Books Genre Prediction using Machine Learning Models

Shweta Shrestha Thapa

March 2021

1 Definition

1.1 Project Overview

The purpose of this project is to predict a book's genre using books' summaries or description and Machine Learning techniques.

Both Machine Learning and Deep Learning has been used in this project to test with different types of algorithm.

The domain chosen for the capstone is **Natural Language Processing (NLP)**. The most famous domain in Machine Learning is perhaps the model used for prediction. The domain chosen for the proposal is Natural Language Processing. Actually, there is a long list of applications (real-life) used in this domain as listed in the article: Natural Language Processing (NLP) for Machine Learning

I, myself had the chance to explore this domain in my previous NanoDegree, Deep Learning by using Recurrent Neural Network to analyze and predict sentiments (using tweets): Sentiment Analysis using RNN.

1.2 Natural Language Processing (NLP)

Elizabeth D. Liddy[1] describes it as: "*Natural Language Processing is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications.*"

According to Gartner[2]:

"*Natural-language processing (NLP) technology involves the ability to turn text or audio speech into encoded, structured information, based on an appropriate ontology. The structured data may be used simply to classify a document, as in "this report describes a laparoscopic cholecystectomy," or it may be used to identify findings, procedures, medications, allergies and participants.*"

It means that the objective is to get "human-like" understanding and processing of natural language spoken or written or heard by a human-being.

NLP starts its journey with Alan Turing's first paper Computing Machine and Intelligence[3] where he asks a simple question: "Can machine think?" and ends now in 2021 with domestic appliances able to understand and process human language.Today, it is one of the fastest growing field in Artificial Intelligence and expected to grow from USD 10.2 billion to USD 26.4 billion by 2024 according to MarketsandMarkets and as shown in figure 1.

Even sarcasm is difficult for some human being to understand or even a joke, how can a machine understand such a complicated part of us? Rules like syntax and semantic can be learned by a machine but there are two many ways to say the same thing. The difficulty is to get the unstructured data in a format that the machine can understand it as we do.

1.3 Motivation

As personal hobbies, I like to read and write. I've seen a couple of applications created by other users to predict movie genres with based on plot summaries using Machine Learning techniques.Also, sentiment analysis projects using Deep Learning techniques. So, my motivation is to predict a book's genre based on plot summaries. Below are some links of some projects:

- Predicting Movie Genres Based on Plot Summaries
- IMDB MOVIE GENRE TAG PREDICTION
- Sentiment Analysis - My Github project

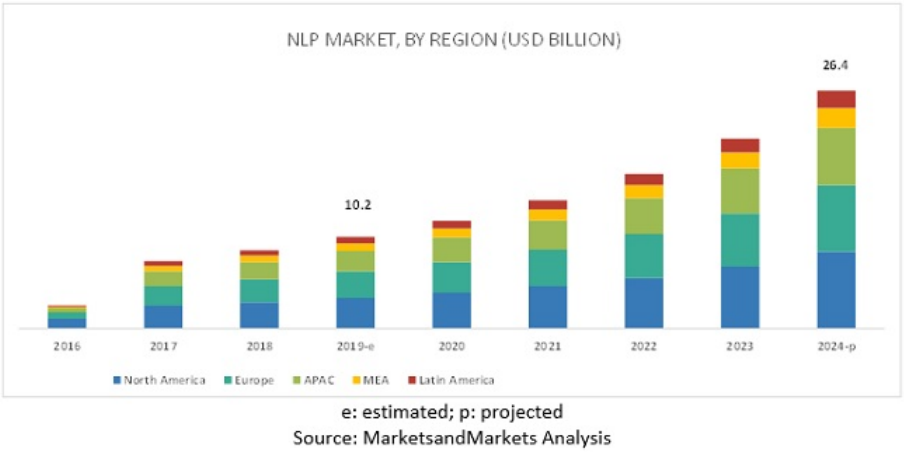


Figure 1: NLP Market, by Region (USD Billion)
Source: MarketsandMarkets

In websites like goodreads.com, the users are the ones who manually vote for the type of genre for each book. This kind of tool could be useful to predict the book's genre automatically when the author uploads the book's metadata to such websites.

1.4 Problem Statement

Predict a book's genre based on the 16,559 plot summaries available from the CMU Book Summary Database[4] and the data obtained from goodreads (web scraping) for different genres, using different supervised Machine Learning methods. It should be able to predict correctly at least 30% from the testing dataset. The testing dataset will be 4% of the clean complete dataset. The results, code and environment details will be available in the github repository in case anyone wants to replicate the results.

1.5 Metrics

As this is a classification resolving problem, the metrics used to meaºsure the different models are:

- Accuracy: Classification accuracy is the ratio of number of correct predictions to the total number of predictions made.

$$Accuracy = \frac{TruePredictedPositives}{TotalNumberofPredictions}$$

- Precision: True positives over all positives. Will be higher when the number of false positives is lower.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

- Recall: True positives over all true positives and false negatives. Will be higher when the number of false negatives is lower.

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

- F1 Score: Used to measure test's accuracy. Defines how precise a classifier is.

$$F1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

2 Analysis

This section will analyse the dataset, CMU Book Summary Database[4] which was used for training the different machine learning models.

2.1 Data Exploration and Visualization

As mentioned in the previous section, the dataset will be the one from CMU Book Summary Database[4] and also from goodreads.

2.1.1 CMU Dataset

Originally the data was in .txt format, available and used from kaggle and a notebook was created for data exploration and analysis in the same environment called book-genre-prediction-data-preparation, a dataframe was created as shown in figure 2.

The "Genres" column is in json format, after cleaning it and converting it to list, there are a total of 2154 types of different multi-genre classification as shown in figure 3. The same genre is repeated in different combinations, for example, the genre "Fiction" is repeated in other different types of combination as shown in figure 4. To avoid the multilabel problem and the complexity it involves, the genres are split individually for this project. If each genres are looked in individually, the number of plots per each one of them are as shown in figure 5. The number of samples for fiction is too high, that is because fiction genre is a generic genre for all other branches of fiction, that is why this genre will be discarded. The same with the genre novel, it will be difficult to distinguish between all the other genres because most of the summaries are from novels, so most of the books can be classified as novels. The objective is to explore more branches of genres. After selecting a list of genres according to number of samples available, the counts of summaries per each genre are as shown in figure 6a. These are the genres that will be identified or classified by the trained models.

[26]:
```
book_summary_df = pd.read_csv("/kaggle/input/cmu-book-summary-dataset/booksummaries.txt",
                              header=None,sep="\t",
                              names=["Wikipedia ID", "Freebase ID", "Book title", "Book author", "Pub date","Genres","Summary"])
book_summary_df.head(5)
```

Out[26]:

	Wikipedia ID	Freebase ID	Book title	Book author	Pub date	Genres	Summary
0	620	/m/0hhy	Animal Farm	George Orwell	1945-08-17	{"/m/016lj8": "Roman \u00e0 clef", "/m/06nbt":...	Old Major, the old boar on the Manor Farm, ca...
1	843	/m/0k36	A Clockwork Orange	Anthony Burgess	1962	{"/m/06n90": "Science Fiction", "/m/0l67h": "N...	Alex, a teenager living in near-future Englan...
2	986	/m/0ldx	The Plague	Albert Camus	1947	{"/m/02m4t": "Existentialism", "/m/02xlf": "Fi...	The text of The Plague is divided into five p...
3	1756	/m/0sww	An Enquiry Concerning Human Understanding	David Hume	NaN	NaN	The argument of the Enquiry proceeds by a ser...
4	2080	/m/0wkt	A Fire Upon the Deep	Vernor Vinge	NaN	{"/m/03lrw": "Hard science fiction", "/m/06n90...	The novel posits that space around the Milky ...

Figure 2: Pandas dataframe created from the .txt file.

Out[25]:
```
[Novel]                                                                                                  839
[Science Fiction, Speculative fiction]                                                                   567
[Science Fiction]                                                                                        526
[Fiction]                                                                                                402
[Fiction, Novel]                                                                                         381
                                                                                                        ...
[Speculative fiction, Young adult literature, Adventure novel]                                             1
[Science Fiction, Children's literature, Historical fiction, Speculative fiction, Fantasy, Fiction]        1
[Historical fiction, Novel, Historical romance]                                                            1
[Crime Fiction, Detective fiction, Children's literature, Mystery, Fiction, Suspense]                      1
[Chick lit, Comedy, Fiction]                                                                               1
Name: Genre, Length: 2154, dtype: int64
```

Figure 3: Total of 2154 types of different multi-genre classifications.

