Automatic Literary Genre Identification

Dilley Christopher **Mian Safeer** **Pirina Inna**

Abstract

In this project we attempt to classify a corpus of fictional novels into eight corresponding literary genres. After acquiring the data files and processing them, we apply a convolutional neural network model to achieve the automatic classification.

1 Premise

The purpose of this project is to train a machine learning model to ascertain the literary genre of a novel based on its textual content. A literary genre (from Latin *genus*, kind, sort), determines the category of a certain literary composition. This category, in turn, describes such features as: the writing style, tone, mood, technique, length, and content of a literary composition. Identifying the literary genre of a novel is something humans can do intuitively (with some subjectivity), but because genres are defined so flexibly and loosely, it would be challenging to explicitly identify them by direct means in a computer program.

We are hoping that the model will be able to pick up on the qualities of a book that cause it to belong to a particular genre, and have some success at labeling unknown texts. To narrow the scope of the identification, we are limiting the model to identify **fiction novels** that belong to one (or more) of the following genres:

- Adventure
- Crime/Mystery
- Fairy Tale
- Fantasy
- Horror
- Romance
- Science Fiction
- Western

2 Data Source

We are primarily utilizing texts acquired from ManyBooks. We also used some texts downloaded directly from Project Gutenberg to fill in some gaps, but the bulk of the data came from the former. The process by which we acquired these texts is detailed below.

2.1 ManyBooks

In order to acquire the data from ManyBooks we had to write our own scraper, since it is practically impossible to manually download all the books of all needed genres, and there is no option to download the books in bulk.

The main purpose of the scraper was to generate data in a format that we could further manipulate and process as desired. For instance, we created data lists, where every book would have a name, an author, and a link to download the textual content.

When the scraper's PHP code gathered all the data, we processed it using simple javascript. For instance, we removed any images, extra line breaks, unnecessary div anchors, and extracted the names of the book from the titles.

We came across a problem in downloading the files from the website, as for each file the site would offer a five-minute valid link, the name of which would represent some encoded reference name of a specific book. However, having downloaded a few files manually, we noticed the download link pattern. So we used the link patterns against each entry and created a download link manually.

Going through the generated data lists with a Python script (using Requests) and utilizing the adjusted download links, we were able to acquire books for six out of eight genres.

The code for this aspect of the project can be

found in the *ManyBooks/* directory.

2.2 Project Gutenberg

After downloading all texts made available from their download page, we extracted as much information from the raw text files as possible and used various methods to assign genre labels to these texts. An explanation of this entire process can be found in *Gutenberg.html*, and all files associated with the process are found in the *Gutenberg/* directory.

By doing so, we managed to collect and successfully label 1,964 texts.

However, we found that most of these texts acquired and labeled in this manner made up a smaller subset of the ManyBooks data set. As a result, only a small handful of these texts were utilized in the final corpus (specifically, those texts belonging to the Crime/Mystery and Fairy Tale genres).

3 Processing of the Data

After acquiring all texts and sorting them by genre, we underwent a process of preprocessing the data. This involved cleaning up unnecessary garbage from the texts (such as the Gutenberg Project intro), making sure all of the files have the same encoding, as well as conforming them all to a standardized tokenization scheme.

3.1 Description of tokenization process

We had to make sure that all of the texts are properly and uniformly separated into sentences and tokens. In order to do that, we chose the already verified tokenization models provided by the Open NLP Tools. Their models are available to download from their official website.

We applied the English Sentence Detection model in a java program to split the texts into correct sentences. Then we used the English Tokenizer model to split each sentence into tokens. We then wrote the tokenized texts back to the files, with each sentence on a new line and each token separated by a space.

All files associated with this tokenization process are located in the *Tokenization/* directory.

4 Final Corpus Results

After preprocessing, we were left with our final corpus of labelled texts. In total, the corpus contains 5,776 individual texts with 376,089,468 individual tokens in 515,323 unique token types. These texts are divided into separate folders representing the 8 genres they were labelled by. Some of the texts are duplicates, as texts that belonged to multiple genres were copied and placed in the folders of all applicable genres.

