DeepGenre: Deep Neural Networks for Genre Classification in Literary Works

Sicong Liu* **Zihan Huang** **Yikang Li** **Zhanlin Sun** **Jiahao Wu** **Hongyi Zhang**
Language Technologies Institute
Carnegie Mellon University
{sicongli,zihanh,yikangli,zhanlins,jiahaowu,hongyiz2}@cs.cmu.edu

Abstract

In this paper, we attempt to address a multi-label text classification problem that predominately features very long text-based inputs. In particular, we focus on using the *Gutenberg Project* dataset and use the main text of an e-book to infer its genre. This is motivated partly because many e-books, especially new ones, may have few or no labeled genres; an automated approach would help curators and librarians assign correct genres for better cataloguing of library resources. We propose to use feature engineering combined with a distributed approach using deep neural networks to tackle long textual inputs. We evaluate and benchmark various models according to our custom metrics in order to determine their effectiveness.

1 Introduction

With the rise of information technologies, the digital revolution has begun to have an unprecedented impact on people's everyday life, particularly on how books can be presented and delivered. Digital books have promoted wide sharing of resources and encouraged distribution of public domain texts. However, the flood of e-books have posed a challenge for e-book maintainers on how these books can be automatically tagged and organized. This is especially important when an e-book is from an anonymous source, and it can take a lot of human effort to correctly label a new book solely based on its contents. We therefore would like to tackle this problem with machine learning techniques, to facilitate cheaper subject labeling for the e-books as they arrive one pile after another.

Formally, our task is to perform multi-label classification on textual inputs. Text classification is a classical task in the domain of Natural Language Processing. The objective is to assign one or more labels to the given textual input. Despite fewer number of works focusing on the multi-label setting in this field, significant progress in Natural Language Processing has been accomplished to enable promising applications, especially those that are contingent on the classification of long documents, where the commonly used methods exhibit significant drawbacks.

We attempt to face this challenge in a distributed way. The dataset we are dealing with is from Project Gutenberg, a library of over sixty thousand free e-books. The size of this dataset is sufficiently large for us to train a machine learning classifier that could be suitable for our task. Details of the dataset will be discussed in Section 3.

We thus set out to answer the follow questions:

- Can we infer the genre of a piece of literary work given its content?
- Can we do this in a tractable and efficient way using limited resources?
- What is the best way to support multiple genres during inference?

*Corresponding author

This project aims at building a machine learning pipeline at scale[2]. To answer the questions above, we will first describe the machine learning techniques we have used as an attempt to solve the task, followed by how we have managed to apply them in a scalable setting for our large e-book collection. Performances and costs are then discussed to compare different methodologies, Finally, we will conclude with a discussion of how multi-genre inference can be best done, as well as room for future improvement.

2 Background and Related Work

Predicting the genre of a book given its content would be useful in the cases where an online book library obtains new books that do not have associated metadata. The heterogeneity of the sources these e-books are coming from suggests that that qualities of the associated book metadata vary significantly, with some missing and others incomplete. Out of all the fields in the metadata, the subject is oftentimes the least straightforward to determine, unless one actually reads through the book itself. As a result, we see this as an excellent opportunity challenge to solve an outstanding problem, with the premise that the online library community would be able to use our tool to streamline the e-book management process.

Multi-label classification has long been a research topic [**? ?**], and it has seen many applications for classifying texts, categorizing audio tracks, and building recommender systems [**? ?**]. Techniques that have been applied include kernel-based methods [**?**], boosting methods [**?**], deep learning [**?**], active learning [**?**], etc. More specifically for deep learning, there has been huge progress in the area of NLP. Recent advancements such as attention-based methods [**? ?**] for text classification set us free to use variable-length inputs without necessarily of compressing them down to a fixed-size vector. On the other hand, many of these methods suffer from an increased computational complexity when dealing with long input sequences, with a couple of models proposed as an attempt to address these issues [**? ? ?**].

In regards to our proposed task, Oramas et al. in their recent work [**?**] has a similar setting. We contrast that they are predicting music genres from data not limited to text. Feature embeddings learned via deep learning techniques are used to classify audio tracks into 250 genre classes, with a focus on the efficacy of different modalities in the data. The dataset we use has also been used by other works [**? ?**], but all works we have seen that work on the Gutenberg collection only focus on single-label classification among all categories, or labels under a specific subcategory.