To recap, the following steps were taken to clean the dataset:

1. Convert .txt file to pandas dataframe.
2. Delete all the rows with null/empty summary and genre values.
3. Process the genre column to convert from json format to list format.
4. Only keep the summaries that are classifies as the selected genres.

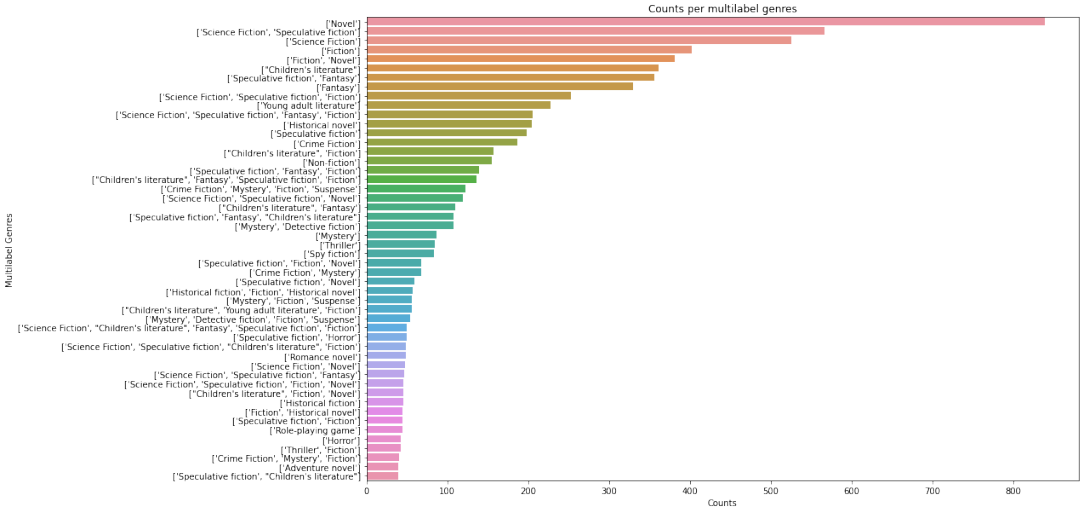


Figure 4: First 30 most repeated multilabel genres. Some genres are repeated in different combinations.

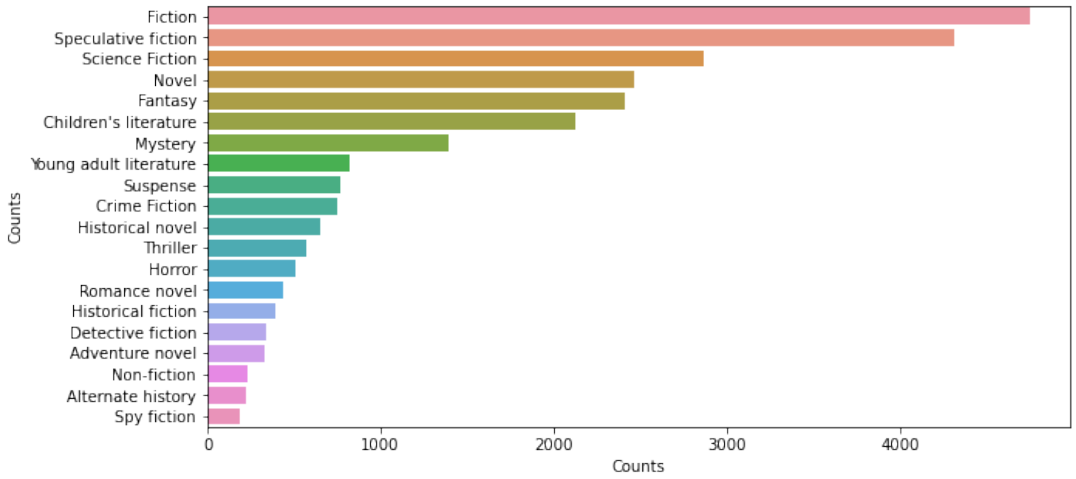


Figure 5: Book Summaries per Genre.

2.1.2 goodreads Dataset

As the data is very unbalanced for each type of genre, web scraping method was used to get book summaries for different genres which can be found in the notebook load-summaries. For each book, the extracted fields were **book title**, **book author** and **book description**. These are the lists from which the book description were extracted for each genre type:

1. Non-Fiction - 15 pages.
2. Mystery - 14 pages.
3. Young Adult literature - 12 pages.
4. Children's Literature - 5 pages.
5. Suspense - 12 pages.
6. Crime Fiction - 14 pages.
7. Science Fiction - 12 pages.

Finally, the dataset obtained from goodreads would be as shown in figure 6b. All the book descriptions were applied with the same technique as for the CMU dataset. Duplicates had to be deleted with multiple entries for different genres always taking in account the number of samples per each genre.

2.1.3 Final Dataset

The final dataset after combining both the CMU and the goodreads dataset can be found in the figure 6c. Now, the data is not that unbalanced as before.

3 Algorithms and Techniques

Both machine learning classification models and a deep learning custom model were used to get the predictions. Please note that the limites resources and the computational cost had to be taken in account while trying to tune the hyperparameters.

3.1 Supervised Machine Learning Classification Models

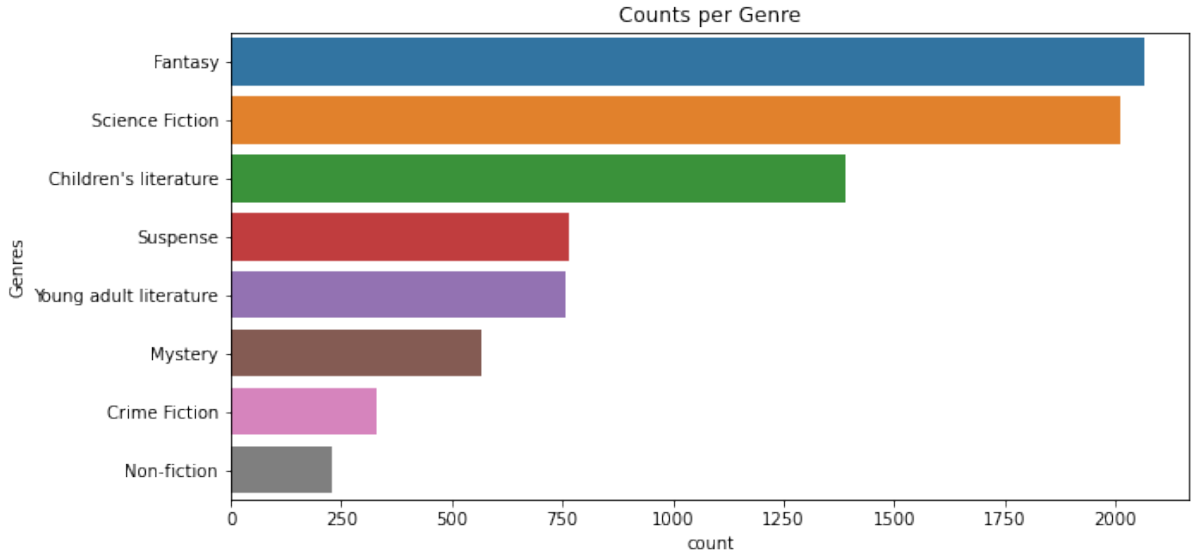
For machine learning models, the scikit-learn library was used. It is important to note that for the supervised machine learning models, the text were transformed first using the TF-iDF method. This is an innovative way of document feature extraction. TF means the terms that occur more frequently and the iDF is the inverse document frequency. iDF highlghts the most uncommon words that occur in each document. For example, the word fairy is most likely to occur fantasy genre texts than in mystery genre texts.

The function used was from the TfidfTransformer from the sciki-learn library. The list of hyperparameters can be found in the link provided.