A breakdown of the corpus by genre is depicted below:

Genre	Files	Tokens	Types
Crime/Mystery:	134	11,126,930	62,274
Fairy Tale:	47	2,680,628	37,717
Adventure:	1,767	134,857,684	287,165
Fantasy:	278	19,780,694	124,992
Horror:	174	7,342,300	70,902
Romance:	1,560	129,766,454	263,058
Science Fiction:	1,321	34,746,711	148,170
Western:	495	35,788,067	113,285
TOTAL:	5,776	376,089,468	515,323

Table 1: Final Corpus Results

5 Implementing the Machine Learning

We considered a couple of different approaches for defining models to train on our corpus. However, we had to exclude the Logistic Regression model, because the literary genres, as mentioned above, are very loosely defined. This would result in a set of very inefficient or ineffective classification features.

Eventually, we made our choice in favor of the convolutional neural network model.

Texts had their most frequent words removed, and were trimmed or padded to a consistent length. All words were then represented as word vectors as defined by GloVe before being fed into the model.

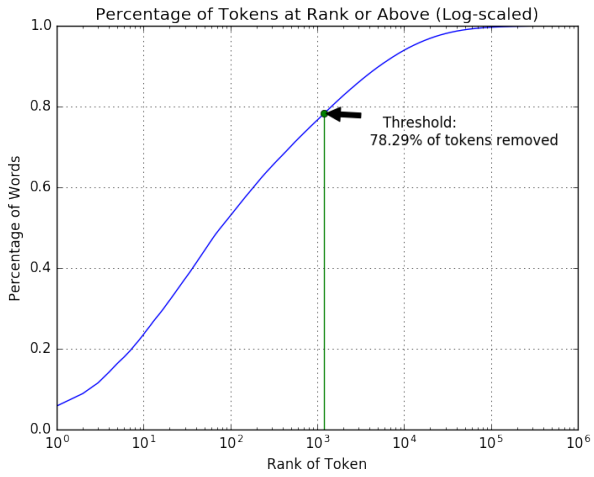
5.1 Convolutional neural network model

We opted to write our CNN model utilizing much of the code used in Assignment #3, tweaked to suit this purpose and overall improved. This code was written in Python, using the the Keras library. All code related to this process is included in the *Learning/* directory.

The first stage of the process involved simplifying the text data from string tokens into integers, and creating a word index linking words to these

integers. The total frequency of all tokens was calculated as part of this process as well. We accomplished this using Keras's tokenizer.

Due the large size of the corpus data, it became difficult to process the entirety of the texts in this manner due to memory constraints. As such, we opted to pare the texts down further by removing the 1200 most frequent tokens. This eliminated approximately 78.29% of the tokens in the documents. This value was arbitrarily chosen after manually looking through the ranked list of words and after consulting the graph depicted below. We feel this would have little impact on the learning capabilities of the model, as most of these words would not hold much predictive power. However, due to limited time, we were unable to tweak this number much in either direction to observe the resulting differences. As a result, this assertion is purely conjecture.



Furthermore, we limited all texts to contain no more than 12,000 tokens after removing the most frequent tokens. Texts that contained fewer than this many were padded with 0-tokens to reach this 12,000 value. In the end, all texts contained exactly 12,000 integer values representing specific words. The word index that linked words to their integer representation was also saved into a CSV file.

After performing this processing, all texts were saved in a separate folder in this format in order to prevent having to repeat this time-consuming task. All code related to this process can be found in *Learning/embed.py*.

The second stage of this process involved loading these processed texts, and feeding them into the CNN model. This was done by forming the texts into a multi-dimensional array of integer values as well as a vector of label values. Word embedding vectors from GloVe were also loaded, and the words represented by the texts' integers were mapped to their corresponding word vectors. After experimenting with the 50-, 100-, and 200-dimensional vectors from GloVe, it was found that the 50-dimensional vectors had the best results (in additional to being processed quicker).

The architecture of the model contains the following layers, in sequence:

- An embedding layer which converts the word identifiers into their corresponding word vectors
- A convolutional layer, with **16** convolutional kernels and a filter depth of **32**.
- A max-pooling layer, pooling the values by a factor of **8**.
- A flattening layer, converting the 2-dimensional output from the previous layers to one dimension.
- A densely-connected layer, with **128** outputs.
- A densely-connected layer, with as many outputs as genres (**8**, generally).