3 Dataset

Project Gutenberg is a large-scale volunteering-based movement to digitize and archive cultural works, so as to "encourage the creation and distribution of e-books" [3]. The project was initially founded by Michael Hart, who had coined the term *e-books* in 1971. As of April 2020, Project Gutenberg has reached over 60,000 books in its collection of digital books in various formats, including plain text, HTML, PDF, EPUB, etc. It is also partially made available in various languages, including English, Latin, Chinese, etc. Summing all these together, the total size of the Project Gutenberg dataset amounts to over 742GB.

For this project; however, we only focus on the plain English texts that are available from the Gutenberg collection. That leaves us with a raw dataset of 65GB that contains 48,954 books. This can be directly downloaded using existing scripts to an *Amazon S3* bucket. Along with the textual content, each digital book comes a unique ID and a corresponding bibliographic record. The record contains metadata about the book, such as title, author, language, and most importantly, a list of subjects. Each book can have one or more subjects (and in rare cases, none) associated with it that describes the content, the genre, the generation, and various other information about the book. We see these subjects as labels, and formulate our task as to predict the multiple subjects given the plain text.

To investigate the label distribution, we performed *data cleaning* and found there were almost 20,000 subjects in total. Due to their highly-skewed distribution, we filtered out the subjects that appear in less than 100 documents. This results in a total of 278 subjects, which capture about 66% of all the

[2] As part of the 10-605 final course project

[3] Quoted from Project Gutenberg Mission Statement

existing subject occurrences. Note that this does not mean we will be removing up to a third of our data. Instead we will only be removing parts of labels from the datapoint themselves, which will not have a large effect on the data size. Details on label cleaning will be discussed in Section 5.2.

4 Methodology

4.1 Overview

We compared numerous machine learning techniques, including traditional classification methods such as decision tree, Naïve bayes, logistic regression, random forest, as well as a modern neural network approach that performs fine-tuning on a pre-trained BERT model – the latter being the state-of-the-art language model for most NLP tasks. A N-nearest neighbor (also known as majority vote) classifier is also run as a baseline to compare performance.

Decision trees are a classic non-parametric learning method for classification. It splits the data into separate nodes using Gini coefficient or information gain. Random forests are ensembles of independent decision trees, each of which is trained using only a subset of the features in the training set to ensure robustness and generalization. Naïve bayes is a generative classifier with the assumption of conditional independence between features. It is a probabilistic model that models the joint distribution and predicts the posterior probability for each possible class. Logistic regression is a discriminative classifier that also models the probability of a certain class, but differs from Naïve Bayes in that it directly models the posterior probability.

BERT (Bidirectional Encoder Representations from Transformers) [**?**] is a language model that pre-trains bidirectional transformers to obtain a better contextualized text-based embeddings used for a variety of downstream tasks. Compared to previous methods using uni-directional training, this architecture is better at capturing context with bidirectional training. BERT has empirically proven itself to be effective in most natural language processing tasks. We view BERT as a promising model that may contend for the best performance in this project, with the possible drawbacks being training time and model size.

4.2 Metrics

Model performances are evaluated using several classification metrics including accuracy, precision, recall and F-1 score. Accuracy is a common metric for classification tasks, which is the proportion of correctly classified samples among all the samples in the dataset. However, each book has varying numbers of labels, making it extremely difficult for the model to perfectly match the target output. To relax this constraint, we first compute, for each label class c, the number of samples the classifier has made in terms of true positive predictions (TP) and true negative predictions (TN). We then define our accuracy metric to be the weighted average of the correctness over all the label classes. More formally, it can be written as

$$\text{accuracy} = \sum_{c \in \mathcal{C}} w_c \cdot \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}[(c \in f(x_i) \wedge c \in \hat{f}(x_i)) \vee (c \notin f(x_i) \wedge c \notin \hat{f}(x_i))],$$

where f denotes the ground truth function, and $\hat{f}$ denotes the learned classifier. $f(x_i)$ would thus represent the multi-label output of the ground truth function for observation x_i; and $\hat{f}(x_i)$ would represent the multi-label output of the learned classifier for the same observation. Weights are determined based on the label class frequency. Considering that labels with higher frequency tend to be more influential than others, labels with more observations are given greater weights. To ensure all weights sum to 1, we normalize the weights through dividing each label frequency by the sum of all frequencies.

We follow the same logic when computing other metrics including precision, recall and F-1 score for the multi-label output. We calculate the metric for each label class, and compute their average weighted by the (normalized) number of samples for each label. In this way, all of our evaluation metrics have managed to take label imbalance into account to reflect the model performance in a more reasonable manner.