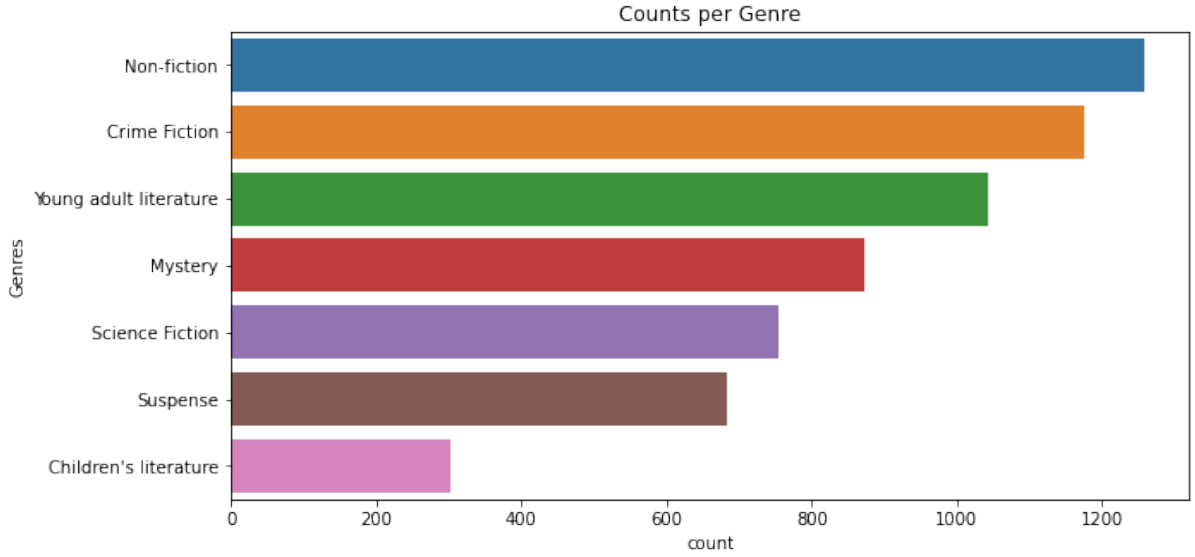
- **ngram_range:** after trying with 1-gram, 2-grams and 3-grams for each different models, different values were used for each one of them.
- **stop_words:** english.
- **use_idf:** enable inverse-document-frequency weighting. As will be explained later, for each type of model, this value was found to be useful if turned on or off.
- **norm:** To normalize each output row with type l1 or l2.
- **max_df:** When building the vocabulary ignore terms that have a document frequency strictly higher than the given threshold (corpus-specific stop words). This is used basically to reduce processing time.

3.1.1 Binary VS. Multiclass Classifier

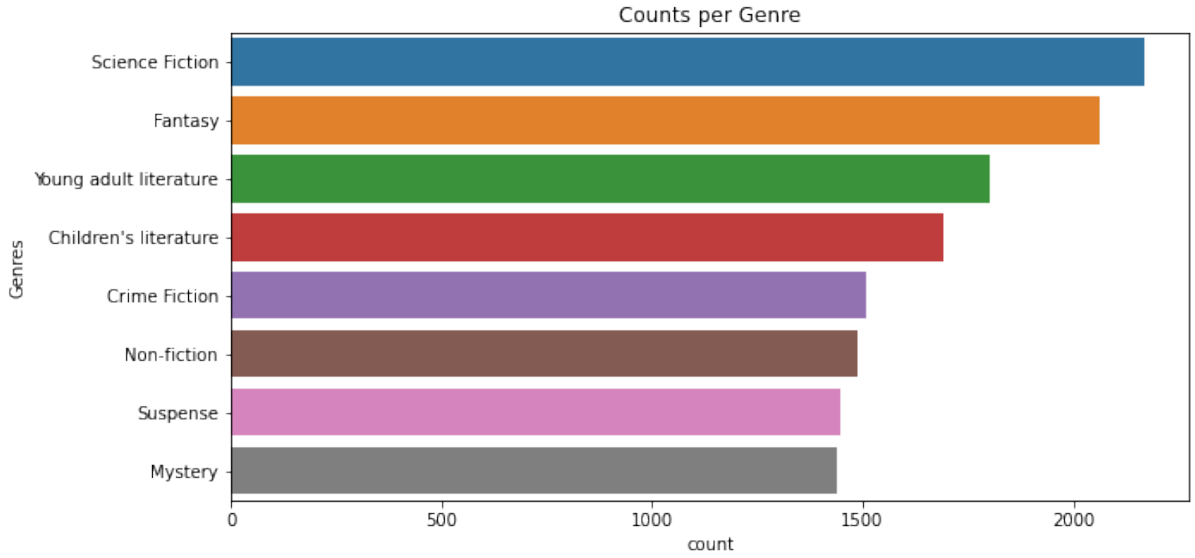
Models such as logistic regression or SVM are meant to use for binary classification. There are methods to use such models for multiclass, for this project the One-VS-Rest classifier is used.



(a) CMU Dataset - Book Summaries per Genre.



(b) goodreads Dataset - Book Summaries per Genre.



(c) Final Dataset - Book Summaries per Genre.

Figure 6: CMU, goodreads dataset and the final combined dataset.

3.1.2 Pipelines

One of the most useful method offered by sklearn is Pipeline. This way, the end-to-end model can be written in only one function. Putting it all together from the previous sections and using this function, the pipeline would look like as shown in figure 7. This is an easy way to adjust hyparameters for both feature extraction and the machine learning models in the same pipeline.

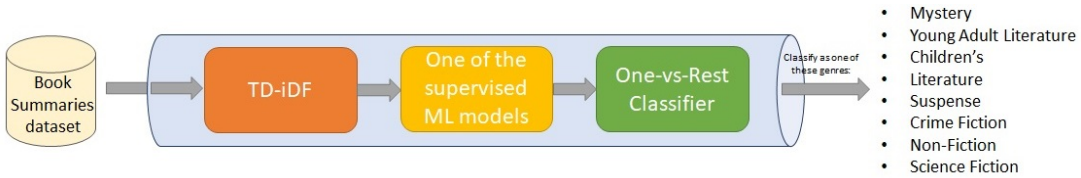


Figure 7: Pipeline for training and testing the text classification models.

3.1.3 Naive Bayes:

It is suitable for binary and multiclass classifications. The multinomial naive bays algorithm is the one most used for text classification. The function used was from the sciki-learn library Naive Bayes MultinomialNB. The hyperparameter tuned in this project for this function is:

- **alpha:** The default value of this parameter is 1.0. It is an additive (Laplace/Lidstone) smoothing parameter.

3.1.4 Support Vector Classification:

Suitable for classification and regression. As mentioned in the paper *Text Categorization with Support Vector Machines: Learning with Many Relevant Features.*[5] by Thorsten Joachims, this algorithm is one of the best one for text classification. The function LinearSVC by sklearn is used. With the defaul parameters, this function worked very well with the dataset and gave the expected results (will be explained further).

3.1.5 Logistic Regression:

Despite its name, the logistic regression is a linear model rather than a regression one. The function LogisticRegression by sklearn was used. The hyperparameters tuned for this model in this project are:

- **penalty:** To avoid overfitting of the data, here are three types of regularization l1, l2 and elasticnet or none. By default the type l2 is chosen.
- **C:** To strength the regularization. Smaller values means stronger regularization.
- **fit_intercept:** If to add a constant or bias to the function.
- **estimator_solver:** There are 5 types of solver. Each one of them for different purposes and with different combinations with the penalty hyperparameter.

3.2 Deep Learning Custom Model

Besides from machine learning models from sklearn, deep learning supervised custom model was also created, trained and tested using pytorch. The model created is as shown in the figure 8. This model was created inspired in previous deep learning projects for sentiment analysis and can be found here. Two types of recurrent neural networks are used: **LSTM** and **GRU**, resulting GRU as faster than LSTMs. To make the training faster, the library torchtext (using PyTorch) was used. torchtext has different pre-trained models and the following were used to create and compare the vocabulary dictionary and the embedding layers:

1. **CharNGram:** Charagram: Embedding Words and Sentences via Charactern-grams (100 dimension vectors) [6]. The ones used in this project was *charngram.100d.*
2. **FastText:** Pre-trained word vectors using fastText (300 dimension vectors) [7]. The ones used in this project were *fasttext.en.300d* and *fasttext.simple.300d.*

3. **GloVe:** Global Vectors for word representations. [8]. The ones used in this project were *glove.6B.50d*, *glove.6B.300d*, *glove.42B.300d*, *glove.840B.300d* and *.twitter.27B.100d*.

