**CSE 404 Project Paper : Sentiments of IMDB Reviews through PLA, RNN & BERT**

**Raduan Moustafhim, Liam McNulty, Ben Blachard, Jeet Jhaveri, Ndiaga Diouf**

**Abstract**

This project explores sentiment analysis of IMDB reviews using a Perceptron Learning Algorithm as a baseline model as a baseline and looks for improved performance using a Recurrent Neural Network. The project also explores a preliminary review of the BERT model for sentiment analysis. Our model should be able to take in a review and determine if it is good or bad based on sentiment analysis. Our goal is to compare the effectiveness of these models using accuracy, f1 score, precision, and recall score. Our models highlight the advantages of deep learning approaches over traditional algorithms. Our models, both Peceptron and RNN, showcased high scores with accuracy, precision, recall score, and F1 score being over 80%, but the BERT model was able to outperform them with the accuracy, precision, recall, and F1 score exceeding 90%.

**1 Introduction**

Sentiments are an integral part of communication. They influence how we as humans view, understand, and react to certain events. With modern-day technology becoming more accessible, millions of terabytes of data are generated daily. From social posts, online forums, comment sections, and reviews, all these types of content contain some form of sentient material. This same data could be analyzed and provide businesses with insights revolving around customer experience, engagement, and overall feedback. With said insights, businesses could improve their product’s efficiency and boost customer satisfaction. Understanding this, our project aims to take advantage of the concept and develop a sentiment-based analysis model for IMDB movie reviews. We chose movie reviews as they are heavily based on a consumer’s perspective/opinion. As for why IMDB, their platform is the gold standard for hosting user-generated reviews and as a result they have a well-structured and varied dataset to train our model.

By the end of this project, our goal is to highlight patterns within consumer sentiment and certain content, evaluate and compare different models, and add on to the ongoing development of high performance sentiment analysis models/techniques.

**2 Related Works**

“A Probabilistic Analysis of the Rocchio Algorithm with TF-IDF for Text Categorization” by Thorsten Joachims provides an explanation for the text vectorizing method that our baseline perceptron model will use. The method is a way to represent a given text as a vector of word weights. Those weights are calculated using term frequency and document frequency. Term frequency is the number of times a word occurs in a text and document frequency is the number of texts that the word appears in. By using term frequency multiplied by inverse document frequency we are able to include the importance of a word to a body of texts that a specific text is found in and the importance of a word to the specific text. Although this model does not capture the context of words in terms of where they appear in the text it is still a highly effective way to vectorize text and combined with perceptron can have impressive performance. We will hope to identify a model that can perform better than this by interpreting the sequential context of words.

“Effective Use of Word Order for Text Categorization with Convolutional Neural Networks” by Rie Johnson and Tong Zhang is an exploration of CNN text categorization capabilities. Low-dimensional word vectors are avoided as opposed to high-dimensional text data for learning on small text regions for classification. Applying multiple convolutional layers also can improve accuracy. This work demonstrates the effectiveness of the CNN text categorization approach when compared to the strongest alternative models.

“Author-Specific Sentiment Aggregation for Polarity Prediction of Reviews” by Subhabrata Mukherjee and Sachindra Joshi takes a different approach in which a Phrase Annotated Author Specific Sentiment Ontology Tree (PASOT) is constructed in order to classify the polarity of IMDB reviews, much like this project intends to do. This approach proved to be more accurate than supervised classification using Support Vector Machines with a 7.55% improvement in accuracy.

All of these previous works differ from this project in that we intend to use a Recurrent Neural Network (RNN) with a Perceptron Learning Algorithm (PLA) as our baseline model. This method will improve upon these works as RNN can capture sequential dependencies which allows the model to understand the context of the data. This is a big advantage for this task as the model would be able to correctly classify things such as negations (difference between “I like” and “I don’t like”), unlike the CNN which detects local features. RNN is also much more flexible than PASOT as it is data driven rather than rule based, allowing the model to adapt to different writing styles.