4.3 Justifications

Let's revisit our questions at the beginning and justify how the machine learning techniques described above are going to help provide an answer.

Question 1. Can we infer the genre of a piece of literary work given its content?

Yes, we can. Our models are capable of classifying segments of e-books into a list of possible genres. The model performance can be quantified by our resulting metrics shown in Table 1: Table that represents metric result. Among all models, BERT has achieved the best performance with an weighted average accuracy of 0.9284. It also improves other metrics including precision, recall and F1-score, which we arguably care more about due to the class imbalance. The resulting metrics effectively show that our BERT Model is able to give good inference on the genre given a paragraph of literary work, with imbalanced classes well-handled.

Question 2. Can we do this in a tractable and efficient way using limited resources?

Our entire dataset contains 65GB of English e-books from Gutenberg, which is challenging to work on with limited resources. In order to reduce the work and time during training, we preprocessed the data on AWS cloud provider using Spark and then stored the data in S3 bucket. As described in section 4.4, we sampled paragraphs from books instead of using the whole text, which resulted in less than 1G of input data, a feasible size to load into the model. To further improve the training efficiency, we applied multi-GPU distributed learning for BERT using the *DataParallel* with *PyTorch*. It makes several copies of model parameters and sends one to each GPU. Each GPU then runs the computations on its own partition of data in parallel, which efficiently speeds up the whole training process.

Question 3. What is the best way to support multiple genres during inference?

For machine learning models, in order to convert probabilities to labels after we obtain the raw scores produced by each model, we initially tried a conservative strategy where we simply picked a threshold and declared classes that have a raw probability larger than the threshold to be the predicted classes. We also tried a more radical strategy where we kept picking probabilities from a sorted list of class probabilities until the sum of the them exceeded a certain threshold (0.2 worked well). Both methods allow us to obtain a flexible set of output labels but can also each demonstrate different behavior depending on the use case.

For our BERT model, a threshold of 0.2 was used on the `sigmoid` output of the model, and we use majority votes to aggregate results of different paragraphs from the same e-book. The threshold was tuned via grid search on a validation set, which optimizes for books having a varied number of output labels. As mentioned in 4.4, we used multiple `(key_paragraph, label)` pairs to represent one e-book. When we generate results, each key paragraph would vote for the labels it predicts for this e-book, and we keep labels that won majority (half) votes from all paragraphs.

4.4 Applicability

As a starting point, we used *term frequency-inverse document frequency* (TF-IDF) as the feature extractor for our input text. TF-IDF is widely used in text mining to reflect the importance of a term to a document in the corpus. Considering that the size of the vocabulary can be extremely large and many models require the inputs to have fixed length, we take advantage of the *HashingTF* method implemented in *Spark* which uses a feature hashing-based technique to reduce input dimensionality. This approach avoids the need to compute a global term-to-index map which can be expensive for large texts, especially in our case.

For model training, one major issue originates from the length of books. Given the fact that most e-books contain hundreds of thousands words, it is difficult to load the full text into the models. In our proposed machine learning models, longer input sequences will inevitably result in increased training difficulty, both in terms of computation costs as well as RAM storage. For the pre-trained BERT model, the performance is sensitive to input sequence length which must to be limited to a maximum of 512 tokens.

Our approach for this was to sample paragraphs from the full text. These paragraphs need to be representative of the whole book. The sampling method we used for each book includes the following steps:

- Remove stopwords and count term frequencies of each word, and discard the words that appear for less than 20 times. Count document frequency of each word that appears in the entire corpus, and discard the words that appear in more than 75% books.
- Within the training set, count the occurrences of words for each class (genre). We discard the words that occur in more than 75% classes.
- Obtain a set of words with steps above, use the intersection of this word set with each book's word set as their keywords.
- Extract paragraphs where the frequency of keywords is greater than a certain threshold. We only take the first 512 tokens of each paragraph under the consideration that this is the limitation of the input sequence length for the BERT model.

In terms of applicability, the random forest model is a reasonable choice for a couple of reasons: It is an ensemble method that is fairly robust against overfitting. It is suitable for high dimensional input since features are randomly selected for each individual decision tree. Plus, all the decision trees in the forest are independent of each other and thus can be trained efficiently in parallel. Another similar tree-based ensemble method, Gradient Boosting Decision Tree classifier, is more robust and less susceptible to stochastic behavior compared to random forest, but we didn't choose this because as a boosting method it is more of a sequential algorithm (there is still possible parallelism within a single decision tree).

