Book Genre Prediction and Sentiment Analysis

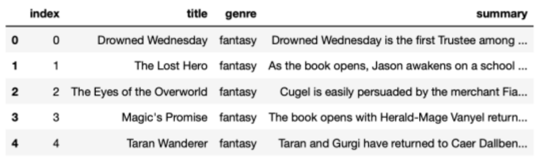
1. Introduction

Text mining and information retrieval are two essential aspects of natural language processing (NLP). Information Retrieval is the process of retrieving relevant information from a large collection of unstructured or semi-structured data. On the other hand, text mining includes the procedure of evaluating and extracting useful insights from textual data that is unstructured or semi-structured.
In this project, the use of information retrieval and text mining techniques has been employed to predict the genre of a book and perform sentiment analysis on it. The project is aimed at determining the genre of the book and identifying the sentiment of the book.

2. Dataset Preprocessing

2.1 Dataset Information

The "Book Genre Prediction Dataset" will be utilized in this project and is collected from Kaggle(Book-Genre-Prediction-Dataset). The dataset consists of a 4656 collection of books along with their respective titles, genres, and summary. The main objective of this proposal is to predict the genre of a book based on its summary retrieved from this dataset. Each book is labeled with one of ten different genres: Thriller, Fantasy, Science, Crime, Psychology, Romance, Sports, Travel, History, and Horror.



	index	title	genre	summary
0	0	Drowned Wednesday	fantasy	Drowned Wednesday is the first Trustee among ...
1	1	The Lost Hero	fantasy	As the book opens, Jason awakens on a school ...
2	2	The Eyes of the Overworld	fantasy	Cugel is easily persuaded by the merchant Fia...
3	3	Magic's Promise	fantasy	The book opens with Herald-Mage Vanyel return...
4	4	Taran Wanderer	fantasy	Taran and Gurgi have returned to Caer Dallben...

Fig 1. Dataset representation

2.2 Data Cleaning

Data cleaning is performed to ensure there are no inconsistencies and inaccuracies in the dataset. The cleaning process involved several steps, including removing unnecessary characters other than texts, converting all the summaries to lowercase, removing stopwords, performing lemmatization, and stemming. To achieve this, various Python libraries and functions were used. The re library was used to remove unnecessary characters, the nltk library was used to remove stopwords, and the WordNetLemmatizer function was used for lemmatization. Additionally, the PorterStemmer function from the nltk.stem library is used to perform stemming on the dataset. These cleaning techniques ensured that the dataset was suitable for further analysis, and that the results obtained would be reliable and accurate.

2.3 Data Analysis

Data analysis is performed to gain insights into the book genre dataset and understand patterns and trends. The analysis involved plotting various graphs to visualize the data and understand its distribution. For instance, bar plots were used to analyze the distribution of genres in the dataset. This also helped to identify the most common words in the dataset.

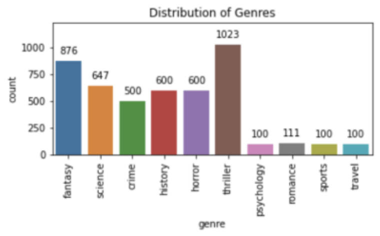


Fig 2. Distribution of Genre

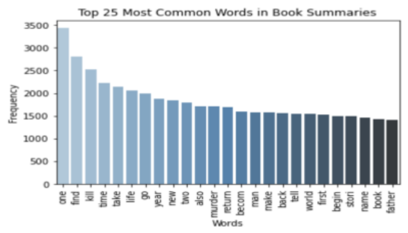


Fig 3. Top 25 words in the summary attribute

The PCA plot is used to visualize the high-dimensional data in a lower-dimensional space. This helps to explore the relationships between variables and identify patterns or clusters in the data. Also, the T-SNE produces high-quality visualizations of high-dimensional data that can reveal hidden patterns or relationships between variables.

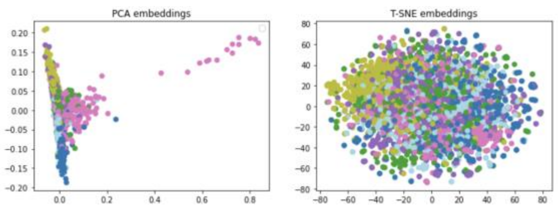
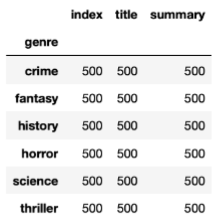


Fig 4. Visualization of PCA and T-SNE embeddings

3. Implementation

Class balance is important when it comes to applying classification models because imbalanced data can negatively affect the performance of the model. In an imbalanced dataset, the number of instances in one class which is the minority class is much smaller than the number of instances in the majority class. When a model is trained on an imbalanced dataset, it tends to have a bias towards the majority class, as it can achieve a higher accuracy by simply predicting the majority class for every instance. This is because the model is optimized to minimize the overall error, and it is penalized more for misclassifying instances from the majority class than from the minority class. Hence class balancing is performed to balance the number of classes.