Continuing to build upon advancements in deep learning in relation to sentiment analysis, Nkhata (2025) explores the integration of a Bidirectional Encoder Representations from Transformers (BERT) model along with a Bidirectional LSTM (BiLSTM) to enhance fine-grained sentiment classification of movie reviews. Specifically, they use a pre-trained BERT model and use a BiLSTM layer to make improvements. Specifically they want to improve the capturing of contextual information to improve the model. Their study was able to produce a binary classification accuracy of 97.67% on the IMDB dataset, which surpassed some of the previous best models. This study aligns with our project's usage of BERT’s contextual understanding capabilities.

In our project's implementation we utilize the “bert-base-uncased” model using the Hugging Face Transformers library, and use the Trainer API for more efficient training and evaluation. We focus specifically on binary sentiment classification of these movie reviews, and Nkhata and their partners take it a step further using a BiLSTM layer. Their usage of that layer is designed to more specifically model sequential relationships. They also do far more preprocessing, and they delved further into labels beyond positive and negative, unlike our model which remained as a binary classifier which allowed us to more directly compare success with our other models. Even though they developed a more advanced model than we used, the success of their approach shows how beneficial using a transformer based model such as BERT can be.

**3 Dataset/Methodology**

In this project we will be taking a dataset of IMDB movie reviews which are classified into separate classes for positive and negative reviews. These strings of text range from 52 characters to 13704 characters. We will preprocess this data by removing duplicate data and convert these strings into a series of vectors to find the frequency of highly important words in relation to if the review is positive or negative. Basically, It will find the words used at the highest frequency which are not commonly found in positive and negative reviews.

This will be accomplished by using the Term Frequency-Inverse Document Frequency (TF-IDF) Vectorizer. Unlike a simple frequency based vectorizer, TF-IDF minimizes the importance of common words (“the”, “is”, etc.) and gives more weight to words which are more meaningful that show more importance in classification. The Vectorizer converts the movie review texts into very long vectors of features (around 75,000 in this case) where each column represents a word in the dataset, with the value being the TF-IDF score for that word in the review.

We will then split this data into training and testing data, and train a perceptron model to attempt to classify the reviews into positive or negative reviews at as high of a success rate as possible. The Perceptron Learning Algorithm is a supervised learning algorithm which is used for classification. It will use the feature vectors to form a perceptron boundary line to attempt to separate the positive and negative reviews. Our boundary line (the Perceptron Boundary Line) can be updated to handle any misclassified points using the Perceptron Update Rule. The perceptron operates by finding a weighted sum of input features, which are the TF-IDF values for the words in each review, and passes it through a sign function to attempt to classify the review as positive or negative. Over a lot of iterations, the perceptron continues to refine the decision boundary to maximize accuracy.

When the perceptron model was running we were able to determine the optimal set of weights for our features. This means that each word has a weight of effectiveness. Extracting these words from our dataset we will see whether certain words have a positive or negative connotation. In figure 3, we explore which words had the highest “positive score” and which words had the highest “negative score” This was done by arranging the features by their weight and extracting the highest and lowest performers. This made it so that reviews that included words like “excellent”, “great”, and “wonderful” are likely positive reviews while reviews that included words like “worst”, “waste”, and “aweful” are likely to come from negative reviews. The perceptron model worked by analyzing the connotation of words in a review. This is not necessarily the case for the BERT model and the RNN model, because they determine their own features unlike the Perceptron Learning model.

Neural Networks are Machine Learning Models that allow us to mimic complex functions that human beings make. It allows us to identify patterns by connecting data with weights and biases to generate data. The Neural Network for our purposes is the Recurrent Neural Networks (RNN). This is an ideal as it allows us to analyze data with context. Rather than looking at one word, we are able to hold the information of the sentence. Unlike our simpler PLA algorithm, our data will feed itself back into our model after each step. Each unit holds a “hidden state” that holds the past words. This data is unfolded for analysis, allowing us to handle greater context. This allows RNN to use feedback loops to help us analyze our language formatted data.