These values were settled upon after trying a number of different configurations and tracking their results. Increasing any of these numbers tended to give worse results, presumably suffering from overfitting issues.

After performing 10-fold cross-validation, with **2** training epochs per fold, we obtained the following results:

Fold	Accuracy	Precision	Recall	F1-Score
Fold 1:	65.06%	36.17%	35.58%	0.3509
Fold 2:	59.31%	29.00%	30.10%	0.2893
Fold 3:	62.24%	48.26%	32.91%	0.3275
Fold 4:	61.21%	24.28%	28.71%	0.2609
Fold 5:	66.26%	42.17%	36.27%	0.3604
Fold 6:	62.05%	29.90%	30.79%	0.2963
Fold 7:	59.62%	31.26%	33.85%	0.3200
Fold 8:	59.65%	37.92%	33.48%	0.3419
Fold 9:	63.94%	37.26%	31.36%	0.3059
Fold 10:	67.94%	36.26%	33.00%	0.3077
TOTAL:	62.73%	35.25%	32.60%	0.3151

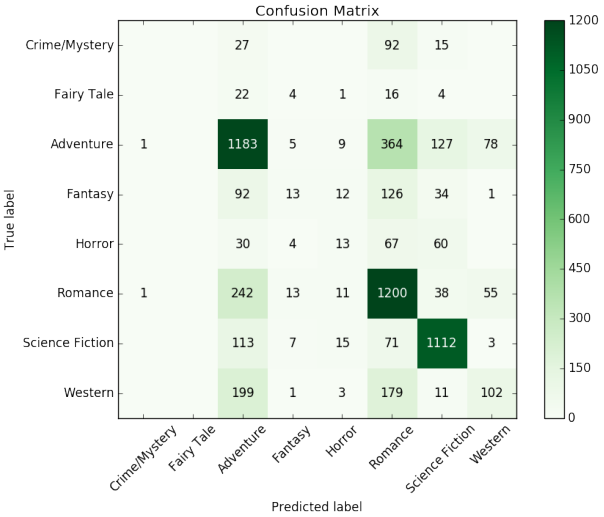
Table 2: Results by Fold (Model 1)

Genre	Accuracy	Precision	Recall	F1-Score
Crime/Mystery:	*97.65%**	0%	0%	0
Fairy Tale:	*99.19%**	0%	0%	0
Adventure:	77.34%	66.95%	62.00%	0.6438
Fantasy:	*94.82%**	4.68%	27.66%	0.0800
Horror:	*96.33%**	7.47%	20.31%	0.1092
Romance:	77.93%	76.92%	56.74%	0.6531
Science Fiction:	91.38%	84.18%	79.37%	0.8170
Western:	*90.82%**	20.61%	42.68%	0.2779

Table 3: Results by Genre (Model 1)

** The high accuracy of these genres can be attributed to the high number of 'true negatives' for this small genre. Precision and recall better reflect the poor quality of these genres' classification.*

Figure 1: Confusion Matrix (Model 1)



It is clear from these results that the imbalance in the number of texts per genre has played a significant role. The model predominantly attempts to classify the texts as either Adventure, Romance, or Science Fiction, as these are most highly populated genres. The other genres are predicted by the model far less often, certainly much less than expected based on their relative proportions.

To try combat this issue, we tried utilizing the same model, but limiting each genre to no more than **400** texts each. In the interests of time, this was performed on only one fold for one training epoch. The results were as follows:

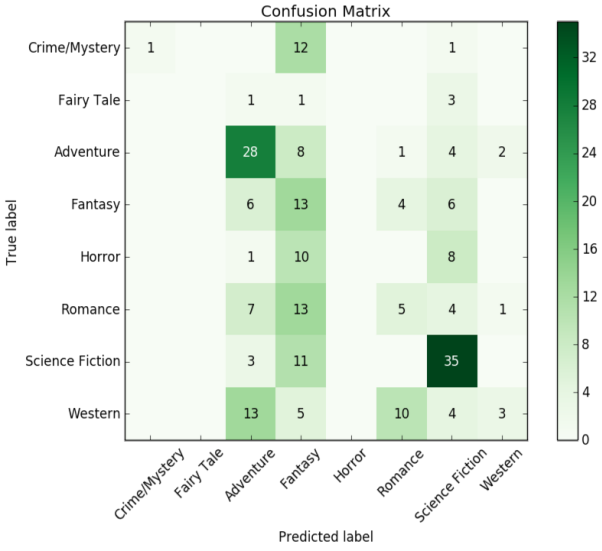
Accuracy	Precision	Recall	F1-Score
37.95%	36.76%	26.72%	0.2372