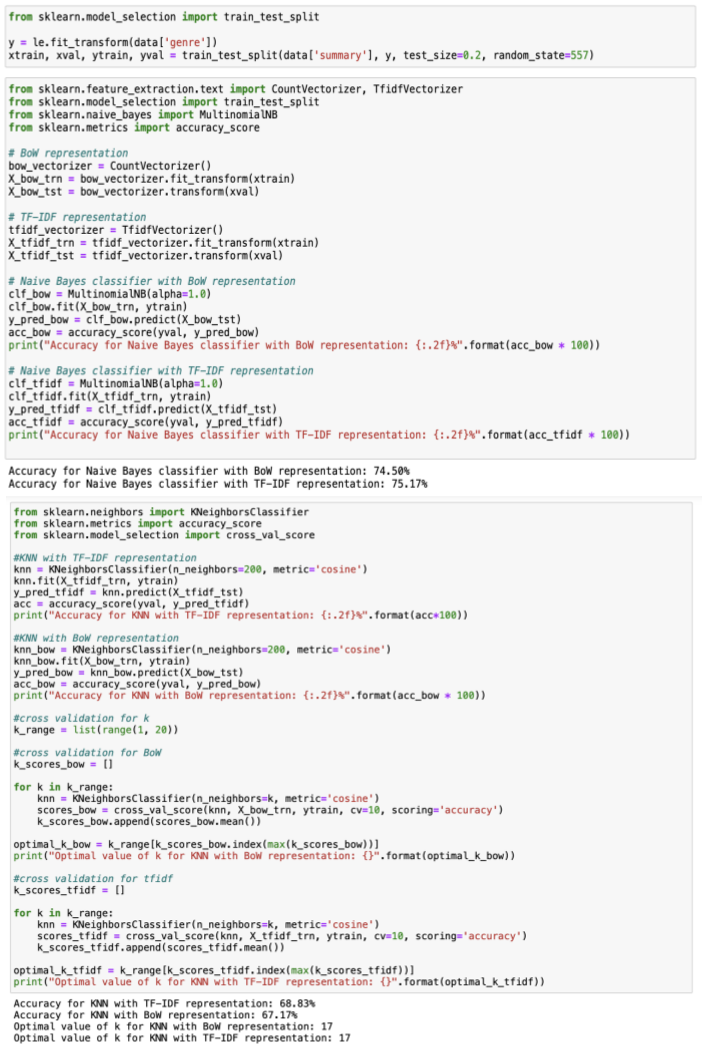
	index	title	summary
genre			
crime	500	500	500
fantasy	500	500	500
history	500	500	500
horror	500	500	500
science	500	500	500
thriller	500	500	500

Fig 5. Representation of balanced genre class

The scikit-learn library LabelEncoder class is used to encode the categorical values in the "genre" column of a Pandas DataFrame to numerical labels. The LabelEncoder assigns a unique integer label to each distinct genre in the "genre" column and replaces the original genre values in the DataFrame with their corresponding numerical labels. This is necessary because machine learning algorithms typically require numerical inputs. The train_test_split function from scikit-learn's model_selection module is used to randomly split the data into training and validation sets. The dataset is split in the ratio 80:20 where 80% data is for training purpose and rest 20% for testing purpose.

