Executive Summary of the Thesis

A Transformer-based Approach to Predicting the Likeability of Books

Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Author: Gianluca Guarro

Advisor: Prof. Marco Brambilla

Co-advisor: Marco di Giovanni

Academic year: 2021-2022

1. Introduction

Are our computers capable of extrapolating what elements make a novel or work of literature compelling to the human reader? Instinctively, one would assume that the "cold" binary foundation of computer logic and analytics is at odds with the appraisal of creativity and the interpretation and "understanding" of any form of human art. However, recent advancements in Neural Networks have demonstrated that even the visual, musical, and literary arts can be modeled to an impressive degree by computers, and that the latter can not only interpret human artistic products, but even generate their own works in diverse artistic domains. The recent breakthroughs in this area of research provide the scientific basis and motivation to seriously consider and investigate the aforementioned question, that is whether and to what extent computers of the current generation and processing capability can analyze books and differentiate between successful and unsuccessful literature.

Beyond the purely scientific challenge presented by the posed question, the development of a system that can effectively discern between good literature and bad literature in an automated fashion would have immense practical value to book publishers. It is no secret that the publishing industry operates with an unprecedented level of competition. The advent of electronic submissions in the digital age has resulted in an explosion of manuscripts that editors need to process. As a result, many of these submissions are not adequately assessed and evaluated.

While there is great motivation to study such a question, the underlying problem to be solved is quite complex and seemingly affected by undefined or unidentified factors. When considering in fact the traditional book evaluation processes, it can be readily noted that even many bestsellers have been rejected several times by publishing houses before finally being accepted and permitted to find their audiences. The qualities that make a novel great are not clear cut and may involve aspects that are difficult to assess in a predictive perspective of likeability, such as the book novelty, style of writing, story line, and character development. In addition, a book success may depend on external factors such as resources available for its marketing, current and transient social trends, and competition from books released simultaneously.

Despite its obvious difficulties, recent research has demonstrated that artificial intelligence is able, to an effective degree, to distinguish good literature from bad literature. The goal of this thesis is to expand the research of book success prediction systems by investigating the applicability and effectiveness of the novel transformer architecture [11] to the problem. Since its inception, the transformer architecture has become commonplace among natural language processing (NLP) researchers thanks to models like BERT [2]. BERT harnesses the full potential of a transformer encoder as it comes out-of-the-box pretrained on millions of English sentences via the masked-language modeling task. In this way, BERT is endowed from the start with a strong understanding of the English language. On the other hand, despite the transformer architecture widespread success across several natural language processing tasks, its applicability to the book likeability prediction problem has not been seriously investigated. Our work seeks to take a step in the direction of filling this gap.

2. Previous Work

Before the boom of Artificial Intelligence, literary experts conducted several studies aimed at extrapolating stylistic aspects and/or content characteristics from great authors and books in order to qualitatively understand the elements of successful writing. In recent years, a handful of studies have been carried out that aim to build statistical models capable of predicting the successfulness of a book from its text alone.
Ashok et al. [1] conducted the first computational study correlating writing style and quality in literature. Their work focused on the construction of an array of handcrafted features that could then be used to train a Support Vector Machine (SVM).
Maharjan et al. are the largest contributors, having published three papers on the topic. Their first paper [4] expands upon the work of Ashok et al. with the construction of additional handcrafted features, as well as, neural ones. Among these neural features are Recurrent Neural Network (RNN) features which were developed keeping RNN shortcomings of dealing with long sequences of text in mind. Additionally, they discovered that the RNN model would perform better when trained in a multitask setting. Their third paper [6] introduces the state-of-the-art Genre-Aware Attention Model which allows the classifier to dynamically weigh the various modalities, while, at the same time, learn genre embeddings that get baked into the final representation of the book.
Khalifa et al. [3] attempted to do away with handcrafted features and instead build an end-to-end Convolutional Neural Network (CNN)-based classifier using pretrained sentence embeddings. While they were able to achieve good results using this method, they found that they could boost the weighted F1 score by nearly 5 points by simply incorporating readability metrics.

3. Task Dataset

An important characteristic of our target task is that it involves the classification of very long sequences. Generally speaking, neural networks designed to process sequential data, including RNNs, LSTMs, and transformers, are ill-suited for the processing of very long sequences. As a result, prior work on this problem has relied on both count-based textual features and/or some unorthodox and ad-hoc training procedure for allowing these neural networks to overcome to some degree this underlying issue. Given the nature of our problem, our work has proceeded along this same paradigm, in that one of its main contributions is an extensive investigation into how to make best use of transformer models when dealing with very long data sequences.
The dataset [4] used in this work is a benchmark dataset procured by Maharjan et al. The dataset consists of 1003 books across eight genres taken from Project Gutenberg. The average rating of the books on Goodreads was used to label the books as successful or unsuccessful. Specifically, if the book had an average rating of 3.5 or greater, it was considered successful, otherwise it was considered unsuccessful. In an effort to reduce noise or bias, certain heuristics were performed to appropriately choose the books, such as not allowing for books with too few ratings or books whose authors have already appeared twice in the dataset. The genre and success label distribution of the dataset can be seen in Table 1.

Genre	Unsuccessful	Successful	Total
Detective Mystery	60	46	106
Drama	29	70	99
Fiction	30	81	111
Historical Fiction	16	65	81
Love Stories	20	60	80
Poetry	23	158	181
Science Fiction	48	39	87
Short Stories	123	135	258
Total	349	654	1003

Table 1: Genre Distribution of Goodreads Maharjan Dataset

While working with this dataset, we had to take into account that the books typically began with a preamble containing information such as the copyright, translation or transcription note, an ASCII title page, etc. In an effort to concentrate our classifier training on the story itself, we studied methods to remove this information from our dataset. Noticing that the preamble often made frequent use of newline characters with respect to the rest of the text, we employed the CUSUM change detection algorithm [7] to identify precisely where the frequency of newline characters would change dramatically. According to our error analysis, this point correlated very well with where the book actually began. Thus, we used this method to trim the unwanted text from all the books in the dataset.

4. Our Work

In our work, we distinguish between first and second stage classifiers. The former process concerns the training of our BERT basic model and of the other BERT-like models that we have also investigated, whereas the latter concerns the training of models that make use of embeddings generated from the former. The motivation to study second stage classifiers comes as a direct result of working with very long sequences. Since transformer-based models typically have a low max sequence length limit, we needed to train our models by segmenting our books into multiple training samples. As a result, our BERT model does not attain a cohesive understanding of each book and instead averages its predictions on the segments to classify each book. The goal of the second stage classifier utilization was to compensate for this BERT limitation by allowing for a more holistic view of the books.

4.1. First Stage Classifiers

The development of the first stage classifier (i.e. transformer-based model) entailed many design decisions, that we investigated in an attempt to either better prepare the model for the target task, or to mitigate the potential issues arising from the segmenting / chunking of our dataset. These design decisions include a) the choice of the base transformer model; b) how to further pretrain the model (if at all); c) whether masking the characters in our dataset is useful; d) how to best tokenize our books into segments; e) whether to train the network in a multi-task setting with the genre; and f) if setting a max segment limit per book, or taking more fine-grained control of the sampling order to guarantee a more fair representation of shorter books, is useful. Decisions c, d, and f aim to target potential issues that surface from segmenting our samples into many parts.

Through Pointwise Mutual Information (PMI), we were able to verify that character names are the most discriminative words that distinguish the successful and unsuccessful classes in our dataset. Concerned about the risks of overfitting our model on character names, we passed our dataset through a named-entity recognition model to detect character names and subsequently mask them. Despite the results from PMI, character masking provided no significant benefit to the classification task, thus suggesting that BERT is robust to unique class identifiers.

We also explored two different tokenization algorithms. One that would ensure sentences did not get split between segments and one that would tokenize books with a moving window allowing for a defined amount of overlap. Moreover, we also studied the effect of truncating longer books to a defined max number of segments so that they are not overrepresented. Our experiments show that the two tokenization algorithms perform similarly on our dataset. Moreover, the insight BERT can extrapolate from each book quickly saturates. That is, we perceive no advantage in using more than 20 segments per book during our training.

Among all the BERT-inspired models, DistilBERT yielded the best performance on our task. This comes as somewhat of a surprise, since the DistilBERT model was designed simply to be smaller and faster to train than other transformer-based models, rather than to seek better performance [9]. Moreover, also as a partial surprise, other models that either claimed to be a better version of BERT (RoBERTa) or advantageous when dealing with very long sequences (BigBird and Longformer) actually performed poorly on our task. These models were larger in size than BERT. We are led to deduce from these contrasting performance records, although we do not have a definitive proof for this conclusion, that such a difficult task as ours, for which such a small dataset as the one at our disposal is available, is actually better addressed by a smaller model with more built-in bias like DistilBERT.

Maharjan et. al report improved performance on the book likeability prediction task when training their models to simultaneously identify / predict the book genre. We have drawn inspiration from their results and modified BERT (DistilBERT) to this multitask setting. Despite not having come across other research that trains BERT to predict classes from two sets of labels simultaneously, our results show that our target task is benefitted by this multitask setup. We believe that simultaneously predicting book genre acts as a form of regularization for the likeability task.

While the investigated transformer-based models have been pretrained on millions of sentences of the English language, they have not necessarily been pretrained to understand literary English. We have therefore explored techniques that further pretrain these models on text within the same domain as our target dataset, making them more adept at handling our target task. In particular we have applied both the within-task and in-domain pretraining paradigms [10]. The former implies pretraining by using the same text as our classification dataset, whereas the latter requires an additional dataset whose text is obtained from a similar distribution as the text of the classification dataset. To explore the latter approach, we procured our own dataset of 2600 books from Project Gutenberg. While our results are not entirely conclusive, they suggest that further pretraining is at least slightly beneficial for the target task.

In comparison with the use of BERT in its simplest form (no pretraining, single-task), the utilization of the design decisions validated by our experiments in combination, to build the first stage classifier, allowed us to achieve a boost in performance from a weighted F1 (W-F1) score of 62.89% to a W-F1 of 72.15% on the test set. This result outperforms other architectures from other work designed to process sequential input as well as our own initial strong baselines.

4.2. Second Stage Classifiers

The classifier model developed so far makes predictions on segments of a book and aggregates them together to make a prediction for the whole book. We have attempted in this fashion to build and investigate a classifier that can consider a representation of the whole book in unison. As a building block, this study makes use of embeddings of the segments of our book extracted from our fine-tuned transformer model. Our research includes A) the training of hierarchical sequential networks, RoBERT and ToBERT , over our segment embeddings; B) the aggregation of the segment embeddings to create book embeddings and subsequently training a shallow feedforward network and an SVM over them; and C) the training of multimodal architectures that not only incorporate our BERT-based book embeddings but also utilize handcrafted features and other useful neural features. Our hierarchical sequential networks like RoBERT and ToBERT [8] train an LSTM layer and transformer encoder respectively over our segment embeddings. Despite the promising performances published in their original paper, these models underperformed our base DistilBERT model. On the otherhand, we were able to achieve our best W-F1 score of 73.63% by averaging the segment embeddings per book and using them to train a support vector machine (SVM). We were able to achieve a similar performance of 73.57% by training Maharjan's Genre-Aware Attention multimodal network using our DistilBERT book embeddings, sentiment concept features, and readability metrics. While the original paper for the Genre-Aware Attention Model [6] reports state-of-the-art results of 75.4% using their own combination of handcrafted and neural features, we found that model to be a difficult and somewhat unpredictable network to train, with its performance being highly dependent on initial conditions and hyperparameters. Lastly, to allow for an easy comparison between our best performing model, our baselines, and the best performing models from prior works, we coalesce their results in Table 2. Model names in normal, italicized, or bolded text represent, respectively, our baselines, the best performing models of prior work, and our best performing model.

Models	**Test W-F1**
Most Frequent	0.506
Stratified	0.542
BERT One Randomized Chunk [3]	0.660
Bag of Words Logistic Regression	0.665
Tf-idf Logistic Regression	0.670
Word2Vec RNN	0.686
Emotion Flow Model [5]	0.690
Doc2Vec SVM	0.691
CNN with Readability [3]	0.720
Maharjan Multimodal With RNN [4]	0.735
SVM w/ BERT-based Features	0.736
Genre-Aware Attention [6]	0.754

Table 2: Model Comparison

5. Bibliography

- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

- Suraj Maharjan, John Arevalo, Manuel Montes, Fabio A González, and Thamar Solorio. A multi-task approach to predict likability of books. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1217–1227, 2017.

- Suraj Maharjan, Manuel Montes, Fabio A González, and Thamar Solorio. A genre-aware attention model to improve the likability prediction of books. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3381–3391, 2018.

- Raghavendra Pappagari, Piotr Zelasko, Jesús Villalba, Yishay Carmiel, and Najim Dehak. Hierarchical transformers for long document classification. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 838–844, 2019.

- Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *China National Conference on Chinese Computational Linguistics*, pages 194–206. Springer, 2019.

6. Conclusion

In conclusion, we found transformer-based models like BERT to be a suitable building block for the realization of a book likeability classifier, even more so when supplemented by the use of second stage classifiers which build off of BERT output, outperforming strong baselines. This result comes despite the fact that our task entails the classification of very long sequences, a domain for which transformer models are typically bottlenecked. In order to realize the true potential of transformer models on the task, we found worthwhile to explore several design decisions, such as a proper choice of the base model, further pretraining the model using a within-task approach, and modifying the transformer to work in a multitask setting. To our surprise, first-stage transformer models that advertise great progress in performance, such as RoBERTa, BigBird, and Longformer, failed in our task to live up to their claimed capabilities, while DistilBERT, whose principal focus was primarily to be just smaller and easier to train than BERT, was actually the most well suited in performance to the task among the base transformer models investigated. Moreover, we were able to show that fine-tuning BERT, so that it simultaneously predicts both book genre and likeability significantly boosts performance in the likeability task. We were also successful in coalescing the BERT embeddings of the book segments together by taking their mean, as evidenced by the enhanced performance of our SVM model. On the other hand, hierarchical sequential models underperformed with respect to our base model, in contrast with the results published in the associated original paper. While our results do not surpass the state-of-the-art, we have been able to achieve a high performance without using count-based features or a multitude of neural features. This fact makes our network in some ways easier to apply, although at the expense of the considerable amount of computational effort needed for an adequate level of BERT training.

7. Acknowledgements

I would like to extend my gratitude to the several people who have been part of my journey during the completion of my master's thesis.
To my superadvisors, Professor Brambilla and Marco Di Giovanni: It has been an honor to work under your guidance and direction. The discussions from our weekly meetings inspired me to keep pushing my work further and further.
I must also thank my parents, Drs. Clorinda Donato and Sergio Guarro and my siblings, Marcello Guarro and Adriana Romero. Without their unconditional love and support, I would have never made it this far.
A special thanks goes to my relatives in Terni: Elisabetta, Vilma, Francesca, and Riccardo, who helped me stay sane and productive in my Polytechnic online studies when I escaped to the Umbrian countryside during the pandemic. I am forever grateful that allowed me to develop strong bonds with all four of you.
Last but not least, I would like to thank all the friends that I met along the way. Because of all of you, I will forever cherish the time I spent in Milan.

References

[1] Vikas Ganjigunte Ashok, Song Feng, and Yejin Choi. Success with style: Using writing style to predict the success of novels. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1753–1764, 2013.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] Muhammad Khalifa and Aminul Islam. Will your forthcoming book be successful? predicting book success with cnn and readability scores. *arXiv preprint arXiv:2007.11073*, 2020.

[4] Suraj Maharjan, John Arevalo, Manuel Montes, Fabio A González, and Thamar Solorio. A multi-task approach to predict likability of books. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1217–1227, 2017.

[5] Suraj Maharjan, Sudipta Kar, Manuel Montes-y Gómez, Fabio A González, and Thamar Solorio. Letting emotions flow: Success prediction by modeling the flow of emotions in books. *arXiv preprint arXiv:1805.09746*, 2018.

[6] Suraj Maharjan, Manuel Montes, Fabio A González, and Thamar Solorio. A genre-aware attention model to improve the likability prediction of books. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3381–3391, 2018.

[7] Ewan S Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, 1954.

[8] Raghavendra Pappagari, Piotr Zelasko, Jesús Villalba, Yishay Carmiel, and Najim Dehak. Hierarchical transformers for long document classification. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 838–844, 2019.

[9] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[10] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *China National Conference on Chinese Computational Linguistics*, pages 194–206. Springer, 2019.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

A Transformer-based Approach to Predicting the Likeability of Books

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Gianluca Guarro**

Student ID: 918696
Advisor: Prof. Marco Brambilla
Co-advisors: Marco di Giovanni
Academic Year: 2021-2022

Abstract

Can machines learn what it is that makes a good book a good book? On the one hand, the characteristics of a computer seem to be intrinsically unsuited for the interpretation of any form of human art, yet recent advancements in Neural Networks demonstrate that computers have effectively been able to understand and even generate their own works of art in the visual, musical, and literary domains. Inspired by the fast-growing achievements in Natural Language Processing (NLP) and its potential use-cases, with the present work we explore the current limits of a book success prediction system. Prior research has demonstrated that Artificial Intelligence (AI) is able to "read" and analyze book contents to generate assessments of their quality that correlate reasonably well with factual indicators of success with human readers. However, although methods of content analysis and classification have been rapidly evolving in the rapidly growing field of NLP, and replacing older approaches, such more modern approaches have not yet been fully applied to the task of assessing book quality and likeability with potential human readers.

In our thesis, we propose a transformer-based approach to the book likeability prediction problem. More specifically, we investigate the applicability of Google's new language model, BERT, and BERT-like models, which have seen wide-scale adoption and application across an abounding number of NLP tasks. Given the inherent limitation on the length of text sequences that NLP algorithms can analyze and classify in one single processing step, in our application we experiment with approaches in which the transformer architecture can be tailored to enable the classification of very long sequences, such as books. In addition, we also explore techniques to generate embeddings and book representations from transformer-based models, so that these can be used in conjunction with more classical text-derived features to train multi-modal networks that produce likeability ranking outputs.

Keywords: Artificial Intelligence, Data Science, Machine Learning, Natural Language Processing, Neural Networks, Transformer Architecture, Multi-modal Networks, Book Likeability

Abstract in Lingua Italiana

Può una macchina elettronica capire cosa è che determina la qualità di un buon libro? Da un lato le caratteristiche di un computer sembrano essere intrinsicamente poco adatte ad interpretare qualsiasi tipo di espressione artistica umana; dall'altro i recenti progressi raggiunti nel campo delle Reti Neurali (Neural Networks) hanno dimostrato che i computers sono stati capaci di capire forme di arte e persino generarne esempi, nei campi delle arti visuali, musicali, e letterarie. Partendo dall'ispirazione fornita dai sempre più notevoli sviluppi nel campo del Natural Language Processing (NLP) e dalle sue potenziali applicazioni, la presente tesi esplora le possibili capacità di un sistema di predizione del livello di successo ottenibile da parte di un libro. Precedente ricerca ha dimostrato che tecniche di Artificial Intelligence (AI) permettono ad un computer di "leggere" ed analizzare il contenuto di un libro per generare una valutazione della sua qualità, in modi che si correlano ragionevolmente bene con reali indici del successo ottenuto con normali lettori umani. Tuttavia, sebbene metodi di analisi e classificazione del contenuto di un libro siano in costante e rapida evoluzione nel sempre più ampio campo di ricerca dell'NLP rimpiazzando progressivamente metodologie precedenti, tali moderni metodi non hanno ancora trovato un'applicazione estesa e completa al problema di valutare la qualita' di un libro e il suo potenziale di apprezzamento da parte dei suoi lettori.

In questa tesi proponiamo un approccio alla valutazione del potenziale di apprezzamento ("likeability") di un libro che è basato sull'uso delle reti "transformers" (Transformer Networks). Più in particolare, la tesi esplora l'applicabilità a questo problema del nuovo modello di linguaggio BERT prodotto da Google, e di vari modelli derivati da BERT, i quali hanno avuto adozione e applicazione su larga scala in una estesa varietà e notevole numero di problemi nel campo dell'NLP. La nostra applicazione, dati gli intrinsechi limiti di lunghezza delle sequenze di testo che un algoritmo NLP può analizzare e classificare in un singolo stadio di analisi, è stata condotta sperimentando con vari metodi tramite i quali l'architettura transformer può essere adattata per permettere la classificazione di sequenze di dati di notevole lunghezza, come è nel caso del testo di un libro. Oltre questo, la tesi usa e valuta tecniche per generare rappresentazioni ed indici delle caratteristiche di un libro ("embeddings"), tramite modelli di tipo transformer, e usare tali rappresentazioni, in congiunzione con altri più tradizionali indici di valutazione basati sul testo stesso, per eseguire la calibrazione e formazione ("training") di reti multi-modali che producono le valutazioni finali del potenziale apprezzamento di un generico lettore del libro.

Parole Chiavi: Intelligenza Artificiale, Scienza dei Dati, Apprendimento Automatico, Elaborazione del Linguaggio Naturale, Reti Neurali, Reti Transformers, Reti Multi-modali, Apprezzamento del Libro

Contents

1 | Introduction

Millions of books are written each year and only a fraction are accepted by traditional publishers, with an even smaller fraction of those becoming renowned successes. There are many factors that go into predicting whether a book will be successful or not, including intrinsic features such as novelty, style of writing, story line, and character development, as well as external factors such as marketing, current trends, and competition from books released simultaneously. All of these factors can make it difficult to estimate a book's true value and reception in the marketplace. In fact, even some best sellers went through several rejections before a publisher finally approved them. The quintessential example is J.K. Rowling's Harry Potter which was rejected by twelve different publishers before being taken on by a thirteenth. The difficulty of accurately assessing how a book might appeal to the public is exacerbated by the sheer number of submissions that publication houses need to process [1]. With these considerations in mind, there is a strong motivation to study the viability of an automated book success prediction system. Additionally, the huge strides the field of natural language processing has taken as of recent, with many state-of-the-art results across several datasets being achieved in just the past few years, provides an equally strong motivation to explore and evaluate their applicability to this specific type of task, and see how they compare to prior, more classical, success prediction models that have been proposed and applied in the past, with limited degrees of success.