The different hyperparameters tuned were:

- **Epochs:** Defines the number of iterations of training for the DL model.
- **Learning Rate:** The steps the DL model should take in order to converge. Defines how fast the model adapts to the solution (to minimize the loss).
- **Number of Layers:** Defines the number of layers of the RNN units in the model to create stacked LSTMs or stacked GRUs.

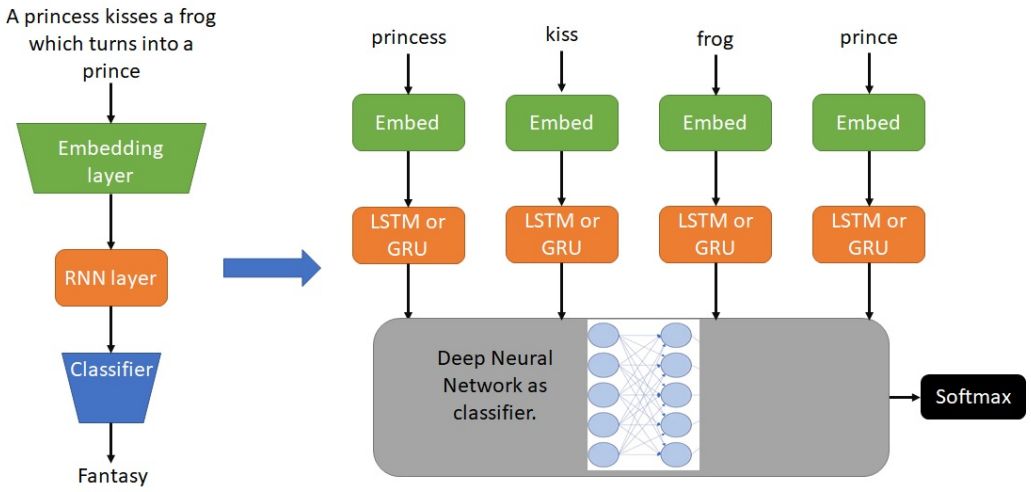


Figure 8: Deep Learning model using Recurrent Neural Networks.

4 Benchmark Model

The kaggle competition Movie Genre Classification will be taken as benchmark. Looking at the leadership board, the winner has a score of about 0.64 of accuracy. The mentioned benchmark model takes in as input movie plot summaries and predicts the movie's genre.

However, each of the models mentioned in the previous section is compared as in training time, testing time and testing accuracy.

5 Methodology

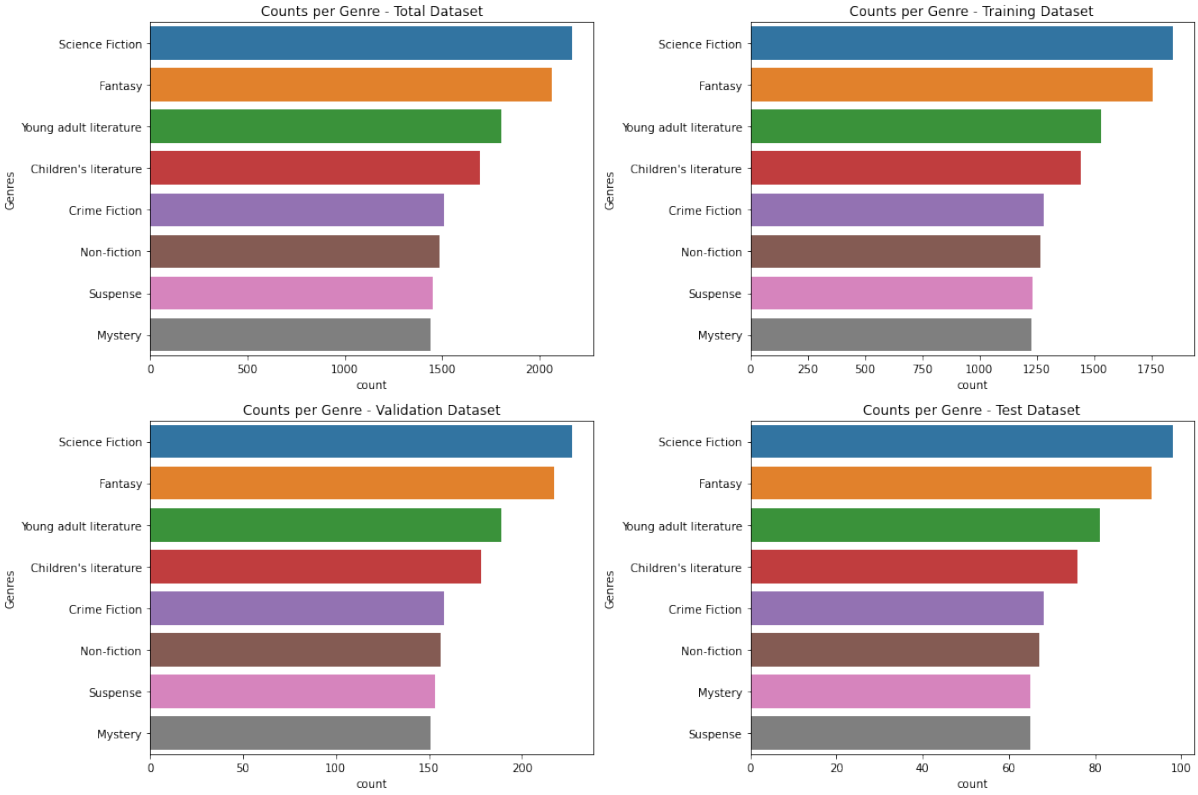
5.1 Data Pre-processing

The data was pre-processed to feed them as input to all the algorithms mentioned in the previous section.

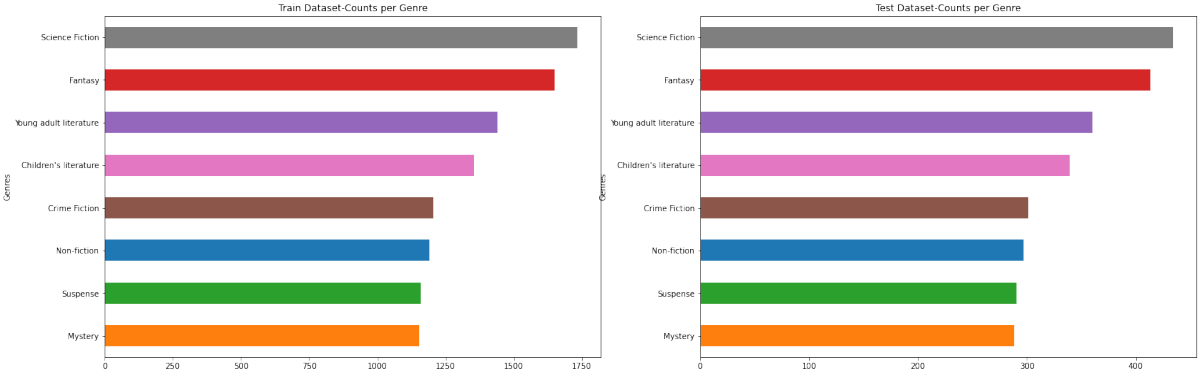
- **TD-iDF:** For all the supervised machine learning models, the feature extraction was done for the text using TD-iDF. As explained in the previous section, this has also been tuned to get the optimized result.
- **Word representing vectors and Embedding:** For the deep learning model, the pre-trained vectors and embedding as listed in the previous section were used to create and compare the classifier.

For the machine learning models, the total dataset was only divided for training and testing. And for the deep learning model, the total dataset was divided into training, validation and testing because the validation dataset is needed to avoid model overfitting.

The data was stratified using the method train_test_split from sklearn to avoid unbalancing of data for training, validation and testing data. Thus, the figure 9 shows how the training, validation and testing is split for deep learning and machine learning methods.



(a) Book Summaries per Genre - Training, Validation and Testing Dataset for Deep Learning.



(b) Book Summaries per Genre - Training and Testing Dataset for Machine Learning

Figure 9: Book Summaries per Genre - Stratified for training, validation and testing.

5.2 Implementation and Refinement

There are four notebooks and one helper script used to implement all the different solutions.

5.2.1 Data Preparation

The notebook load-summaries scraps the web goodreads and gets the book title, author and description as mentioned before and saves them in a csv file format called book_summaries_goodreads.csv.