We use a tokenizer to process our RNN data. This text-based data is translated into sequences of integers. These sequences are then padded or truncated to ensure they all have a uniform length. Similar to the Perceptron Learning Algorithm, we now have very long vectors of features where each column represents a word in the database. For testing purposes. Our dataset is split into a training and testing dataset.

Our initial RNN model has 5 layers. An embedded layer, a LSTM (Long Short Term Memory) layer, a dropout layer, a dense layer, and another dense layer that gives us our output. The embedding layer converts our tokenized words into dense vectors of size 16. We have our LTSM layer of 32 units to help understand long term dependencies of our texts. The dropout layer reduces our neurons (by 20%) to tackle overfitting. The next layer uses ReLu to process our data with nonlinear transformations. Our output layer uses sigmoid to give us a singular output that represents its probability. We compile with the Adam optimizer and use the binary\_crossentropy so that we have an adaptive learning environment that can be applied on binary classification.

In order to train the initial RNN model we fit the model to our training data, running on 20 epochs and using a batch size of 2. This effort aims to maximize accuracy while minimizing the loss.

Our final RNN model has 8 layers. An embedded layer, a dropout layer, a bidirectional LSTM layer, another dropout layer, another LSTM layer, a third dropout layer, a dense layer, and another dense layer that gives us our output. Unlike the initial model, the final model introduced two additional dropout layers each at 40% to better repel against overfitting. These dropout layers were placed between the embedding layer, bidirectional LSTM layer, LSTM layer, and the first dense layer. For our final model a bidirectional LSTM layer was introduced such that the input sequence is read in both directions to obtain more context. Both LSTM layers are 32 units to help understand long term dependencies of our texts like the initial model, however unlike the initial model the final RNN applies L2 regularization of 0.001 to both LSTM layers as a means to prevent overfitting. Both dense layers remain the same from the initial model. The final RNN model continues to use the Adam optimizer to compile with the addition of a learning rate of 0.0001 in order to prevent overfitting.

In order to train the final RNN model we fit the model to our training data, running on 20 epochs and using a batch size of 64. The increased batch size may have led to a worse accuracy and issues with overfitting, however the vastly improved run time allowed us to experiment more with the model with far greater ease.

Both of these models will be evaluated using the precision, recall, accuracy, and F1 score. We are able to calculate our accuracy, precision, recall score, and F1 score through functionality within sklearn. Accuracy measures out of all predictions how many are correct. Precision measures how many of the predicted positive cases are actually positive. Recall measures how many actual positives are correctly predicted. F1 score is an equation which is a balance between precision and recall, and works to capture false positives and false negatives.

**4 Experimental Analysis**

As we designed these models the majority of our experimentation was conducted on our RNN model. For our PLA and BERT implementations little experimentation was needed to improve the models to reach the scores mentioned in our results. However, our initial implementation of the RNN was unable to surpass our baseline model. We conducted a large amount of experimentation to find ways to improve the RNN. These experiments included methodically adjusting variables in our model and observing consequenting changes in performance metrics. The variables that we experimented with included batch size, number of epochs, LSTM parameter size, directional LSTM implementation, activation functions, early stopping, max words, regularization, learning rate, and drop out in various layers. By adjusting batch size we hoped that our model would be able to generalize better despite increasing training duration. Through changing the number of epochs we hoped we could decrease overfitting, but instead found it difficult to avoid both underfitting and overfitting at once so we moved on to alternatives to remedy our overfitting issue. Other variables we experimented with were LSTM parameter size and a directional LSTM implementation, but as expected neither could prevent overfitting of the model as a whole. We also tried changing activation functions and although a good choice of activation is essential for non linear data, changing from one non linear activation to another did little to improve our model. Adjusting max words may have been a possible way to decrease complexity, but it did not solve our primary issue. Finally, we did significant experimentation with learning rate, dropout, and regularization since these are all effective ways to improve generalization. Through this experimentation we found that the variable that led to the most significant improvement was increasing the regularization rate. With this change performance metrics had the following improvements: accuracy increased from 82.4% to 85.02%, precision increased from 82.4% to 83.6%, recall increased from 85.5% to 87.1%, and F1 score increased from 82.4% to 85.3%. Other variable adjustments had negligible positive impact on performance metrics for the RNN model or were detrimental.