1.1. General Overview

With the massive and constantly growing quantity of written text that has become directly accessible online, Natural Language Processing has emerged as an exciting field of AI research and development that aims to extract true essential content and information from this knowledge-rich, albeit unstructured and massive amount of data. From machine translation, to semantic analysis, to part-of-speech tagging, NLP has been effectively applied to several text analysis and manipulation tasks. Among these, the Book Success

[1]`https://jerichowriters.com/if-an-agent-accepts-your-work-what-are-chances-of-getting-published-2/`

Prediction problem is particularly intriguing and difficult, due to the multi-faceted factors involved and its partly subjective nature. The goal of this thesis is to build upon previous work carried out by NLP researchers in this area, more specifically by exploring different ways of generating book neural features through use of the transformer architecture, to evaluate and compare their efficacy.

Since its publication in 2017 [32], the transformer neural architecture has been applied to a great variety of tasks, often yielding state-of-the-art results in the processing and classification of large amounts of data, especially in the NLP domain. In fact, at the time of writing this paper, the GLUE benchmark leaderboards are overrun by transformer-based approaches [2]. For this reason, it is natural to consider an application of this type of technique to the book success prediction problem as a research subject well worth of serious investigation. However, transformer models can typically handle only a short sequence of words and were thus not poised to address the classification of long texts such as books. Thus, a central part of our work has focused on an extensive exploration and evaluation of how to best overcome this limitation. More specifically, a lot of effort first went into the study of how to best prepare the training data for consumption by BERT, and second, into the study of how to best consolidate BERT's inherently segmented judgment into a wholistic one. As a result, we have succeeded in extracting from the books selected for our case study powerful neural embeddings which can be used as input features for Maharjan's state-of-the-art Genre-Aware Attention Model (along with other second stage classifiers) to determine whether we can further boost performance.

Our experiments have shown that the transformer architecture can be adapted to deal with long-sequence text, such as books, to recognize their quality; moreover, our experiments also reveal transformer architecture to be competitive with other purely neural classifiers. The validity of our approach has been confirmed by the fact that, by training a support vector machine on these neural representations, we have been able to produce results that are less than two percentage points from the state-of-the-art's weighted- f1 score on the Goodreads Dataset – more specifically, 73.6% vs. 75.4%.

1.2. Brief Description of the Work

The elements readers esteem to be essential when they evaluate the quality of a book are perceived to be purely subjective and therefore difficult, if not impossible, to quantify. Ultimately, as readers, we look for books that make us feel intellectually or emotionally engaged with the storyline and the characters, a response that is predicated upon seem-

[2]https://gluebenchmark.com/leaderboard

ingly unquantifiable features, such as an engaging plot, a literary style that "matches" the setting of the book, or well-developed characters that are relatable to one's own experiences [3]. Despite obvious difficulties, the development of an autonomous system able to effectively discern whether readers will like a book or not would be of immense value for book publishers and readers alike. At the same time, testing the limits of the current state-of-the-art of Natural Language Processing with such an inquisitive task is a fascinating study in its own right, as it seeks to negotiate questions of taste, emotion, culture, and content usefulness which all converge in reader appraisal of a book.

With these goals in mind, we draw inspiration from former researchers on the topic, namely Ashok et. al [2], Maharjan et. al ([19], [21] [20]), and Khalifa et. al. [16]. Among these pioneers in the field, our work builds primarily from that of Maharjan et. al who have published extensive research on the book success prediction problem. As part of their work, they assembled and made publicly available a benchmark dataset to enable researchers to better assess and understand the performances of their models. In their experiments, they have studied the efficacy of several handcrafted features, as well as RNN and BoW based neural features. By designing a multimodal neural architecture that is able to incorporate all of these features (and learn the importance of each) while at the same time also learn genre embeddings, they were able to achieve a weighted-f1 score of of 75.4%.

To our knowledge, prior to our own research, application of the novel transformers model and in particular BERT, has rarely been studied to predict the likeability of books. Thus, the scope of this paper is to study the ways in which BERT can be applied to this problem. This means using BERT directly for classification, as well as extracting embeddings from it to be used in second stage classifiers. We have explored a multitude of ways to best prepare BERT for the target task, according to what is explicitly detailed in *How to Fine-Tune BERT for Sequence Classification* [30]. Having established a strong BERT model as the foundation, we have implemented and assessed the techniques discussed in *Hierarchical Transformers* [25], in which the authors report success in training an LSTM or a Transformer Encoder over fixed BERT embeddings. Finally, we have also investigated whether our BERT embeddings are able to provide any additional information to all of the other features being considered, by using them in multi-modal settings such as in Maharjan's Genre-Aware Attention Model [21].

[3]`https://www.masterclass.com/articles/the-elements-of-a-good-book#the-5-elements-of-a-good-book`

1.3. Outline of the Thesis

The remainder of this thesis is structured as follows:

- *Chapter 2* summarizes related research and state-of-the-art results on the subject of book success prediction;
- *Chapter 3* presents and discusses the technical foundation of NLP and of the techniques investigated in this thesis;
- *Chapter 4* describes the research questions that this thesis addresses and seeks to answer;
- *Chapter 5* details our approach to answering the research questions presented in the previous chapter;
- *Chapter 6* explains the details of the datasets, including all associated data pre-processing steps, that were used for our experiments;
- *Chapter 7* describes the experiments that were set-up and performed to answer the research questions defined in Chapter 3, and presents / discusses the results obtained;
- *Chapter 8* summarizes the entire work carried out, drawing conclusions on our approach to the problem, on the validity of the results obtained, and on the possible direction of any future research;

2 | Related work

"You shall know a word by the company it keeps."

John Rupert Firth

We present here a survey of the different approaches that have been applied to the book likeability prediction problem to date.

2.1. Book Success Prediction

The understanding of what makes a good book a good book has intrigued literary enthusiasts for decades. Literary experts have conducted several studies aimed at extrapolating stylistic aspects and/or content characteristics from great authors and books. These studies are qualitative in nature and rely on expert domain knowledge ([14], [13]). In recent years, a handful of studies have been carried out that aim to build statistical models capable of predicting the successfulness of a book from its text alone. As this paper builds off these more objectively and quantitatively-oriented approaches, the following sections document the survey and review of the more recent work in this area that we have conducted at the onset of our research. This material provides an overview of the approaches that have recently been explored by other researchers, and includes a summary discussion of the key features that have been formulated and applied.

2.1.1. A brief overview of quantitative approaches

Ashok et al. (2013) [2] conducted the first computational study correlating writing style and quality. Their dataset was composed of books from Project Gutenberg [1] with the success label being defined as a function of the number of downloads a book had received. Their work concentrated on the construction of handcrafted features intended to encapsulate the stylistic information of the book.

[1]Project Gutenberg is an online repository of cultural works with most items consisting of books within the public domain

Maharjan et al. (2017) [19] argued that downloading a book is not a good indicator of a book's likeability or its commercial success and in turn, composed a new dataset in which the success label was derived from the book average rating on Goodreads. In their analysis, they considered a combination of handcrafted stylistic features and neural features (including RNN features). In order to incorporate RNN features, they developed a strategy to overcome RNN limitations in dealing with long sequences of text. Moreover, they framed the problem in a multi-task learning setting, by simultaneously seeking to identify the book genre as well as to predict its level of success.

Maharjan et al. extended their work from the prior year with two additional papers in 2018 ([20] [21]). The former proposes an approach in which they focus on modeling the emotion flow throughout the books, with the hypothesis that a novel's success is highly correlated with the flow of emotions it makes its reader feel. The latter proposes the "Genre-Aware Attention Model", a multi-modal neural architecture that is well-equipped to flexibly attend to the variety of handcrafted and neural features they extracted from the books. In their work, they were able to achieve the current state-of-the-art on the Goodreads dataset using this model with a weighted F1 score of 75.4%.

Khalifa (2020) proposed a model in which they embedded the books' sentences with Universal Sentence Encoder (USE) and split them into chunks to then feed through a convolutional neural network (CNN). The output from the CNN along with various readability metrics are then fed through a final classification layer. Interestingly, they report the best results when they only consider the first 1K sentences of each book. This may indicate that the stylistic essence that their neural network learns from each book is quickly saturated or that only a portion of a book needs to be read in order to assess its general quality.

2.1.2. Long Text Classification

With the advent of neural architectures such as LSTMs and Transformers, end-to-end deep learning has become increasingly commonplace for several NLP tasks. However, a purely neural learning process is not practical for the book success prediction problem, where one is basically dealing with very long sequences of text. RNNs, for instance, suffer from vanishing gradients during training by back-propagation through time (BPTT). LSTMs, in turn, were designed to circumvent the shortcomings of RNNs, but they have their limits as well, and will still eventually succumb to the vanishing gradient problem at some point, when dealing with very long sequences of data to analyze and classify. Transformers like BERT abandon the sequential left-to-right or right-to-left processing paradigm of

RNNs/LSTMs and instead use a self-attention mechanism that allows each word to attend to all others to discover its importance in a parallel processing mode. However, this self-attention mechanism has the problem of consuming memory quadratically with the length of the input sequence. As a result, BERT has a sequence limit of 512 tokens.

Given the neural networks' shortcomings of handling long sequences of text, all prior works make use of purely handcrafted features (Ashok et al. (2013)) or a mix of both handcrafted and neural features (Maharjan (2017), Maharjan (2018), Khalifa (2020)) which are then used to train a classifier. These features scale well with the length of the book as they are count-based. However, they are typically only able to capture aspects of linguistic style such as lexical choices, distribution of part-of-speech tags, distribution of grammar rules, etc. and thus they overlook the plot, semantics, and emotions of the book. It is therefore, often best to use a combination of neural and handcrafted features, as each seems to intuitively support the shortcomings of the other. A natural question is how to generate the most expressive neural representations of a book. This paper will explore various techniques that aim to answer this question using BERT.

Given the neural networks' shortcomings in handling long sequences of text, all prior works make use of purely handcrafted features (Ashok et al. (2013)), or a mix of both handcrafted and neural features (Maharjan (2017), Maharjan (2018), Khalifa (2020)) which are then used to train a classifier. These features scale well with the length of a book being analyzed, since they are count-based. However, they are typically only able to capture aspects of linguistic style such as lexical choices, distribution of part-of-speech tags, distribution of grammar rules, etc. While they do that relatively well, they overlook other features of a book that may be equally, if not more important, in having a substantial impact on likeability, e.g., characteristics of the plot, the overall meaning and message the book seeks to convey, and the type of emotions it is likely to evoke in the reader. One is thus led to the conclusion that it is best to use a combination of neural and handcrafted features, as each type seems to intuitively compensate for the shortcomings of the other. A natural question along this line of investigation is whether it is possible to generate a neural representations of a book sufficiently expressive and effective for the purpose of likeability and success prediction by including in its representation both count-based and non-count based relevant features. This thesis explores and discusses some techniques that aim to provide an answer to this question using BERT.

3 | Technical Approach Foundations

In the previous chapter, we introduced and summarized prior approaches to the book likeability prediction problem. In this chapter, we aim to provide the technical foundation background that is needed to understand the solutions adopted in our research approach to the problem. This is initiated with a brief general description of Natural Language Processing (NLP) techniques and is continued with a more specific and detailed discussion of the techniques that we have identified and applied as potentially best suited for the task at hand. Of particular relevance is the section dedicated to BERT, as BERT constitutes the primary foundation for the research described in this thesis. Additionally, we provide descriptions of other statistical methods used in this work that do not have an immediate prior affiliation with past NLP applications or research.

3.1. Natural Language Processing

Natural Language Processing (or NLP for short) is the field of study at the crossroads between linguistics and artificial intelligence. In essence, it is the study of how computers can be programmed to interpret and manipulate human language, e.g., to translate from one language to another. The field emerged in the 1950s and until the 1980s to accomplish its basic set of tasks, among which parsing and translation figured as primary ones, relied mainly on a complex set of hand-written rules. More recently, due to massive increases in available computational power and the massive quantity of unstructured textual data appearing and being stored in decentralized web storage, NLP has boomed both in practical applications and as a research field, alongside its more general parent field of Artificial Intelligence (AI). More specifically, as the capability of our computers to store and process data has continued to grow, so has the research to architect machine learning models that are able to better "understand" and process human language for a variety of purposes. The subsections below discuss some of the AI techniques that have found frequent use in NLP.

3.1.1. LSTM

LSTM (Long Short-Term Memory) is a particular type of artificial Recurrent Neural Network (RNN) architecture that has been successfully applied in a wide-range of NLP applications, from image captioning [33] to machine translation [31] to question answering [35]. "Recurrent" in this context means that an RNN, unlike a standard feedforward neural network, has feedback connections in its neural grid. Within the RNN family, LSTM units are typically considered superior to other more traditional types of RNNs because they contain memory cells, which allows them to maintain information for longer periods of time. The structure of a typical LSTM unit is depicted in 3.1.

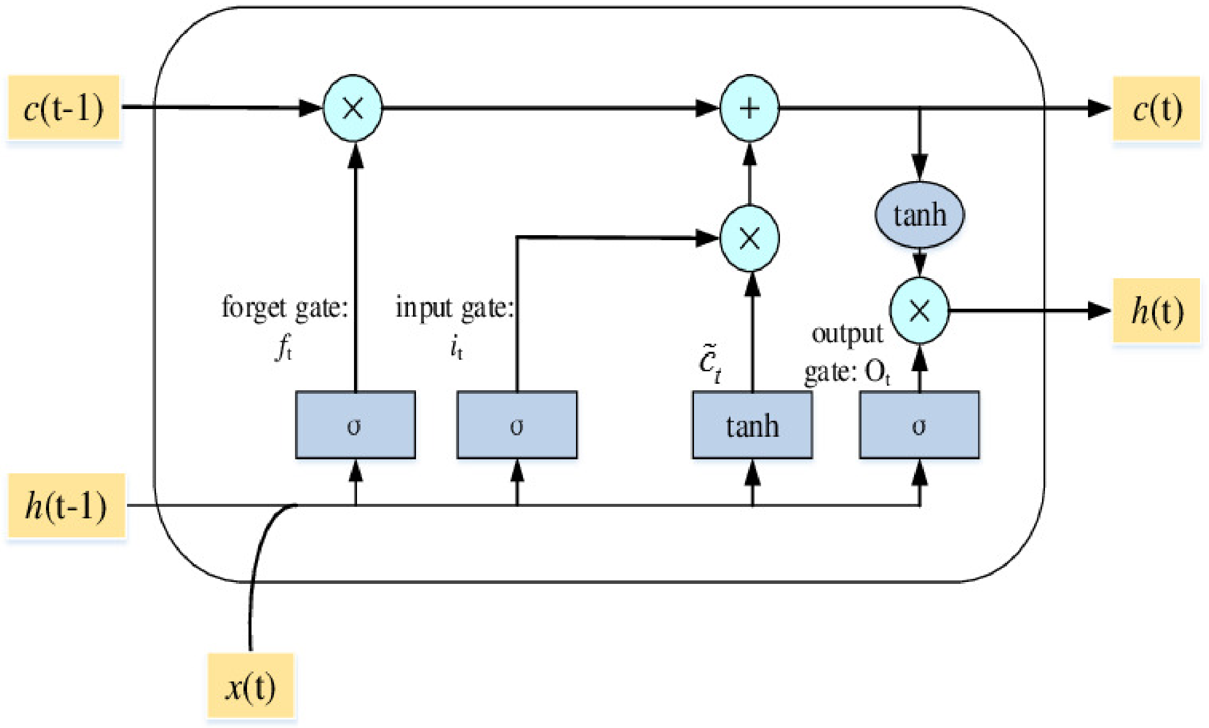


Figure 3.1: LSTM Unit [36]

The basic LSTM unit contains three external inputs:

1. its previous cell state, c_{t-1}
2. its previous hidden state, h_{t-1}
3. the current input vector, x_t

The jobs of the forget gate, input gate, and the output gate are to regulate the flow of information into and out of the cell. The three gates are the output of a nonlinear activation function taking as input a linear combination of the previous hidden state and the current input vector.

3.1.2. Transformer

Like LSTMS, Transformers are intended to process sequential data. However, in order to recognize the importance of each part of the input they use the mechanism of self-attention rather than recurrence. The Transformer architecture has largely replaced RNN-based networks in NLP, as it does not suffer from parallelization limitations as RNNs do and thus can consume and process much more data in a given amount of time. As self-attention is really the cornerstone to the Transformer's popularity, we shall explain the concept more in depth.

The motivating concept behind self-attention is to allow the model to look at all other tokens when encoding each token in a sequence, thereby learning more context-aware embeddings for each. The way this is done in practice is to score all tokens at each position to represent how relevant each token is at each position. As discussed in *Attention is all you need*, the scoring is done via the following equation:

$$Z = softmax(\frac{Q \times K^T}{\sqrt{d_k}})V \tag{3.1}$$

where Q, K, and V represent the Query, Key, and Value matrices respectively. Q, K, and V are found by taking our embeddings X and multiplying them by trainable weight matrices W_Q, W_K, W_V respectively. The resulting matrix, Z is then ready to be processed by further layers.

The self-attention layer can be further endowed with a mechanism known as "multi-headed attention," which the original authors have demonstrated improve performance. In this variation of the concept, at each layer of the net self-attention is performed by N heads which are individually and randomly initialized, thus allowing the layer to learn multiple representations of the inputs. In order to prepare the output for the dimensions of the next layer, the outputs from the attention heads are concatenated together and multiplied by a weight matrix.

With the concept of multi-head attention explained in its basic characteristics, we can present in Fig. 3.2 an illustration of the typical architecture of a Transformer model:

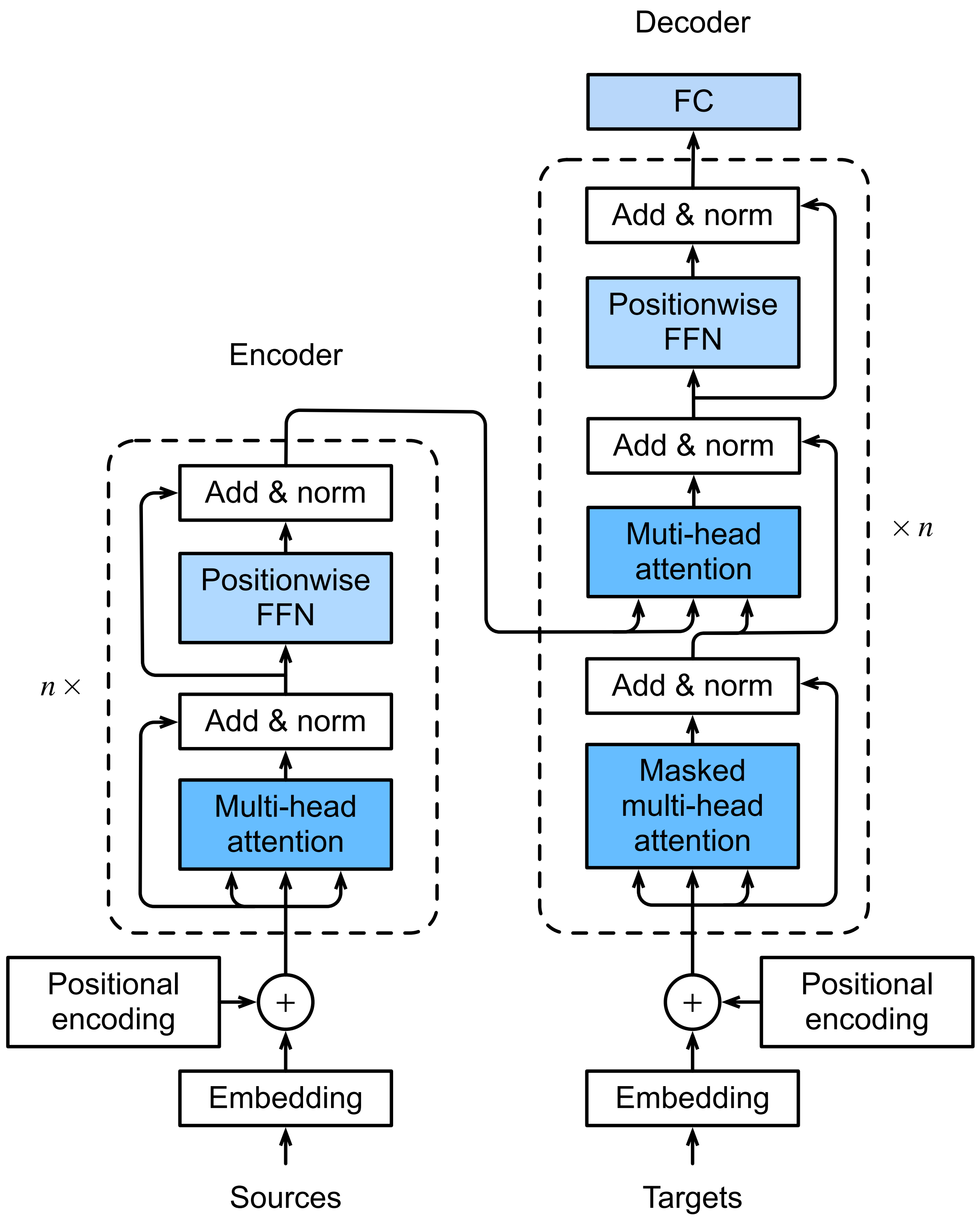


Figure 3.2: Transformer Architecture [32]

With reference to the figure, it is to be noted that the decoder portion of the transformer is only needed when the output of the model is itself a sequence, such as in tasks like text summarization, machine language translation, and closed-book question answering.

Since our problem is instead one of sequence classification, only the encoder portion of the transformer is needed in our type of application. It is for these purposes that the BERT tool (presented and explained in the next subsection) was designed and developed.

For a more detailed explanation of self-attention and the construction of the Transformer as a whole, we refer the reader to the original paper that introduced the concept [32].

Bidirectional Encoder Representations from Transformers (BERT)

BERT is a relatively new transformer-based language model introduced by Google, which has been pushing the bounds of NLP since its release in 2018. It has marked a new era in the field of NLP and is widely considered to be one of the greatest milestones reached in the field. This reputation is bolstered by its state-of-the-art performance in eleven research-standard Natural Language Understanding (NLU) benchmarks (GLUE, which consists of 9 tasks [34]; SQUAD [26]; and SWAG [38]).

