UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *<Ergina Dimitraina>*
sdi: <sdi2100031>

Course: *Artificial Intelligence II (M138, M226, M262, M325)*
Semester: *Spring Semester 2025*

Contents

1. Abstract

<Briefly describe what's the task and how you will tackle it.>
In this project, we were asked to extend our previous work, where we implemented a sentiment classifier for tweets using the TF-IDF method and Logistic Regression. In the current phase, we were required to use the Word2Vec method and incorporate a Neural Network to address the problem.

2. Data processing and analysis

2.1. Pre-processing

<In this step, you should describe and comment on the methods that you used for data cleaning and pre-processing. In ML and AI applications, this is the initial and really important step.

For example some data cleaning techniques are: Dropping small sentences; Remove links; Remove list symbols and other uni-codes.>

In this step, we focused on cleaning and preparing the text data before training our machine learning models. Effective pre-processing is crucial for achieving high performance in NLP tasks.

First, we standardized the text by converting all characters to lowercase. We applied a series of corrections to replace common slang, misspellings, and abbreviations (e.g., "u" to "you", "gr8" to "great", "pls" to "please"). We also removed URLs, numbers, punctuation marks, and excessive whitespace.

After the initial cleaning, we tokenized the text by splitting it into words. We removed custom stopwords (such as "to", "the", "I", "and") to reduce noise, helping the model generalize better over different word forms.

For the word representation, we trained a Word2Vec model on the tokenized corpus, generating dense embeddings with a vector size of 100. These embeddings were later averaged to represent each text as a fixed-length vector.

Furthermore, we ensured reproducibility by setting random seeds for Python, NumPy, and PyTorch. We also applied basic data augmentation by experimenting with different hyperparameters during training (such as varying hidden dimensions, learning rates, and epochs).

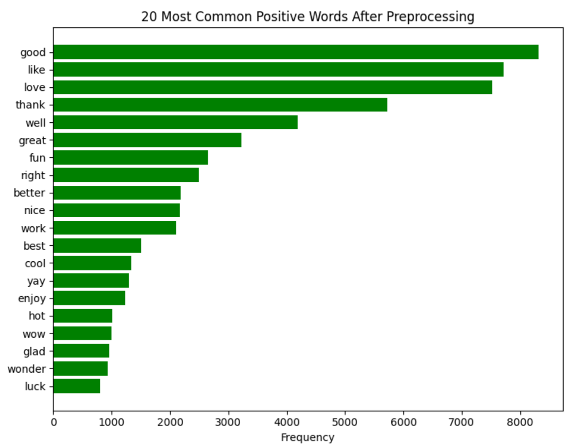
Overall, these pre-processing techniques helped us create cleaner, more meaningful input data, crucial for the performance of the downstream models.

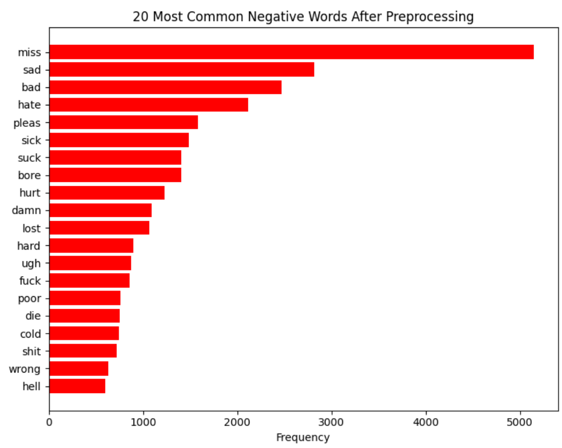
But again, due to the variety and the amount of test data, the preprocessing is not perfect. Some imperfections include words such as "quot", which do not have any meaning but seem to appear quite often in the dataset. I tried to remove the word manually, but again nothing happened (same problem as in the first project).

Also, because the data cleaning is the most important procedure to improve the models accuracy, we experimented with some techniques and tried to improve the previous version.

