with a slight decline in precision observed at 0.1. At a noise factor of 1, significant performance degradation was evident across all metrics. Interestingly, the LSTM-CNN model demonstrated a better performance across all noise levels. Despite experiencing some performance decline at a noise factor of 1, the LSTM-CNN model showed relatively better resilience compared to the other models.

Comparison with the original results

The original results showed strong performance across all three models, indicating their effectiveness in classifying text data. However, when introducing noise at three different levels, the performance of all models was affected to varying degrees. At noise level 0.01, the impact on model performance was relatively minimal. While there were slight fluctuations in precision, recall, and accuracy, the overall performance remained relatively stable compared to the original results. As the noise level increased to 0.1, there was a noticeable decline in performance across all models. Precision, recall, and accuracy scores decreased, indicating that the models struggled to classify text data accurately in the presence of higher noise levels.
At noise level 1, the performance degradation was most pronounced. All models experienced significant drops in precision, recall, and accuracy, suggesting that the high level of noise severely impacted their ability to classify data effectively.

Imbalanced Dataset

Introducing imbalance into a dataset can be useful to identify potential weaknesses in the models' performance. When a dataset is imbalanced, it means that one class is significantly more prevalent than the others and this scenario often reflects real-world situations where certain classes occur more frequently than others.
By creating an imbalanced dataset, we can evaluate how well a model generalizes to minority classes, which are typically underrepresented. Models trained on imbalanced data may exhibit biases towards the majority class, leading to poor performance on minority class instances.
Also, imbalanced datasets challenge models to learn meaningful patterns from limited data, forcing them to adapt and prioritize features that discriminate between classes effectively. This process helps reveal the robustness of the model's learned representations and its capacity to handle real-world scenarios where class distributions are skewed.

To create an imbalanced dataset, I filtered out a subset of positive reviews and a smaller subset of negative reviews. Specifically, I sampled 15,000 positive reviews and 10,000 negative reviews randomly from the original dataset. After that, I concatenated these sampled reviews to form a new dataset, resulting in an imbalanced distribution where the number of positive reviews is greater than the number of negative reviews. Finally, I tokenized and padded the text data for model training and evaluation.

Results

	Accuracy	Precision (Class 0)	Recall (Class 0)	F1-score (Class 0)	Precision (Class 1)	Recall (Class 1)	F1-score (Class 1)
LSTM	0,855	0,86	0,76	0,81	0,85	0,92	0,88
CNN	0,864	0,87	0,78	0,82	0,86	0,92	0,89
LSTM-CNN	0,867	0,86	0,79	0,83	0,87	0,92	0,89

Accuracy: The LSTM model achieved an accuracy of approximately 85.5%, the CNN model achieved around 86.4%, and the LSTM-CNN model achieved approximately 86.7%.

Precision: For class 0 (negative sentiment), the LSTM model achieved a precision of about 86%, the CNN model achieved around 87%, and the LSTM-CNN model achieved approximately 86%. For class 1 (positive sentiment), the LSTM model achieved a precision of about 85%, the CNN model achieved around 86%, and the LSTM-CNN model achieved approximately 87%.

Recall: For class 0, the LSTM model achieved a recall of approximately 76%, the CNN model achieved around 78%, and the LSTM-CNN model achieved approximately 79%. For class 1, the LSTM model achieved a recall of approximately 92%, the CNN model achieved around 92%, and the LSTM-CNN model achieved approximately 92%.

F1-score: For class 0, the LSTM model achieved an F1-score of about 81%, the CNN model achieved around 82%, and the LSTM-CNN model achieved approximately 83%. For class 1, the LSTM model achieved an F1-score of approximately 88%, the CNN model achieved around 89%, and the LSTM-CNN model achieved approximately 89%.

Overall, the LSTM-CNN model showed the highest accuracy, precision, recall, and F1-scores across both positive and negative sentiment classes, making it the most effective model for sentiment classification in this scenario.