Logistic regression is also suitable for this task. To address scalability, distributed implementations mainly focus on implementing sparse updates and reducing storage cost. One major drawback of the logistic regression model is that it is a linear predictor by nature. We choose the Naïve Bayes model because it is also interesting in that it is a generative model and it only requires very simple computation. The resulting model is very memory-efficient, compared to the large number of trees that need to be stored by random forest.

We found that the most suitable BERT-based model was a pre-trained Bert-uncased model with one additional dense layer appended. The loss function we applied is the `BCELossWithLogits`, which combines a `sigmoid` layer and the `BCELoss` in one single calculation stage for it to be more numerically stable. In PyTorch, `torch.nn.BCELoss` has a parameter named `pos_weight` that scales the magnitude of the weight update based on this parameter. This was also a way for us to address the class imbalance issue. Output labels for each paragraph was determined by a certain probability threshold. Finally, majority votes were taken among paragraphs to obtain the final predicted labels for each book.

5 Computation

5.1 Pipeline

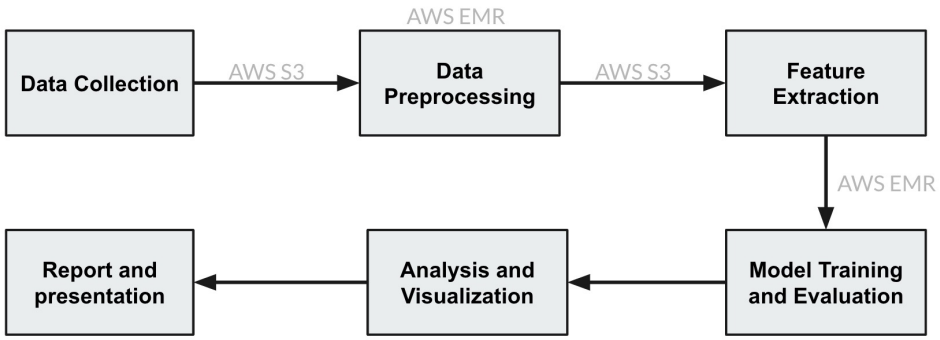

Figure 1: Project pipeline diagram

As illustrated in Figure 1, our machine learning pipeline composes of roughly six steps: Data Collection, Data Preprocessing, Feature Extraction, Model Training and Evaluation, Analysis and Visualization, and the Final Report. Data cleaning is done in both the Data Preprocessing stage, where we focus on cleaning the text itself, and the Feature Extraction stage, where we focus on cleaning the subjects associated with each book which we take as labels.

5.2 Data Preprocessing

We first downloaded 48,954 e-books from the Gutenberg Project to an Amazon S3 bucket. We then launched an AWS EMR cluster to preprocess the data and make feature-label pairs. For data cleaning, we found that some e-books were simply empty. We removed these empty e-books (11 books). For each individual book, we removed the unnecessary headers and footers for the books which are just for introducing the Gutenberg Project. Texts are then processed by uncasing all the letters, removing all the punctuations, and tokenizing the words. Note that the books themselves are almost too long to be feasible for training the classification models, with most of them having tens of thousands of words, so we plan to classify the paragraphs in these books, which are relatively short and simple. To achieve this, we transformed every book into a "list of list", where the outer list represents paragraphs, and the inner list represents the tokens in each paragraph.

Besides cleaning the texts, we also needed to clean and transform the subject information. There exists many complex subjects such as `United States - Population - Statistics`. Moreover, most of the books have more than one labels, for example, `[United States - Population - Statistics, United States - Census, Housing - United States - Statistics]`. To handle such labels, we first split them by the dashes, and then select all distinct words as the final set of labels to consider. The previous example would thus be transformed into `[statistics, unitedstates, housing, population, census]`. Despite having 19,692 different labels in total, these labels follow a distribution that is heavily right-skewed. We took advantage of this fact and filtered out labels that occur in fewer than 100 books: this greatly decreased the label space to just 278 labels. After the aforementioned data cleaning phase, we were left with a total of 25GB of text data.

5.3 Points of Interest

One interesting computation was deciding how to make a relatively fair evaluation scheme given the imbalanced label distribution that we have. Initially, we started with just the raw accuracy; however, this was rather misleading in the sense that we did not take into account the false positives and false negatives. We then added the precision and recall metrics, but these are usually only defined on a per-class basis. It is certainly difficult trying to visualize 278 precision-recall pairs. One naive method might have been just taking the average, but this discounts the fact that we have more examples in some classes over others. Finally, we defined our own "weighted" version of accuracy, precision and recall, such that the weight is proportional to the amount of datapoints in that particular class.