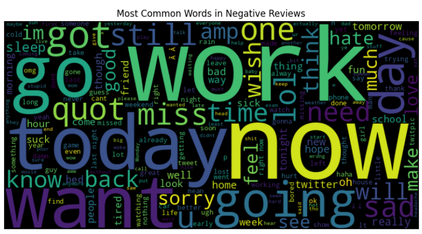
2.2. Analysis

<In this step, you should also try to visualize the data and their statistics (e.g., word clouds, tokens frequency, etc). So the processing should be done in parallel with an analysis. >
Following the analysis made in the first project, we again chose to display the data in these two types of diagramms.
20 Most Common Positive/Negative Words
The first set of diagrams presents the 20 most common positive and 20 most common negative words, both before and after preprocessing. These four diagrams help us observe which words tend to appear more frequently in the dataset prior to preprocessing, potentially leading to less accurate results. After preprocessing, we ensure that the majority of the dataset is 'clean' and contains words that contribute meaningfully to the sentiment classification process.





Word CLouds Word clouds provide a general overview of the words that most significantly influence our model. They serve as an effective visualization tool, helping us identify key terms that impact sentiment classification.



2.3. Data partitioning for train, test and validation

<Describe how you partitioned the dataset and why you selected these ratios>
For the development of the model, we used three datasets: training, validation, and test sets. The initial dataset was partitioned into:
Training Set: 72% of the total data
Validation Set: 8% of the total data

Test Set: 20% of the total data

We first split the original dataset into a training set and a test set, reserving 20% of the data for testing. Next, we further split the training set into a new training set and a validation set, again using stratification. We allocated 10% of the training data to validation. This resulted in approximately 72% of the full dataset for training and 8% for validation.

The reasoning behind these ratios was:

Training Set (72%): A sufficiently large amount of data was required for the model to learn robust patterns.

Validation Set (8%): A smaller, separate validation set allowed us to tune hyperparameters and prevent overfitting without using the test data.

Test Set (20%): A relatively large test set was reserved to fairly and reliably evaluate the final performance of the model after all training and tuning steps.

This partitioning strategy helped ensure that the model could generalize well to unseen data and that the evaluation metrics were meaningful and not biased by model tuning.

2.4. Vectorization

<Explain the technique used for vectorization>
For the vectorization of the text data, we used the Word2Vec model from the Gensim library. Word2Vec is a popular word embedding technique that transforms words into dense, continuous vector representations based on their semantic context.
After preprocessing and tokenizing the text data, we trained a custom Word2Vec model with the following parameters:Vector size: 100 dimensions,Window size: 5 words,Minimum word count: 2 (words appearing less than twice were ignored) and Architecture: Skip-gram (sg=1)
Once the Word2Vec model was trained, each sentence was converted into a vector by averaging the embeddings of all the words in the sentence. This approach enabled us to represent text inputs as fixed-size numerical vectors, making them suitable for training a simple feedforward neural network classifier.

3. Algorithms and Experiments

3.1. Experiments

<Describe how you faced this problem. For example, you can start by describing a first brute-force run and afterwords showcase techniques that you experimented with. **Caution:** we want/need to see your experiments here either they increased or decreased scores. At the same time you should comment and try to explain why an experiment failed or succeeded. You can also provide plots (e.g., ROC curves, Learning-curves, Confusion matrices, etc) showing the results of your experiment. Some tech-

niques you can try for experiments are cross-validation, data regularization, dimension reduction, batch/partition size configuration, data pre-processing from 2.1, gradient descent>

To tackle the sentiment classification task, we initially performed a brute-force approach by training a basic feedforward neural network using default hyperparameters. This served as a baseline to evaluate later improvements. The network used Word2Vec embeddings averaged over each text input as features.

Several experiments were conducted to optimize the model:

TextBlob

TextBlob is a simple and user-friendly Python library that facilitates various natural language processing (NLP) tasks, particularly in the preprocessing stage. It offers a range of convenient methods for cleaning and preparing textual data, such as tokenization, part-of-speech tagging, noun phrase extraction, sentiment analysis, and text correction. These built-in features simplify the preprocessing pipeline and make it more efficient, especially for users who are new to NLP or need to implement quick prototypes.

Hidden Dimensions: We experimented with different hidden layer sizes (100, 120, 130, 140, 150). We observed that increasing the hidden size up to around 140-150 improved performance, but further increases caused overfitting without significant accuracy gains. Based on validation accuracy, we selected a hidden dimension of 150.