The notebook book-genre-prediction-data-preparation is where the data exploration, data analysis and data processing it done. This notebook mainly uses pandas library basic functions like creating dataframe, concat, dropna, drop duplicates, etc. The steps taken in this notebook are:

- Load the .txt file from the CMU dataset available in kaggle. Clean and pre-process the data as explained the Analysis section.
- Load the csv output file from the load-summaries notebook. The data is already cleaned here so no need to clean it again.
- Next, is to combine both datasets and create the final dataset by using the concat function from pandas library and save it as book_summaries_goodreads.csv.
- For the deep learning model, the final dataset is split in train, valid and testing dataset as shown below using the stratify attribute in order to get the same range of samples for each genre for each dataset.

```
        from sklearn.model_selection import train_test_split
        book_summary_train, book_summary_valid_test = train_test_split(book_summary_df,
        ↪  test_size=0.15, random_state=42, stratify=book_summary_df['Genres'])
        book_summary_valid, book_summary_test = train_test_split(book_summary_valid_test,
        ↪  test_size=0.3, random_state=42, stratify=book_summary_valid_test['Genres'])
```

- Save as each dataset as train, valid and test in csv format as follows:

```
        book_summary_train.to_csv('./book_summaries_train.csv')
        book_summary_valid.to_csv('./book_summaries_valid.csv')
        book_summary_test.to_csv('./book_summaries_test.csv')
```

5.2.2 Machine Learning

The notebook predict-genre-machine-learning-models implements the machine learning models, the testing and the metrics for each one of them. First of all, it loads the book summaries total dataset generated by the book-genre-prediction-data-preparation notebook and divides it to train and valid data using the same function as shown in the previous section.

The cleaning of the text is done by using the following function clean which removes the punctuation and keeps only the alphabets, numbers and spaces. Then it converts all the text to lowercase. This function is used for each row in the dataframe for the field "Summary".

```
        import string
        def clean(summary):
            table = str.maketrans(dict.fromkeys(string.punctuation))
            text = summary.translate(table)
            text = re.sub(r"[^a-zA-Z0-9\s]"," ", summary.lower())
            return text
        book_summary_df['Summary'] = book_summary_df['Summary'].map(lambda summary : clean(summary))
```

The pipelines already mentioned in the section 3.1.2 were implemented as follows for each type of models using some default values for each of them:

```
        # Naive Bayes
        NB_pipeline = Pipeline([
                ('tfidf', TfidfVectorizer(stop_words=stop_words, ngram_range=(1,1), use_idf=True)),
                ('clf', OneVsRestClassifier(MultinomialNB())),
            ])
```

```
        NB_pipeline.fit(train_x, train_y) #fit model
        # Linear SVC
        SVC_pipeline = Pipeline([
                ('tfidf', TfidfVectorizer(stop_words=stop_words, ngram_range=(1,2))),
                ('clf', OneVsRestClassifier(LinearSVC(), n_jobs=1)),
            ])
        SVC_pipeline.fit(train_x, train_y) #fit model
        #Logistic Regression
        LogReg_pipeline = Pipeline([
                ('tfidf', TfidfVectorizer(stop_words=stop_words, ngram_range=(1,1))),
                ('clf', OneVsRestClassifier(LogisticRegression(solver='liblinear'), n_jobs=1)),
            ])
        LogReg_pipeline.fit(train_x, train_y) #fit model
```

However, to get the optimize results, the function GridSearchCV was used for hyperparameters tuning for both feature extraction and the machine learning algorithms. Each of the hyperparameters are already explained in the section 3.1. For the extensive search, for each hyperparameter, different possible values are given and then the grid search cross validation function searches for the best parameter combination for the whole pipeline within the provided ones.

```
        # Naive Bayes
        parameters = {
        'tfidf__use_idf': (True, False),
        'tfidf__norm': ('l1', 'l2'),
        'clf__estimator__alpha': (1, 0.1, 0.01, 0.001, 0.0001)
         }
        NB_grid = GridSearchCV(NB_pipeline, param_grid=parameters, n_jobs=-1, verbose=5)
        NB_grid.fit(train_x, train_y) #fit model
        # Linear SVC
        parameters = {
              'tfidf__use_idf': (True, False),
              'tfidf__max_df': [0.3, 0.5, 0.8, 1.0],
              'clf__estimator__loss' : ['hinge', 'squared_hinge'],
              'clf__estimator__penalty' : ["l1", "l2"],
              'clf__estimator__fit_intercept': [True, False],
              'clf__estimator__C': [0.01, 1.0, 2.0]
             }
        SVC_grid = GridSearchCV(SVC_pipeline,param_grid=parameters, n_jobs=-1, verbose=5)
        SVC_grid.fit(train_x, train_y)  #fit model
        #Logistic Regression
       parameters = {'clf__estimator__penalty' : ['l1', 'l2', 'elasticnet', 'none'],
              'tfidf__use_idf': (True, False),
              'clf__estimator__fit_intercept': [True, False],
              'clf__estimator__C': [0.01, 1.0, 2.0],
              'clf__estimator__solver': ('newton-cg', 'sag','saga','lbfgs')
             }
        LogReg_grid = GridSearchCV(LogReg_pipeline,param_grid=parameters, n_jobs=-1, verbose=5)
        LogReg_grid.fit(train_x, train_y) #fit model
```

Each of these models are saved in .sav format using the joblib library.
To create a classification report, the metrics function from sklearn was used:

- metrics.classification_report: Builds a text with the basic classification report metrics already mentioned in section 1.5 metrics.
- metrics.confusion_matrix: Generates the confusion matrix to evaluate the accuracy of the model.

Lastly, the inference function is created to test random book descriptions also saved in a file called "summaries_for_testing.csv" in the load-summaries notebook. It returns prediction for each type of saved/tuned model.

```
       def predict_genre(plot):
        s = clean(plot)
        lr_est = joblib.load('./LR_tuned_model.sav')
        svc_est = joblib.load('./SVC_tuned_model.sav')
        nb_est = joblib.load('./NB_tuned_model.sav')
        return (lr_est.best_estimator_.predict([s])[0],
```

```
                svc_est.best_estimator_.predict([s])[0],
                nb_est.best_estimator_.predict([s])[0])
```

5.2.3 Deep Learning

The notebook predict-genre-pretrained-dl implements the deep learning model creation and training for the book genre prediction.

It also uses the additional script (for clean coding) genre-prediction-models where the LSTM and GRU models are defined. In this script, also the train and test functions are defined.

```
    class GRUClassifier(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, n_classes, n_layers, batch_size,
    ↪  bidirectional = True,  dropout=0.3):
        super(GRUClassifier, self).__init__()
        self.hidden_dim = hidden_dim
        self.batch_size = batch_size
        self.name = "GRU Classifier"
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.gru = nn.GRU(embedding_dim, hidden_dim,
                            num_layers=n_layers,
                            dropout=dropout,
                            bidirectional = bidirectional,batch_first=True)
        if bidirectional:
            self.fc = nn.Linear(hidden_dim*2, n_classes)
        else:
            self.fc = nn.Linear(hidden_dim, n_classes)


    def forward(self, sentence):
        embeds = self.embedding(sentence)

        packed_outputs, hidden = self.gru(embeds)
        hidden = torch.cat((hidden[-2,:,:], hidden[-1,:,:]), dim = 1)
        outputs = self.fc(hidden)
        return outputs
```

```
    class LSTMClassifier(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, n_classes, n_layers, batch_size,
    ↪  bidirectional = True,  dropout=0.3):
        super(LSTMClassifier, self).__init__()
        self.hidden_dim = hidden_dim
        self.batch_size = batch_size
        self.name = "LSTM Classifier"
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim,
                            num_layers=n_layers,
                            dropout=dropout,
                            bidirectional = bidirectional,batch_first=True)
        if bidirectional:
            self.fc = nn.Linear(hidden_dim*2, n_classes)
        else:
            self.fc = nn.Linear(hidden_dim, n_classes)


    def forward(self, sentence):
        embeds = self.embedding(sentence)

        packed_outputs, (hidden,cell) = self.lstm(embeds)
        hidden = torch.cat((hidden[-2,:,:], hidden[-1,:,:]), dim = 1)
        outputs = self.fc(hidden)
        return outputs
```

The implementation is done as explained and shown in figure 8. The embedding layer is assigned with the pretrained weights/vectors from the different solutions mentioned in the section 3.2 as follows:

```
    # Choose from following list
    pretrained = ['glove.6B.50d','glove.6B.300d',
              'charngram.100d', 'fasttext.simple.300d',
              'fasttext.en.300d', 'glove.42B.300d',
              'glove.840B.300d', 'glove.twitter.27B.100d']
    vocab.load_vectors(pretrained[4])
    embedding = vocab.vectors.to(device)
    #Initialize the pretrained embedding
    pretrained_embeddings = vocab.vectors
    model_lstm.embedding.weight.data.copy_(pretrained_embeddings)

    model_gru.embedding.weight.data.copy_(pretrained_embeddings)
```

It is worth mentioning that after the fully connected layer, the softmax function returns the probability of belonging to each genre for the given text (or vector). So, in the training function, the prediction is saved as the genre which has the maximum probability:

```
    output = model(feature).squeeze()
    #Calculate the loss
    loss = loss_fn(output.squeeze(), label)
    loss.backward()
    #Clip to avoid exploding of gradients.
    nn.utils.clip_grad_norm_(model.parameters(), clip)
    #Optimize the model
    optimizer.step()
    # Save prediction with the maximum probability.
    pred = output.data.max(1, keepdim=True)[1]
```

The model is saved each time the validation loss decreases. If the validation goes on increasing while the training loss decreases, it means that the model is overfitting and this should be avoided.