Table 4: Overall Results (Model 2)

Genre	Accuracy	Precision	Recall	F1-Score
Crime/Mystery:	94.20%	7.14%	100%	0.1333
Fairy Tale:	97.77%	0%	0%	0
Adventure:	79.46%	65.12%	47.48%	0.5490
Fantasy:	66.07%	44.83%	17.81%	0.2549
Horror:	91.52%	0%	0%	0
Romance:	82.14%	16.67%	25.00%	0.2000
Science Fiction:	80.36%	71.43%	53.85%	0.6140
Western:	84.38%	8.57%	50.00%	0.1463

Table 5: Results by Genre (Model 2)

Figure 2: Confusion Matrix (Model 2)



Making this change greatly reduced the accuracy of the model, but did improve its precision and recall. However, this is a worse result overall.

Out of curiosity, we ran the model using only those three most frequent genres in order to observe how well the model differentiates between them when alone. As with the previous experiment, this was performed on only one fold for one training epoch. These were the results:

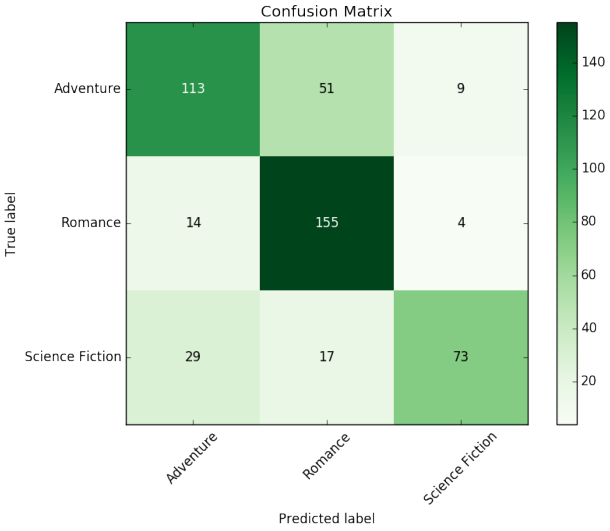
Accuracy	Precision	Recall	F1-Score
73.33%	75.61%	72.09%	0.7273

Table 6: Overall Results (Model 3)

Genre	Accuracy	Precision	Recall	F1-Score
Adventure:	77.85%	65.32%	72.44%	0.6869
Romance:	81.51%	89.60%	69.51%	0.7828
Science Fiction:	87.31%	61.34%	84.88%	0.7122

Table 7: Results by Genre (Model 3)

Figure 3: Confusion Matrix (Model 3)



These results show improved accuracy and greatly improved precision and recall, and shows the models overall power in differentiating genres when trained with a larger set of data. However, this level of accuracy is still not terribly reliable, and would likely not be good enough for a real-world classification system.

Some improvements may be found by tweaking the processing to include more (or less) of the tokens found in the texts, as well as potentially by combining the results of several models into one larger model. If given time and reason to expand on this system, these are the most likely paths for optimization we would follow.

6 Conclusion

We have attempted to classify the dataset of fiction novels into 8 conforming genres: Adventure, Crime/Mystery, Fairy Tale, Fantasy, Horror, Romance, Science Fiction, and Western. The data corpus was acquired from two websites, then cleaned and tokenized.

In order to fulfill the classification task, we applied the convolutional neural network model and came to the following conclusions: it is clear that our model is capable of identifying features indicative of particular literary genres given sufficient data, and making reasonably accurate predictions once trained. However, while certainly much better than if it were to guess randomly, it leaves room for considerable improvement.

This project can be further updated and improved to also use a different classification model (such as recurrent neural network model), include more languages and even accept individual files as input for further classification.