Learning Rate Tuning: Different learning rates (0.01, 0.001, 0.0005) were tested. A learning rate of 0.01 produced the best trade-off between convergence speed and stability. Lower learning rates led to slower learning without clear performance benefits, accuracy was decreased to 0.5.

Epochs: We ran experiments with 20, 30, 40, 50 and 60 epochs. Performance steadily improved up to around 50 epochs, beyond which the model started to overfit. Therefore, we selected 60 epochs as the best setting.

Data Regularization: To address overfitting, we added a dropout layer with a rate of 0.3 between hidden layers. But this significantly decreased validation accuracy and generalization, so I decided to keep this part commented in the code and not use it.

Early Stopping: Although early stopping was initially considered, in the final version we opted to manually monitor validation accuracy across epochs instead, as validation curves were relatively stable after 60 epochs.

3.1.1. Table of trials. In this section I will display the results of each experiment with the diffent parameters.

Hidden Dim: 100

```
Epoch 20/20 | Loss: 0.6672 | Val Accuracy: 0.7038
Learning Rate: 0.001
Hidden Dim: 100
Epochs: 20
Test Accuracy: 0.7066
Classification Report:
              precision    recall  f1-score   support

           0     0.6663    0.8277    0.7383     14839
           1     0.7726    0.5856    0.6662     14839

    accuracy                         0.7066     29678
   macro avg     0.7195    0.7066    0.7023     29678
weighted avg     0.7195    0.7066    0.7023     29678
```

```
Epoch 40/40 | Loss: 0.6244 | Val Accuracy: 0.7219
Learning Rate: 0.001
Hidden Dim: 100
Epochs: 40
Test Accuracy: 0.7261
Classification Report:
              precision    recall  f1-score   support

           0     0.7163    0.7488    0.7322     14839
           1     0.7369    0.7035    0.7198     14839

    accuracy                         0.7261     29678
   macro avg     0.7266    0.7261    0.7260     29678
weighted avg     0.7266    0.7261    0.7260     29678
```

```
Epoch 50/50 | Loss: 0.6014 | Val Accuracy: 0.7199
Learning Rate: 0.001
Hidden Dim: 100
Epochs: 50
Test Accuracy: 0.7213
Classification Report:
              precision    recall  f1-score   support

           0     0.7139    0.7387    0.7261     14839
           1     0.7293    0.7039    0.7164     14839

    accuracy                         0.7213     29678
   macro avg     0.7216    0.7213    0.7212     29678
weighted avg     0.7216    0.7213    0.7212     29678
```

```
Epoch 60/60 | Loss: 0.5895 | Val Accuracy: 0.7316
Learning Rate: 0.001
Hidden Dim: 100
Epochs: 60
Test Accuracy: 0.7271
Classification Report:
              precision    recall  f1-score   support

           0     0.7255    0.7306    0.7280     14839
           1     0.7287    0.7236    0.7262     14839

    accuracy                         0.7271     29678
   macro avg     0.7271    0.7271    0.7271     29678
weighted avg     0.7271    0.7271    0.7271     29678
```

Hidden Dim: 120

```
Epoch 20/20 | Loss: 0.6649 | Val Accuracy: 0.7067
Learning Rate: 0.001
Hidden Dim: 120
Epochs: 20
Test Accuracy: 0.7057
Classification Report:
              precision    recall  f1-score   support

           0     0.6758    0.7906    0.7287     14839
           1     0.7478    0.6207    0.6784     14839

    accuracy                         0.7057     29678
   macro avg     0.7118    0.7057    0.7035     29678
weighted avg     0.7118    0.7057    0.7035     29678
```

```
Epoch 40/40 | Loss: 0.6154 | Val Accuracy: 0.7249
Learning Rate: 0.001
Hidden Dim: 120
Epochs: 40
Test Accuracy: 0.7219
Classification Report:
              precision    recall  f1-score   support

           0     0.7175    0.7321    0.7247     14839
           1     0.7265    0.7118    0.7191     14839

    accuracy                         0.7219     29678
   macro avg     0.7220    0.7219    0.7219     29678
weighted avg     0.7220    0.7219    0.7219     29678
```

```
Epoch 50/50 | Loss: 0.5963 | Val Accuracy: 0.7226
Learning Rate: 0.001
Hidden Dim: 120
Epochs: 50
Test Accuracy: 0.7241
Classification Report:
              precision    recall  f1-score   support

           0     0.7214    0.7301    0.7257     14839
           1     0.7268    0.7180    0.7224     14839

    accuracy                         0.7241     29678
   macro avg     0.7241    0.7241    0.7241     29678
weighted avg     0.7241    0.7241    0.7241     29678
```

```
Epoch 60/60 | Loss: 0.5823 | Val Accuracy: 0.7302
Learning Rate: 0.001
Hidden Dim: 120
Epochs: 60
Test Accuracy: 0.7289
Classification Report:
              precision    recall  f1-score   support

           0     0.7284    0.7299    0.7292     14839
           1     0.7294    0.7279    0.7286     14839

    accuracy                         0.7289     29678
   macro avg     0.7289    0.7289    0.7289     29678
weighted avg     0.7289    0.7289    0.7289     29678
```