6 Results

The results can be found in this section for all the models mentioned in the previous sections.

6.1 Model Evaluation and Validation

To evaluate each of the model, first each model was trained and then was tested using the testing dataset to evaluate the accuracy. The best model is considered valid when it has reached an accuracy of at least 64%.

6.1.1 Machine Learning Models

The machine learning models accuracy were quite good even without tuning the hyperparameters in some cases. The results and benchmarks can be found as shown below for each model are shown in table 1.

The model SVC with the parameters tuned is the one that took the longest but the SVC model with the defcault parameters took less than one minute with similar results.

The linear regression model even with the default values seem to perform well with 65.52% of accuracy score.

It is clear that the best model is the **SVC tuned** model with an accuracy score of 64.14%.

Models	Trained Duration	Testing Duration	Accuracy(%)
NB	0.070424	0.016258	55.547392
NB_tuned	2.845161	0.016388	62.637766
SVC	0.394362	0.028475	64.033799
SVC_tuned	90.322383	0.038774	64.144012
LR	0.133835	0.016189	62.527553
LR_tuned	65.917469	0.015956	63.335783

Table 1: Benchmarks for all the three machine learning models.

The Grid Search CV function determined the following hyperparameters to be the best ones for each model as shown in table 2.

Models	Best parameters
NB_tuned	• alpha: 0.01, • tfidf_norm: l2, • tfidf_use_idf: False
SVC_tuned	• C: 2.0, • fit_intercept: True, • loss: hinge, • penalty: l2, • tfidf_max_df: 0.3, • tfidf_use_idf: True
LR_tuned	• C: 2.0, • fit_intercept: True, • loss: hinge, • penalty: l2, • estimator_solver: lbfgs, • tfidf_use_idf: True

Table 2: Best parameters for all the three machine learning models.

It is also worth mentioning that the for each model, the accuracy changes when the ngram is defined for the tfidf function. For example, for Naive Bayes and Logistic Regression, the best accuracy was obtained with ngram (1,1). Meanwhile, for the SVC model, the best accuracy was obtained with ngram (1,2). The training time also increases when the ngram increases.

As shown in figure 10, the number of correct predictions is proportional to the number of training samples available for each genre. For example, for the genre Science Fiction, the number of samples is greater as well as the number of correct predictions. So, it is possible that if the number of training samples is increased the model's accuracy can increase with it.

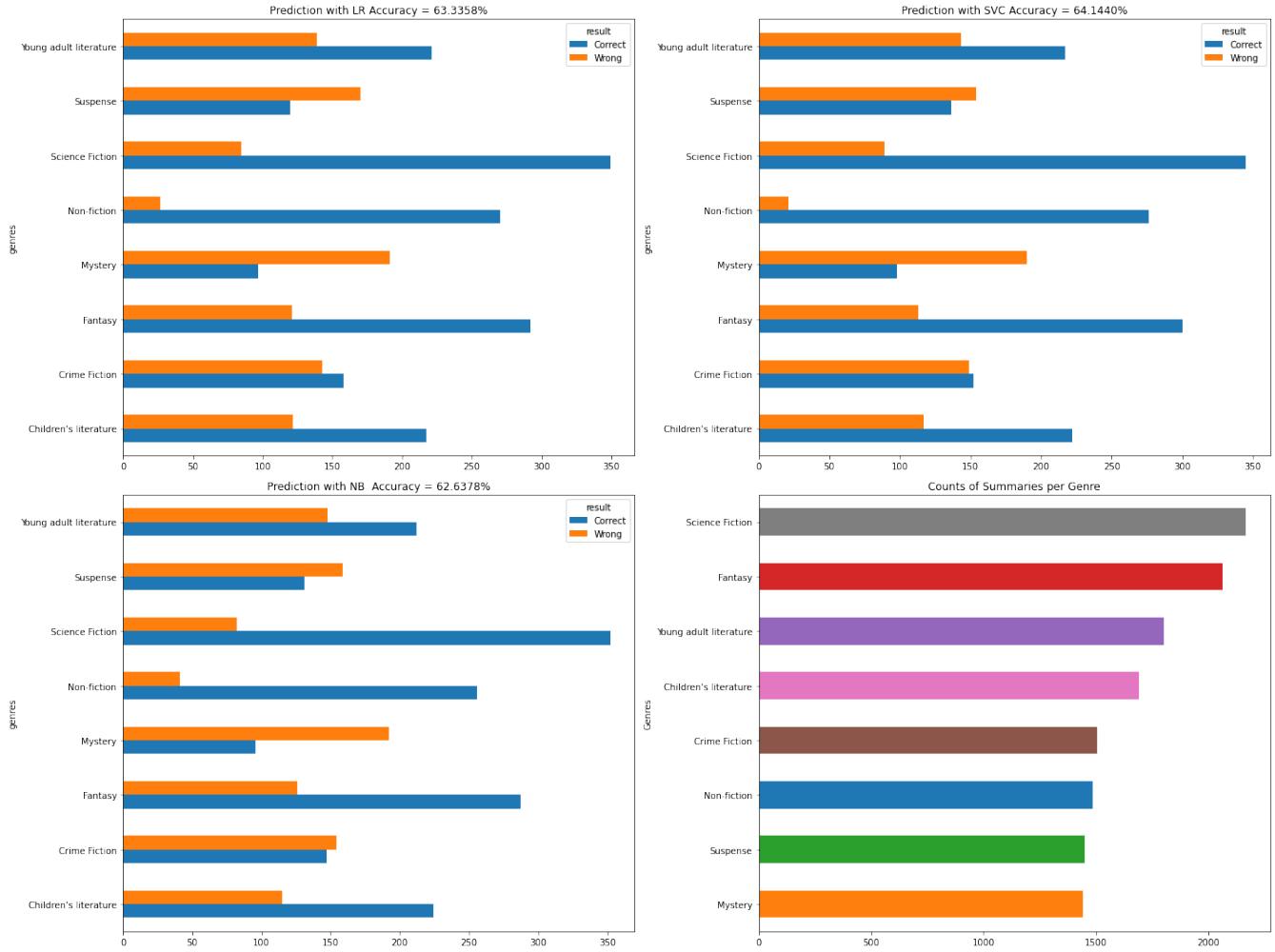


Figure 10: Correct and Wrong Predictions per Genre and model - Testing Dataset.

The confusion matrix for each model is quite interesting. As shown in figures 11 for the Naive Bayes model seems like the Crime Fiction, Children's Literature and Suspense has the least accuracy. They easily get confused with other genres. Crime Fiction is often confused with Children's literature and vice-versa. For the suspense genre, it is quite confused with Crime Fiction genre. This model is quite good at predicting Mystery genres.