Comparison with the original results

The LSTM model trained on the imbalanced dataset achieved an accuracy of 85.5%, which is slightly lower than the accuracy achieved on the original balanced dataset. When examining precision, recall, and F1-score, particularly for negative sentiment classification, there is a notable decrease in performance for the imbalanced dataset compared to the original dataset. This suggests that the model trained on the imbalanced dataset struggles more to correctly classify negative sentiment instances. In our imbalanced dataset there are considerably fewer samples of the minority class (negative sentiment) compared to the majority class (positive sentiment). This class imbalance often results in the model becoming biased towards the majority class, which makes it difficult to accurately identify instances belonging to the minority class. With fewer examples to learn from, the model may struggle to discern the patterns and characteristics of the minority class, leading to lower performance in correctly classifying negative sentiment instances. However, it performs reasonably well in classifying positive sentiment instances, with slightly lower precision and recall compared to the original dataset.

Similar to the LSTM model, the CNN model trained on the imbalanced dataset achieved slightly lower accuracy compared to the original balanced dataset. Again, while precision, recall, and F1-score for positive sentiment classification remain relatively consistent between the two datasets, there is a noticeable decrease in performance for negative sentiment classification in the imbalanced dataset. This indicates that the CNN model trained on the imbalanced dataset is less effective in correctly identifying negative sentiment instances.

The LSTM-CNN model also exhibits a slight decrease in performance when trained on the imbalanced dataset compared to the original balanced dataset. Although the accuracy is slightly lower than the original accuracy, the differences in precision, recall, and F1-score, particularly for negative sentiment classification, are more pronounced. Just like for the previous models, it shows lower precision, recall, and F1-score for negative sentiment instances compared to the original dataset.

In summary, while the models trained on the imbalanced dataset still perform reasonably well, there is a noticeable decrease in performance, especially in metrics related to negative sentiment classification (minority class).

BATCH SIZE

After trying to test the robustness and find weaknesses of the models by creating an imbalanced dataset, I attempted to further stress test the systems by varying the batch sizes during training and evaluation. This approach aimed to assess how the models performed under different operating conditions, particularly in terms of robustness, scalability, and efficiency. By subjecting the models to varying batch sizes, we can potentially expose vulnerabilities or limitations that may not be apparent under typical training conditions. For example, the models may exhibit instability or increased error rates when processing data in certain batch size configurations. For stress testing, it's beneficial to explore a wide range of batch sizes to assess how the model responds to varying computational demands and input data sizes, in this case I used 8 (small batch size), 64 (medium batch size), 512 (large batch size).

Batch Size 8

	Accuracy	Precision (Class 0)	Recall (Class 0)	F1-score (Class 0)	Precision (Class 1)	Recall (Class 1)	F1-score (Class 1)
LSTM	0,8687	0,90	0,82	0,81	0,84	0,91	0,88
CNN	0,8698	0,90	0,83	0,82	0,85	0,91	0,88
LSTM-CNN	0,8769	0,89	0,85	0,83	0,86	0,90	0,88

Batch Size 64

	Accuracy	Precision (Class 0)	Recall (Class 0)	F1-score (Class 0)	Precision (Class 1)	Recall (Class 1)	F1-score (Class 1)
LSTM	0,8749	0,89	0,85	0,87	0,86	0,90	0,88
CNN	0,8624	0,82	0,92	0,87	0,91	0,81	0,86
LSTM-CNN	0,8815	0,89	0,87	0,88	0,87	0,90	0,88

Batch Size 512

	Accuracy	Precision (Class 0)	Recall (Class 0)	F1-score (Class 0)	Precision (Class 1)	Recall (Class 1)	F1-score (Class 1)
LSTM	0,8796	0,87	0,88	0,88	0,89	0,88	0,88
CNN	0,8711	0,86	0,89	0,87	0,88	0,86	0,87
LSTM-CNN	0,8773	0,88	0,86	0,87	0,87	0,89	0,88

Accuracy: The accuracy generally increases with larger batch sizes, with the highest accuracy achieved with a batch size of 512.

Precision (Class 0): Precision for class 0 shows a slight decrease with larger batch sizes, although the difference is minor.

Recall (Class 0): Recall for class 0 improves with larger batch sizes, showing a positive correlation between batch size and recall.

F1-score (Class 0): F1-score for class 0 follows a similar pattern to recall, improving with larger batch sizes.

Precision (Class 1): Precision for class 1 slightly increases with larger batch sizes, indicating better classification of positive sentiment.

Recall (Class 1): Recall for class 1 remains relatively consistent across different batch sizes.

F1-score (Class 1): F1-score for class 1 remains consistent or slightly improves with larger batch sizes.