Another point of interest lies in the feature extraction. We knew that it would be futile to directly use the full text as inputs for our models, especially since the BERT model could only handle token sequences less than or equal to length 512. It is natural to just sample some representative paragraphs from within a book and use those as inputs instead. Initially, we chose to randomly sample 10 consecutive paragraphs from within any point in the main text. But this method proved to be quite ineffective as it did not produce good results during our evaluation stage, which indicates that the paragraphs that were selected may not have been indicative of the book's genre. We thus proposed a new method to sample paragraphs: 1. Remove all the stop words and low frequency words in every book. 2. Count the document frequency of every words and remove words that exist in more than 75% of paragraphs. 3. Count the genre frequency of each word and discard those appear in more than 75% genres. 4. For each book, we generate its keywords using the intersection of the global word set obtained from the previous steps (50k) with its own words. 5. Extract paragraphs that contain the most keywords. This method was able to yield far better results according to our evaluation.

5.4 Challenges

Out-of-memory errors were one of the more frequent errors that we encountered, both in data preprocessing and feature extraction stages. This can be observed from Spark logs. To address this,

we had to carefully plan the memory allocated for the whole cluster, for the driver, as well as for the executors. The total memory of a worker machine should be preferably at least twice as large as Spark requires based on our empirical experience. With careful calculations, we could specify the `spark.driver.memory` as well as `spark.executor.memory`, and/or launch a bigger cluster with more worker machines despite its higher price. Another approach was to avoid performing actions on RDDs and take advantage of lazy evaluation. For example, when we were processing raw data and trying to store the processed data to S3, we encountered one such out-of-memory problem. The way we got around with it was to transform the RDD into a DataFrame and directly store the DataFrame to S3. We believe this is due to the more efficient memory using that underlies the DataFrames data structure.

While we appreciated the lazy evaluation feature of Spark to reduce memory requirement, it also posed an additional challenge for us. The exception thrown on one line might originate from several lines that came before this line, making it harder to debug when there was a lot of code involved and each trial could take quite a long time to run. The unit test was also difficult to implement, as sometimes calling `take(1)` was far from sufficient to ensure every element in the RDD was transformed as expected. We solved this by breaking down the code step by step and being extra cautious before moving on to writing progressive lines of code. One particular subtle bug was that the type returned by a UDF for PySpark DataFrame had to be restricted to native Python types, and thus `numpy` functions that return numpy.int64 or numpy.float64 types could cause trouble in later serialization.

Another challenge was about version control. When we use multiple GPUs to train our model, the program will often unexpectedly stop. By examining the original Bert code and searching for related tutorials, we realized that we had been using PyTorch of an incompatible version to run Bert. Our PyTorch was the latest version, and in this version, a number of functions that were previously offered became deprecated. Since backward compatibility was not handled perfectly, the program could possibly halt with errors. By launching the instance with a previous version of PyTorch, this problem was resolved. The complexity of Bert also made it harder for us to spot errors earlier, and it slowed down our iterative process of trying out other different approaches and tackling our real task.

5.5 Resources and Expenditures

We went with *AWS* as our only cloud service provider for both our CPU servers and GPU servers. CPU servers were used to process the data and extract the features, while both types of servers were used to train the models.

In the data preprocessing and feature extraction stage, we launched an EMR cluster of m5.xlarge instances with 60GB disk. A notebook was created upon the cluster, opened in JupyerLab as provided by AWS, and run on top of Spark installed in the cluster. The EMR clusters were not the destination for our data—instead we stored and loaded all the data from and to our S3 bucket, as it would be more convenient for us to share the intermediate data. We used Linux as the main operating system and Python as the main language. The libraries we used include PySpark general, `pyspark.ml` which was the favorable choice over the original `pyspark.mllib`, and PyTorch for deep learning. For the data preprocessing stage, the whole process was completed in 4 hours; for the feature extraction stage, it took nearly 7 hours to finish.

During our model training process, we used memory-efficient r5.xlarge instances for the baseline models that do not require many matrix multiplications. For the BERT model, we chose GPU instances since GPUs will offer much faster matrix computations for deep learning compared to CPUs. In addition, we also used parallel training techniques to accelerate the training process. To achieve this, we launched a g4dn.12xlarge instance with four GPUs which support multi-GPU learning to realize data parallelism. Computations in this stage are largely dependent on previous stage. One complete training process for Bert-based model took 4 to 5 hours.