BERT has become ubiquitous in NLP applications and research, not only due to its high performance on a large variety of datasets, but also to its ease-of-use. BERT (base) is a massive model employing twelve encoders and twelve bidirectional self-attention heads, which translates into hundreds of millions of parameters that would be impractical for any NLP researcher to tune. However, BERT and models of a similar scale typically employ a "transfer learning" approach, in which the model parameters have been pretrained on a language modeling task with a mas sive corpus of text, before any fine-tuning is to be ultimately carried out for a researcher's downstream task. More specifically, the version of BERT that we have used in this research had been pretrained using two such tasks:

1. **Masked Language Model (MLM)** objective task, in which random words are masked from the input, and the model aim is to predict what these masked words may be, using the rest of the unmasked input. By use of this training task, BERT was able to abandon the hitherto standard training paradigms of only using the words to the left of the target word (or vice versa, only the words to the right of the target word) for prediction purposes. Referring to Figure 3.3, if the i-th token is chosen to be masked, T_i will be used to predict the original token with cross-entropy loss.

2. **The Next Sentence Prediction (NSP)** objective task, in which sentence A is used to predict whether sentence B logically follows it. During the actual pre-training, sentence B was drawn randomly from the text corpus 50% of the time; otherwise it was selected as the actual subsequent sentence to A. Referring to Figure 3.3, C is the BERT node used for the NSP output.

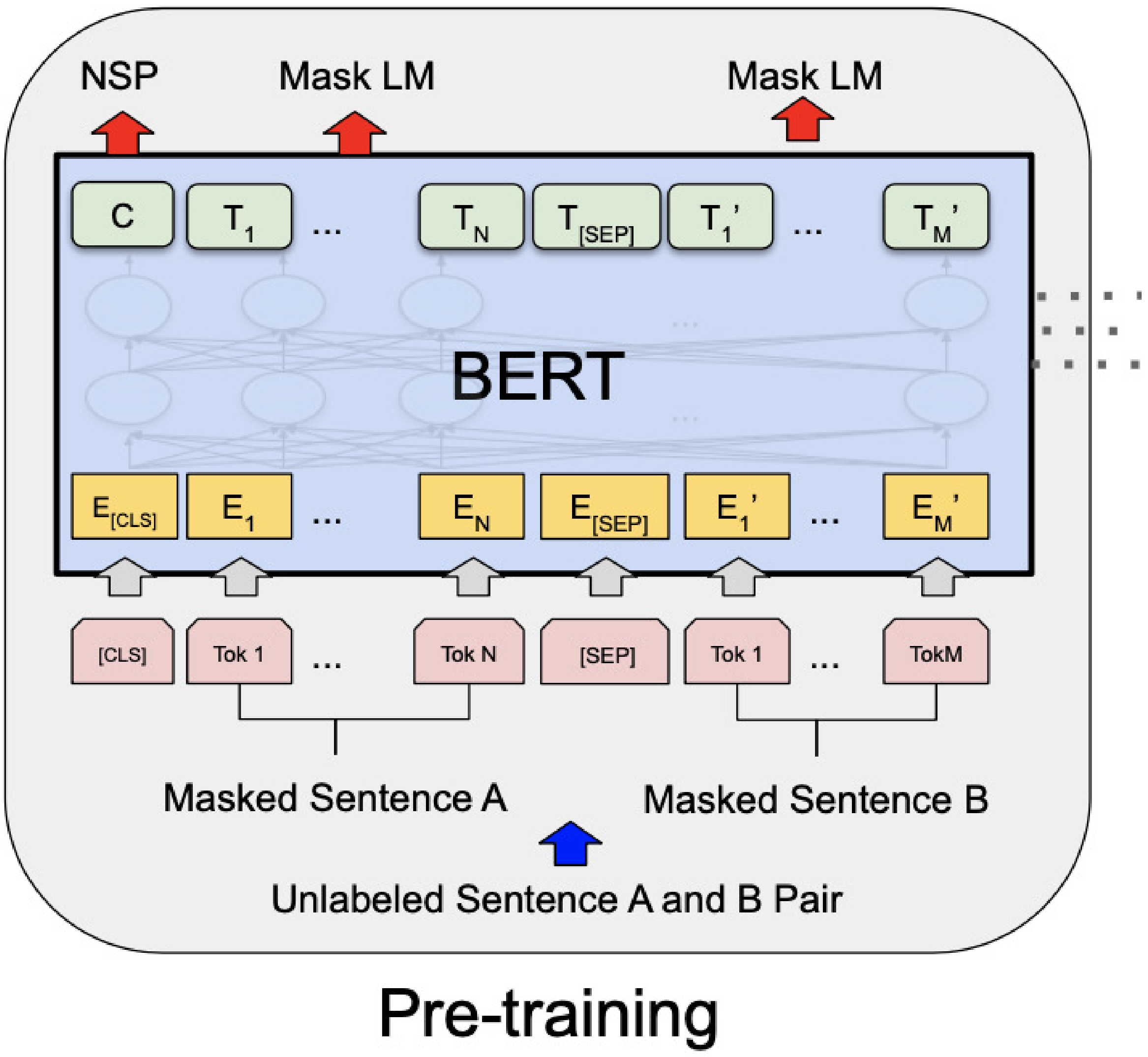


Figure 3.3: BERT High-Level View for Pre-training [32]

3.1.3. Dialects of BERT

Since the initial publication of BERT in 2018, several BERT "dialects" have been subsequently introduced. These either aim to enhance the BERT pretraining process, or address certain specific BERT shortcomings. In the following we provide a brief survey of BERT dialects that we have used in our experiments.

RoBERTa

The authors of RoBERTa (**Ro**bustly **o**ptimized **BERT a**pproach) [18] more thoroughly investigate the pretraining process of BERT. According to the findings of their experiments, BERT was originally significantly undertrained, and its authors overlooked certain potentially useful design choices. The design decisions the RoBERTa developers identify as improvements of their model over BERT in the execution of several standard NLP tasks include:

- Training the model with bigger batches and more data
- Removing the NSP objective training task; i.e., the RoBERTa designers pretrained their model using only the MLM training task.
- Training on longer sequences
- Dynamically changing the masking pattern of the samples with each epoch.

DistilBERT

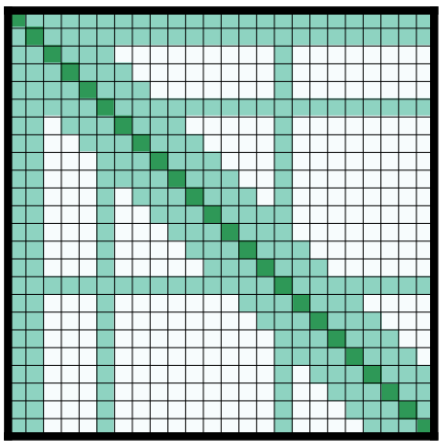
The authors of DistilBERT [27] sought to develop a significantly smaller version of BERT while still retaining most of its natural language understanding capabilities. They leveraged "knowledge distillation" during the pre-training phase to realize a version of BERT that is 40% smaller than the original, but still maintains 97% of the BERT performance on the GLUE benchmark. As a result, DistilBERT can be fine-tuned 60% faster than BERT, which is appealing to NLP practitioners with limited resources. The concept of knowledge distillation closely follows a student-teacher analogy, in which a smaller model (the student) is trained to replicate the behavior of a larger model or of an ensemble of models (the teacher). In this way, an effective form of model compression can be achieved [6].

Electra

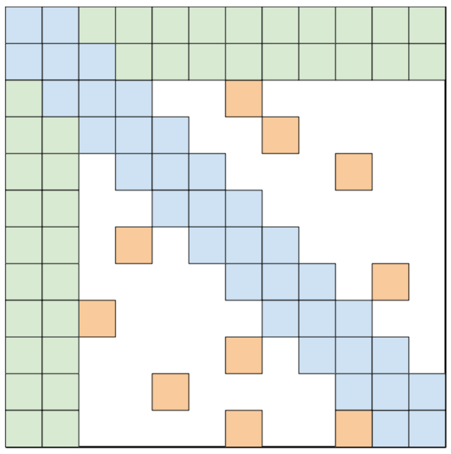
The authors of Electra [7] take inspiration from Generative Adversarial Networks (GANs) [11] to develop a more efficient approach to pretraining than the combination of MLM and NSP used in the development of BERT. Instead of masking the input as done in MLM, the Electra developers instead corrupted the input by replacing some of the tokens with plausible alternatives selected by a small GAN. The discriminator (Electra) is then trained to predict whether each token comes from the original source or was generated by the adversarial network. In this approach, the generator and discriminator networks are working in constant competition with each other. Using this form of pretraining, the authors of Electra were able to achieve cresults that are comparable in quality with those obtained by RoBERTa while using less than one quarter of the processing power, or significantly superior when using a similar amount of computational resources.

BigBird, Longformer

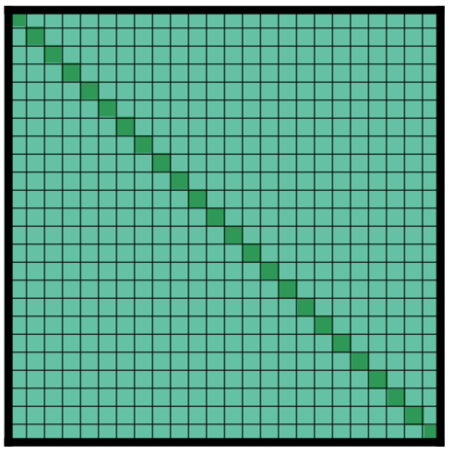
BigBird and Longformer are two transformer-based models that strive to remedy the sequence length limitations of BERT and other BERT dialects. The underlying cause for this limititation is the memory hungry nature of the self-attention mechanism which scales quadratically with the sequence length. The BigBird and Longformer models seek to address this problem by modifying the self-attention mechanism. This is accomplished by “sparsifying” the corresponding self-attention matrix according to an “attention pattern” that identifies a subset of tokens, significantly reduced in size with respect to the original attention set, which is allowed to “attend” to each token being processes. Figure 3.4 below illustrates the attention patterns of both Longformer (a) and BigBird (b) comparing them to the standard full-attention set (c).



(a) Longformer Attention Pattern [4]



(b) BigBird Attention Pattern [37]



(c) Full n^2 Attention (Standard) [4]

Figure 3.4: Attention Patterns

The standard attention scheme is of course the most expressive, as all tokens are allowed to attend to all other tokens. However, as explained above, the effect is that its application does not scale well to long sequences. Longformer and BigBird alike extend the transformer model usability to longer sequences, albeit at the cost of expressiveness, by defining and utilizing both a "window attention" and a "global attention." With window attention, each token can attend to its close neighbors, while, as explained earlier, global attention defines a subset of tokens that can attend to every other token in the sequence being analyzed. BigBird takes this concept one step further than Longformer by additionally also using "random attention," i.e., a mechanism by which each token may randomly attend more tokens in the sequence than allowed by the global attention pattern.

3.2. Additional Technical Background

In this section, we group together and provide summary descriptions of certain non-NLP-specific models and methods that we have used in support of the research of this thesis. Although these methods are applicable for the target NLP task which is the thesis subject, and as such we have used them in our research experiments, they generally have widespread application outside the NLP field. Accordingly, we have deemed appropriate to present them and discuss them in a dedicated section.

3.2.1. Support Vector Machine

A Support Vector Machine (SVM) is a type of supervised learning model that is generally used for classification tasks. Its objective is to find a separating hyperplane that maximizes the margin between the data points of two distinct classes. Maximizing the separation margin has the effect of classifying data points with more confidence. What makes SVMs particularly powerful is that they are able to identifying a separating hyperplane in a new feature space of arbitrary dimensions, based on and evolved from the original definition of features, without actually executing the space transformation. This makes SVMs well-suited to classify data points that are non-linearly correlated in the original feature space.

3.2.2. Pointwise Mutual Information

In general terms, Pointwise Mutual Information metrics can be used as a measure of association between two events. In our type of application, these two events can be the instances of a word or phrase, and their belonging to the successful or unsuccessful class (likeable or non-likeable use of the word or phrase).

The general PMI metric formulation takes the form:

$$pmi(x;y) \equiv \log \frac{p(x,y)}{p(x)p(y)} = \log \frac{p(x|y)}{p(y)} = \log \frac{p(y|x)}{p(y)} \tag{3.2}$$

In our use cases we have applied PMI to understand to what degree a certain word associates with the successful book class or unsuccessful book class. For this specific purpose, we can rewrite the above as:

$$pmi(w;c) \equiv \log \frac{p(w,c)}{p(w)p(c)} \tag{3.3}$$

where $p(w,c)$ denotes the joint probability of observing word w and class c, $p(w)$ denotes the probability of observing w in the whole corpus and $p(c)$ denotes the prior probability of a word belonging to class c.

The formulas provided below define how we can calculate each of these probabilities. Additionally, we apply smoothing such that each word appears in each class α additional times. This is done to force non-zero probabilities, and more generally, to smooth probabilities of words which have very little support [1] in the corpus or subcorpus.

$$\begin{aligned} p(w,c) &= \frac{C(w,c)+\alpha}{n+2\alpha * |V|} \\ p(w) &= \frac{C(w)+2*\alpha}{n+2\alpha * |V|} \\ p(c) &= \frac{n_c+\alpha * |V|}{n+2\alpha * |V|} \end{aligned} \tag{3.4}$$

where α is the smoothing factor; n_c is the number of words in class c; n is the number of words in the entire corpora; $|V|$ is the total number of words of our vocabulary in the corpora; $C(w,c)$ is the number of occurences of the word, w, in class c's subcorpus; and $C(w)$ is the number of occurences of the word, w, in the entire corpus.

3.2.3. Change Point Detection

Change points are abrupt variations in sequence data. These change points are suggestive of a transition that has occurred between states. Change Point Detection (CPD) is the set of techniques developed for the purpose of finding exactly when/where these transitions occur. CPD is a well-studied topic within which several methods and applications have been developed. For our application we have experimented with a very simple algorithm known as CUSUM.

CUSUM

CUSUM is a simple CPD algorithm developed by E.S. Page in 1954 [24]. It involves the calculation of a cumulative sum of the differences between data points and their expected value. It has two formulations, depending on whether a positive anomaly or a negative anomaly is to be detected. If the expected anomaly does occur, then there

[1]Support in this context is a term from Association Rule Learning to denote how frequently the itemset appeared in the dataset

should be a distinct point at which the cumulative sum is at a minimum (maximum) for a negative (positive) anomaly. This point is identified as the point where the change has been detected.

The general formula for detecting the change point of a negative anomaly is as follows:

$$\begin{aligned} S_0 &= 0 \\ S_{i+1} &= \min(0, S_i + x_i - \omega_i) \end{aligned} \tag{3.5}$$

Where x_i represents the i-th sample from a process and ω_i represents its likelihood value which is often taken to be the mean of the data points.

4 | Research Questions

In Chapter 3 we have introduced the Transformer architecture and provided a survey of the current state of the book success prediction problem. In the discussion, we underscored the difficulty of classifying the successful-ness of books, and the limitations of modern neural architectures with regard to processing very long sequential data.

The overarching goal that the research documented in this thesis paper has pursued is how to best utilize BERT for the book success prediction task. In pursuit of this goal, we set out to determine how BERT stacks up against strong baselines of results and see if it can be used to move forward the current state of the art of book success prediction. Keeping this goal as the primary focus of our work, we have sought to explore a set of related subsidiary questions, which are listed in the following.

1. How critical is the choice of the base Transformer model for the target task?

Since the initial development of BERT, many extensions and modifications have been proposed to address its known shortcomings. Many of these models, which still employ transfer learning, claim superior performance over BERT on NLU benchmarks. These claims have provided the motivation for to explore the feasibility of using these BERT-evolved or modified models for our task and determine whether any of them may offer significant performance advantages.

2. What is the best way to pretrain BERT (or BERT-like models) for the target task?

We have mentioned earlier that BERT and its dialects employ transfer learning, whereby their millions of parameters are first trained via a masked language modeling task, before being fine-tuned on the principal downstream task. The developers of BERT used unlabeled data from BookCorpus with 800M words and English Wikipedia with 2500M words for the training task. However, the distribution of style and lexicon between books and Wikipedia articles are very different. Thus it intuitively makes sense to further pretrain Bert using text from sources within the same domain as the actual classification task.

3. Can we preprocess the data to filter out bias and/or noise and improve the prediction?

Bert's limit of 512 input tokens, forces us to train on segments of the books at a time. This causes us to have many training samples for each book which may introduce bias. As we would like to use as much of the books as possible, we will explore techniques that aim to mitigate this bias by preprocessing the dataset.

4. Can the target task execution benefit from carrying out the model training in a multitask setting?

The motivation for exploring this question is drawn from prior research (Maharjan et. al 2017), in which the authors were able to improve book success prediction task execution performance by designing the classifier to identify the book genre at the same time. We have investigated if the same result may hold true in using BERT.

5. How can we best extend the BERT fine-tuning procedure to achieve good results in tackling long-text classification?

With the goal of making a classifier operate on a more wholistic view of the books being analyzed, we have investigated the possible ways in which we can aggregate the embeddings of the book chunks processed by BERT. We have explored two principal methods for doing this:

- Train another sequential input neural network directly on the chunk embeddings.
- Compresses all of the segmented book embeddings into a fixed-sized book embedding for each book and subsequently use them to train another classifier.

5 | Our Approach

5.1. Preparing BERT for the task at hand

One of the main reasons for the widespread success of BERT is that it employs a transfer learning approach. BERT comes out-of-the-box pretrained on 16GB of unstructured English text data, thus it already has an internal understanding of the English language, making it well suited to the processing of English datasets. While this inherent knowledge of the English language is key to its success, we can, in theory, make the BERT model even more suitable for our research objectives, by extending its pretraining onto the sub-domain of English text more specifically pertaining to our task. In our case, we want our BERT model to have a more nuanced understanding of literary prose and poetic verse. This is the motivating idea behind the further pretraining of a transformer-based model like BERT with the target domain of text. Furthermore, because we masked character names in our dataset with the special token '[CHARACTER]' (the reason, for which, is explained in Section 6.3.2), the further pretraining of our model is designed to make the model properly learn the embedding of this newly added token.

5.1.1. Further Pretraining of BERT

We consider two pretraining approaches suggested by Sun et. al (2020):

1. Within-task pre-training in which BERT is further pretrained using the text from the target task dataset.
2. In-domain pre-training, in which BERT is further pretrained using text from the same domain as that of the text of the target task.

Within-task pre-training is the simplest because it does not require acquisition of further textual data, whereas in-domain pre-training does. The advantage of in-domain pre-training is that we can presumably use a lot more textual data, as long as we have sources that come from similar distributions as the target dataset. We have explored both these methods. For the latter, we constructed a new dataset ourselves. The details of this dataset will be explained in Section 6.2.

5.1.2. Training in a Multitask Setting

We have also studied what effect training BERT in a multitask setting has on learning the target task. Because the dataset provided by Maharjan is provided with associated genre information, we can train BERT to predict the genre as well as the success label of the books being analyzed. We note here that this type of multitask learning is different from the one detailed in Sun et. al [30]. In their version of multitask learning, they studied the effect of sharing the knowledge obtained from several related supervised tasks. Thus they used multiple datasets that had similar target tasks (e.g. the sentiment classification of Yelp reviews and IMDB reviews). In our case we use the same text to predict two different labels (the genre and the success).

There are two ways in which we can set up the multitask learning approach.

1. First train BERT on the genre classification task and only then train BERT for the target task of predicting the success label. This method can also be seen as a special form of pretraining in which we perform sequence classification on a task different from the target task using the same explanatory variables (i.e. the text). Since we have not come across this methodology in the literature, we shall refer to it as "other-task pretraining".

2. The perhaps more natural idea is to train BERT to predict both the genre and the success label simultaneously. This idea is inspired by Maharjan et. al 2017 in which they were able to improve their results with such a setting. We take the approach one step further by introducing a hyperparameter α with range from 0 to 1. It allows one to linearly weigh the importance of the genre classification versus the success label classification when computing the loss function. $\alpha = 0$ tells the model to "care" only about the genre classification task whereas $\alpha = 1$ tells the model to "care" only about the success label classification task. The default value, $\alpha = 0.5$ allows the model to equally weigh the importance of each.

5.2. Extending BERT to Sequences longer than 512 tokens

In discussion of the following topics, we find it useful to provide again a high-level depiction of BERT (Figure 5.1), this time in the context of fine-tuning on a sequence classification task. In this context, C, which was used for the next sentence prediction task in the original pretraining, is now used to predict the success label of books (or more accurately, segments of books).