Hidden Dim: 130

```
Epoch 20/20 | Loss: 0.6644 | Val Accuracy: 0.7095
Learning Rate: 0.001
Hidden Dim: 130
Epochs: 20
Test Accuracy: 0.7082
Classification Report:
              precision    recall  f1-score   support

           0     0.6738    0.8070    0.7344     14839
           1     0.7594    0.6093    0.6762     14839

    accuracy                         0.7082     29678
   macro avg     0.7166    0.7082    0.7053     29678
weighted avg     0.7166    0.7082    0.7053     29678
```

```
Epoch 40/40 | Loss: 0.6124 | Val Accuracy: 0.7243
Learning Rate: 0.001
Hidden Dim: 130
Epochs: 40
Test Accuracy: 0.7246
Classification Report:
              precision    recall  f1-score   support

           0     0.7165    0.7433    0.7297     14839
           1     0.7333    0.7059    0.7194     14839

    accuracy                         0.7246     29678
   macro avg     0.7249    0.7246    0.7245     29678
weighted avg     0.7249    0.7246    0.7245     29678
```

```
Epoch 50/50 | Loss: 0.5910 | Val Accuracy: 0.7308
Learning Rate: 0.001
Hidden Dim: 130
Epochs: 50
Test Accuracy: 0.7337
Classification Report:
              precision    recall  f1-score   support

           0     0.7314    0.7387    0.7351     14839
           1     0.7361    0.7288    0.7324     14839

    accuracy                         0.7337     29678
   macro avg     0.7338    0.7337    0.7337     29678
weighted avg     0.7338    0.7337    0.7337     29678
```

```
Epoch 60/60 | Loss: 0.5817 | Val Accuracy: 0.7306
Learning Rate: 0.001
Hidden Dim: 130
Epochs: 60
Test Accuracy: 0.7325
Classification Report:
              precision    recall  f1-score   support

           0     0.7318    0.7339    0.7328     14839
           1     0.7331    0.7310    0.7321     14839

    accuracy                         0.7325     29678
   macro avg     0.7325    0.7325    0.7325     29678
weighted avg     0.7325    0.7325    0.7325     29678
```

Hidden Dim: 140

```
Epoch 20/20 | Loss: 0.6625 | Val Accuracy: 0.7096
Learning Rate: 0.001
Hidden Dim: 140
Epochs: 20
Test Accuracy: 0.7090
Classification Report:
              precision    recall  f1-score   support

           0     0.6804    0.7884    0.7304     14839
           1     0.7485    0.6296    0.6839     14839

    accuracy                         0.7090     29678
   macro avg     0.7144    0.7090    0.7072     29678
weighted avg     0.7144    0.7090    0.7072     29678
```

```
Epoch 40/40 | Loss: 0.6116 | Val Accuracy: 0.7218
Learning Rate: 0.001
Hidden Dim: 140
Epochs: 40
Test Accuracy: 0.7207
Classification Report:
              precision    recall  f1-score   support

           0     0.7148    0.7346    0.7245     14839
           1     0.7270    0.7069    0.7168     14839

    accuracy                         0.7207     29678
   macro avg     0.7209    0.7207    0.7207     29678
weighted avg     0.7209    0.7207    0.7207     29678
```

```
Epoch 50/50 | Loss: 0.5929 | Val Accuracy: 0.7269
Learning Rate: 0.001
Hidden Dim: 140
Epochs: 50
Test Accuracy: 0.7255
Classification Report:
              precision    recall  f1-score   support

           0     0.7228    0.7315    0.7272     14839
           1     0.7283    0.7195    0.7239     14839

    accuracy                         0.7255     29678
   macro avg     0.7256    0.7255    0.7255     29678
weighted avg     0.7256    0.7255    0.7255     29678
```

```
Epoch 60/60 | Loss: 0.5809 | Val Accuracy: 0.7295
Learning Rate: 0.001
Hidden Dim: 140
Epochs: 60
Test Accuracy: 0.7331
Classification Report:
              precision    recall  f1-score   support

           0     0.7311    0.7374    0.7342     14839
           1     0.7351    0.7288    0.7319     14839

    accuracy                         0.7331     29678
   macro avg     0.7331    0.7331    0.7331     29678
weighted avg     0.7331    0.7331    0.7331     29678
```