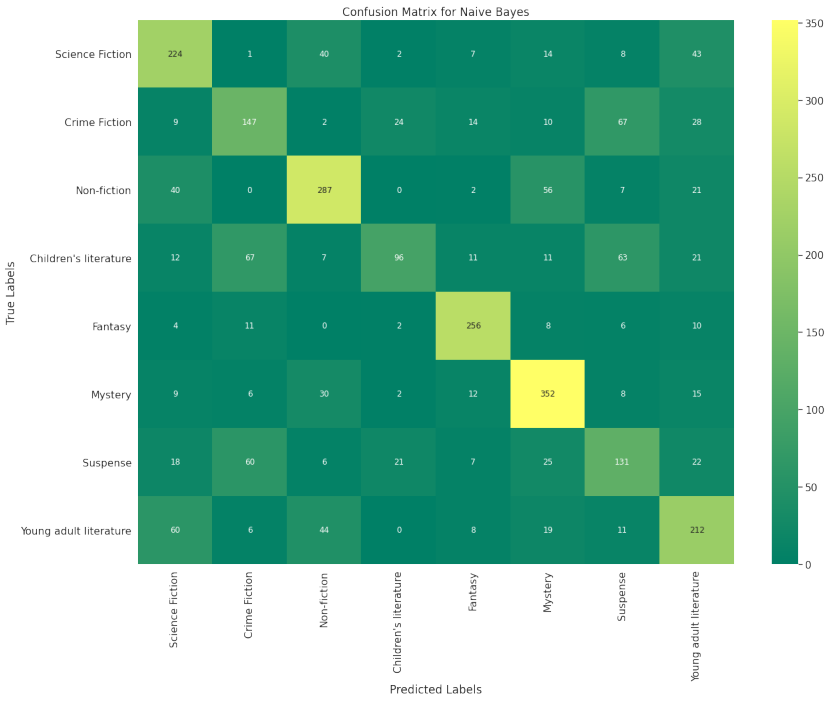


Figure 11: Confusion Matrix for Naive Bayes model - Testing Dataset.

As shown in figures 12 for the Support Vector Classification model seems like the Crime Fiction, Children's Literature and Suspense has the least accuracy. Same as before, the confusion is within the same genres as for the Naive Bayes.

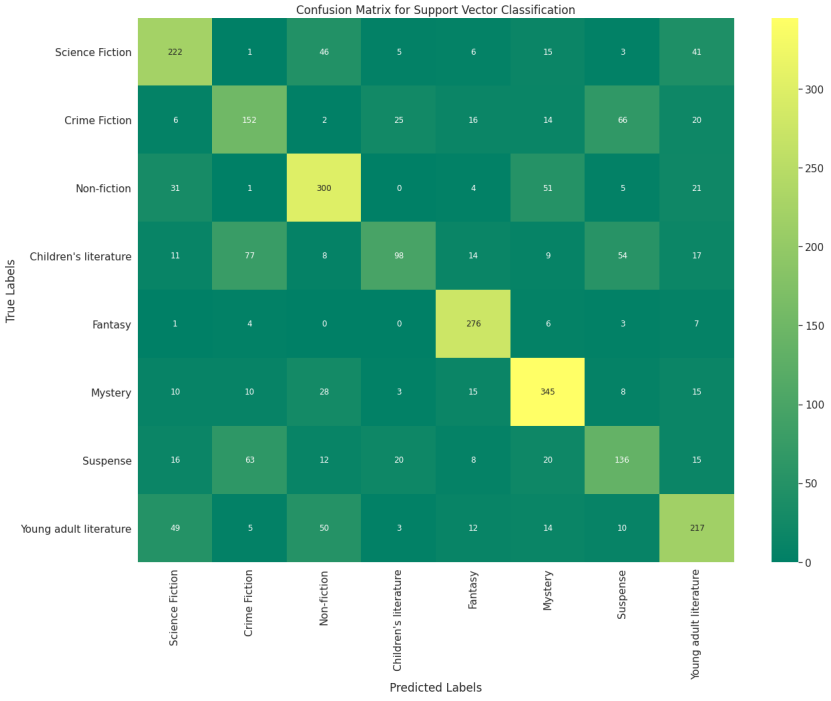


Figure 12: Confusion Matrix for SVC model - Testing Dataset.

As shown in figures 13 for the Logistic Regression model seems like the Crime Fiction, Children's Literature and Suspense has the least accuracy. Same as before, the confusion is within the same genres as for the Naive Bayes and SVC models.

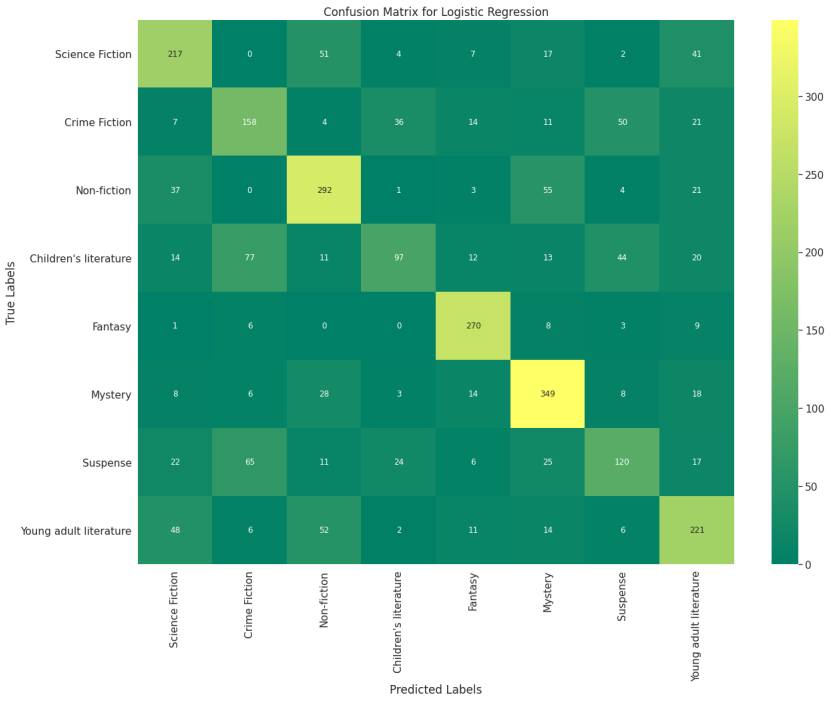


Figure 13: Confusion Matrix for Logistic Regression model - Testing Dataset.

6.1.2 Deep Learning Model

The results with each pre-trained word vectors were quite similar but the desired accuracy score of 64% was not obtained with the deep learning model. It is quite possible that the deep learning model needs more dataset to increase the accuracy.

For each model, it is considered to be the best model when the validation stops decreasing at a certain epoch. When the validation loss increases while the training loss keeps decreasing means that the model is overfitting and this has to be avoided.

As shown in table 3, for the LSTM Classifier, the best pre-trained vectors that works for it is the GloVe 6Billion 300 dimension vector with 56.77% of test accuracy. This model fits the best on epoch 4.

As shown in table 4, for the GRU Classifier, the best pre-trained vectors that works for it is the GloVe 42Billion 300 dimension vector with 59.03% of test accuracy. This model fits the best on epoch 2.

If both classifiers are compared, the fastest is the GRU Classifier to fit as it fits in less epoch than the LSTM Classifier. GRU Classifier's test accuracy is also better than the LSTM ones.

Pre-trained	Epoch	Training Loss	Validation Loss	Testing Accuracy(%)
glove.twitter.27B.100d	5	1.013966	1.258968	55.5556
glove.840B.300d	3	1.092867	1.208814	51.9097
glove.42B.300d	3	1.104383	1.232457	53.2986
fasttext.en.300d	4	0.898906	1.226587	53.1250
fasttext.simple.300d	2	1.394718	1.308129	45.6597
charngram.100d	4	1.090440	1.387638	45.8333
glove.6B.300d	4	0.916517	1.178452	56.7708
glove.6B.50d	6	1.118116	1.276597	50.6944

Table 3: Benchmarks for LSTM Classifier.

Pre-trained	Epoch	Training Loss	Validation Loss	Testing Accuracy(%)
glove.twitter.27B.100d	3	1.008377	1.148285	55.3819
glove.840B.300d	2	1.034216	1.061049	58.3333
glove.42B.300d	2	1.057161	1.080249	59.0278
fasttext.en.300d	2	1.058304	1.109307	58.6806
fasttext.simple.300d	2	1.094810	1.161578	53.9931
charngram.100d	2	1.328072	1.230578	49.6528
glove.6B.300d	2	1.082108	1.114147	54.5139
glove.6B.50d	4	0.971570	1.170154	55.0347

Table 4: Benchmarks for GRU Classifier.

The figure 14 shows the confusion matrix for the best LSTM Classifier with testing accuracy 56.77%. It best works with Mystery and Non-fiction. Science Fiction is easily confused with youn adult literature. That perhaps, is because within young adult literature genre there can be science fiction literature as well.