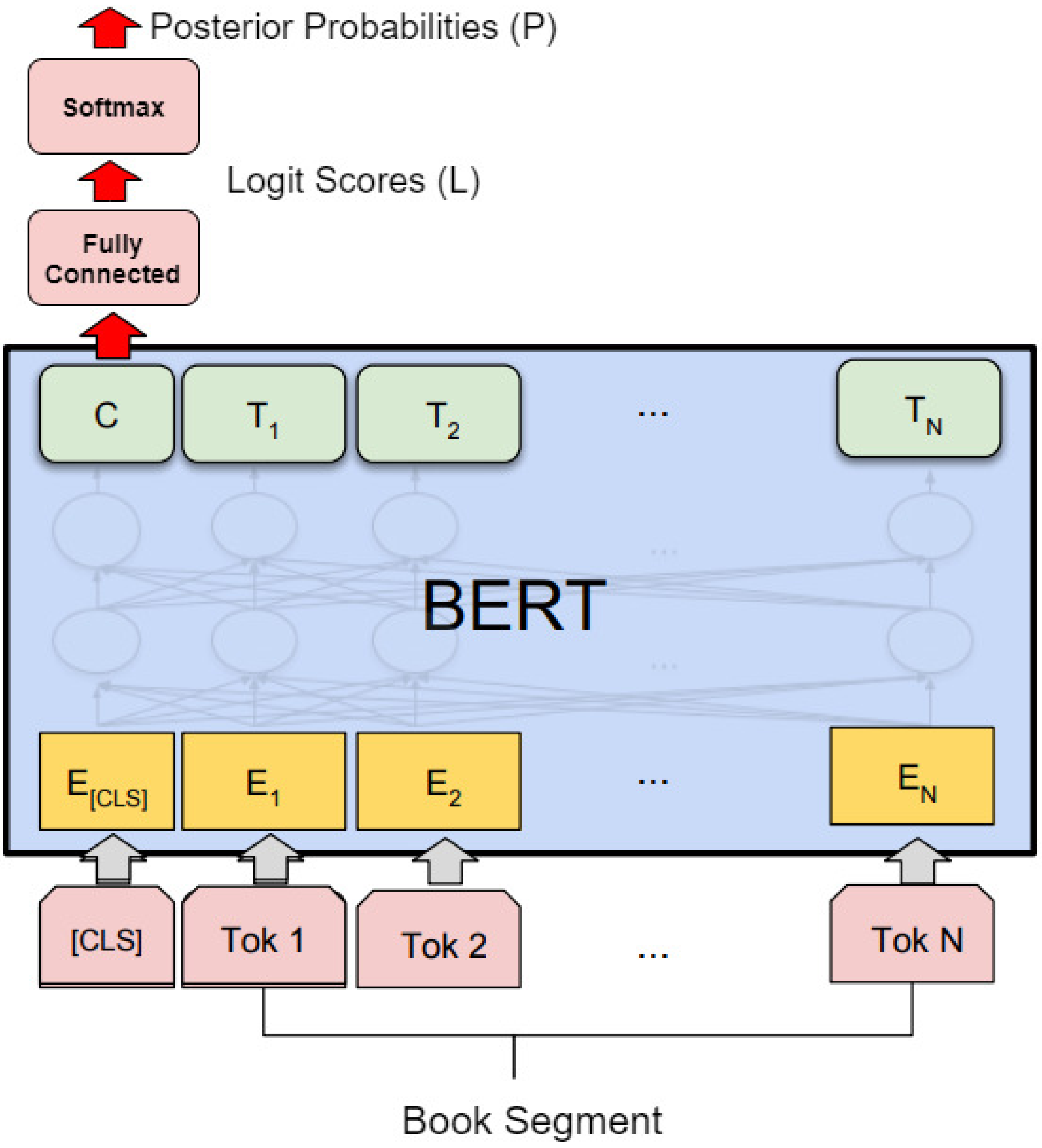


Figure 5.1: BERT High-Level View for Sequence Classification (modified from [8])

There are several techniques one can employ to adapt BERT to work with longer sequences than its internal limit of 512 tokens. The first and most obvious is to simply truncate the data samples to their first 512 tokens. This method has two obvious problems:

- Most books are much longer than 512 tokens thus we are removing a lot of information.
- The sample of each book comes exclusively from just the first couple of pages alone, thus it will, most likely, not be representative of the book as a whole.

Khalifa et. al [16], in their experiment with BERT, split the books into segments ("chunks"), and randomly sampled sentences from each chunk so that the sample would fit into the 512 token limit. While this may better represent the book as a whole, it is conditioned by the randomndess of sampling and still relies on a very small subset of text for the training task.

To overcome the second issue, one can choose to partition the books into segments of 512 tokens and then train BERT using each segment as a data sample. Despite the simplicity of this approach, no prior work that we know of has employed this technique for the task at hand. And although the idea is simple, it is complicated by having to make some design decisions, whose effects need to be investigated, as a result of the chunking. The specific design decisions that we have explored are:

- The choice for a max limit on the number of chunks used for each book to reduce biases of longer books
- The choice of the sampling procedure
- The choice of a tokenization algorithm to split the books into chunks of 512 tokens
- How to coalesce the predictions on the segmented books to come up with the overall prediction on the books

The first two choices presented are prompted by the fact that, because each book will generate a variable number of samples, the resulting dataset that will be used for training grossly violates the i.i.d. (independent and identically distributed) assumption usually applied to the samples. One resulting issue is that within the Goodreads dataset that we use for training, books can be split into from as little as one 512-token chunk to as many as 100. Therefore, the books on the lower side of this spectrum can be severely underrepresented in the analysis, with respect to the longer books. To compensate for this, we have envisioned and applied a compromise approach, by which a middleground between reducing the bias of longer books and using as much textual data to make a prediction is sought by setting an upper limit to the number of chunks used for each book. Moreover, we have conjectured that a more fine-grained control over the order in which data points are sampled and fed to the model may be advantageous. A standard random sampler, for example, may result in the model observing too many samples coming from the same books in quick succession. This problem may be exacerbated by the large discrepancy in the length of the books in our dataset. Therefore we have studied the effect of "forcing" the model to process a segment of all the books before moving on to the next batch of segments consisting, again, of one new segment from each book. We refer to this sampler in later chapters as the "Sequential Book Sampler".

The choice of a tokenization chunking algorithm also requires some consideration. In our experiments, we consider two different algorithms.

- **Sentence tokenizer:** In this algorithm, we first split the book into sentences and then we tokenize the book sentence by sentence, fitting as many tokenized sentences into a segment before it reaches the cap of 512 tokens. If the newly tokenized sentence were to push the segment beyond the 512 token limit, it is instead inserted into a new segment as the segment beginning.
- **Overlap tokenizer:** In this algorithm, we ignore sentence structure altogether. The whole book gets tokenized as a whole and is segmented into chunks of 512 tokens with a variable overlap parameter. The overlap parameter, x, is set such that the last x tokens from the previous segment appear as the first x tokens of the subsequent segment.

Each algorithm comes with their own perceived advantages and disadvantages. The advantage of the sentence tokenizer is that a sentence cannot be split between two chunks, thereby removing the risk of losing important contextual information within each sentence. Apart from being more time consuming, the sentence tokenizer forces the model to disregard any dependencies between neighboring sentences that were placed in adjacent segments. The overlap tokenizer, on the otherhand, minimizes this effect through use of the overlap parameter. The disadvantage of the overlap tokenizer is that it introduces another hyperparameter (the overlap parameter). Another middleground must then be found between training the model on duplicated information and providing more contextual information for each piece of text.

Another speculative issue that comes from training a model on sections of a sample at a time, is that the model could theoretically overfit to repeating distinguishing words that are unique to the sample. In the case of books, we have investigated whether the model may overfit to character names. To explore this concern, we have experimented with a preprocessing step in which we mask character names with the special keyword '[CHARACTER]'. The details of this process is explained in Section 6.3.2.

We have also explored different ways in which the predictions on the segmented books could be coalesced into one prediction for the entire book. Some simple approaches are:

1. Choose the majority class for each book.
2. Softmaxing the average of the logit scores from the output of the classification layer (denoted by L in Figure 5.1) for each book
3. Similar to the previous method, but flipping the operations. Instead of doing what 2 does, this approach takes the average of the softmaxed logit scores of the segments of each book (denoted by P in Figure 5.1).

The first method listed is the most obvious. However, it may not be the preferred choice when working with misbalanced datasets. Moreover, for methods 2 and 3, when working with misbalanced datasets, one may want to find a validated threshold instead of simply using 0.5. Otherwise, the model may be at risk of overpredicting the majority class.

Beyond predicting books using the predictions on their chunks directly, we also explored approaches which use BERT to extract embeddings for each chunk of the books that are then used to train a second stage classifier. These methods are discussed in the following section.

5.3. Second Stage Classifier

While we were able to achieve quite satisfactory results using the class predictions on the segments directly, with that approach our model is not taking into consideration the book as a whole. The approaches explained in this section aim to do just that, by working with the compressed representations of book segments generated by BERT instead of the actual book segments themselves. As an additional benefit of this process, we can generate fixed-sized embeddings of the books as a whole. These can be further used to train an SVM, shallow neural network, or more complex multimodal architectures like Maharjan's Genre-Aware Attention Model.

5.3.1. Simple Models Fit Over Average Segment Embeddings

The simplest idea to coalesce the chunk embeddings into fixed-sized book representations is to simply take the average of the chunk embeddings for each book. These averages which effectively serve as the book embeddings can then be used to fit a Support Vector Machine (SVM) or a Shallow Neural Network.

5.3.2. Hierarchical Sequential Models

RoBERT (Recurrence over BERT) and ToBERT (Transformers over BERT) are two possible extensions of BERT studied by Pappagari et. al [25]. With RoBERT, an LSTM layer is trained on the segment embeddings coming from BERT, whereas with ToBERT, the encoder portion of a transformer is trained on the segment embeddings coming from BERT.

In the original paper, the researchers trained these second-stage neural networks with the embedding of the '[CLS]' token (denoted by C in Figure 5.1), as well as, using the class probabilities (denoted by P in Figure 5.1). In our research, we use the embedding of the '[CLS]' token exclusively as it makes the most intuitive sense and because the paper claims much better performance using them, over using the class probabilities.

RoBERT

RoBERT (Recurrence over BERT) is an extension of BERT which trains one to many LSTM layers over the segment embeddings.

With the LSTM unit as its core building block, we provide a graphical representation of the RoBERT model in Figure 5.2. Here, X_0 to X_n represent the segment embeddings that have already been extracted from BERT.

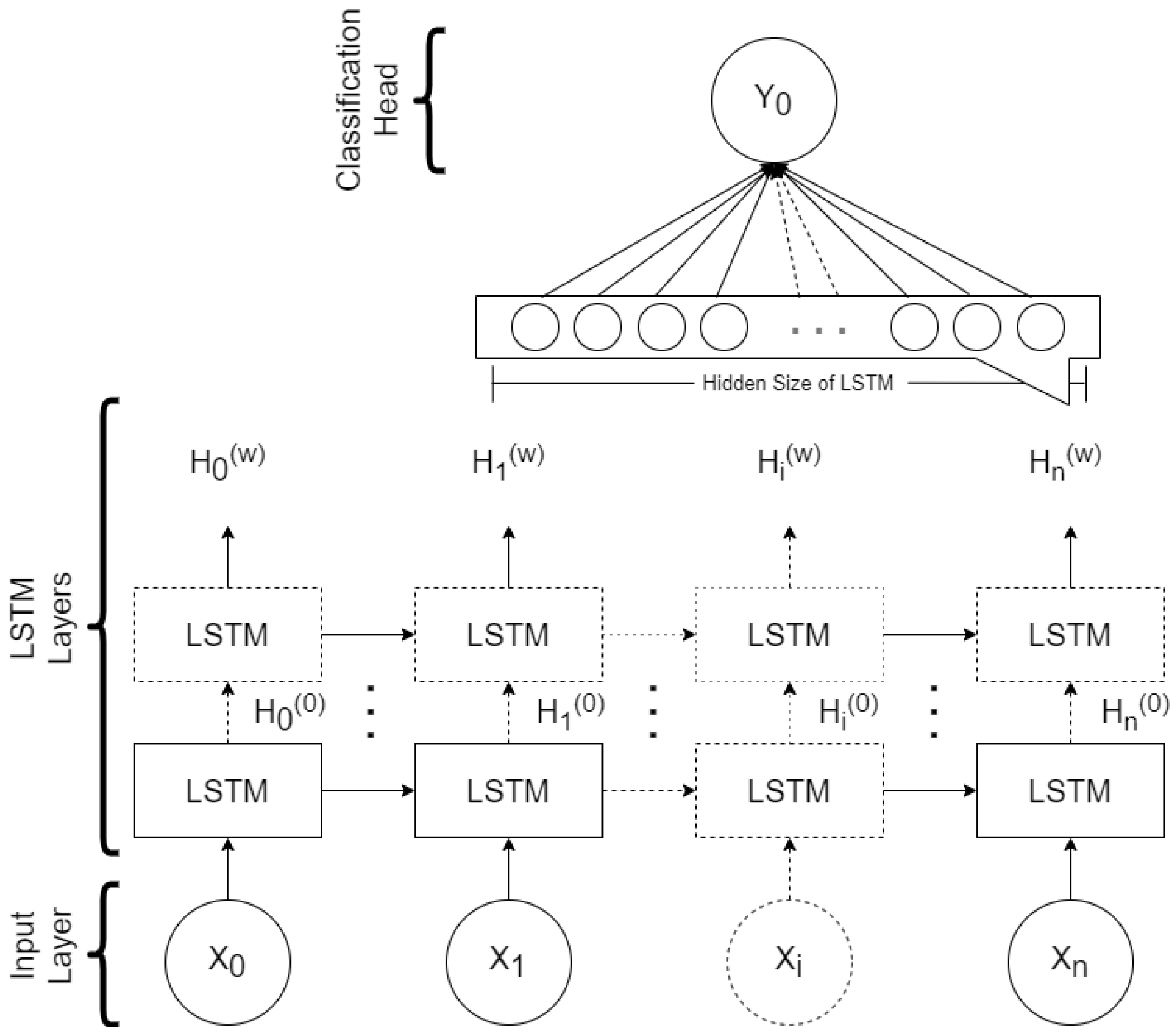


Figure 5.2: RoBERT High-Level Architecture

ToBERT

ToBERT (Transformers over BERT) is another possible extension of BERT, perhaps the most natural as it uses the same neural architecture as BERT: the encoder portion of the transformer. With this model, the same self-attention mechanism that BERT uses (explained in Section 3.1.2) is used to allow the segments of each book to attend to one another. A key difference between ToBERT and RoBERT is that RoBERT uses a left-to-right training paradigm allowing for the positional information of each segment to be ingrained in the architecture. Transformers, on the other hand, are not sequential in nature and on their own, lose all positional information. To overcome this issue, transformer architectures inject positional information in the form of positional encodings. This is done by adding these positional encodings to the input embeddings (which in the case of ToBERT are just the chunk embeddings themselves). In our implementation of ToBERT, we use the same sine and cosine functions that were used in the original paper [32]:

$$\begin{aligned} PE_{pos,2i} &= \sin(pos/10000^{2i/d_{model}}) \\ PE_{pos,2i+1} &= \cos(pos/10000^{2i/d_{model}}) \end{aligned} \tag{5.1}$$

where pos is the position of the chunk and i is the dimension in the chunk. The authors justification for this function is that "it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}. [32]"

In our experiments, we have tested the computational performance both with and without positional encodings.

5.3.3. Multimodal Network

As our world delves more and more into the big data era, the trend of end-to-end deep learning has become increasingly predominant. End-to-end deep learning is the concept of using neural networks to find a hypothesis function that directly maps the inputs to the output. However, in order for end-to-end deep learning to outperform the more classical approach of using handcrafted features, one typically needs a large quantity of data. Since the dataset we use to assess our classifiers contains only 1003 samples, we suspect that additional handcrafted representations can further boost performance. Moreover, because our data samples are very long, current textual end-to-end deep learning methods are not able to effectively consider the whole text. As a result, we have conducted experiments that incorporate our neural features and handcrafted features together through use of multimodal networks.

Multimodal networks are a type of heterogenous artificial neural network that are able to encapsulate multiple and different representations of the data. Unlike other multimodal architectures which aim to use data coming from different sources (e.g. text, sound, pictures, etc.), the features that we will make use of are derived from the text of the books alone. Nonetheless, different textual features may provide varying insights into the data as each has distinct statistical properties. Moreover, a system may benefit from learning joint representations of these different modalities.

In this subsection, we will provide a discussion of the different handcrafted features and neural features that we have adopted, and the architectures of the multimodal networks with which we have conducted our experiments.

Handcrafted Text Features

Below, we divide the handcrafted features into five aspects of linguistic styles. The handcrafted features aim to represent these aspects in a purely quantitative count-based way. The features chosen are those also used in prior work ([2], [19]).

(I) Constituents: To quantify the syntactic style that the books were written in, we compute the normalized counts of the clausal tags in the parse tree of each book. These tags correspond to $'SBAR'$, $'SQ'$, $'SBARQ'$, $'SINV'$, and $'S'$. [1]

(II) Lexical: The goal of lexical features is to correlate the vocabulary in the book to its success. For this purpose, we use char n-grams with term frequency-inverse document frequency (TF-IDF) as the weighting scheme.

(III) Readability: Readability metrics aim to quantify how easy the book is to read. One can imagine that a book with a simple style has the opportunity to appeal to a larger audience and this may have an effect on its success. In fact, Khalifa et al. were able to improve the performance of their neural network by a margin of five points on the weighted F1 metric by incorporating readability metrics. [16]. We use readability metrics calculated in different ways, and their mean normalized values for training. [2]

(IV) Sentiment: We use SentiWordNet 3.0 [3] to compute the average sentiment score across the entire document. SentiWordNet defines a way to quantitatively ex-tract the polarity, sensitivity, attention, pleasantness, and aptitude from sentences. We compute the average of these scores over the sentences for each book, exploring the hypothesis that the average mood of the book may provide some indication of its success.

(V) Writing Density: We hypothesize that metrics providing insight into the length of the book may aid in the prediction task. For example, perhaps readers may struggle to get through long books, thus making them less successful. To this effect, we have computed metrics such as the number of words and characters. We also have computed metrics that give a sense of how dense or complex the writing is (a type of metric similar to but more straightforward than the readability metrics). These metrics include average word length, words per sentence, sentence length, and number of words in the book vocabulary.

[1]Please see A.1 for the meaning of these tags

[2]Please see A.2 for a list of the readability metrics used along with their equations

Neural Features

Since BERT is the focus of this thesis, its book features can be expected to be the most prominent among all others. However, we have also made use of other neural features that permit a different approach to be taken in developing representations of our books.

(I) BERT Features: We generate book embeddings from our BERT-based model using the same approach described in Section 5.3.1.

(II) Doc2Vec (DMM) Features: We use Doc2Vec [17] to train a distributive memory (DM) model with a summing of context word vectors (DMM) for 50 epochs. Doc2Vec is a neural network algorithm that learns fixed-length feature representations for variable length pieces of text. These feature representations were trained to best predict words in the document being considered. Doc2Vec has a few different configurations. We adopted DMM per the findings of the research done by Maharjan et. al [19] who found this setting to be the most beneficial. For more information about Doc2Vec, we refer the reader to the original paper [17].

Feedforward Standard Concatenation

To fuse together the different features we have gathered from our books, we first took a very simple approach. We designed a classical feedforward neural network in which each set of features pertaining to the same type are fed through their own feed-forward network (1-2 layers in depth depending on feature size), and are then concatenated together and passed to a classification head. The network is then trained end-to-end. Since our dataset contains only about 1000 samples, we opted to keep the network relatively shallow. This network provides us with a term of comparison with the more complex Genre-Aware Attention model.

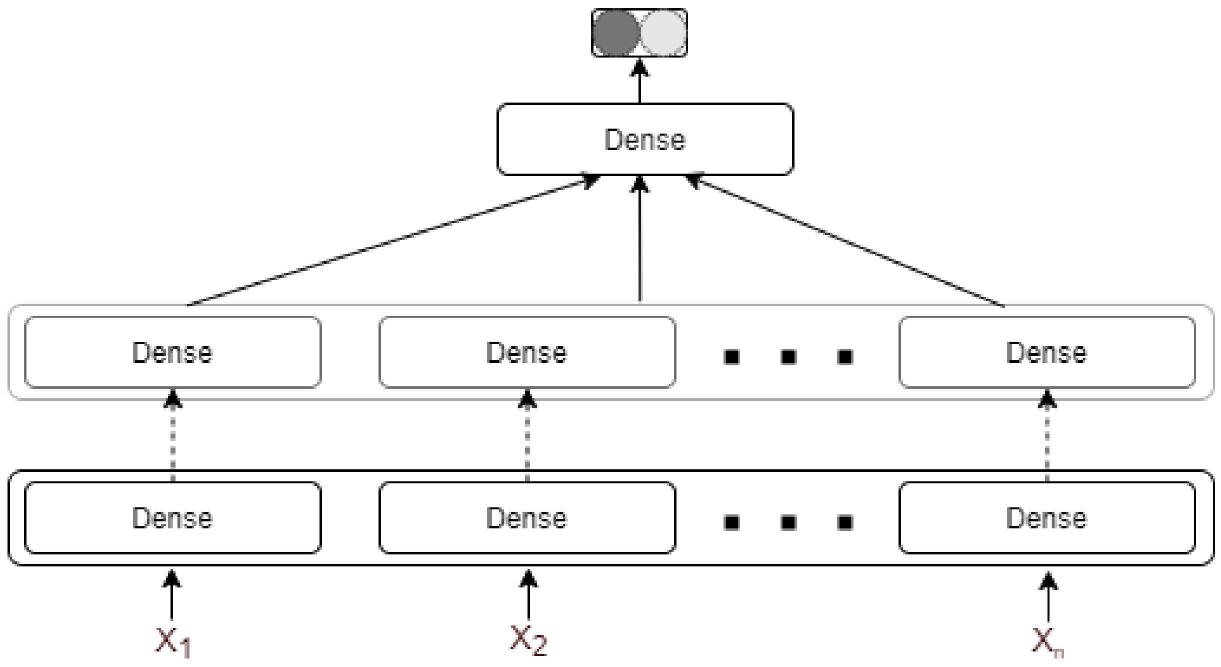


Figure 5.3: Feedforward Standard Concatenation

Figure 5.3 is a schema of our Feedforward Standard Concatenation network where X_i represents the i-th set of features used to train the network (e.g. BERT features, readability metrics, char-5-grams, etc.).

Genre-Aware Attention Model

The genre-aware attention model developed by Maharjan et al. [21] holds the state-of-the-art performance on the Goodreads dataset. For the actual training, they used all the best features from each category where 'best' is determined by that feature's performance when training the network in isolation. Their results have provided proof that a classifier has much to gain from features encapsulating specific aspects of the dataset.

The Genre-Aware Attention Model is more complex than the previously presented multimodal network for two main reasons:

- The model learns weights of how much "attention" it should provide to each feature.
- The model learns a genre embedding that gets baked into the overall book embedding.

In Figure 5.4 below, we show the model schematic provided in the original paper [21].