6 Results

In this section, we carried out a comprehensive suite of experiments to evaluate the performance of our models on the real-world e-book dataset collected from project Gutenberg. We also showed that our distributed machine learning models can perform reasonably well on large datasets, in terms

id	actualLabel	predictedLabel
30428	[2, 67, 198, 246]	[0, 1, 3, 2, 7, 9]
32667	[0, 10, 7]	[16]
33013	[64]	[0, 1, 3, 2, 7, 9]
33760	[]	[0, 1, 3, 2, 7, 9]
37261	[0, 33, 71]	[0, 2, 1]
38108	[0, 6, 71, 14, 50, 57]	[0, 1, 3, 2, 7, 9]
39094	[2, 46]	[0, 1, 3, 2, 7, 9]
39520	[0]	[0, 1, 3, 2, 7, 9]
44134	[]	[0, 1, 2, 3, 4]
45166	[25]	[0, 1, 3, 2, 7, 9]
47492	[0, 1]	[0, 1, 3, 2, 7, 9]

(a) Outputs from Decision Tree

id	actualLabel	predictedLabel
30428	[2, 67, 198, 246]	[0, 2, 38, 1]
32667	[0, 10, 7]	[0, 1, 3, 2]
33013	[64]	[16, 0]
33760	[]	[0, 1, 37, 12, 3]
37261	[0, 33, 71]	[0]
38108	[0, 6, 71, 14, 50, 57]	[0, 3, 2, 1]
39094	[2, 46]	[0, 2, 1, 3, 7, 16]
39520	[0]	[0, 1, 16, 3, 2, 7]
44134	[]	[1, 0, 5, 38]
45166	[25]	[0, 1, 3, 4, 17, 5]
47492	[0, 1]	[3, 0, 1, 16, 2]

(b) Outputs from Logistic Regression

id	actualLabel	predictedLabel
30428	[2, 67, 198, 246]	[0, 16, 2]
32667	[0, 10, 7]	[0, 3, 1]
33013	[64]	[16]
33760	[]	[37, 43]
37261	[0, 33, 71]	[0]
38108	[0, 6, 71, 14, 50, 57]	[3, 7, 0]
39094	[2, 46]	[0, 7, 1, 2, 9]
39520	[0]	[0, 1, 3, 12, 4]
44134	[]	[40, 0, 38]
45166	[25]	[1, 43, 54, 17, 0]
47492	[0, 1]	[222, 16, 0, 3, 273]

(c) Outputs from Naïve Bayes

id	actualLabel	predictedLabel
30428	[2, 67, 198, 246]	[0, 1, 3, 2, 16]
32667	[0, 10, 7]	[0, 3, 1, 2]
33013	[64]	[0, 1, 3, 2]
33760	[]	[0, 7, 1, 9, 11]
37261	[0, 33, 71]	[0, 2, 1]
38108	[0, 6, 71, 14, 50, 57]	[0, 7, 9, 11]
39094	[2, 46]	[0, 3, 1, 2, 7]
39520	[0]	[0, 3, 1, 7, 16]
44134	[]	[0, 1, 2, 3]
45166	[25]	[0, 1, 2, 3]
47492	[0, 1]	[0, 3, 1, 2, 16]

(d) Outputs from Random Forest

Figure 2: Sample inference results from different machine learning models

of both performance and efficiency with limited resources. And finally, we will conclude with our thoughts on the best way to perform inference for multiple genres.

Setup. We split our dataset into a training set, a validation set and a test set, with an approximate $3:1:1$ ratio. In our setup, a preprocessed text was provided to a model and was tasked to output a multi-label prediction $\mathbf{Y} \in 2^{\{0,1,\cdots,C-1\}}$ where the total number of classes to consider is C. We used this setup for all models we were to explore: random forest, logistic regression, decision tree, Naïve Bayes, as well as the BERT-based deep network. We compared their performance using our custom metrics, and then looked into the generalization capabilities of our models by inspecting the distribution of predicted labels.