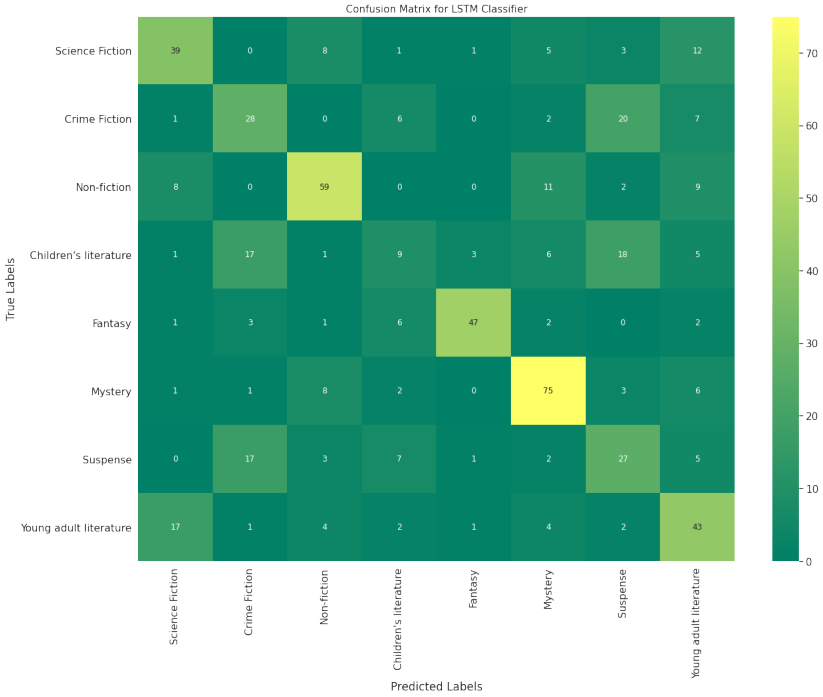


Figure 14: Confusion Matrix for the best LSTM Classifier - Testing Dataset.

The figure 15 shows the confusion matrix for the best GRU Classifier with testing accuracy 59.03%. It best works with Mystery, Fantasy and Non-Fiction. Where it gets more confused is with Crime Fiction, Children's literature and Suspense.

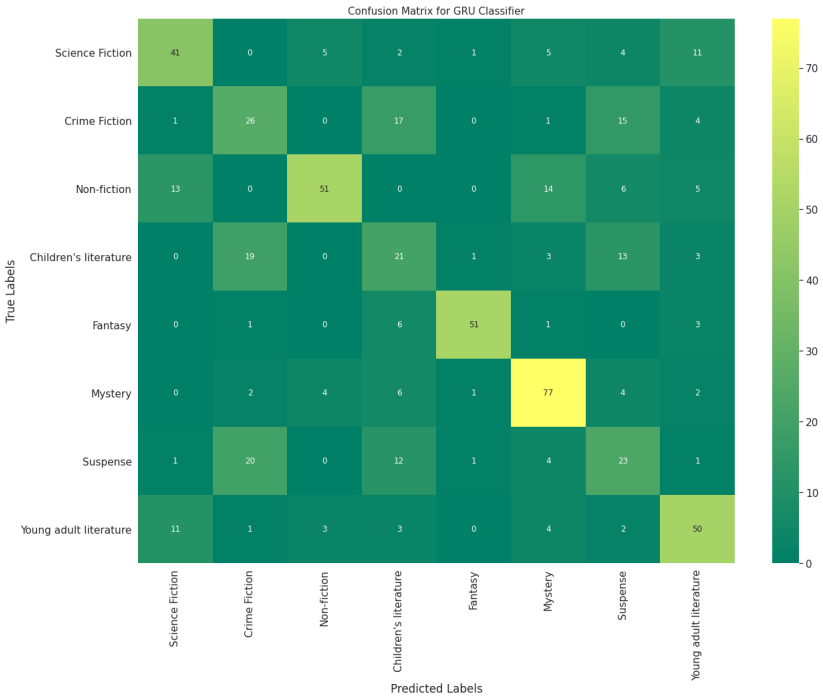


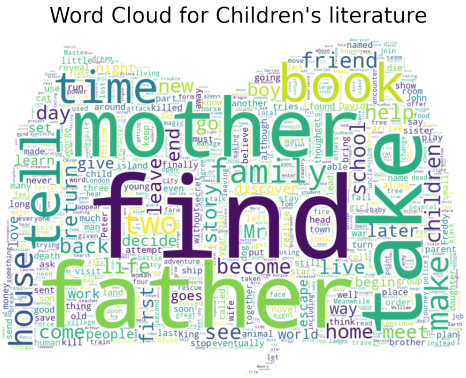
Figure 15: Confusion Matrix for the best GRU Classifier - Testing Dataset.

7 Conclusion

The conclusion after comparing all the models and the benchmarks is that the best model is the Support Vector Classification model along with the TF-iDF feature extraction method. It provides an accuracy of almost 64% which was the initial goal.

7.1 Free-Form Visualization

As a curiosity, word cloud was created for each type of genres and are shown in the below figures 16. For suspense, mystery and crime-fiction the key words seem to be find and murder which makes sense. For non-fiction words like book, life, world are the keywords. For young adult and children's literature keywords like father, mother and find are the ones more repeated.



(a) Word Cloud for Children's Literature - Total Dataset.

(b) Word Cloud for Crime Fiction - Total Dataset.

(c) Word Cloud for fantasy - Total Dataset.

(d) Word Cloud for Mystery - Total Dataset.

(e) Word Cloud for Non-Fiction - Total Dataset.

(f) Word Cloud for Science Fiction - Total Dataset.

(g) Word Cloud for Suspense - Total Dataset.

(h) Word Cloud for Young Adult Literature - Total Dataset.

Figure 16: Word Cloud per each genre.

As an extra, a webpage was created using the flask framework and was deployed in the pythonanywhere.

The webpage is called fakebooks. The user will just have to insert a book summary and the prediction is given back to the user. The model used in this webapp is the SVC model with accuracy of 64%.

8 Reflection

The entire process can be summarized as follows:

1. Usage of publicly available dataset.
2. Collection of additional dataset from goodreads to balance the data.
3. The dataset was analysed and preprocessed.
4. A benchmark was created for all the classifiers. The metrics to evaluate the classifiers were also defined.
5. All the classifiers were trained using the dataset. Various models were trained to get the optimized result.
6. All the models were analysed and concluded being the SVC tuned model as the best one.
7. Additionally as an extra a webapp was created so that it could be user-friendly.

The most difficult part was to adjust the classifiers because there are a lot of possibilities. The limited resources also didn't help because the usage of limited resources was not the appropiate method to do an extensive grid search for the machine learning models. Perhaps, even getting more data would have improved the model.

The most important part of the project perhaps, was testing with real random data and seeing how the model predicted the correct genre. It is also true that a book can have different genre, for example even if the model predicts a book summary as a suspense genre, the correct answer could also have been crime fiction or science fiction or even mystery. That could also be on the reader's decision.

References

[1] Elizabeth D. Liddy. *Natural Language Processing*. 2001. URL: https://surface.syr.edu/cgi/viewcontent.cgi?referer=https://scholar.google.es/&httpsredir=1&article=1019&context=cnlp.

[2] Gartner. *Natural Language Processing (NLP)*. URL: https://www.gartner.com/en/information-technology/glossary/natural-language-processing-nlp.

[3] A.M. Turing. *Computing Machinery and Intelligence*. 1950. URL: https://www.csee.umbc.edu/courses/471/papers/turing.pdf.

[4] Carnegie Mellon University. *CMU Book Summary Database*. 2013. URL: http://www.cs.cmu.edu/~dbamman/booksummaries.html.

[5] Thorsten Joachims. *TextCategorization with Support Vector Machines:Learning with Many Relevant Features*. 1998. URL: https://www.cs.cornell.edu/people/tj/publications/joachims_97b.pdf.

[6] John Wieting et al. "Charagram: Embedding Words and Sentences via Character n-grams". In: *CoRR* abs/1607.02789 (2016). arXiv: 1607.02789. URL: http://arxiv.org/abs/1607.02789.

[7] Tomas Mikolov et al. "Advances in Pre-Training Distributed Word Representations". In: *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*. 2018.

[8] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.