**5 Results**

After training and testing both the PLA and RNN models, we evaluated the model’s performance by calculating the accuracy, precision, recall, and F1 score. The resulting scores for the PLA model (figure 1) is as follows: Accuracy: 85.1%, Precision: 84.8%, Recall: 85.5%, F1 score: 85.2%. As shown by the visual representation of the evaluation metrics in figure 1, all metrics come out as about 85%, demonstrating that the model is well balanced and a good generalization of the data that can reliably classify that data. The resulting scores for our initial implementation of the RNN model (figure 2) is as follows: Accuracy: 82.4%, Precision: 82.4%, Recall: 85.5%, F1 score: 82.4%. As shown by the visual representation in figure 2, all metrics come out to be around the same, with accuracy, precision, and F1 score actually coming out to be the same result at 82.4% while recall comes out to be 85.5%. Like with the PLA model, these metrics demonstrate that the RNN model is fairly balanced and a good generalization of the data, however unlike the PLA model, the RNN model tends to favor detecting positive cases as demonstrated by the slightly higher recall percentage than precision percentage. This could result in false positives being detected by the model, therefore the recall percentage should be brought down in future versions of this model if possible.

After applying our adjustments to the RNN model, we were able to increase their overall scores. This gave us the following results seen in figure 4: Accuracy: 85%, Precision: 83.6%, Recall 87.1%, F1: 85.3%. This is an overall increase in our initial RNN model and on par with our perceptron. When modeling the changes in accuracy loss in figure 5, we see that the increase in epochs lead to an increase in accuracy and a decrease in loss until about 18 when change is stagnant.

After vectorizing the dataset through TF-IDF, we were able to apply the Perceptron model to gather the top 5 best, neutral, and worst words. This effectively allowed us to quantitatively format qualitative information, providing greater analysis and error checking of our model. As shown in figure 3, the bar chart words like, "masterpiece" and "superb" correlate to movies with highly-rated reviews. On the other hand, a word such as "worst" has a negative connotation in terms of reviews. Meanwhile, neutral words such as 'ordinance' hold little to no sentiment influence.

Our model has further applications than classifying reviews. We can have our models review and judge an individual review. After the RNN model was fully trained and evaluated, we wanted to see how our model would predict five uniquely written movie reviews that were not a part of either the training or testing dataset. Given five reviews consisting of positive, negative, and neutral sentiments, the example reviews were tokenized and given to the model to predict each reviews sentiment. Here are the results:

Review: Absolutely loved this movie, the story was touching and the performances were brilliant! Predicted Sentiment: Positive (Confidence: 0.98).

Review: This was a complete waste of time. Terrible acting and worse direction. Predicted Sentiment: Negative (Confidence: 0.02).

Review: It was okay, not great but not awful either. Had its moments. Predicted Sentiment: Negative (Confidence: 0.05).

Review: One of the best thrillers I’ve seen in a while. Kept me on the edge of my seat. Predicted Sentiment: Positive (Confidence: 0.98).

Review: I couldn’t even finish it. So boring and predictable. Predicted Sentiment: Negative (Confidence: 0.04).

Our RNN model was able to accurately predict the sentiment of each movie review that was clearly either positive or negative. What we found to be of interest was how our model would predict the sentiment of a review which was neutral. Both our RNN and PLA models had only been trained to categorize positive and negative sentiments, therefore when our model was tasked with predicting the neutral review, it had to predict it as either positive or negative. In this particular instance the model found the neutral review to be a negative sentiment. If given more time, finding a way to not only predict positive and negative sentiment, but neutral sentiment as well would be optimal in making a more accurate model.