Baseline model. We evaluate the performance of our models against the N-Nearest Neighbor method which is an extremely easy and efficient baseline for multi-label prediction. In our experiments, this model simply predicts

$$f(x) = \left\{ \underset{0 \le c < C}{\operatorname{argmax}} \sum_{x_i \in X_{train}} \mathbb{1}[c \in f(x_i)] \right\}$$

6.1 Content-based literary genre inference

System output. Figure 2 provides some example outputs of our machine learning classifiers. From the figures we can get a sense of what the system is able to produce and some apparent behavior discrepancies across different models, which will analyze later in this section. Given a text from a

Table 1: Performance metrics for different machine learning models applied

Model	Accuracy	Precision	Recall	F1 score
N-Nearest Neighbor	0.8965	0.0250	0.1037	0.0399
Random Forest	0.8043	0.0408	0.2131	0.0623
Logistic Regression	0.8282	0.0584	0.2045	0.0749
Decision Tree	0.8003	0.0406	0.2144	0.0641
Naïve Bayes	0.8744	0.0529	0.1380	0.0698
Fine-tuned BERT	**0.9284**	**0.2517**	**0.3419**	**0.2746**

book as input, our model will produce an array of index that maps to a label pool with a size of 278. This array represents all the possible genres that the model believes this book should belong to.

Performance. Table 1 compares the performance of the various models using the evaluation metrics we have described in Section 4.2. The BERT-based model achieved the best performance in all of the metrics we have considered, outperforming the second-best models by 5.4% absolute (6.1% relative) in terms of accuracy, 20% absolute (342.5% relative) in terms of precision, 12.8% absolute (59.5% relative) in terms of recall and 20% absolute (266% relative) on F1 score. This provides great evidence that deep neural networks are extremely powerful tools for complex prediction tasks.

It is never easy for the neural network to achieve such a remarkable result. Combining IGF (Inver-Genre Frequency) and TF-IDF as input features for BERT works much better than our previous feature engineering approaches. Also, whether to use the first couple of paragraphs or to sample paragraphs from the middle for training can also yield a noticeable difference in prediction results.

It could be fairly surprising at first sight to see that the N-nearest neighbor baseline is on top in terms of accuracy among all non-deep learning models. But this happens to reflect the exact drawback of this metric—large number of true negatives predicted by this baseline can inflate its accuracy score to even be comparable to more complicated models. Therefore, it would make more sense to look at the F1 score, which is a harmonic average over precision and recall. From F1 scores, we observe that logistic regression and Naïve Bayes outperform the two decision-tree based methods by a noticeable margin. This might be caused by the fact that probabilistic models are operating in the real fields which give more model capacity, while decision-tree based methods are mostly based on counting and statistics.

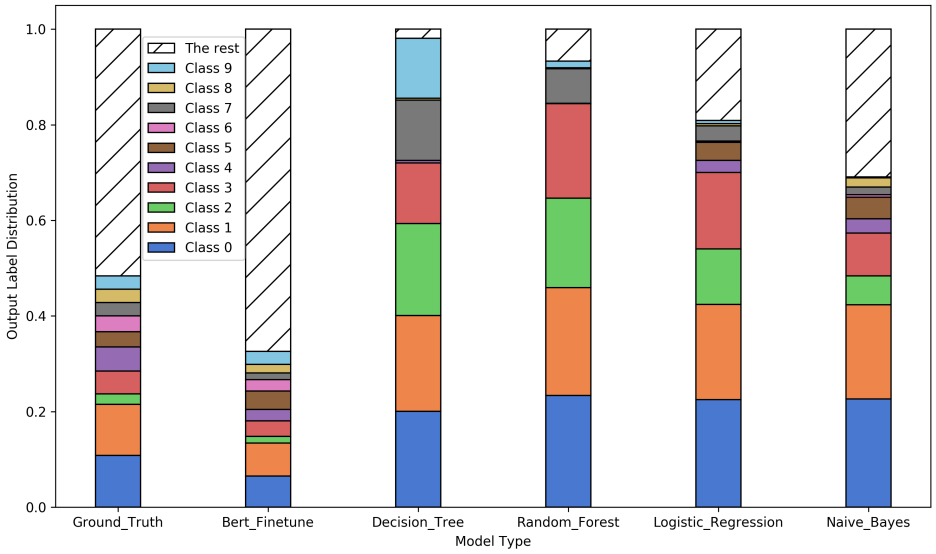

Figure 3: Distributions of label classes predicted by different models

Generalization. Due to the severe imbalance of labels in the training data set, we found that many of the machine learning models over-fitted as we might expect. Figure 2 shows some sample predictions from these models. Take Figure 2(a) as an example, the predictions from Decision Tree can be found to make similar or even identical predictions for different books, while for other models the predictions exhibit a bit more variety. To further inspect such phenomenon, we made Figure 3 which plots the distribution of different genres predicted by our different models to demonstrate their generalization capability and how well these models can fit the actual data distribution. The legend in the figure was sorted by their occurrence frequency in the whole dataset in descending order, with class 0 being the most frequent label class.