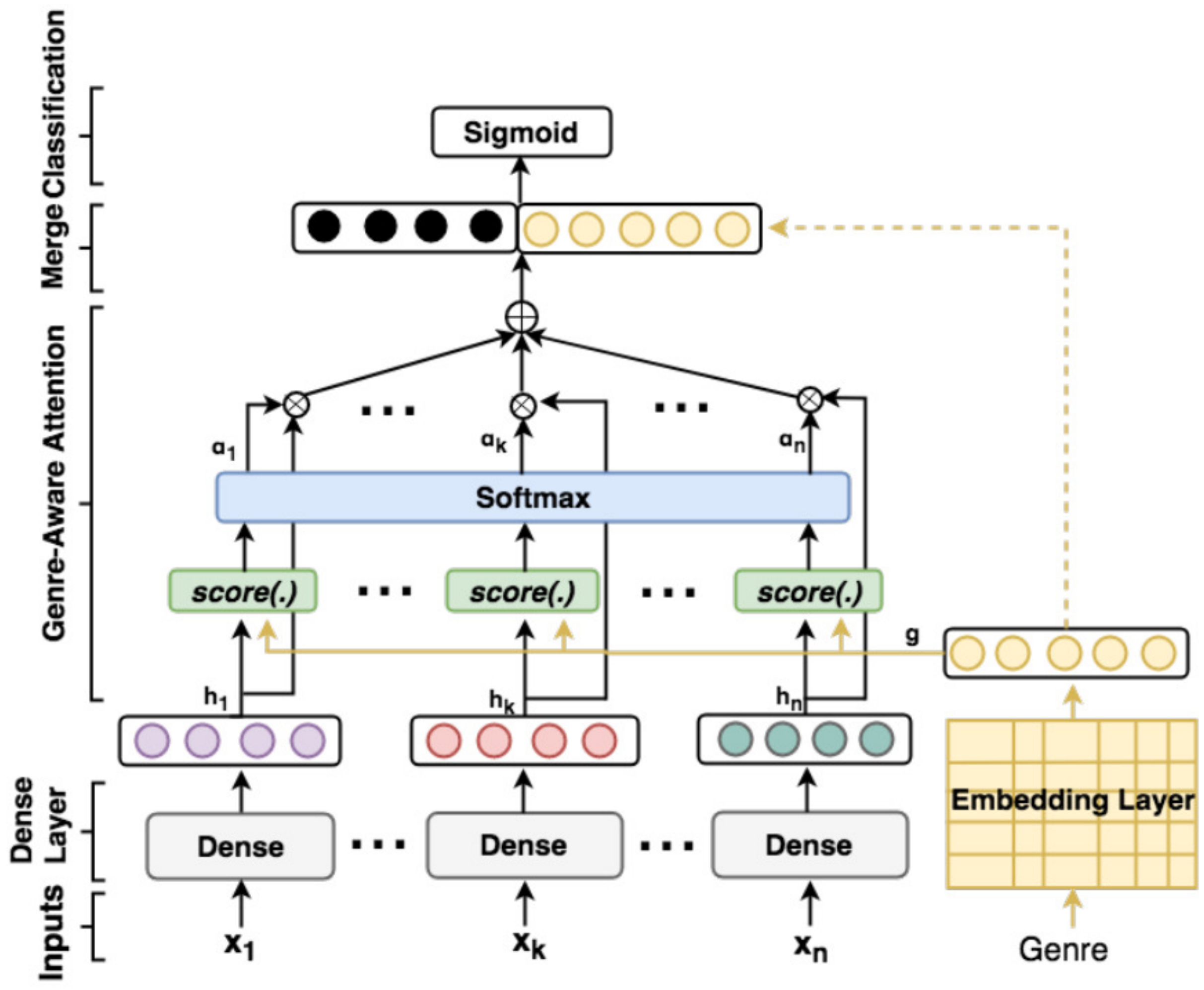


Figure 5.4: Genre-Aware Attention Model [21]

$$h_i = selu(W_h x_i + b_h) \tag{5.2a}$$

$$g = genre \times W_g \tag{5.2b}$$

$$score(h_i, g) = v^T selu(W_a h_i + g + b_a) \tag{5.2c}$$

$$\alpha_i = \frac{exp(score(h_i \cdot g))}{\sum_{i'} exp(score(h_{i'}, g))} \tag{5.2d}$$

$$r = \sum_i \alpha_i h_i \tag{5.2e}$$

$$\hat{p} = \sigma(W_c r + b_c) \tag{5.2f}$$

As in the model presented earlier, in Figure 5.4, x_i represents the feature representations coming from the i-th modality. As the features have different dimensions, they are first projected into a space with the same dimensions via a dense layer (not formulated above).

Then they are each passed through the same linear layer with a selu activation (5.2a). At the same time, we use the one-hot genre encoded vector to retrieve the learned genre embedding (5.2b). The feature embeddings at this stage (h_i) are then passed through another linear layer that bakes together the genre embeddings (5.2c). The outputs are then passed through a softmax function to get the attention weights (5.2d) which are used to incorporate the appropriate fraction of information from each feature (5.2e). At this point, the learned genre embedding is optionally concatenated to r before it is passed to a non-linear layer with sigmoid activation to obtain the class probabilities (5.2f).

6 | Datasets

In this chapter, we present the datasets used and the various preprocessing steps implemented to prepare them for utilization in model training. In particular, we discuss the dataset provided by Maharjan et. al [19] and the dataset we have procured ourselves. The dataset created by Maharjan et. al is used to both pretrain BERT and assess the performance of our classifiers, allowing us to compare our results with prior work. The dataset we assembled ourselves, on the otherhand, was used strictly to further pretrain BERT.

All datasets discussed used Project Gutenberg, an online library of over 60,000 books that are in the public domain. Thus, most books on Project Gutenberg are old books whose copyright has expired.

6.1. Dataset for the Classification Task

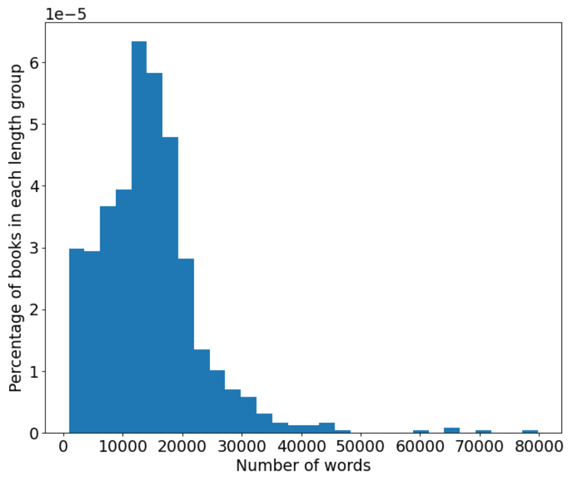
As just mentioned, we have used the dataset catered by Maharjan et. al to conduct our experiments and assess our results in comparison with prior research. We refer to this dataset as the "Goodreads Maharjan" dataset. Ashok et. al also proposed a dataset, however, their criteria for success depended on the download count each book had on Project Gutenberg. Maharjan et. al's dataset, on the otherhand, uses the book ratings from Goodreads, which we deem to be a more viable measure of the success of the books. Goodreads is a social-cataloging website that allows users to search its database of books and write reviews. The site is generally regarded as an online community of bibliophiles with the main focus being that of discussing books and getting book recommendations. Because of the assumption made on the demographic of its users, and the fact that the reviews pertain solely to the books themselves, we believe that the data is less noisy than other online alternatives. As a comparative example on this aspect, in Amazon reviews the users may leave a negative book purchase rating for reasons that do not pertain to the content of the book, but to aspects of the purchase itself, such as the shipping being slow, or the book being delivered in a damaged condition.

In an effort to reduce noise, Maharjan et. al used only books that have had at least 10 ratings. Moreover, in order to reduce authorship bias, they allowed an author to appear in a maximum of two books in the dataset. They used an average rating of 3.5 as the threshold for success: i.e., if its average rating was less than 3.5, a book was labeled as unsuccessful; otherwise, it was labeled as successful. In the end, they used 1003 books across 8 different genres. In Table 6.1, we provide the distribution of both the genre and the success label in the dataset.

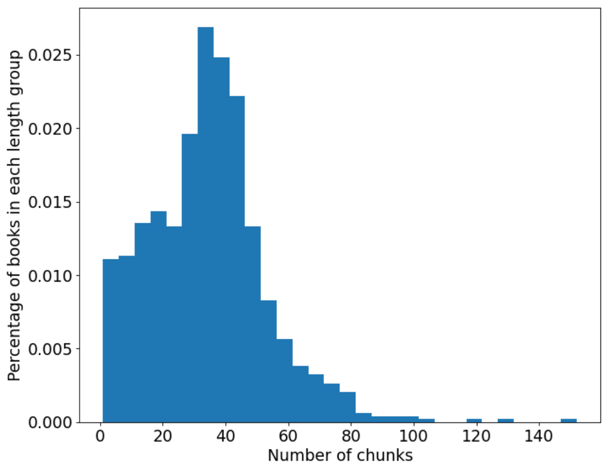
Genre	Unsuccessful	Successful	Total
Detective Mystery	60	46	106
Drama	29	70	99
Fiction	30	81	111
Historical Fiction	16	65	81
Love Stories	20	60	80
Poetry	23	158	181
Science Fiction	48	39	87
Short Stories	123	135	258
Total	349	654	1003

Table 6.1: Genre Distribution of Goodreads Maharjan Dataset

As the length and the variance in length of the books is a cumbersome aspect that needs to be overcome when applying BERT to the dataset, we show in Figure 6.1 histograms of the length of the books both as a number of words and as a number of 512-token chunks (using the BERT tokenizer with no overlap).



(a) Word Count Histogram



(b) Chunk Count Histogram

Figure 6.1: Length Histograms for Preprocessed Goodreads Dataset

We note that the lengths depicted in Figure 6.1 above are for the books after they have already been trimmed by the preprocessing step discussed in 6.3.1.

6.2. Datasets for Pre-training BERT

As part of our work, we looked into building a larger version of the Goodreads Maharjan dataset using the same resources (Project Gutenberg and Goodreads). However after filtering candidate selections to remove duplicate authors, books with too few ratings, and books in an undesirable genre, we were left with too few samples to justify creating another classification dataset. Instead, we opted to develop a dataset without labels, to be used strictly for the further pretraining of BERT in an in-domain context. The goal is simply to gather a large quantity of textual data whose distribution is similar to that of the dataset used for the classification task. To satisfy this criteria, we selected books pertaining to the genres contained in the Goodreads Maharjan dataset. We note that, since we are using the same source (Project Gutenberg) as the Goodreads Maharjan dataset, the books in our dataset are still older books that are in the public domain.

Since Project Gutenberg does not include genre information directly, we have used the genre labeling scraped from Goodreads. We opted to simulate Maharjan et. al's filtering process of selecting only the books that pertained to the eight genres previously listed in Table 6.1, and allowing each author to appear a maximum of two times. For more details regarding the procurement of this dataset, please see appendix B.1. In the table below, you can see our distribution of genres.

Genre	Count
Detective Mystery	181
Drama	52
Fiction	1254
Historical Fiction	108
Love Stories	74
Poetry	410
Science Fiction	326
Short Stories	203
Total	2608

Table 6.2: Genre Distribution of pg-2600 Dataset

With a total of 2608 books, we were able to increase the amount of potential pretraining by 260%, going from a within-task pretraining schema to an in-domain pretraining one. We refer to this dataset in later sections as the "pg-2600 dataset" (**P**roject **G**utenberg **2600** books).

6.3. Dataset Preprocessing

6.3.1. Noise Trimming

The text files provided by Project Gutenberg contain a lot of information that do not pertain to the books themselves. Via manual inspection of the files, we noticed that many books began with a copyright note, information about the translation, a note about the author, an ASCII cover page, the table of contents etc. We wanted to ensure that our classifiers were trained on the actual literary prose alone (or verse, in the case of poetry). To achieve this, we have investigated the following two approaches to prune the unwanted information from our dataset.

1. Noticing that the books often began after a line stating only the title of the book, the first approach considered was to search for the first occurrence of the book title and remove all the text that came before.

2. We also noticed that the frequency of the newline characters changed greatly from the beginning of the text files and when the book actually began. Thus, the second approach was based on the use of a simple change detection algorithm to identify exactly where the newline character frequency changes and prune the text before it.

Since the latter method, although still simple, is more sophisticated, we explain its details in the following subsection.

Change Detection of Newline Characters

In order to effectively detect the change in the frequency of newline characters, we employed a classical CUSUM [24] sequential analysis technique. We were interested in detecting a "negative anomaly" as the frequency of our newline characters starts off high and then is significaly reduced once the book actually starts. Referring to equation 3.5, in our formulation, x_i represents the positional difference between two subsequent newline characters and w_i is simply taken as the mean of all the x_i's. The perceived change point occurs at the index of the minimum of the S's. This can be intuitevly visualized by making a plot of the values of S. Also, since we can be quite certain that the change occurs early on, we only look for a change in the first 20% of the text file. In Figure 6.2, we provide a sample plot of the change point in newline frequencies for one of the books. The red line marks a distinct point in which the frequency of newline characters changes.

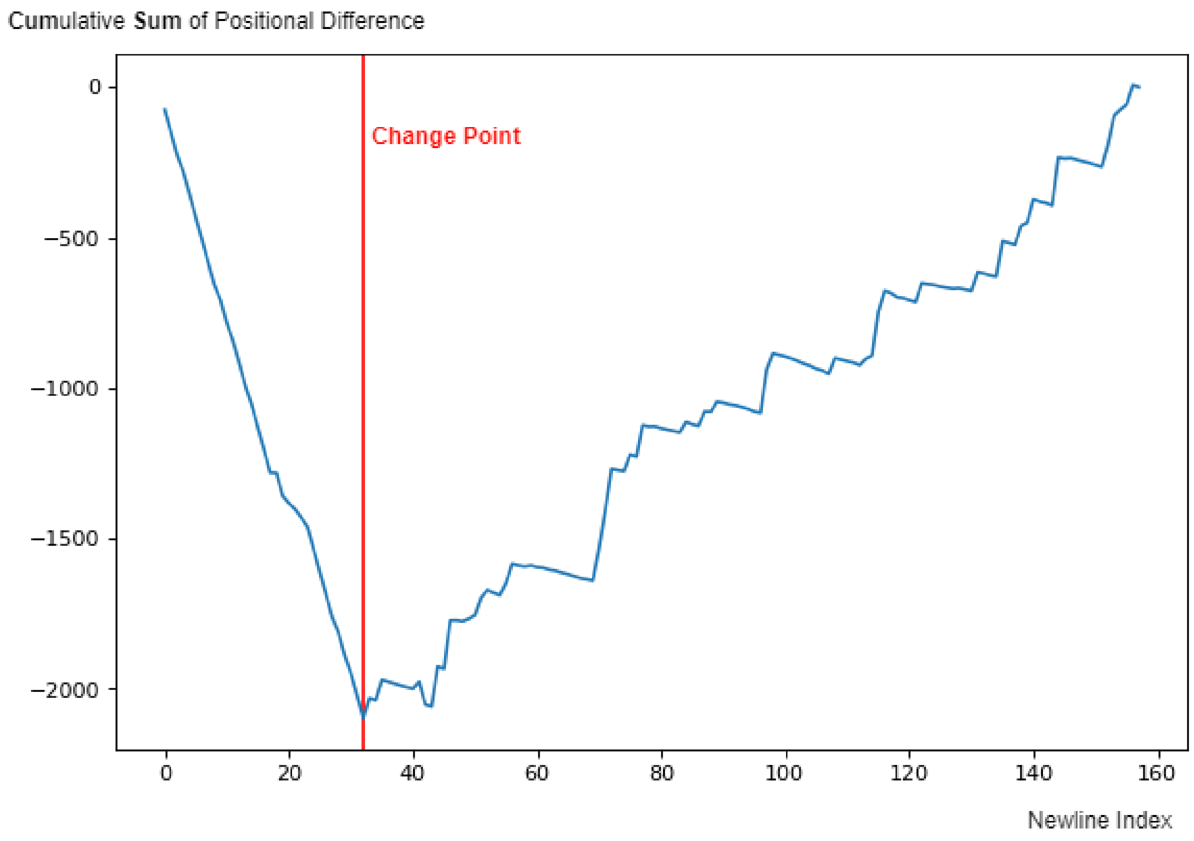


Figure 6.2: Change Point Detection for 'In the Shadow of the Glen' by J.M. Synge

To assess and compare the results of the change detection approach with that of the book title search, we manually inspected the performance of each method on a random sample of the same 50 books. We classify the performance into three categories:

- **Excellent**: The trimming occurred exactly where desired.
- **Suboptimal**: The trimming occurred within a couple sentences of where desired.
- **Poor**: A good chunk (more than 3-4 sentences) of the book was erroneously trimmed or a significant amount of the noise was kept.

	Performance		
Method	Excellent	Suboptimal	Poor
Title Search	22	15	14
Newline Freq.	44	3	4

Table 6.3: Performance Count of Trimming Methods

As the change point detection algorithm performed quite well on these 50 samples, and much better than the title search method, we adopted it to trim our datasets.

6.3.2. Masking Character Names

Because our books need to be separated into segments of 512 tokens for the training of BERT, we deemed likely that the model may overfit to unique book identifiers. In particular, we feared that the model may choose to give an undue amount of attention to character names while ignoring more important qualities such as the book style or vocabulary choice. Before committing to masking the character names, we first checked that character names are indeed highly discriminative between the two classes by using Pointwise Mutual Information.

We set the smoothing factor, α, to 10 and Pointwise Mutual Information confirms our suspicions that character names are highly discriminative between the two classes. In Table 6.4, we provide the most discriminative words for each class along with their corresponding PMI scores. As we can see, they seem to be almost exclusively character names. Please note that the PMI scores are significant in terms of understanding the presence of a word in a given class, however their numerical value can only be used for comparing the discriminating value of words within the same class, not across classes.

We proceeded then to explore the effect that masking character names has on training BERT. To perform the character masking, we used a BERT-large model that has already been fine-tuned on the named entity recognition task using the CoNLL-2003 Dataset. The model was made available out-of-the-box by Huggingface. To perform the actual character masking, we passed all books in our dataset through the model, sentence by sentence, and for each word or group of words that were understood to be a 'Person' entity, we masked it with the key word '[CHARACTER]'.

We then used Pointwise Mutual Information on this further preprocessed dataset to gain insight into the level of effect the character masking technique had on our books. In Table 6.5 we provide the most discriminative words in each class along with their corresponding PMI scores for the character-masked dataset.

On inspection, one can see that a lot of the character names previously appearing in Table 6.4 are no longer present, thus providing evidence that the named-entity recognition model used was effective. However, we noticed that character names were still, mostly, the most discriminative words in our dataset; hence, the model was not able to detect all character names.

Successful Class		**Unsuccessful Class**	
Word	**PMI Score**	**Word**	**PMI Score**
heav	0.4830	jonson	1.6915
thir	0.4746	doul	1.6911
vanslyperken	0.4697	forrester	1.6822
spake	0.4671	darrell	1.6798
daoud	0.4656	pemberton	1.6655
chad	0.4656	olof	1.6639
retief	0.4654	pepita	1.6632
hiawatha	0.4644	milner	1.6614
evelina	0.4621	dorriforth	1.6603
bernick	0.4617	parr	1.6472
jimmie	0.4611	hillcrist	1.6467
mosby	0.4601	mitchener	1.6467
smallbones	0.4598	rynason	1.6433
jerry	0.4588	garin	1.6433
troy	0.4579	lenora	1.6397
stockmann	0.4571	ransford	1.6391
artagnan	0.4560	theodora	1.6380
anna	0.4532	honath	1.6378
nattie	0.4528	petter	1.6371
gan	0.4525	kieran	1.6351
brett	0.4523	tarling	1.6337
cornelia	0.4500	norgate	1.6330
hilary	0.4494	heredith	1.6323
patty	0.4491	loudwater	1.6316
barney	0.4468	wycherly	1.6312

Table 6.4: 25 Most Discriminative Words for Each Class Before Character Masking Using PMI Metric

Successful Class		Unsuccessful Class	
Word	**PMI Score**	**Word**	**PMI Score**
heav	0.4787	doul	1.6713
thir	0.4743	jonson	1.6643
spake	0.4669	hillcrist	1.6362
troy	0.4565	winsor	1.6282
gan	0.4534	ermyntrude	1.6232
oedipus	0.4466	royster	1.6158
dicaeopolis	0.4453	cler	1.6117
tesman	0.4398	strammfest	1.6073
anna	0.4388	ventidius	1.6062
lysistrata	0.4299	mitchener	1.5976
wor	0.4268	inca	1.5966
pg	0.4246	mommy	1.5859
solness	0.4246	lof	1.5815
keith	0.4240	balsquith	1.5781
barabas	0.4240	rling	1.5764
wallenstein	0.4216	levis	1.5738
lhari	0.4216	darrell	1.5728
thro	0.4196	pacha	1.5722
yow	0.4194	lenora	1.5710
repeller	0.4178	_king_	1.5652
spear	0.4174	dorriforth	1.5611
morn	0.4153	marquise	1.5599
som	0.4141	mos	1.5586
wel	0.4135	hornblower	1.5568
yo	0.4128	timmy	1.5564

Table 6.5: 25 Most Discriminative Words for Each Class After Character Masking Using PMI Metric

7 | Experiments

As with many deep learning projects, in a task like the one at hand one can quickly be overwhelmed by the number of decisions that need to be made and the combinatorial number of permutations of those decisions that can theoretically be explored. This is especially the case with our setup, in which we are faced with many task-specific decisions, beyond the standard model design decisions common to all deep learning or machine learning projects. The list below serves to better identify and define this problem in our context. In the compilation of the list, we distinguish between first and second stage classifiers, whereby the former concerns the training of our BERT (or BERT-like) model and the latter concerns the training of models that make use of embeddings generated from the former.

Decisions concerning the development of the first stage classifier:

- Whether to use the character-masked dataset or not
- Choice of the base transformer model (BERT, DistilBERT, Electra, Roberta, etc.)
- What kind of further pretraining should be done (None, In-domain, Within-task, Both In-domain and Within-task)
- Should the model be trained in a single-task setting or a multi-task setting with the genre?
- Whether to use the sentence tokenizer or the overlap tokenizer
    - In case of the overlap tokenizer, how much overlap?
- What is the max number of segments allowable for each book

For the development of the second stage classifier:

- Which first stage classifier should be used to generate our BERT-based neural embeddings
- For the training of the SVM / Shallow NN, how can we best generate book embeddings from their chunk embeddings

- For RoBERT and ToBERT, how much overlap should be used to generate the chunk embeddings
- For multimodal approaches, which set of features should we use together with our BERT-based embeddings

Note that in addition to the decisions in this list, we also had to address the complexity of tuning standard model hyperparameters that define the widths and depths of the model architectures at each layer, and the learning processes of these models.

To overcome the complexity of the questions identified in this section, we adopted a very straight-forward approach, which consisted of working on one issue at a time, keeping all other hyperparameters fixed to a default value. Once we had quantified which hyperparameter value or setting works best for that issue we used it in the training of our final model. For the most part, for sake of simplicity we assumed independence of these tuning variables. However, there are certain cases where this assumption of independence is inappropriate. For example, the decisions of choosing the masked character dataset or not and of choosing whether to further pretrain the model or not are clearly dependent on one another. By adding a special token to denote the presence of a character in the character-masked dataset, further pretraining is needed to learn the embedding of this token. In such cases, we explored these interdependencies.