3.1 Classification

To implement the classification models, methods such as Term Frequency-Inverse Document Frequency (TF-IDF) and Bag-of-Words (BoW) have been utilized. TF-IDF is a technique used to determine the importance of a word in a document by calculating the frequency of the word in the document and comparing it to the frequency of the word in the entire corpus. BoW, on the other hand, involves the creation of a vocabulary of words present in the corpus, and each document is represented as a bag of words from the vocabulary. The CountVectorizer() and TfidfVectorizer() functions from scikit-learn are used to create the BoW and TF-IDF representations, respectively. Two classification models Naïve Bayes and KNN are implemented to calculate the accuracy using both the text representation techniques which are TF-IDF and BoW. The MultinomialNB() function is used to instantiate the Naïve Bayes model with an alpha value of 1.0, and two instances of the KneighborsClassifier() function are used with k=200 and the cosine similarity metric.



```python
from sklearn.model_selection import train_test_split

y = le.fit_transform(data['genre'])
xtrain, xval, ytrain, yval = train_test_split(data['summary'], y, test_size=0.2, random_state=557)
```

```python
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

# BoW representation
bow_vectorizer = CountVectorizer()
X_bow_trn = bow_vectorizer.fit_transform(xtrain)
X_bow_tst = bow_vectorizer.transform(xval)

# TF-IDF representation
tfidf_vectorizer = TfidfVectorizer()
X_tfidf_trn = tfidf_vectorizer.fit_transform(xtrain)
X_tfidf_tst = tfidf_vectorizer.transform(xval)

# Naive Bayes classifier with BoW representation
clf_bow = MultinomialNB(alpha=1.0)
clf_bow.fit(X_bow_trn, ytrain)
y_pred_bow = clf_bow.predict(X_bow_tst)
acc_bow = accuracy_score(yval, y_pred_bow)
print("Accuracy for Naive Bayes classifier with BoW representation: {:.2f}%".format(acc_bow * 100))

# Naive Bayes classifier with TF-IDF representation
clf_tfidf = MultinomialNB(alpha=1.0)
clf_tfidf.fit(X_tfidf_trn, ytrain)
y_pred_tfidf = clf_tfidf.predict(X_tfidf_tst)
acc_tfidf = accuracy_score(yval, y_pred_tfidf)
print("Accuracy for Naive Bayes classifier with TF-IDF representation: {:.2f}%".format(acc_tfidf * 100))
```

```
Accuracy for Naive Bayes classifier with BoW representation: 74.50%
Accuracy for Naive Bayes classifier with TF-IDF representation: 75.17%
```

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score

#KNN with TF-IDF representation
knn = KNeighborsClassifier(n_neighbors=200, metric='cosine')
knn.fit(X_tfidf_trn, ytrain)
y_pred_tfidf = knn.predict(X_tfidf_tst)
acc = accuracy_score(yval, y_pred_tfidf)
print("Accuracy for KNN with TF-IDF representation: {:.2f}%".format(acc*100))

#KNN with BoW representation
knn_bow = KNeighborsClassifier(n_neighbors=200, metric='cosine')
knn_bow.fit(X_bow_trn, ytrain)
y_pred_bow = knn_bow.predict(X_bow_tst)
acc_bow = accuracy_score(yval, y_pred_bow)
print("Accuracy for KNN with BoW representation: {:.2f}%".format(acc_bow * 100))

#cross validation for k
k_range = list(range(1, 20))

#cross validation for BoW
k_scores_bow = []

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k, metric='cosine')
    scores_bow = cross_val_score(knn, X_bow_trn, ytrain, cv=10, scoring='accuracy')
    k_scores_bow.append(scores_bow.mean())

optimal_k_bow = k_range[k_scores_bow.index(max(k_scores_bow))]
print("Optimal value of k for KNN with BoW representation: {}".format(optimal_k_bow))

#cross validation for tfidf
k_scores_tfidf = []

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k, metric='cosine')
    scores_tfidf = cross_val_score(knn, X_tfidf_trn, ytrain, cv=10, scoring='accuracy')
    k_scores_tfidf.append(scores_tfidf.mean())

optimal_k_tfidf = k_range[k_scores_tfidf.index(max(k_scores_tfidf))]
print("Optimal value of k for KNN with TF-IDF representation: {}".format(optimal_k_tfidf))
```