While developing our model, we can recognize some of the biases the model may hold. Our RNN and PLA model were trained on the hugging face data on imdb. When we take a look at this data we can see by looking at our Perceptron model, that there is an equal distribution of positive and negative reviews. We can see this on figure 8. 50 percent of the reviews are positive and 50 percent of the reviews are negative. This is important as the distribution of reviews on a site or a movie is not likely to be this distribution. From outside research, online reviews on IMDB tend to skew positive. The average rating of the review is a 6.9 out of 10.

We can also take a direct look at the dataset by seeing how imdb moderates its content. According to its Digital Services Act, imdb seeks to moderate its content using automated and human systems. This means that some reviews that are on imdb will be removed for violating their set of violations. Firstly, spam messages and reviews are taken down. They will also take down a review if they see that one person is repeatedly making multiple reviews on the same movie. This seeks to make the reviews on imdb feel more natural, but it makes the reviews less reflective of what people want to put on the review site.

Additionally, content that is flagged by others will be more likely to be removed from imdb’s page. Most specifically, if a review has some form of hate speech then it would also be removed for violating imdb’s policies. This can directly impact our model, as reviews that include hate speech would likely be the most negative. This can impact the criteria and impact of how review sentiments are seen among words and phrases. It is likely that if our dataset included hate speech language, that our worst performing words would likely include some of that language. Reviews with hate speech would likely be on a movie that the reviewer did not like. While this moderation improves the quality of the movie, it seeks to remove some user’s true thoughts. This removal causes a distinct difference of how language is used on IMDB, compared to a site that has looser hate speech policies.

These biases in the dataset make it so that the reviews on the dataset are more authentic and realistic, but they do not make the dataset as representative of the reviews on a dataset. In the perspective of our goal, the dataset does resemble the reviews on imdb, but the distribution of these reviews in terms of being positive and negative could be better. Ultimately, these systems of content moderation seem to eliminate data that would likely be outliers which would have been better to have been handled during the development of our models.

**6 Future Work**

Even though the RNN performed slightly better than our baseline PLA model. Due to a number of limiting factors such as complex model configuration in terms of finding the ideal architectural combination, the RNN isn't the best model for our sentiment analysis use case.

In the future, the team plans on experimenting more with the BERT model as it is naturally good at understanding nuanced and subtle context which is ideal for sentiment analysis. Instead of receiving information sequentially like traditional RNNs, the BERT model analyzes every word within the input at the same time with the help of self-attention. Down the line we plan to shift our attention to developing a sufficient BERT model that's capable of performing better than its counterparts. As it stands with our current experimental BERT model it has an overall accuracy of ~95.8% as seen in figure 6. In terms of learning performance, in figure 7 the BERT model is shown to have a stable drop of training loss per epoch. Also considering the eval loss is low, we can infer that the model isn't prone to overfitting. Alongside the promising results, one of the ways we plan to improve the model is by combining it with a lightweight BERT model such as "ALBERT". This would enable us to reduce the number of overall parameters and lower the complexity of our approach and lower memory usage while also prioritizing high accuracy scores.

Another application of our model is to give a certain movie an overall sentiment score. With some tweaking of our model, we can take the sentiment score of all the imdb reviews under a movie and use the distribution of positive scores and negative scores to determine if the movie is seen positively or negatively. This is effectively taking the average of all the reviews to get a “thumbs up” or “thumbs down” score. This can be useful as IMDB reviews movies by stars rather than this system.