The overall structure of this chapter begins with a description of our experimental setup and baselines. Following that are our experiments and results of the first and second stage classifiers respectively.

7.1. Setup for Conducting Experiments

Google Colab

For our experiments we used Google Colab which provides a Jupyter notebook environment that runs on Google's servers. It is very popular among Machine Learning researchers as it grants users access to Google's powerful GPUs. Most of our models were trained on a Tesla T4 or a Tesla P100 depending on what was allocated to us at the time of training.

Train, Validation, and Test Splits

Before carrying out our experiments, we first set up a fixed training, validation, and test split on the Goodreads Dataset. Given the magnitude of the BERT model and the limited computation resources available to us, we opted to use hold-out validation to assess our models even though k-fold validation is more statistically reliable. In addition to the dataset, Maharjan et al.'s work also provided us with the information defining their 70/30 train and test dataset split, which maintains the distribution of classes per genre. We used the same split so that we have a consistent basis of comparison with prior work.

We have then taken the train set and made an 80/20 split to get our true training set and validation set. As we want our validation set to have a similar distribution to that of the test set, we ensure that the distribution of genres and success labels in each genre is roughly the same across the two sets.

Our training set, validation set, and test set have 555, 139, and 290 samples respectively.

Hyperparameter Tuning

For the standard hyperparameters that pertain to the model architecture or to the learning process, we performed hyperparameter tuning using the "Median Stopping Rule" scheduler on all of our neural networks. The Median Stopping Rule employs the simple strategy of ending a training process if its performance falls below the median of all other concurrent trials at similar points in time [10]. For the training of the stage 1 classifiers, we only performed hyperparameter tuning for the very last step; that is after we had already conducted experiments to decide all other non task-specific hyperparameters (what version of the preprocessed dataset we will use, which tokenizer we will use, etc.). Ideally, we would perform hyperparameter tuning at all steps, however transformer-based language models are typically very large and we did not have the resources to perform such thorough experimentation. When we did not perform hyperparameter tuning, we used the default values that are common throughout the literature [8].

7.2. Evaluation Metric

In the evaluation of our models we have used the weighted F1-score (w-F1). Since the dataset is misbalanced, the weighted F1 score is used to give each class an equal representation in the metric.

This choice follows the example in the original paper where the Goodreads dataset was first described and used [19]. Thus, we also chose it to allow for easy comparison with prior work.

7.3. Classifiers for Comparison

Before beginning the discussion on our experiments and their performances, it is first important to establish baseline classifiers and the state-of-the-art classifier. These baselines provide comparative meaning to the results our classifiers obtain. They give us a basis of comparison to understand if our models are learning at all or if they can generalize better than simpler approaches. Moreover, to gain insight into the competitiveness of our models, we were interested in comparing their performances with the best models documented thus far in the literature.

"Dummy" Baseline Classifiers

Comparing our performances to that of the most naïve of classifiers is useful to ensure that our models are even capable of learning at all.

We have used two dummy classifiers:

- **Most Frequent:** always predicts the most frequent label in the training set
- **Stratified:** generates random predictions in accordance with the training set's class distribution

Strong Baseline Classifiers

The "dummy" classifiers show us if our models are capable of generalizing at all, but they do not provide us any insight as to whether our models are actually any good. For this reason, we also considered the performance of more classical classifiers. Additionally, we included a simple approach, using BERT as a baseline to gain inisight as to whether the BERT training procedures we have investigated are worthwhile. These baselines have provided us with an indication as to whether our more sophisticated neural architectures and training approaches are effective.

The simple classifiers that we have used as baselines are the following:

- **Bag of Words Logistic Regression:** Extracted features using a simple bag of words model and used them to train a logistic regression classifier with L2 regularization. We used 5-fold cross-validation with grid search to fine-tune the regularization strength.
- **Tf-idf Logistic Regression:** Extracted features using a Tf-Idf vectorizer and used them to train a logistic regression classifier with L2 regularization. We used 5-fold

cross-validation with grid search to fine-tune the regularization strength.

- **Doc2Vec SVM** We used Doc2Vec [17] to generate embeddings for our documents and then trained a hyperparameter-tuned SVM. For this we used the same Doc2Vec configuration explained in Section 5.3.3.
- **BERT One Randomized Chunk [16]:** Khalifa et al. fine-tuned a BERT uncased base model with one segment per book. To generate the segment, they split each book into 50 chunks and randomly sampled a sentence from each to get 50 representative sentences of the whole book.
- **Word2Vec RNN [19]:** Maharjan et al. developed a strategy to overcome RNNs difficulty with representing long sequences. They split each book into chunks of 128 sentences and represented each sentence with the average of the Word2Vec [23] representation of its constituent words. Each chunk was used as a sample to train the RNN in a multitask setting with the labels deriving from the corresponding book.

State-of-the-Art Classifier

We were also of course interested in seeing how our models would stack up to the best one studied so far.

- **Genre-Aware Attention (all best handcrafted + RNN) [21]:** Maharjan et al. trained their genre-aware attention model with the genre embeddings concatenated to the feature embeddings ('r' in 5.2e). They used the following features to achieve the state-of-the-art results:
    - Word Bigram
    - 2 Skip 2 gram
    - Char 3 gram
    - Typed mid-word 3 gram
    - Clausal
    - Writing Density (WR)
    - Sentic Concepts and Scores (SCS)
    - Book2Vec (based off of Doc2Vec [17])
    - RNN

Some of these features are the same as the ones we have used and described in 5.3.3.

Performance of Baseline and SotA Classifiers

Table 7.1 presents the performance of these baseline and state of the art classifiers on the Goodreads dataset by means of the weighted F1 score.

	Baseline	**W-F1 on Test Set**
Dummy	Most Frequent	0.506
	Stratified	0.542
Strong	BERT One Randomized Chunk	0.660
	Bag of Words Logistic Regression	0.665
	Tf-idf Logistic Regression	0.670
	Word2Vec RNN	0.686
	Doc2Vec SVM	0.691
SotA	Genre-Aware Attention	0.754

Table 7.1: Performance of Baselines

The *Most Frequent* classifier performs at just about 50% despite there being many more positive samples than negative ones. This is because the weighted F1 metrics values the classification of the negative samples more than the classification of positive samples. For this same reason, the stratified dummy classifier outperfoms the most frequent one. Apart from "BERT One Randomized Chunk", the other "strong" baselines we used adopt more classical approaches to sequence classification before the advent of the transformer model. An especially important baseline is the *Word2Vec RNN*. As we continue to see the transformer architecture replace RNNs in more and more tasks, we were interested in determining whether our transformer-based model can outperform RNNs in the book likeability prediction task as well. The "BERT One Randomized Chunk" baseline gives an indication of the performance of a transformer-based model adopting a very simple approach.

7.4. First Stage Classifiers

We present here our analysis and study of each design decision listed in the beginning of this chapter for the first stage classifiers. If not otherwise mentioned in the description of each experiment, the default hyperparameters that we use at each iteration are the following:

- DistilBert as Base Transformer Model
- Use of standard dataset (e.g. not character-masked)
- Single task (model is not trained to predict the genre)
- Use of overlap tokenizer with 0 overlap
- No segment max limit
- Random sampler
- Learning rate of 5×10^{-5} as recommended by the original paper [8]

7.4.1. Comparing preprocessed datasets

We found important to first decide what version of the dataset we would use for all further experiments. In particular, we trained a BERT-base-uncased model with the character-masked version of the dataset and without. Both versions of the dataset include the trimming of the noise as explained in 6.3.1. As mentioned in the introduction of this chapter, the decision of training on the character-masked version of the dataset or not cannot be completely disentangled from the decision of whether to further pretrain our model on the masked-language modeling task or not. This is because BERT first converts the tokens into fixed-sized vector representations in the embedding layer before they are passed to the attention layers, as can be visualized in the depiction of the transformer architecture in Figure 3.2. Thus, when using the character-masked dataset, we were introducing a new token (more precisely, the token corresponding to the string '[CHARACTER]') that would be randomly initialized. In order for the model to more effectively incorporate this token in the downstream task, we further pretrained the model on the masked language model task using a corpus that appropriately uses this keyword.

In order to study these dependencies and to have a more fair basis of comparison, we compared the results of a BERT model (base, uncased), fine-tuned on the character-masked dataset and the trimmed dataset, with both in-domain further pretraining and no pretraining. Table 7.2 below shows our results.

The Table 7.2 results do not provide conclusive evidence that character masking provides reasonable benefit to the training task. One would need to perform k-fold cross validation or obtain a larger dataset to draw more confident conclusions. However, since the performance on the validation set is best (albeit slightly) when adopting character-masking with in-domain pretraining, we opted to use this variation of the dataset in our final model.

Model	Val. (w-F1)	Test (w-F1)
No Character-Masking and no Further Pretraining	0.7363	0.6289
No Character-Masking with In-Domain Pretraining	0.7270	0.6684
Character-Masking without Further Pretraining	0.7098	0.6758
Character-Masking with In-Domain Pretraining	0.7374	0.6690

Table 7.2: W-F1 Scores Comparing Preprocessed Datasets

7.4.2. Deciding which transformer model to use as the foundation

Althougth BERT has to some extent become the trademark name for encoder-based transformer models, there are many variations of BERT that have been developed in order to address specific pitfalls of BERT. Many of these models have claimed state-of-the-art results on various datasets, motivating us to study their applicability to our task. We took into consideration those model developments that appeared to be the most promising. These are the models presented in Section 3.1.3 (RoBERTa, DistilBERT, Electra, BigBird, Longformer). Since BigBird and Longformer are intended to be used with sequence lengths greater than 512, in their use we increased the length of our segments to 2048 and 1024 tokens respectively.

Model	**Num Parameters**	**Val.** (w-F1)	**Test** (w-F1)
Longformer	149M	0.5104	0.5093
BigBird	127M	0.5117	0.5079
RoBERTa	125M	0.5980	0.5572
Bert (base)	110M	0.7363	0.6289
DistilRoBERTa	82M	0.7200	0.6243
DistilBERT	66M	0.7374	0.6607
Electra (small)	14M	0.7200	0.6643

Table 7.3: W-F1 Scores Comparing Preprocessed Datasets

Using the default hyperparameters, RoBERTa seems incapable of learning. We even tried to train it for 10 epochs to see if it would show learning process, but without success. Since the authors of RoBERTa claim that their model performs better than BERT in a large number of standard NLP benchmarks, we made an exception to the aforementioned note of keeping hyperparameters constant and tinkered with the learning rate. Despite all efforts, we were not able to get RoBERTa to learn our problem to any satisfactory degree.

As quantified by the table above, we noticed similar issues with the models that make use of sparse attention patterns like Longformer and BigBird. The issues with these models were exacerbated by their substantially larger size over their transformer model counterparts like Bert and RoBERTa. As a result, we were forced to work with a very small batch size of 1 or 2 using the hardware mentioned in 7.1. This constraint is likely to have contributed to the nonperformance of these models.

On the other hand of the spectrum, the smaller models seemed to perform quite well on our task (Distil-RoBERTa, DistilBERT, Electra). Notably DistilRoBERTa, which was designed with the intention of mimicking the behavior of RoBERTa using 30% less parameters, performs quite well, in contrast with the full RoBERTa nonperformance. We also witnessed a significantly better performance of DistilBERT in comparison to its respective "father model" BERT. We attribute this to the fact that, given our small dataset and the bias induced by generating multiple samples from the same document, the training process is benefited by a smaller model with inherently more bias, which makes the model less likely to overfit.

Since DistilBERT gets the best performance on our task, it was selected as the base model of choice for the final stage 1 classifier.

7.4.3. Deciding whether to further pretrain our models or not

In a prior subsection, we studied the effects of masking the characters in our dataset. As part of the study, we needed to further pretrain a model to learn a more appropriate embedding for the special token we added in order to make possible a fairer comparison. The results showed some slight indication that further pretraining may improve performance. This subsection discusses the evidence obtained on the effect that further pretraining has on our downstream task.

As mentioned in the background discussions, there are different ways in which one can further pretrain our transformer model to prepare it for the downstream task. The standard approaches are to continue training by means of the masked-language modeling task via a within-task and in-domain masked schema.

In the experiment conducted, we compared the performances of using no pretraining, within-task pretraining, in-domain pretraining, and within-task after in-domain pretraining. For both within-task and in-domain, we trained our model on the masked-language modeling task for one epoch and did not use a validation set. We conducted our experiment using DistilBERT as the base model.

Model	Val. (w-F1)	Test (w-F1)
No Further Pretraining	0.7374	0.6660
Within-Task Pretraining	0.7497	0.6689
In-Domain Pretraining	0.7318	0.6772
Within-Task after In-Domain Pretraining	0.7460	0.6748

Table 7.4: W-F1 Scores Comparing Pretraining Methods

Although, the results shown in Table 7.4 are not entirely conclusive but they do suggest that further pretraining with the within-task dataset is at least slightly beneficial for the downstream task. Also, since in-domain pretraining seems to at least not hurt the performance and the validation scores between in-domain and within-task after in-domain are so similar, we decided to use the latter strategy when training our final model. Moreover, we believe that the training on more data is beneficial to learning the embedding of the special '[CHARACTER]' token that we added.

7.4.4. Deciding how to best incorporate the genre information

The experiment described in this section aimed to discover if the target task is benefitted by having the model learn the genre of the text in addition to its success label. This approach can be implemented in one of two ways. The first is to take an "other-task pretraining" approach in which the model is trained first on the genre prediction task and then on the success label task. The second is to train the network to simultaneously predict the genre and the label.

Model	Val. (w-F1)	Test (w-F1)
No genre incorporation	0.7460	0.6748
Other-task Pretraining: predict genre, then success label	0.7415	0.6791
Multitask: predict genre and success simultaneously ($\alpha = 0.5$)	0.7644	0.6732

Table 7.5: W-F1 Scores Comparing Genre Incorporation Methods [1]

[1] The experiment was executed using a DistilBERT model pretrained using both in-domain and within-task with the character-masked dataset.

The scores on the validation set, shown in Table 7.5, seem to indicate that training in a multitask setting is beneficial to the target task. As the multitask setting introduces the new hyperparameter α, this was included when we performed hyperparameter tuning.

7.4.5. Deciding how to segment our books

Inherent to the constraint of segmenting our books, to adhere to the memory limitations of transformer models, comes the choice of how to best perform this segmentation. We implemented and compared two different types of tokenizer algorithms (the sentence tokenizer and the overlap tokenizer) whose rationale has been provided in Section 5.2. In summary description, the sentence tokenizer ensures that a sentence does not get split between two chunks and the overlap tokenizer segments the books with a moving window to allow a defined amount of overlap between two consecutive segments.

We compared the sentence tokenizer and the overlap tokenizer at four different values of its associated overlap parameter (0, 50, 100, and 250 tokens). We used the character-masked dataset and an out-of-the-box DistilBERT model trained in a single-task setting to make our comparisons. We intentionally did not use a further pretrained model as we did not want to introduce any potential bias towards the tokenization algorithm that was used to further pretrain the model. Another aspect of this experiment worth noting is that when using the model for prediction, one can choose a different overlap parameter than the one originally used by DistilBERT for training. To keep things simple, we opted to use the same overlap tokenizer for both training and prediction. That is, if the model was trained with an overlap of 50 then inference was also executed using an overlap of 50.

Tokenizer Algorithm	**Overlap Amount**	**Val.** (w-F1)	**Test** (w-F1)
Sentence Tokenizer	NaN	0.7395	0.6760
Overlap Tokenizer	0	0.7374	0.6690
Overlap Tokenizer	50	0.7448	0.6605
Overlap Tokenizer	100	0.7326	0.6850

Table 7.6: W-F1 Scores Comparing Tokenizers

Definitive conclusions cannot be drawn from the experiment results, shown in Table 7.6. This is because the highest score on the validation set corresponds to the lowest score on the test set and vice versa. We have performed one more experiment in which we have used the overlap tokenizer at different overlap values using the decisions already made in the above experiments. Namely, we used a DistilBERT model in the multitask setting, that was pretrained on both the within-task and in-domain masked language modeling strategies.

Tokenizer Algorithm	**Overlap Amount**	**Val.** (w-F1)	**Test** (w-F1)
Overlap Tokenizer	0	0.7645	0.6702
Overlap Tokenizer	25	0.7774	0.6940
Overlap Tokenizer	50	0.7579	0.6773
Overlap Tokenizer	75	0.7644	0.6830
Overlap Tokenizer	100	0.7693	0.6912

Table 7.7: W-F1 Scores Comparing Tokenizers

The experiment results, shown in Table 7.7, suggest that a small amount of overlap may be helpful to the target task, whereas adding a large overlap (50+) does not provide any additional benefit. This result is in agreement with the findings of research by Joshi et. al [15], although it appears to be in contrast with the design choices of other BERT researchers [25], [29]. This discrepancy may suggest that the amount of optimal overlap to be used is task dependent. Based on our results, we have used in going forward a small overlap of 25.

7.4.6. Deciding how to best alleviate biases of long books

In conducting the above experiments, we noticed the unexpected trend constituted by the validation loss increasing well before one epoch of the training data was processed. In contrast with this finding in our task, in the original paper discussing BERT, the authors found 2-4 epochs to be the optimal range when fine-tuning on the downstream task. We conjecture that this difference in findings is due to the fact that we are splitting our training data of 555 books into several segments each. As a result, the samples we use to train the model are not independent and identically distributed. The great dependence among subsets of these samples thus causes the model to overfit before one epoch of the training data has passed. By the same token, another related issue is that longer books will generate more samples and thus have an overrepresentation with respect to the shorter books.

The first remedy we applied in order to alleviate these issues is to set a limit to the max number of segments any book can have during the training procedure (starting from the beginning of the book). In Table 7.8 below, we provide the results for using no limit; and a max segment limit of 40, 35, 30, 25, 20, and 1 segment(s).

$*$ All results were obtained using a pretrained (within-task and in-domain) DistilBERT model pretrained using character-masking in the multitask setting.

Max # of Segments	**Val.** (w-F1)	**Test** (w-F1)
No Limit	0.7774	0.6940
40	0.7674	0.6777
35	0.7710	0.6613
30	0.7792	0.7028
25	0.7698	0.6990
20	0.7725	0.6849
1	0.7395	0.6537

Table 7.8: W-F1 Scores Comparing Max Segment Lengths

The results in Table 7.8 indicate that there is not a significant advantage between using the whole book or setting a limit of 20 segments (and everything inbetween). One troublesome result is that we could still obtain relatively good results (within just a few points of the other experiments) when we only used the first segment of each book. This observation is suggesting that perhaps we were not feeding the training data to the model in an appropriate way.

By this initial insight we were thus motivated to develop and test the "Sequential Book Sampler" approach discussed in Section 5.2. The key concept in this approach is that the model is driven to first see a segment from every book in the training set before seeing a segment of the same book again. This sampling procedure can be used both with and without replacement. When we used replacement, segments of shorter books may appear multiple times during the training process. We report our related results in Table 7.9 below.

Using Replacement	**Val.** (w-F1)	**Test** (w-F1)
No Replacement	0.7776	0.7214
Replacement of 2 (A segment can appear up to two times)	0.7693	0.6964

Table 7.9: W-F1 Scores of Sequential Book Sampler Approaches

We note that we did get a particularly high score on the test when using the Sequential Book Sampler without replacement. However, the score on the validation set does not quite confirm this improvement.

7.4.7. Hyperparameter Tuning

In the preceding subsections we have studied several model design choices, in separate independent experiments for each. The results of those experiments have led us to make the following model design decisions:

- Use of the character-masked dataset
- Use of the DistilBERT model as our base transformer
- Pretraining of our model using datasets corresponding to both in-domain and within-task pretraining
- Training the model in a multitask setting (both genre and success label)
- Use of the overlap tokenizer with an overlap of 25
- Setting a max threshold of 30 segments per book

Having made these design decisions, we then perform a hyperparameter search on the hyperparameters that define the learning process. In particular, we did a hyperparameter search over the learning rate; the dropout rate; the attention dropout rate; and our multi-task loss weight, α.

The following list represents the range of values that we explore for each of these hyper-parameters:

Learning Rate (lr) $[1 \times 10^{-5}, 1 \times 10^{-4}]$

Dropout Rate (dr) $[0.1, 0.4]$

Attention Dropout Rate (adr) $[0.1, 0.4]$

Multi-task Weight, α $[0.3, 0.7]$

We execute 10 trials using a scheduler with the median stopping rule. Below, in Table 7.10 we show the configuration and the results of the best performing trial on the validation set.

lr	dr	adr	α	**Val.** (w-F1)	**Test.** (w-F1)
3.969×10^{-5}	0.3877	0.2436	0.5928	0.7998	0.7215

Table 7.10: W-F1 Score of Best Performing Model as a Result of Hyperparameter Tuning

The model yielding the results presented in Table 7.10 has been used to extract the BERT embeddings for the second stage classifiers described in the following section.

7.5. Second Stage Classifiers

In the previous section, we performed an extensive hyperparameter search to find the transformer model best suited for the book likeability prediction task. This section describes experiments to study second stage classifiers that are trained using the embeddings coming from this selected transformer model.

7.5.1. Extracting the embeddings for our second stage classifiers

The choice of the transformer model used to extract chunk embeddings is determined by the stage 1 classifier that yielded the highest weighted-F1 score during the standard hyperparameter tuning stage. As described in the background on BERT, the '[CLS]' token was prepended to every chunk for the purposes of sequence classification and it is the final representation of this token (referred to as the pooled output) that was passed to the final classification layers. Therefore, the most natural idea for generating the chunk embeddings is to simply use these pooled outputs. Using BERT (base) and many of its related models (including DistilBERT), results in an embedding of 768 dimensions coming from this pooled output for each chunk of text. For our second stage classifiers that were designed to handle sequential data such as RoBERT and ToBERT, we were able to use these chunk embeddings directly. For the other models (SVM, shallow Neural Network, and the multimodal networks), we needed a fixed dimension size. To satisfy this constraint, for each book, we averaged its chunk embeddings. Correspondingly, we obtained book embeddings with 768 dimensions each.