From Figure 3, we found that all non-deep learning models showed an extreme tendency towards classes that have the most number of samples, especially for classes 0, 1, 2 and 3. The decision tree model exaggerated the distribution the most, followed by random forest, as both are more dependent on counting. Naïve Bayes and logistic regression shows a distribution slightly closer to the original. BERT has the best generalization ability as it exhibits the closest distribution of class predictions. With this, we conclude that our deep learning model can successfully outperform the baseline models as well as produce meaningful multi-genre predictions.

6.2 Efficient machine learning on large dataset

To evaluate the efficiency of our distributed machine learning pipeline, we also compared the cost-effectiveness for each model, as is shown in Table 2.

Table 2: Cost-effectiveness for all the machine learning models applied

Model	Machine type	Cost per minute	Training time	Total cost
Random Forest	6 × r5.xlarge	0.0252	5.5 minutes	$0.1386
Logistic Regression	6 × r5.xlarge	0.0252	20.1 minutes	$0.5065
Decision Tree	6 × r5.xlarge	0.0252	4.0 minutes	$0.1008
Naïve Bayes	6 × r5.xlarge	0.0252	1.3 minutes	$0.0328
BERT-based Model	1 × g4dn.12xlarge	3.9120	5 hours	$19.56

Table 2 compares the cost-effectiveness for each model based on cost per minute, training time and total cost. Random Forest, Logistic Regression, Decision Tree and Naïve Bayes all had the lowest cost per minute and Naive Bayes had the lowest training time and the lowest total cost. While the BERT model had to use the most expensive machines and requires more time to train, the overall cost was still acceptable considering the budgets assigned to us. Note that our models were trained on a large dataset with a size over 25 GB, so our distributed training pipeline was highly cost-effective and it was able to perform the required task with limited resources.

Down to specific training time required for each model, a closer look into the *Scala* source code revealed that the `DecisionTreeClassifier` implemented in Spark was actually invoking the `RandomForestClassifier` with number of trees equal to 1. Since random forest is highly parallelizable due to the mutual independence among all of its trees, an ensemble of 100 decision trees would not bring much extra overhead during training compared to one single tree. In contrast, logistic regression as an iterative optimization problem has to perform weight updates following a bulk synchronous parallel (BSP) model. This synchronization at each iteration greatly slows down the training process compared to other baseline methods presented here. One remedy would be to try asynchronous optimization algorithms so that more time could be spent on computation instead of coordination.

6.3 Future work

During the training and inference process, we noticed that there were still possible improvements in our model training pipeline. We list the following ideas that we chose to not expand upon in this project due to our time constraint, but could be interesting to explore as future work.

Paragraph Selection. Taking a whole book as input, our inference models need to decide which paragraphs to use as input features for the machine learning model. Out of the several methods we

had explored, using a collection of keywords as the query as input to a best-matching algorithm to extract key paragraphs worked very well. Some possible alterations include but are not limited to: 1. Randomly select N paragraphs across the book. 2. Combine the predictions from all of the paragraphs and select the top K genres. 3. Choosing five consecutive paragraphs in the middle of book. 4. Develop neural network to select the best paragraphs.

Threshold Decision. Since the inference models only produce probability instead of direct label predictions, we need a way of mapping those results to real genres. For now, we tried a conservative strategy where we simply picked a threshold and declared classes that have a raw probability larger than the threshold to be the predicted classes. We also tried a more radical strategy where we kept picking probabilities from a sorted list of class probabilities until the sum of the them exceeded a certain threshold. Both allowed for a flexible size of the labels produced but can show different behaviors when it comes to particular scenarios. We leave finding the optimal way to map from probability space to multi-label space as future work.

7 Conclusion

In this paper, we explored how to predict an e-book's genre given it's content. We implemented various machine learning models to facilitate this process, including N-Nearest Neighbor, Random Forest, Logistic Regression, Decision Tree, Naïve Bayes and a fine-tuned BERT model. The fine-tuned BERT model achieved the best results, but it was also the most demanding in terms of time and resources. The other models were more cost-effective especially in our distributed setting. We made contributions in feature engineering, by using TF-IDF and IGF (Inverse Genre Frequency) features and a paragraph selection heuristic to cut down training and inference time and boost model performance. In this way, we were able to perform text classification on very-long documents. To tackle challenges within the multi-label classification setting, we set a probability threshold and employed paragraph-majority votes for multi-genre inference. We developed custom evaluation metrics so as to better interpret our results in a fair way.