**7 Conclusion**

Implementing and tuning PLA, RNN, and BERT models to the IMDB review sentiment analysis task provided insight into the strengths and limitations of each model. The PLA and RNN implementations both achieved scores in the middle of the eighty percent range with the RNN performing on par after experimentation with hyperparameters and ultimately increasing L2 regularization rate in order to prevent severe overfitting. The change resulted in the following improvement for the RNN model’s performance metrics: accuracy increased from 82.4% to 85.02%, precision increased from 82.4% to 83.6%, recall increased from 85.5% to 87.1%, and F1 score increased from 82.4% to 85.3%. In the end it is clear that the PLA model is not complex enough to capture information from the sequential nature of our data resulting in less accurate predictions than the BERT model. The RNN is able to perform similarly to the PLA model due to its ability to capture important yet complex sequential information contained in the text input, however, BERT was clearly the best suited for the task. BERT’s ability to identify historic reviews that are most similar to the current input allows it to use relevant context in making predictions. The result is that BERT scored 93.7% accuracy, 93.7% F1 score, 93.3% precision, and 94.1% recall. In the future BERT should be used for text based sentiment analysis over either PLA or RNN models.

**8 Figures**

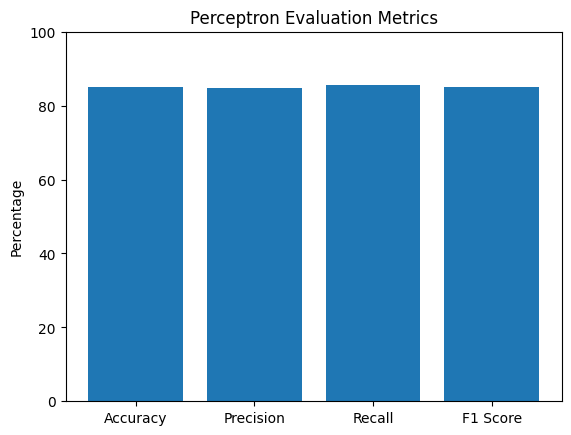


Figure 1: Comparison of resulting scores with Perceptron Learning Algorithm implementation

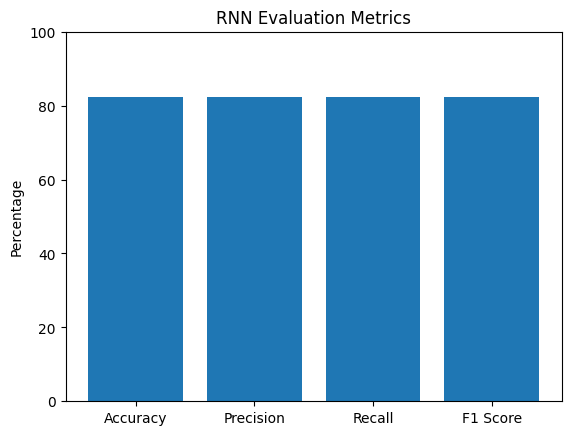


Figure 2: Comparison of resulting scores with original Recurrent Neural Network implementation

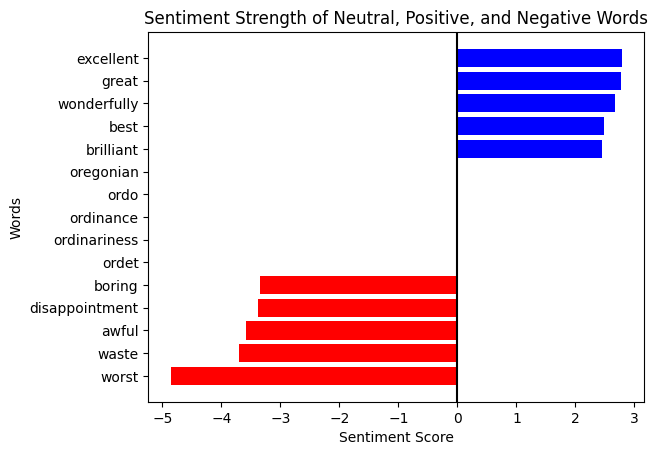


Figure 3: Shows sentiment value to word relationship with the horizontal line representing neutral boundary