To gain some visual insight as to how well these embeddings distinguish the successful and unsuccessful class labels, we used Principal Component Analysis (PCA) to plot them in two dimensions (as shown in Figure 7.1).

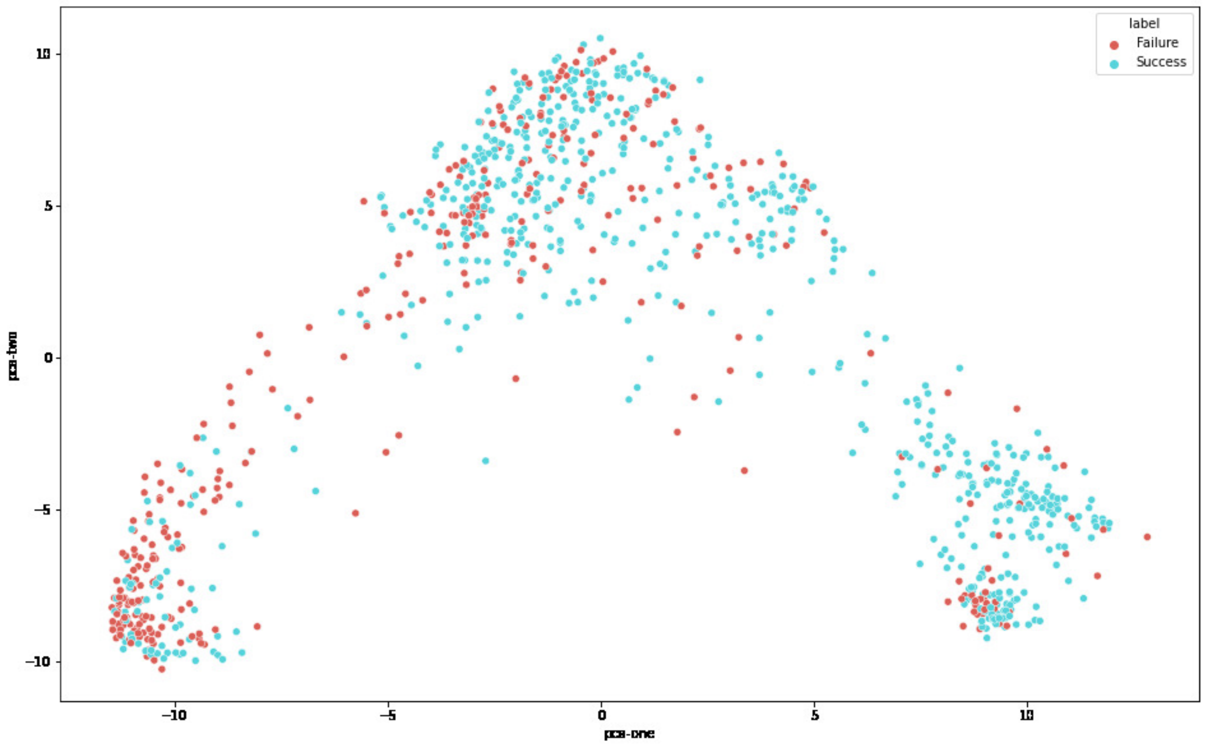


Figure 7.1: Book Embeddings Compressed to Two Dimensions Using PCA

Since the embeddings were also learned to distinguish the genres of the books, it is also interesting to see how well the embeddings can separate the genres. In Figure 7.2, we show a compression of the dataset to two dimensions using t-distributed stochastic neighbor embedding (t-SNE).

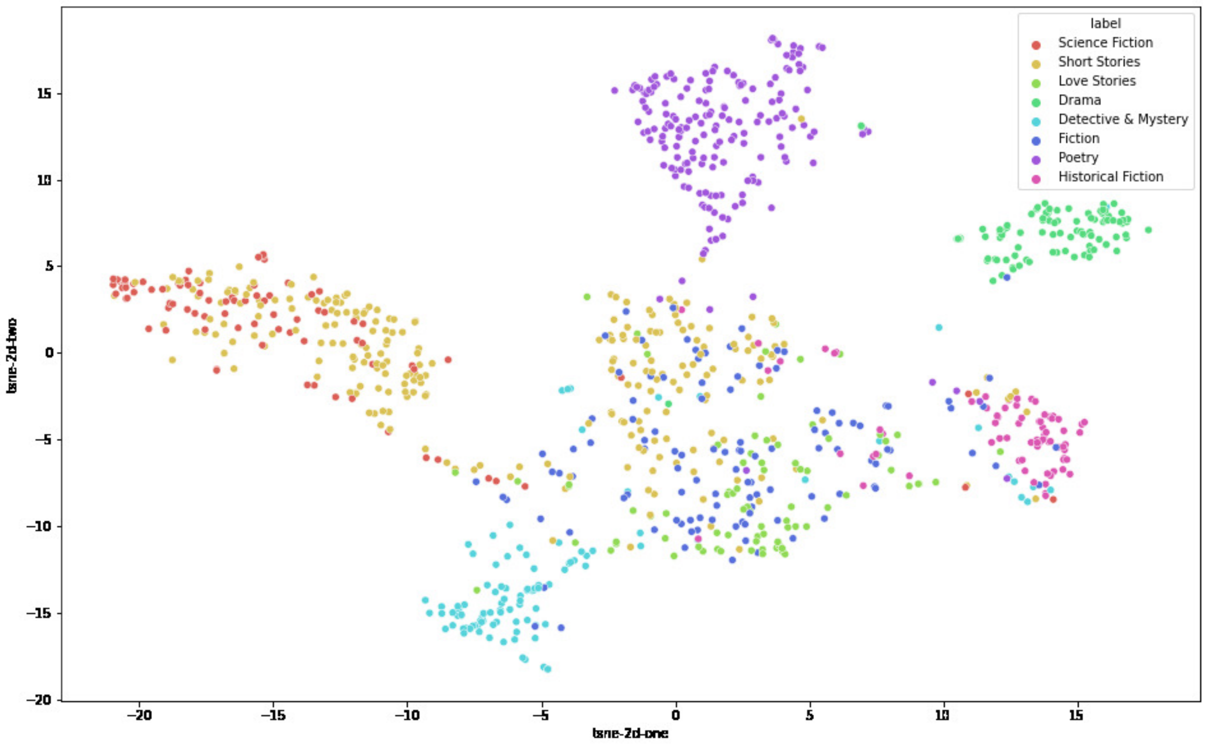


Figure 7.2: Book Embeddings Compressed to Two Dimensions Using t-SNE

The principal finding from this experiments is that the genres that have very distinct themes or writing styles such as Poetry, Detective and Mystery, and Drama are very well clustered, whereas genres that have less concrete themes such as Fiction and Short Stories are more scattered.

7.5.2. Simple Models Trained on Book Embeddings

Shallow Neural Networks

The simplest idea for making use of our book embeddings is to use them to train a shallow neural network. We replicated the classification head of the DistilBERT model we used for classification on the book segments and trained the neural network in both the single-task and multi-task settings. We performed hyperparameter tuning on the learning rate, the number of epochs, dropout rate, and in the case of the multitask setting, alpha as well. Table 7.11 shows our results for the best trials on the validation set for both the single task and multitask settings.

Model	**Val.** (w-F1)	**Test** (w-F1)
Shallow NN (ST)	0.7939	0.7132
Shallow NN (MT)	0.8019	0.7090

Table 7.11: W-F1 Scores of Shallow Neural Networks Trained on Book Embeddings

Support Vector Machine

Support Vector Machines tend to perform well when the number of training samples is relatively small in comparison to the number of features. This is the case for our task as we have 555 training samples and 768 features. We used the validation set to tune the kernel choice and C. We obtained the best results, shown in Table 7.12 using an RBF kernel with $C = 6.0$.

Model	**Val.** (w-F1)	**Test** (w-F1)
SVM with RBF Kernel	0.7805	0.7363

Table 7.12: W-F1 Scores of SVM Trained on Book Embeddings

7.5.3. Sequential Data Models on Chunk Embeddings

Just as we used sequential data models to take chunks of text and predict the successfulness of chunks, we could also use the same type of models to take the embeddings of the chunks and predict the successfulness of the entire book as a whole. This is the idea behind our use of RoBERT and ToBERT, which use an LSTM and a transformer encoder respectively.

RoBERT

We performed hyperparameter tuning with 12 trials on the learning rate, batch size, and hidden layer size of the lstm units. We obtained the best results, shown in Table 7.13, using the following hyper-parameters:

- batch size: 64
- hidden lstm layer size: 32
- learning rate: 0.01282

Model	**Val.** (w-F1)	**Test** (w-F1)
RoBERT	0.7871	0.7041

Table 7.13: W-F1 Scores of Best RoBERT Model

ToBERT

The corresponding information and results relative of our experiment with ToBERT are as shown below, including Table 7.14.

- batch size: 128
- learning rate: 1.6418×10^{-5}
- number of heads: 6
- feedforward layer dimension: 256
- number of layers: 2
- dropout: 0.2714

Model	**Val.** (w-F1)	**Test** (w-F1)
ToBERT	0.7831	0.6994

Table 7.14: W-F1 Scores of Best ToBERT Model

Overall, the performances of RoBERT and ToBERT on our task was less than impressive, as they even underperformed our first stage classifier.

7.5.4. Multimodal Networks

The use of multimodal networks requires more experimentation as there are many different com-binations of features that can be used to train the network. For each combination of features, we performed hyperparameter tuning.

The hyperparameters that we tuned for the Feedforward Standard Concatenation model were:

- Batch Size
- Learning Rate

- Standardized Dimension Size: This corresponds to the final embedding size that each modality was standardized to before concatenation.

The hyperparameters that we tuned for the Genre-Aware Attention model were:

- Batch Size
- Learning Rate
- Standardized Dimension Size: This corresponds to the final embedding size that each modality was standardized to before processed by the attention layer.
- Genre Embedding Size

Feedforward Standard Concatenation

Table 7.15 presents our results on the Feedforward Standard Concatenation multimodal network using various combinations of neural and handcrafted features.

Features Used	**Val.** (w-F1)	**Test** (w-F1)
Bert Features	0.7805	0.7126
Bert Features, Readability	0.7806	**0.7315**
Bert Features, Char 5 Grams	0.7674	0.6695
Bert Features, Bigram	0.7939	0.7173
Bert Features, Clausal	0.7776	0.7163
Bert Features, Concepts	**0.8043**	0.7278
Bert Features, Writing Density	0.7908	0.7234
Bert Features, Book2Vec	0.7628	0.7286
Bert Features, Book2Vec, Writing Density, Readability	0.7792	7123
Bert Features, Char 5 Grams, Writing Density, Sentiment Concepts, Clausal, Readability	0.7729	0.7089
Bert Features, Writing Density, Sentiment Concepts, Clausal, Readability	0.7654	0.7036
Bert Features, Sentiment Concepts, Readability	0.7776	0.7132

Table 7.15: W-F1 Scores Comparing Different Inputs to Feedforward Standard Concatenation Model

Genre-Aware Attention

In the use of the genre-aware attention model, it makes sense to train the network with just one feature, since the model simultaneously learns genre embeddings that get baked into the feature embeddings and are also fed directly to the classification unit. Therefore, we experimented with using the BERT features on their own, to draw insight in what improvements can be realized through use of additional features. Since Maharjan et. al [21] report the best results when feeding the genre embeddings to the classification head directly, we experimented exclusively with this configuration. Our results are shown in Table 7.16 below.

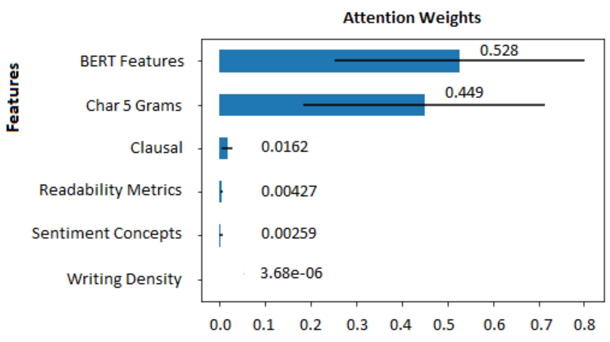
Features Used	**Val.** (w-F1)	**Test** (w-F1)
BERT Features	0.7740	0.7234
BERT Features, Readability	0.7842	0.7265
BERT Features, Char 5 Grams	0.7740	0.7181
BERT Features, Bigram	0.7939	0.7173
BERT Features, Clausal	0.7872	0.7191
BERT Features, Concepts	0.7991	0.7326
BERT Features, Writing Density	0.7792	0.7184
BERT Features, Book2Vec	0.7792	0.7292
BERT Features, Book2Vec, Writing Density, Readability	0.7842	0.7947
BERT Features, Char 5 Grams, Writing Density, Sentiment Concepts, Clausal, Readability	0.7872	0.7139
BERT Features, Writing Density, Sentiment Concepts, Clausal, Readability	**0.8072**	0.7120
BERT Features, Sentiment Concepts, Readability	0.7991	**0.7357**
Char 5 Grams, Writing Density, Sentiment Concepts, Clausal, Readability	0.7145	0.6677

Table 7.16: W-F1 Scores Comparing Different Inputs to Genre Aware Attention Model

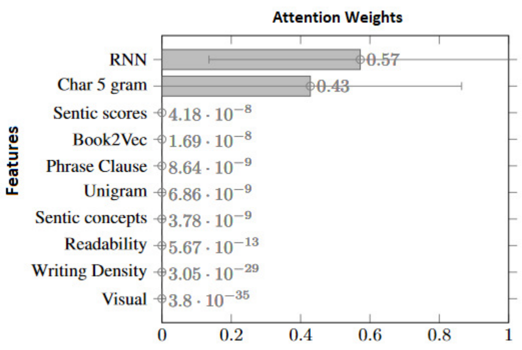
If we compare this multimodal network with the more simplistic one discussed in the previous chapter, we notice small improvements in the performance. This gives evidence that the Genre-Aware Attention model may provide a more suitable architecture for fusing various modalities together for the prediction task.

The results indicate that the BERT features are the most practical for the task at hand. In fact the BERT features on their own significantly outperform all the different categories of handcrafted features used together. We do observe small improvements when including the readability metrics and the sentiment concepts. Our results seem to be consistent with the distribution of results reported in the original genre-aware attention paper [21] but we are unable to reach such a high score as 75.4% as they do with their RNN features.

Since this network internally computes the amount of attention to give to each modality, we can inspect the attention values to gain insight into how much importance the model is giving to each feature. For this analysis, we considered the model that makes use of all the handcrafted features as well as the BERT features. Our results are shown in Figure 7.3 with a side-by-side of the corresponding results from the original paper. Our results agree with those of Maharjan et. al's in that our models attend mainly to the neural features (Bert, RNN) and the char 5 grams. However, we notice that the model tends to perform worse when using the char 5 grams. We notice the same effect with the simpler Feedforward Standard Concatenation multimodal network.



(a) Attention weights with BERT features



(b) Attention weights from Maharjan et. al [21]

Figure 7.3: Average Attention Weights with Standard Deviations

8 | Conclusions

In this thesis have we discussed and investigated the book likeability prediction problem, exploring and testing Transformer-based techniques to determine their suitability for developing a system capable of successfully executing the attending task. In Chapter 4 we posed five derivative research questions, the answers to which can provide the foundation of an effective approach to building such a system. For the reader's convenience, we list again these questions here:

1. How critical is the choice of the base Transformer model for the target task?
2. What is the best way to pretrain BERT (or BERT-like models) for the target task?
3. Can we preprocess the data to filter out bias and/or noise and improve the prediction?
4. Can the target task execution benefit from carrying out the model training in a multitask setting?
5. How can we best extend the BERT fine-tuning procedure to achieve good results in tackling long-text classification?

Based on our study and experiments, we are able to provide answers to the question that, although certainly not altogether complete and definitive, nevertheless allow us to draw some useful conclusions and provide indications and insight for the paths that may be followed in future more extensive research on the subject of this thesis.

In addition to BERT, we investigated a handful of its derivatives that have been developed with the intent of addressing some of the BERT perceived shortcomings. These BERT-evolved models often claim performance significantly superior to BERT in some NLU benchmarks. Our results, however, stand for the most part in opposition to key claims and conclusions drawn by the BERT-evolved model developers.

Models that claim to be more adept at handling long sequences, such as Longformer and BigBird, did not provide any tangible benefits over BERT to our long sequence classification task. In fact, the task execution performance dropped significantly when we used these models. The RoBERTa derivative, which claims to drastically improve upon the pretraining process of BERT, actually performed much worse than BERT in the execution of our task. Rather interestingly, DistilBert, whose declared focus is not to improve in performance over BERT but to reduce BERT size, actually yielded the best results.

In general, we noticed that smaller models like DistilBert and Electra were able to generalize the training data better than their much larger counterparts like the Longformer, BigBird, and RoBERTa. With regard to this evidence, we believe that, given the difficulty of the task at hand and the small dataset available, the training process is carried out more efficiently and effectively by a model with more bias.

The driving force behind the inference capabilities of transformer-based models is their large-scale adoption of transfer learning. These models are typically pretrained on massive quantities of data via some form of the MLM task. By means of the training carried out in this fashion, the model becomes inherently endowed with a strong understanding of the English language. This understanding is, of course, based on the distribution of text data that the models were pre-trained on.

While a strong general knowledge of the language is critical to performance in the book likeability prediction task, we have explored additional pretraining methods to give the models a better understanding of English in a literary context. For this purpose, we further pretrained our model on the task dataset and on our self-procured pg-2600 dataset. While our results are not entirely conclusive, they suggest that this form of pretraining did indeed give the model a more nuanced knowledge of English literature, thus making it more adept for the task at hand.

Maharjan et. al [19] boosted performance of their multi-modal classifiers by training them to learn the success label and the genre simultaneously. Drawing from their work and successful results, we have adopted the same approach in the training of our BERT and BERT-like models. While a number of model developers have trained BERT on multiple tasks simultaneously, with different datasets, we could not find any other published research that has trained BERT to infer classes from multiple sets of labels. Our findings indicate that this type of multitask setting is beneficial to the target task. We believe that given the difficulty of the likeability prediction problem and the scarce amount of data we are able to work with, having the model contemporarily identify the book genre acts as a form of regularization.

A considerable drawback to Transformer-based models is their poor scalability to the analysis of long sequences. Consequently, we were forced to segment our books into smaller sequences in order for BERT to be able to process them. This has the obvious drawback of drastically reducing the context window used for inference. Moreover, we were also concerned that such a procedure may induce the model to give excessive consideration and importance to unique book identifiers, such as character names, and/or become biased towards the longer samples in the dataset. In our attempt to counter the anticipated problems that may be produced by segmenting the long sequence data samples, we explored several different ways to train BERT in ways that may mitigate such issues.

To address the potential problem of book identifiers, we filtered our dataset by processing it through a named-entity recognition model in order to mask character names, after first verifying by means of pointwise mutual information that character names were indeed the most discriminative words in a book. Such a preprocessing step did not appear to have a discernable outcome on the target task, leading us thus to concluding that BERT is robust in its capability of not overfitting to sample identifiers that get split up across multiple chunks.

We have also explored any potential biases that may be associated with the length of books. Long books do generate more data segments and are thus more likely to be sampled by a random sampler than shorter books. To mitigate any potential biases this may cause, we experimented with setting a maximum number of segments limit for each book and defining a sampler in such a way that it prevents the model from seeing too many segments coming from long books before the rest of the dataset. The outcome of this experiment was that setting a max number of segments limit does not appear to improve performance; at the same time, however, this demonstrated that the amount of learning that BERT can extract from each book quickly saturates. That is, we saw no significant drop in performance between setting no limit to using a number of segments limit as low as 20 segments.

Another drawback originated by the segmentation of data sequences, possibly a most significant one in the context of our task, is that it inhibits the model from considering the book as a whole. To counteract and mitigate this, we experimented with second stage classifiers, which either: A) treated the segment embeddings as another sequential layer; or B) made use of a condensed representation of the segment embeddings.

For the investigation of approach "A" we implemented the Pappagari et. al's RoBert and ToBert models [25]. The original paper describes the successful application of these models to three different tasks; however, similarly to most of the other BERT-evolved models we have investigated, the performance of RoBert and ToBert in our task was below that observed when using BERT in its current standard version.

For approach “B,” we averaged the segment embeddings coming from BERT for each book to create our book embeddings. Using these book embeddings, we trained a SVM to find a hyperplane of best-separation between the two classes in the dataset. This achieved the yielding of our best weighted F1 score of 73.63%. Notably, we were able to achieve a similar performance of 73.57% F1 score by training Maharjan et. al’s state-of-the-art genre-aware attention model [21] on our book features; and the sentiment concepts, and readability metrics handcrafted features (described in 5.3.3). While our results do not surpass the state-of-the-art of 75.4% F1 score, given the small size of the dataset that we were able to use we believe that the best of the approaches we have developed is performing essentially at the same level as the state-of-the-art best-scoring method. It is also to be noted that the genre-aware attention model proved to be a model that is rather tricky to effectively train, as performances appears to be highly sensitive to initial conditions and hyperparameters.

In conclusion, the performances of the best versions of our models outperform our strong baselines and are comparable to the state-of-the-art. At the same time, however, our results confirm the shortcomings that transformer-based models exhibit when dealing with very long sequences. While BERT and BERT-like models have been achieving state-of-the-art results across many tasks, they are typically only optimized for shorter texts. We believe that more in-depth research is necessary to make transformer-based models more effectively and efficiently applicable to the analysis and classification of long sequences.

8.1. Future research directions

As natural language processing continues to take great strides forward, it is all but certain that we will continue to see more data-driven approaches to old problems, of which the book likeability prediction problem is a prime example. The potential “return on investment” for more in-depth research of this problem remains large as the development of a robust book likeability prediction system has immense potential value to the publishing industry. Thus, research efforts in this direction will continue as the field of NLP research in general expands and progresses.

Having conducted our research, we project on the basis of the results, positive and “negative,” that we have obtained that further research and development work on the following points is worth of consideration:

- **Procure a broader book dataset incorporating more modern books.** The Goodreads dataset procured by Maharjan et. al allows us to demonstrate that machines are able to understand what type of lexicon or style is associated with better selling books. However, the dataset is composed of books that are in the public domain (i.e. their copyright has expired) so they are generally not representative of modern literature. It would be intriguing, and important for any future practical use of book likeability models, to see how such models perform with newer books. Besides the dated nature of the dataset, its being composed of only 1000 books constitutes a serious limitation for how a model can "understand" what makes a novel good across the range of styles and diverse genres.