Hidden Dim: 150

```
Epoch 20/20 | Loss: 0.6538 | Val Accuracy: 0.7090
Learning Rate: 0.001
Hidden Dim: 150
Epochs: 20
Test Accuracy: 0.7098
Classification Report:
              precision    recall  f1-score   support

           0     0.6926    0.7544    0.7222     14839
           1     0.7304    0.6651    0.6962     14839

    accuracy                         0.7098     29678
   macro avg     0.7115    0.7098    0.7092     29678
weighted avg     0.7115    0.7098    0.7092     29678
```

```
Epoch 40/40 | Loss: 0.6059 | Val Accuracy: 0.7226
Learning Rate: 0.001
Hidden Dim: 150
Epochs: 40
Test Accuracy: 0.7234
Classification Report:
              precision    recall  f1-score   support

           0     0.7149    0.7433    0.7288     14839
           1     0.7327    0.7036    0.7178     14839

    accuracy                         0.7234     29678
   macro avg     0.7238    0.7234    0.7233     29678
weighted avg     0.7238    0.7234    0.7233     29678
```

```
Epoch 50/50 | Loss: 0.5899 | Val Accuracy: 0.7269
Learning Rate: 0.001
Hidden Dim: 150
Epochs: 50
Test Accuracy: 0.7283
Classification Report:
              precision    recall  f1-score   support

           0     0.7242    0.7374    0.7308     14839
           1     0.7326    0.7192    0.7258     14839

    accuracy                         0.7283     29678
   macro avg     0.7284    0.7283    0.7283     29678
weighted avg     0.7284    0.7283    0.7283     29678
```

```
Epoch 60/60 | Loss: 0.5135 | Val Accuracy: 0.7465
Learning Rate: 0.01
Hidden Dim: 150
Epochs: 60
Test Accuracy: 0.7528
Classification Report:
              precision    recall  f1-score   support

           0     0.7512    0.7560    0.7536     14839
           1     0.7545    0.7496    0.7521     14839

    accuracy                         0.7528     29678
   macro avg     0.7529    0.7528    0.7528     29678
weighted avg     0.7529    0.7528    0.7528     29678
```

3.2. Hyper-parameter tuning

<Describe the results and how you configured the model. What happens with under-over-fitting??>
In order to improve model performance, extensive hyper-parameter was conducted. The parameters we experimented with included:

Hidden layer dimensions: 100, 120, 130, 140, 150, 160
Learning rates: 0.01, 0.001, 0.0005
Number of epochs: 20, 30, 40, 50, 60

Through systematic experimentation, the best results were achieved with a hidden dimension of 150, a learning rate of 0.01, and training for 60 epochs.

During the experiments, we observed:
Underfitting when using very small hidden dimensions (e.g., 64) or very low learning rates (e.g., 0.0005), leading to poor training and validation performance.
Overfitting when training for too many hidden dimensions with less epochs, where validation accuracy would start to decline even though training loss kept decreasing.
The final model showed stable learning behavior, with a validation accuracy that consistently improved until around epoch 60.