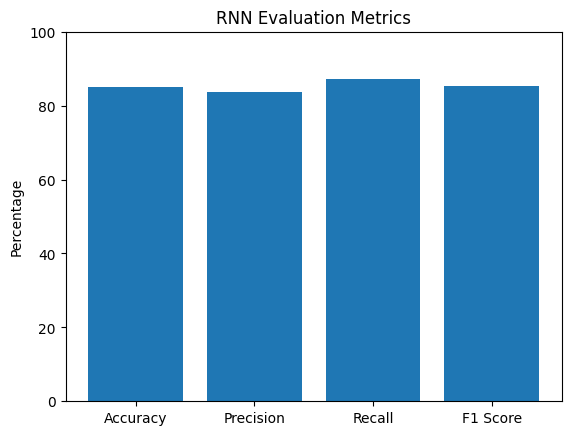


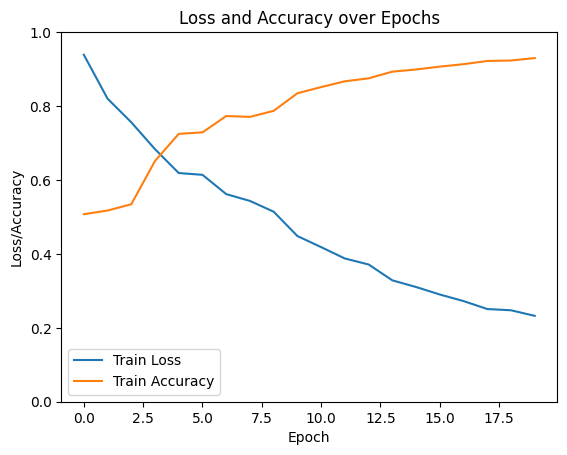
Figure 4: Comparison of resulting scores with improved Recurrent Neural Network implementation

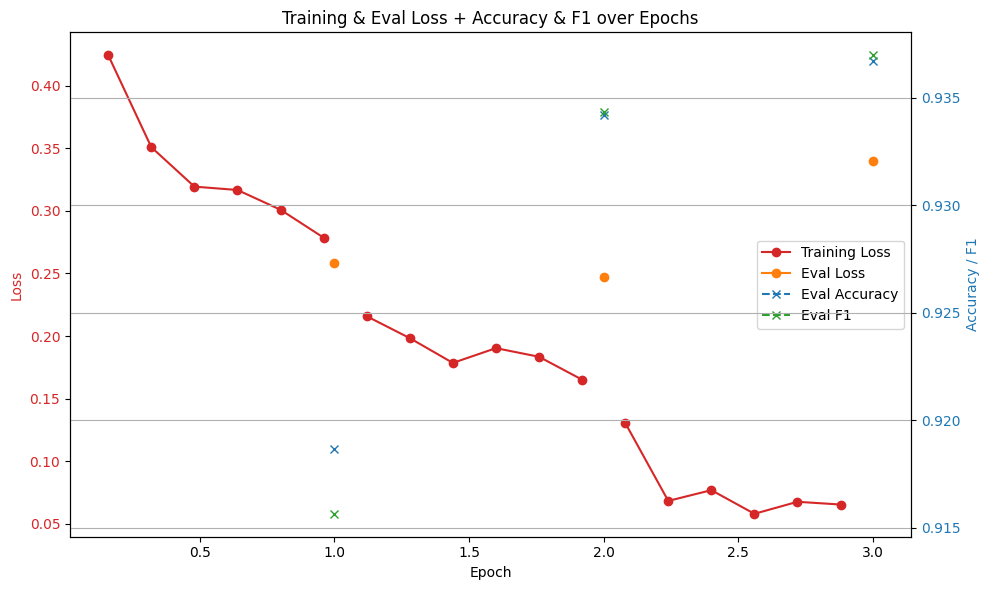
Figure 5: Change in loss and accuracy over each epoch in improved Recurrent Neural Network implementation

Figure 6: Performance of BERT implementation in terms of training loss, eval loss, eval accuracy, and eval F1 score