```
Accuracy for KNN with TF-IDF representation: 68.83%
Accuracy for KNN with BoW representation: 67.17%
Optimal value of k for KNN with BoW representation: 17
Optimal value of k for KNN with TF-IDF representation: 17
```

Fig 6. Implementation code for classification

Cross-validation is used to determine the optimal value of k for KNN. A list of k values ranging from 1 to 20 is created and for each value of k, 10-fold cross-validation is performed using the cross_val_score() function from scikit-learn. The mean of the accuracy scores for each fold is then calculated, and the k value with the highest mean accuracy is chosen as the optimal k. The optimal value obtained for k is 17.

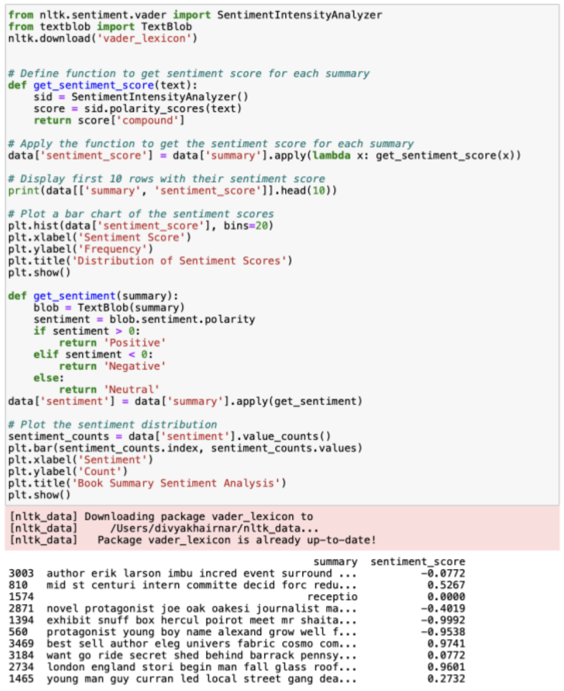
	TF-IDF	**BoW**
Naïve Bayes	75.17%	74.50%
KNN	68.83%	67.17%

Fig 7. Accuracy score comparison of Naïve Bayes and KNN for test data

3.2 Sentiment Analysis

The python libraries nltk.sentiment.vader for sentiment analysis and TextBlob for sentiment polarity calculation are used, the vader_lexicon data required for SentimentIntensityAnalyzer to work is downloaded. The get_sentiment_score function is defined which takes a text string as input and returns the sentiment score as a compound value between -1 and 1, with -1 being very negative and 1 being very positive. The function uses the SentimentIntensityAnalyzer object from NLTK to calculate the sentiment score for each summary. Then, the function is applied to the summary column of the data DataFrame using the apply method and the resulting scores are stored in a new column called sentiment_score. The first 10 rows of summary and sentiment_score are displayed using the head method. The sentiment scores are then visualized using a histogram generated with plt.hist from matplotlib.

Finally, the get_sentiment function is defined which takes a summary as input and returns a string indicating the sentiment, either 'Positive', 'Negative', or 'Neutral'. The function uses TextBlob to calculate the sentiment polarity of each summary. The function is applied to the summary column of the data DataFrame using the apply method and the resulting sentiment is stored in a new column called sentiment. The sentiment distribution is then visualized using a bar chart generated with plt.bar from matplotlib.



```python
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from textblob import TextBlob
nltk.download('vader_lexicon')


# Define function to get sentiment score for each summary
def get_sentiment_score(text):
    sid = SentimentIntensityAnalyzer()
    score = sid.polarity_scores(text)
    return score['compound']

# Apply the function to get the sentiment score for each summary
data['sentiment_score'] = data['summary'].apply(lambda x: get_sentiment_score(x))

# Display first 10 rows with their sentiment score
print(data[['summary', 'sentiment_score']].head(10))

# Plot a bar chart of the sentiment scores
plt.hist(data['sentiment_score'], bins=20)
plt.xlabel('Sentiment Score')
plt.ylabel('Frequency')
plt.title('Distribution of Sentiment Scores')
plt.show()

def get_sentiment(summary):
    blob = TextBlob(summary)
    sentiment = blob.sentiment.polarity
    if sentiment > 0:
        return 'Positive'
    elif sentiment < 0:
        return 'Negative'
    else:
        return 'Neutral'
data['sentiment'] = data['summary'].apply(get_sentiment)

# Plot the sentiment distribution
sentiment_counts = data['sentiment'].value_counts()
plt.bar(sentiment_counts.index, sentiment_counts.values)
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.title('Book Summary Sentiment Analysis')
plt.show()
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /Users/divyakhairnar/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
                                                 summary  sentiment_score
3003  author erik larson imbu incred event surround ...          -0.0772
810   mid st centuri intern committe decid forc redu...           0.5267
1574                                            receptio           0.0000
2871  novel protagonist joe oak oakesi journalist ma...          -0.4019
1394  exhibit snuff box hercul poirot meet mr shaita...          -0.9992
560   protagonist young boy name alexand grow well f...          -0.9538
3469  best sell author eleg univers fabric cosmo com...           0.9741
3184  want go ride secret shed behind barrack pennsy...           0.0772
2734  london england stori begin man fall glass roof...           0.9601
1465  young man guy curran led local street gang dea...           0.2732
```

Fig 8. Implementation code for sentiment analysis

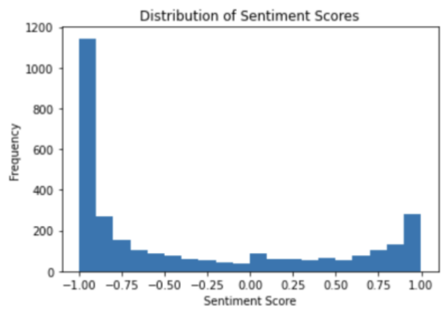


Fig 9. Distribution of sentiment score

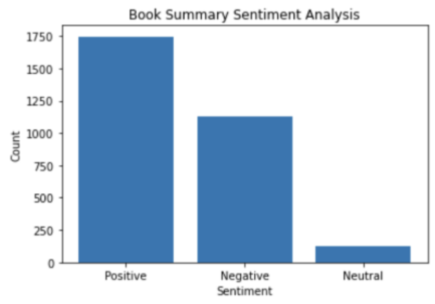


Fig 10. Bar plot for summary sentiment analysis