- **Procure a dataset designed for unbiased benchmarking and comparatively evaluate models** This point can overlap with the previous but is important because at the present time, it can be difficult to confidently compare massive language models like BERT and its derivatives, since fair comparison methods, e.g., those employing cross-validation, are computationally very costly.

- **Develop multi-modal architectures that take in consideration a combination of different neural features.** Such a development would enable a more wholistic appraisal of book likeability. In our experiments, we have shown that multi-modal architectures favor our BERT features over handcrafted features. Maharjan et. al [21] claim a similar result with their RNN features. Thus, an interesting research project would be to study the effects of using various neural representations, in conjunction with a multimodal network such as the genre-aware attention model.

- **Develop architectures tailored for the analysis of long sequences as the NLP field advances.** In general, there is great interest in the study of models that can process very long sequences of text. As research on this topic continues, new models with such capabilities should be more suitable and easier to optimize for application to the book success prediction problem.

- **Test and benchmark model results against an actual book agent or publisher "expert opinion."** Once a book success prediction system achieves a satisfactory accuracy in research test settings, a natural follow-on would be to compare its capability and accuracy with those of human experts operating in the real publishing industry world. By testing and benchmarking the model results against book agent and publisher predictions and opinions, developers will gain insights into the strengths and weaknesses of their book success prediction systems.

Bibliography

[1] J. Anderson. Lix and rix: Variations on a little-known readability index. *Journal of Reading*, 26(6):490–496, 1983.

[2] V. G. Ashok, S. Feng, and Y. Choi. Success with style: Using writing style to predict the success of novels. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1753–1764, 2013.

[3] S. Baccianella, A. Esuli, and F. Sebastiani. Sentiwordnet 3.0: an enhanced lexical resource for sentiment analysis and opinion mining. In *Lrec*, volume 10, pages 2200–2204, 2010.

[4] I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[5] A. Bies, M. Ferguson, K. Katz, R. MacIntyre, V. Tredinnick, G. Kim, M. A. Marcinkiewicz, and B. Schasberger. Bracketing guidelines for treebank ii style penn treebank project. *University of Pennsylvania*, 97:100, 1995.

[6] C. Bucilua, R. Caruana, and A. Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006.

[7] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[9] R. Flesch. A new readability yardstick. *Journal of applied psychology*, 32(3):221, 1948.

[10] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM*

SIGKDD international conference on knowledge discovery and data mining, pages 1487–1495, 2017.

[11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[12] R. Gunning et al. Technique of clear writing. 1952.

[13] J. W. Hall. *Cracking the Code of the Twentieth Century's Biggest Bestsellers*. Random House Digital, Inc., 2012.

[14] J. Harvey. The content characteristics of best-selling novels. *Public Opinion Quarterly*, 17(1):91–114, 1953.

[15] M. Joshi, O. Levy, D. S. Weld, and L. Zettlemoyer. Bert for coreference resolution: Baselines and analysis. *arXiv preprint arXiv:1908.09091*, 2019.

[16] M. Khalifa and A. Islam. Will your forthcoming book be successful? predicting book success with cnn and readability scores. *arXiv preprint arXiv:2007.11073*, 2020.

[17] Q. Le and T. Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR, 2014.

[18] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[19] S. Maharjan, J. Arevalo, M. Montes, F. A. González, and T. Solorio. A multi-task approach to predict likability of books. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1217–1227, 2017.

[20] S. Maharjan, S. Kar, M. Montes-y Gómez, F. A. González, and T. Solorio. Letting emotions flow: Success prediction by modeling the flow of emotions in books. *arXiv preprint arXiv:1805.09746*, 2018.

[21] S. Maharjan, M. Montes, F. A. González, and T. Solorio. A genre-aware attention model to improve the likability prediction of books. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3381–3391, 2018.

[22] G. H. Mc Laughlin. Smog grading-a new readability formula. *Journal of reading*, 12 (8):639–646, 1969.

[23] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[24] E. S. Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, 1954.

[25] R. Pappagari, P. Zelasko, J. Villalba, Y. Carmiel, and N. Dehak. Hierarchical transformers for long document classification. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 838–844, 2019. doi: 10.1109/ASRU46091.2019.9003958.

[26] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[27] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[28] R. Senter and E. A. Smith. Automated readability index. Technical report, CINCINNATI UNIV OH, 1967.

[29] F. Souza, R. Nogueira, and R. Lotufo. Portuguese named entity recognition using bert-crf. *arXiv preprint arXiv:1909.10649*, 2019.

[30] C. Sun, X. Qiu, Y. Xu, and X. Huang. How to fine-tune bert for text classification? In *China National Conference on Chinese Computational Linguistics*, pages 194–206. Springer, 2019.

[31] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[32] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[33] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[34] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[35] D. Wang and E. Nyberg. A long short-term memory model for answer sentence selection in question answering. In *Proceedings of the 53rd Annual Meeting of the*

Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers), pages 707–712, 2015.

[36] X. Yuan, L. Li, and Y. Wang. Nonlinear dynamic soft sensor modeling with supervised long short-term memory network. *IEEE transactions on industrial informatics*, 16(5):3168–3176, 2019.

[37] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, et al. Big bird: Transformers for longer sequences. In *NeurIPS*, 2020.

[38] R. Zellers, Y. Bisk, R. Schwartz, and Y. Choi. Swag: A large-scale adversarial dataset for grounded commonsense inference. *arXiv preprint arXiv:1808.05326*, 2018.

A | Additional Background Information

A.1. Constituent Tags

The following information comes from "Bracketing Guidelines for Treebank II Style Penn Treebank Project". [5]

These are the different types of clausal tags that were distinguished in the Penn Treebank Project:

- **S:** Simple declarative clause, i.e. one that is not introduced by a (possible empty) subordinating conjunction or a wh-word and that does not exhibit subject-verb inversion.
- **SBAR:** Clause introduced by a (possibly empty) subordinating conjunction.
- **SBARQ:** Direct question introduced by a wh-word or a wh-phrase. Indirect questions and relative clauses should be bracketed as SBAR, not SBARQ.
- **SINV:** Inverted declarative sentence, i.e. one in which the subject follows the tensed verb or modal.
- **SQ:** Inverted yes/no question, or main clause of a wh-question, following the wh-phrase in SBARQ.

A.2. Readability Metrics

The readability metrics provide ways of measuring the difficulty of a text. Since a lot of them use similar variables, we will set forth the notation beforehand:

C	= the total number of characters
L	= the total number of syllables
P	= the total number of polysyllables
S	= the total number of sentences
W	= the total number of words
W_{c}	= the number of complex words (containing three or more syllables)
W_{long}	= the number of long words (more than 6 letters)

- **Gunning Fog Index:** The Gunning Fog Index estimates the number of years of education needed to understand a text on the first reading. It is formulated as follows:

  $$Gunning - Fog = -.4[(\frac{W}{S}) + 100(\frac{W_c}{W})] \tag{A.1}$$

  [12]

- **Flesch Reading Ease Score:** The Flesch Reading Ease Score outputs a value between 0 and 100 with the higher values indicating an easier text. It uses two variables to determine a text's readability:

  - The average length of the sentences.
  - The average number of syllables per word.

  It is formulated as follows:

  $$FKG = 206.835 - 1.0.15 \times \frac{W}{S} - 84.6 \times \frac{L}{W} \tag{A.2}$$

  [9]

- **Flesch Kincaid Grade Level:** The Flesch Kincaid Grade Level is a variant of the Flesch Reading Ease Score that aims to present the score as a U.S. grade level. It is formulated as follows:

  $$FRES = 0.39 \times \frac{W}{S} - 11.8 \times \frac{L}{W} - 15.59 \tag{A.3}$$

  [9]

- **RIX:** RIX and LIX do away with syllable counting as they aim to provide readability metrics for foreign languages as well. Words with high syllable counts may be indicative of difficult reading for the english language but this is not true in general for all languages.

  $$RIX = \frac{W_{long}}{S} \tag{A.4}$$

[1]

- **LIX:**

$$LIX = \frac{W}{S} + \frac{W_{long} \cdot 100}{W} \tag{A.5}$$

[1]

- **ARI:** ARI standing for "Automated Readability Index" is a readability test for English texts. It also provides an approximation of the US grade level needed to understand a text. It is formulated as follows:

$$ARI = 4.71(\frac{C}{W}) + 0.5(\frac{W}{S}) - 21.43 \tag{A.6}$$

[28]

- **SMOG:** SMOG is a measure of readability that estimates the years of education needed to understand the text. It is calculated as follows:

$$grade = 1.0430\sqrt{P \times \frac{30}{S}} + 3.1291 \tag{A.7}$$

[22]

B | Technical Details

This appendix section serves to provide technical details to precisely define how certain aspects of this thesis were performed.

B.1. Building the pg-2600 Dataset

In this section we describe exactly how we procured the pg-2600 dataset that was used for further pretraining BERT.

The authors of Project Gutenberg allow for robot scraping of their repository of cultural works and books and provide an http route for doing so [1]. To download all English books in a txt format, we ran the following command using *wget*:

```
$ wget -w 2 -m http://www.gutenberg.org/robot/harvest?filetypes
    []=txt&langs[]=en
```

Since there are over 50,000 English documents on Project Gutenberg, the download took a couple of days to complete. The documents are organized in folders corresponding to a unique identifier. Project Gutenberg also provides metadata for their documents in XML/RDF format [2]. From this metadata, we were able to extract the title and author for each document. Unforunately, no global unique identifier such as the ISBN was provided. Thus, in order to map each document to its page on Goodreads,we were forced to rely on the search functionality of Goodreads using the title and author as the search query. Then, during the scraping process we use the edit distance [3] with a fixed threshold of 25 between the title from the RDF file and the title from goodreads to infer whether we had an actual match (with a reasonable degree of confidence). To remove any potential authorship bias, we limited the number of books each author can have in the dataset to two. Lastly, we filtered the books such that they roughly pertained to the genres of novels used in the Goodreads Maharjan dataset. This was the most cumbersome part of

[1]For more information, visit: `https://www.gutenberg.org/policy/robot_access.html`

[2]For more information, visit: `https://www.gutenberg.org/ebooks/offline_catalogs.html`

[3]The edit distance is a way of quantifying how dissimilar two strings are by counting the minimum number of operations it takes to turn one string into the other.

the process since Project Gutenberg does not provide genre information as part of their metadata and Goodreads determines the genre by crowd-sourcing their users' "shelves". That is each book has a number of user votes for each genre. To filter the genre using this format, we first normalized the genres such that related genres would be grouped together (e.g. "Love" and "Romance"). Then, we filtered out the books if one of their top three genres was not in the subset of genres used in the Goodreads Maharjan dataset.

The table below shows how many candidate books were filtered out after each filtering stage.

Filter Step	**Candidate Books Left**
Only english books	50127
Title matching	42401
Max 2 books per author	23161
Genre filter	2608

Table B.1: Number of Books at Each Filtering Step

Thus at the end of all the filtering steps, we were left with 2608 books which made up our dataset. One could probably have come up with a larger dataset with a more careful approach to the genre normalization strategy and title matching but for the purposes of a dataset to be used strictly for further pretraining BERT, we were satisfied with the quantity.

We will also make note that the original intention of this workflow was to generate another classification dataset. However, after having already filtered the dataset as above, to then filter books that had too few ratings left us with only a couple hundred more books than the Goodreads Maharjan dataset which we did not deem significant enough.

C | Code Implementation

This appendix section serves to provide the code implementations of a selection of algorithms or neural architectures developed in the realization of this thesis.

C.1. Pointwise Mutual Information

The following function calculates the pointwise mutual information for each class in the corpus. The inputs **text_class_1** and **text_class_2** represent the subcorpora of text corresponding to each class. The input **corpus_vectorizer** is an instantiation of scikit-learn's CountVectorizer [1] which associates tokens with frequency.

```
def get_PMIs(text_class_1, text_class_2, corpus_vectorizer):

    len_vocab = len(corpus_vectorizer.vocabulary_)
    vector_class_1 = np.array(corpus_vectorizer.transform([text_class_1]
                                        ).toarray()[0])
    vector_class_2 = np.array(corpus_vectorizer.transform([text_class_2]
                                        ).toarray()[0])
    id_to_word = {v: k for k, v in corpus_vectorizer.vocabulary_.items()
                                        }

    n_s = np.sum(vector_class_1)
    n_u = np.sum(vector_class_2)
    n = n_s + n_u

    alpha = 10
    word_probs_class_1 = (vector_class_1 + alpha)/(n_s + alpha*len_vocab
                                        )
    word_probs_class_2 = (vector_class_2 + alpha)/(n_s + alpha*len_vocab
                                        )
    word_probs_corpus = (vector_class_1 + vector_class_2 + 2*alpha)/(n +
                                         2*alpha*len_vocab)
```

[1] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

```
    PMI_class_1 = np.log(word_probs_class_1 / (word_probs_corpus * (n_s
                                           / n)))
    PMI_class_2 = np.log(word_probs_class_2 / (word_probs_corpus * (n_u
                                           / n)))

    return PMI_class_1, PMI_class_2
```

C.2. Extending BERT to the Multitask Setting

This code was adapted from the Hugging Face implementation of "BertForSequenceClassification" [2] to handle the classification of two sets of labels simultaneously. The main change occurs in the *forward* method in which we compute the cross entropy loss for each task and sum them together with the weight parameter **alpha**.

```
from transformers import BertPreTrainedModel, BertModel

class BertForMultipleSequenceClassification(BertPreTrainedModel):
    def __init__(self, config, num_labels1 = 2, num_labels2 = 8):
        super().__init__(config)
        self.num_labels1 = num_labels1
        self.num_labels2 = num_labels2
        self.alpha = .5
        self.config = config

        self.bert = BertModel(config)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.classifier1 = nn.Linear(config.hidden_size, self.
                                    num_labels1)
        self.classifier2 = nn.Linear(config.hidden_size, self.
                                    num_labels2)

        self.init_weights()


    def forward(
        self,
        input_ids=None,
        attention_mask=None,
        token_type_ids=None,
        position_ids=None,
        head_mask=None,
```

[2] https://github.com/huggingface/transformers/blob/master/src/transformers/models/bert/modeling_bert.py

```
        inputs_embeds=None,
        labels=None,
        output_attentions=None,
        output_hidden_states=None,
        return_dict=None,
    ):

        return_dict = return_dict if return_dict is not None else self.
                                                    config.use_return_dict

        outputs = self.bert(
            input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids,
            position_ids=position_ids,
            head_mask=head_mask,
            inputs_embeds=inputs_embeds,
            output_attentions=output_attentions,
            output_hidden_states=output_hidden_states,
            return_dict=return_dict,
        )

        pooled_output = outputs[1]

        pooled_output = self.dropout(pooled_output)
        logits1 = self.classifier1(pooled_output)
        logits2 = self.classifier2(pooled_output)
        logits = torch.cat([logits1, logits2], 1)
        loss = None
        if labels is not None:

            if self.num_labels1 > 1:
                loss_fct1 = CrossEntropyLoss()
                loss1 = loss_fct1(logits1.view(-1, self.num_labels1),
                                                    labels[:, 0].view(-1
                                                    ))
            else:
                loss_fct1 = MSELoss()
                loss1 = loss_fct1(logits1.view(-1), labels[:, 0].view(-1
                                                    ))

            if self.num_labels2 > 1:
                loss_fct2 = CrossEntropyLoss()
                loss2 = loss_fct2(logits2.view(-1, self.num_labels2),
                                                            labels[:, 1].view(-1
                                                            ))
            else:
                loss_fct2 = MSELoss()
                loss2 = loss_fct2(logits2.view(-1), labels[:, 1].view(-1
                                                            ))
            loss = self.alpha*loss1 + (1-self.alpha)*loss2

        if not return_dict:
            output = (logits,) + outputs[2:]
            return ((loss,) + output) if loss is not None else output

        return SequenceClassifierOutput(
            loss=loss,
            logits=logits,
            hidden_states=outputs.hidden_states,
            attentions=outputs.attentions,
        )
```

We also adapted the DistilBertForSequenceClassification model class to extend to two tasks as well. Since the adaptation is very similar, we exclude it.

C.3. RoBERT

RoBERT from the paper, *Hierarchical Transformers* [25], consists of an LSTM layer with a classification head. The model processes sequences of bert embeddings which is why the input size of the LSTM units is 768.

```
class LightningRoBERT(pl.LightningModule):
    def __init__(self, layer_size = 100):
        self.layer_size = layer_size
        super(RoBERT_Model, self).__init__()
        self.lstm = nn.LSTM(768, layer_size, num_layers=1, bidirectional
                                            =False)
        self.out = nn.Linear(layer_size, 1)

    def forward(self, grouped_pooled_outs):

        seq_lengths = torch.LongTensor([x for x in map(len,
                                            grouped_pooled_outs)])
        batch_emb_pad = nn.utils.rnn.pad_sequence(grouped_pooled_outs,
                                            padding_value=-91,
                                            batch_first=True)
        batch_emb = batch_emb_pad.transpose(0, 1)  # (B,L,D) -> (L,B,D)
        lstm_input = nn.utils.rnn.pack_padded_sequence(batch_emb,
                                                       seq_lengths, batch_first=
                                                       False, enforce_sorted=False)

        packed_output, (h_t, h_c) = self.lstm(lstm_input, )

        h_t = h_t.view(-1, self.layer_size)

        return self.out(h_t)
```

C.4. ToBERT

ToBERT from the paper, *Hierarchical Transformers* [25], consists of an optional Positional Encoding layer, Transformer Encoder layer(s), and a classification head.

```
class PositionalEncoding(nn.Module):

    def __init__(self, d_model: int, dropout: float = 0.1, max_len: int
                                                     = 5000):
        super().__init__()
        self.dropout = nn.Dropout(p=dropout)

        position = torch.arange(max_len).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2) * (-math.log(
                                                       10000.0) / d_model))
        pe = torch.zeros(max_len, d_model)
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0)
        self.register_buffer('pe', pe)

    def forward(self, x):
        """
        Args:
            x: Tensor, shape [batch_size, seq_len, embedding_dim]
        """
        x = x + self.pe[:, :x.size(1), :]
        return self.dropout(x)

class LightningToBERT(pl.LightningModule):
    def __init__(
        self,
        d_model=768,
        nhead=2,
        nhid=512,
        num_layers=2,
        dropout=0.1,
        classifier_dropout=0.1,
    ):

        super().__init__()

        assert (
            d_model % nhead == 0
        ), "nheads must divide evenly into d_model"

        self.pos_encoder = PositionalEncoding(
            d_model, dropout=dropout, max_len=200
        )

        encoder_layers = nn.TransformerEncoderLayer(
            d_model=d_model, nhead=nhead, dim_feedforward=nhid, dropout=
                                                      dropout, batch_first=
                                                      True
        )
        self.transformer_encoder = nn.TransformerEncoder(
            encoder_layers, num_layers=num_layers
        )

        self.dropout = nn.Dropout(classifier_dropout)
        self.pre_classifier = nn.Linear(d_model, d_model)
        self.classifier = nn.Linear(d_model, 1)

        self.softmaxer = nn.Softmax(dim=1)

    def forward(self, x, src_key_padding_mask):

        x = self.pos_encoder(x)
        x = self.transformer_encoder(x, src_key_padding_mask=
                                                  src_key_padding_mask)  #
                                                  self.src_mask)

        # calculates mean taking into account the padding
        x = torch.unsqueeze(1-src_key_padding_mask,2)*x
        x = x.sum(dim=1)/(1-src_key_padding_mask).sum(dim=1).unsqueeze(1
                                                  )
        x = self.pre_classifier(x)
        x = nn.ReLU()(x)
        x = self.dropout(x)
        return self.classifier(x)
```

D | Tools and Platforms

This appendix section itemizes the different technological tools and platforms that were used throughout this work.

Python 3.6: Python was the sole programming language used through all stages of the research including data scraping, data analysis, data visualization, model building, etc.

PyTorch 1.9: PyTorch is an open source machine learning framework developed by Meta. We used PyTorch to train all neural models in our research.

Scikit-learn 0.22.2: Scikit-learn is a machine learning library containing a collection of various classification, regression, and clustering algorithms. We used it to train all support vector machine and logistic regression models.

Hugging Face Transformers 4.12.2: Hugging Face Transformers is an open-source python package that provides a variety of pretrained transformer models that can be easily used by NLP researchers. It supports both PyTorch and TensorFlow 2.

WandB: WandB is a central dashboard to keep track of hyperparameters, performance metrics, and system metrics of machine learning models allowing one to easily compare different models. Moreover, it served as a repository to save model artifacts so that they could be reused later.

Ray Tune 1.8.0: Ray Tune is a hyperparameter optimization framework designed for long-running tasks such as deep learning training. It is easily configurable and includes a variety of scheduling algorithms. We used it to tune hyperparameters of all neural models.

Scrapy 2.4.0: Scrapy is a free and open-source web-crawling python framework. We used it to scrape Goodreads and generate the pg-2600 dataset.

SQLite: SQLite is a lightweight relational database management system. We used it to store data extracted from the scraping process for the pg-2600 dataset.

List of Figures

List of Tables

Acronyms

NLP	Natural Language Processing
NLU	Natural Language Understanding
NSP	Next Sentence Prediction
MLM	Masked-Language Modeling
PMI	Pointwise Mutual Information
SVM	Support Vector Machine
W-F1	Weighted F1

Acknowledgments

I would like to extend my gratitude to the several people who have been part of my journey during the completion of my master's thesis.

To my superadvisors, Professor Brambilla and Marco Di Giovanni: It has been an honor to work under your guidance and direction. The discussions from our weekly meetings inspired me to keep pushing my work further and further.

I must also thank my parents, Drs. Clorinda Donato and Sergio Guarro and my siblings, Marcello Guarro and Adriana Romero. Without their unconditional love and support, I would have never made it this far.

A special thanks goes to my relatives in Terni: Elisabetta, Vilma, Francesca, and Riccardo, who helped me stay sane and productive in my Polytechnic online studies when I escaped to the Umbrian countryside during the pandemic. I am forever grateful that allowed me to develop strong bonds with all four of you.

Last but not least, I would like to thank all the friends that I met along the way. Because of all of you, I will forever cherish the time I spent in Milan.