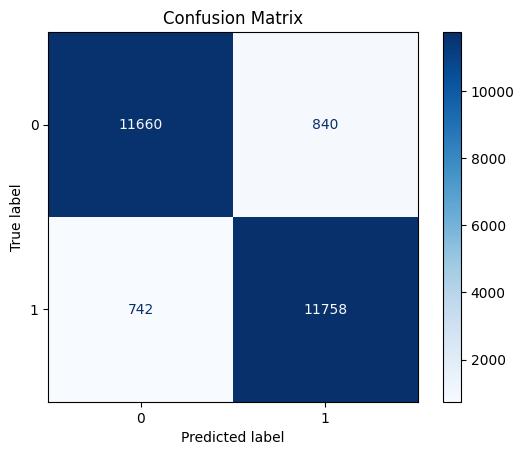


Figure 7: Confusion matrix of BERT implementation

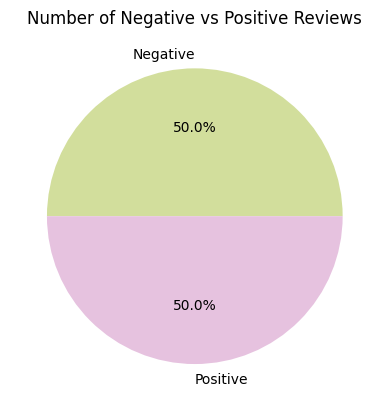


Figure 8: Distribution of sentiment scores across the dataset from our Perceptron Learning Model

**9 Researchers**

Raduan Moustafhim (NetID: moustafh) is a third year student in Computer Science at Michigan State University. He was responsible for training and testing the Recurrent Neural Network including all experimentation to improve the model’s accuracy.

Ben Blanchard (NetID: blanc131) is a fourth year student studying Computer Science and Business at Michigan State University. He was responsible for work on the Recurrent Neural Network.

Jeet Jhaveri (NetID: jhaveri9) is a fourth year student studying Computer Science Engineering at Michigan State University. His responsibilities for this project were developing the coding for the perceptron and working on the initial research on using the RNN and BERT model.

Ndiaga Diouf (NetID: dioufndi) is a fourth-year student studying Computer Science and Business at Michigan State University. He was responsible for formatting and cleaning the required datasets and researched BERT model and implementation techniques.

Liam McNulty (NetID: mcnult48) is a third year student studying Computational Data Science at Michigan State University. He was primarily responsible for the development of the Perceptron Learning Algorithm, along with researching related projects and usages of the BERT algorithm.

**10 References**

Distribution of Things. 2021. IMDb movie ratings: An analysis of over 85,000 films. Published December 16, 2021. <https://distributionofthings.com/imdb-movie-ratings/>.

GeeksforGeeks. n.d. Introduction to recurrent neural network. <https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/>.

Hugging Face. 2024. Kwaai/IMDB\_Sentiment · Datasets at Hugging Face. <https://huggingface.co/datasets/Kwaai/IMDB_Sentiment/viewer/plain_text/train>. Accessed April 11, 2025.

IMDb. 2024. Digital Services Act Information. Retrieved April 16, 2025, from <https://help.imdb.com/article/imdb/general-information/digital-services-act-information/GDAKKSDKCPU25H86>.

Rie Johnson and Tong Zhang. 2015. Effective use of word order for text categorization with convolutional neural networks. In Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 103–112, Denver, Colorado. Association for Computational Linguistics.

Thorsten Joachims. 1997. A probabilistic analysis of the Rocchio algorithm with TF-IDF for text categorization. In Proceedings of the Fourteenth International Conference on Machine Learning, pages 143–151, Pittsburgh, Pennsylvania.

Subhabrata Mukherjee and Sachindra Joshi. 2014. Author-specific sentiment aggregation for polarity prediction of reviews. In Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14), pages 3092–3099, Reykjavik, Iceland. European Language Resources Association (ELRA).

Gift Nkhata, Susan Gauch, Urooj Anjum, and Jialiang Zhan. 2025. Fine-tuning BERT with Bidirectional LSTM for fine-grained movie reviews sentiment analysis. arXiv:2502.20682. <https://arxiv.org/abs/2502.20682>.

‌

‌