3.3. Optimization techniques

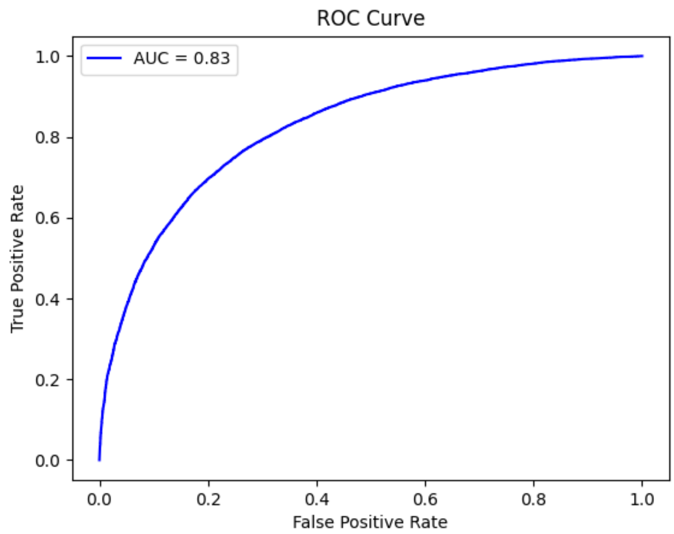
<Describe the optimization techniques you tried. Like optimization frameworks you used.>
For this section of the project we tried to optimize the model not only by experimenting with different parameters but also by incorporating some other techniques, such as early stopping and dropout. Early stopping was implemented, but when used the accuracy of the model decreased in a very important amount.(from 75% to 50%) which tells us

3.4. Evaluation

<How will you evaluate the predictions? Detail and explain the scores used (what's fscore?). Provide the results in a matrix/plots>
<Provide and comment diagrams and curves>
To evaluate the performance of our classification model, we employed a variety of metrics commonly used in natural language processing tasks, including accuracy, precision, recall, and the F1-score. These metrics provide complementary insights into how well the model generalizes and handles class imbalances. The model achieved a final test accuracy of 75.28%, with nearly balanced performance across both classes. This is a good result, indicating that the model performs consistently and does not significantly favor one class over the other.

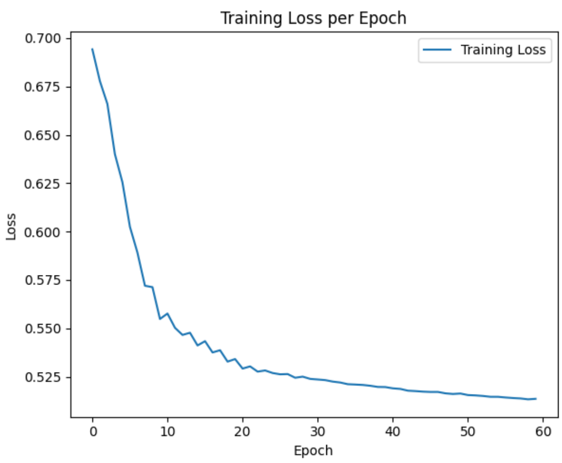
Additionally, the learning curve shows convergence over 60 training epochs, with a final training loss of 0.5135 and a validation accuracy of 74.65%, which is close to the test performance, suggesting good generalization without overfitting.

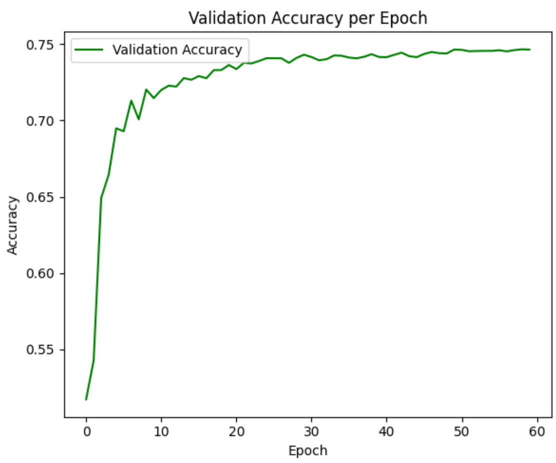
3.4.1. ROC curve.



The Roc curve illustrates the performance of the classification model across different threshold settings by plotting the True Positive Rate against the False Positive Rate. The curve depicted in the figure demonstrates a strong performance, with the model achieving an AUC of 0.83. This AUC score indicates that the model has a good ability to distinguish between the positive and negative classes.