Comparing these results with the original performance metrics revealed both enhancements and deviations. While some models exhibited improved accuracy and precision, particularly with larger batch sizes, others experienced minor fluctuations or slight decreases in performance. In terms of scalability, the results indicate that the models exhibit scalability to some extent, as they generally show improved performance with larger batch sizes. This suggests that the models can handle larger amounts of data efficiently without significant decreases in performance. Regarding efficiency, larger batch sizes can lead to more efficient model trainings, which is reflected in the improved accuracy and performance metrics observed with larger batch sizes. However, there may be decreasing returns in efficiency beyond a certain batch size, as the computational workload increases without substantial performance improvements. Therefore, finding the optimal batch size for a given dataset and model architecture is crucial to balance scalability and efficiency and to ensure optimal model performance under varying conditions.

Comparison with a different dataset

In order to further test the robustness of the models developed and to identify potential weaknesses, an evaluation was conducted using the LSTM, CNN, and LSTM-CNN models on a different dataset. The dataset under consideration contains sentiments for financial news headlines from the perspective of a retail investor. This dataset has two main columns: "Sentiment" and "News Headline". The sentiment labels include negative, neutral, and positive. However all neutral reviews were excluded from the dataset. This preprocessing step resulted in a dataset with 1363 positive reviews and 604 negative reviews.
The methodology replicated the setup used for the IMDb dataset analysis, so preprocessing steps involved tokenization, padding, and vectorization of the financial news headlines. The neutral sentiment reviews were omitted to focus exclusively on discerning between positive and negative sentiments.
One of the main challenges encountered during the analysis was the imbalance in sentiment labels, with a significantly larger number of positive reviews compared to negative reviews. This imbalance posed potential bias issues.

Results:

	Accuracy	Precision (Class 0)	Recall (Class 0)	F1-score (Class 0)	Precision (Class 1)	Recall (Class 1)	F1-score (Class 1)
LSTM	0,7681	0,71	0,48	0,57	0,79	0,91	0,84
CNN	0,7766	0,73	0,49	0,59	0,79	0,91	0,85
LSTM-CNN	0,7445	0,75	0,32	0,45	0,75	0,94	0,83

By applying the LSTM, CNN, and LSTM-CNN models on this new dataset we can see that these findings align with our prior observations during the creation of the imbalanced IMDb dataset. The results affirm our initial expectations, as they demonstrate that the models perform better in detecting positive sentiment compared to negative sentiment, because the models exhibited a bias towards the majority class, which in this case was the positive one.

I chose not to introduce noise into the dataset given the existing complexities encountered by the models in accurately classifying sentiments. While noise addition can be beneficial in some cases, I thought that it might complicate the models' learning process even more, potentially resulting in a decline in performance. Instead, I tried to modify the batch size during training to investigate its effect on model performance and see if it would improve the results but it failed to make a notable impact on the models' effectiveness. This suggests that although batch size plays a role in training dynamics, it may not be the primary determinant of model performance in sentiment analysis tasks. Therefore, exploring alternative strategies or refining the model architecture may be necessary to achieve more significant advancements.
I maintained the other parameters consistent with the configuration described in the article that inspired my study, as my aim was to replicate the experimental setup, test its robustness and find potential weaknesses. However, it would be interesting to explore how variations in parameters, such as the test size, could impact model performance.

Conclusion

In conclusion, this study investigated the robustness and limitations of sentiment analysis models following methodologies inspired by existing research. Given the initial success of the sentiment analysis models with the IMDb dataset I tried to stress test their strength and uncover potential weaknesses. Firstly, I experimented with adding noise to the dataset, which highlighted the models' susceptibility to external disturbances. The outcomes of this test revealed that the models struggled to maintain consistent performance in the presence of noise, indicating a degree of vulnerability to external interference. Then, I deliberately introduced an imbalance in the dataset, which revealed a notable weakness in the models' ability to handle class imbalances effectively. Lastly I explored the effects of varying batch sizes during training and found that these changes lead to both improvements and variations, with some models performing better with larger batch sizes. This highlights the importance of batch size in training and evaluation. To validate these findings I applied them to a different dataset containing sentiments for financial news headlines. The application of the models on this dataset revealed findings consistent with the previous observations made while exploring the imbalanced IMDb dataset. The sentiment analysis models, when confronted with a dataset characterized by imbalanced sentiments, encountered notable challenges in generalization, showing a bias towards the majority class.