3.4.2. Learning Curve.





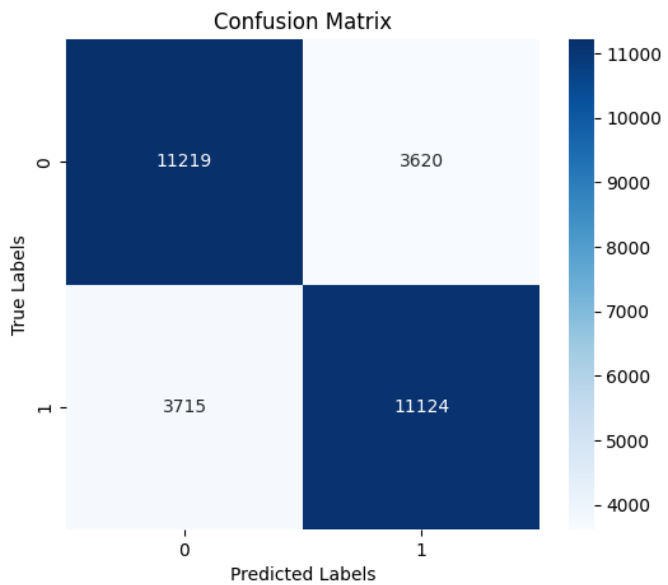
Training Loss and Validation Accuracy Analysis:

The Training Loss plot demonstrates a steady and continuous decrease throughout the epochs. Initially, the loss drops rapidly, which is expected as the model quickly learns basic patterns in the data. After the first few epochs, the rate of decrease becomes slower, indicating that the model is fine-tuning its learning. The smooth and consistent downward trend in training loss suggests that the model is effectively minimizing error without signs of instability or divergence.

The Validation Accuracy plot shows a gradual and overall consistent increase across epochs. Although there are some fluctuations during the initial epochs, these stabilize relatively quickly, and the validation accuracy improves steadily from around 70% to above 74%. The absence of significant drops or volatility after the early stages indicates that the model generalizes well to unseen data and that there are no major signs of overfitting.

Overall, both plots suggest that the training process was successful. The model learned effectively without overfitting, as evidenced by the simultaneous decrease in training loss and increase in validation accuracy over time.

3.4.3. Confusion matrix.



4. Results and Overall Analysis

4.1. Results Analysis

<Comment your results so far. Is this a good/bad performance? What was expected? Could you do more experiments? And if yes what would you try?>
In the current experiments, the SentimentClassifier model achieved a test accuracy of approximately 0.75%, along with reasonable precision, recall, and F1-scores according to the classification report.

Overall, this performance is moderately good.However, there is still room for improvement. The confusion matrix and ROC-AUC analysis showed that while the model distinguishes between positive and negative sentiments reasonably well,but sometimes still makes a noticeable amount of errors.

<Provide and comment diagrams and curves>

4.1.1. Best trial. <Showcase best trial>

```
Epoch 60/60 | Loss: 0.5135 | Val Accuracy: 0.7465
Learning Rate: 0.01
Hidden Dim: 150
Epochs: 60
Test Accuracy: 0.7528
Classification Report:
              precision    recall  f1-score   support

           0     0.7512    0.7560    0.7536     14839
           1     0.7545    0.7496    0.7521     14839

    accuracy                         0.7528     29678
   macro avg     0.7529    0.7528    0.7528     29678
weighted avg     0.7529    0.7528    0.7528     29678
```

4.2. Comparison with the first project

<Use only for projects 2,3,4>
<Comment the results. Why the results are better/worse/the same?>

4.3. Comparison with the second project

<Use only for projects 3,4>
<Comment the results. Why the results are better/worse/the same?>

4.4. Comparison with the third project

<Use only for project 4>
<Comment the results. Why the results are better/worse/the same?>

5. Bibliography

References

[1] Steven Loria. Textblob: Simplified text processing. https://textblob.readthedocs.io/en/dev/, 2013. Accessed: 2025-04-30.

[2] Joaquin Vanschoren. Lecture 6. neural networks. https://ml-course.github.io/master/notebooks/06%20-%20Neural%20Networks.html. Accessed: 2025-04-30.

[1]. [2]