1 Introduction

In this report, we describe an approach for predicting movie genres, based on descriptive and categorizing information, using natural language techniques. First, we discuss several experiments that provided valuable insights during the development of the final model. Afterward, we discuss the final model and the results based on our own, as well as the provided test set.

This report is outlined as follows:

- We provide a summary of the different models and preprocessing techniques we experimented with and how they improved the results.
- We provide a detailed description of our final model and summarize the relevant metrics in a table.

2 Data

For our analysis, we used a data set based on the Wikipedia Movie Plots from Kaggle. The dataset contains approximately 8041 observations. Each entry represents a movie and includes several key features. The primary variable of interest is the movie's genre, which will serve as the target variable for the classification task. In total, there are nine distinct genres in the dataset, though their distribution is imbalanced (see Figure 1). In addition, the dataset provides additional features, such as the movie title, country of origin, director, and a brief plot description, which will be used as explanatory variables to model and predict the genre. In several instances, we identified the director to be labeled as "Unknown", which introduces some inconsistencies in the data. The plots are written in English, with an average length of approximately 370 words. Many of these descriptions contain special characters, such as hyphens and periods (e.g., Ph.D., back-story), which may pose challenges during preprocessing, particularly during tokenization.

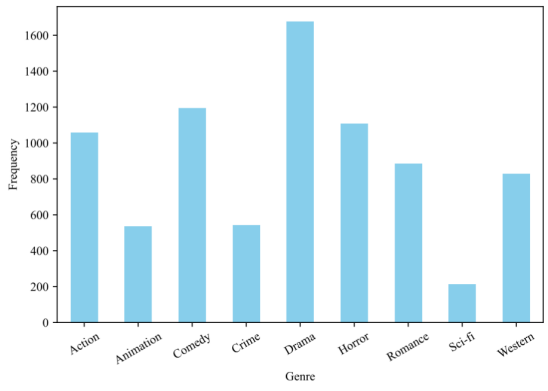

Figure 1: Frequency of Genres

3 Models

The goal of this project was to predict the genre of movies. For this reason, we applied several natural language techniques to classify the movie into a set of genres and compared the results of our classification approaches with the baselines given in the project description. In the following, we discuss our approach to achieving this goal. Our approach can be split into two categories: *Word vector embedding* and *Sentence vector embedding*.

3.1 Word Vector Embedding

The word vector embedding approach focuses on the processing of individual words or tokens in an input text and prediction/classification based on a numerical representation of an input text that considers the (weighted) word frequency, but does not account for context.

Preprocessing We implemented separate functions for tokenizing, lowercasing, punctuation removal, and stopword removal, to test different combinations of these preprocessing techniques to evaluate how they impact the results of the classification task at hand. We used **RegexpTokenizer** from the **nltk** package, which allowed us to define our custom separators for tokenization. After thoroughly analyzing the data, we realized that certain words like *back-story*, should not be split into separate tokens. Additionally, we removed periods within words to avoid tokenizing words like Ph.D. or S.H.I.E.L.D. We used the **TfidfVectorizer** from the **scikit-learn** Python package to create the TF-IDF vectors of the preprocessed data.

For some models, we selected a subset of the resulting features - specifically, the TF-IDF vocabulary - that either improved or didn't change the results of the model but reduced the computational complexity.

Models We used `Naive Bayes`, `SVM`, `RandomForrest`, and a basic `Information Retrieval` using cosine similarity. It should however be noted that information retrieval is primarily used for recommender systems and in web-browser to retrieve to top k results and not for classification by itself.

3.2 Sentence Vector Embedding

The sentence vector embedding approach focuses on creating a numerical representation of an input text, where each word is processed in relation to other words in the text. These numerical representations are used to perform predictions/classification with models that are not only trained on individual word information but also the context in which they appear.

Preprocessing We used the contextual pre-trained model `BERT`, to create a vector representation of each plot in the dataset. We didn't perform any preprocessing on the plots, as pre-trained contextual models make use of all information in a sentence/text, including the information given by the morphemes of a word, stopwords, and punctuation.

Models We trained `SVM` and `Random Forest` models to perform the classification task, based on the sentence embeddings.

4 Experimental Setup and Results

The data set was randomly divided into a training and a test set. The training set consists of 80% of the observations (6432 entries), and the testing set consists of 20% of the observations (1609 entries). To ensure the effectiveness of the evaluation of the performance of the different machine learning approaches, we applied 5-fold cross-validation. To identify the optimal combination of hyper-parameters, we used a grid search for each model. We measured the performance of the models in terms of accuracy. In the following, for each approach presented in chapter 3, we will present the results of the model with the best performance. For these models, we will also provide the hyperparameters.

4.1 Word Vector Embedding

After preprocessing the plots and calculating the TF-IDF scores, we employed cross-validation combined with grid search to identify the optimal model. Our analysis indicated that the Support Vector Classifier (SVC), with the hyperparameter set summarized in Table 1, achieved a test accuracy of 69.0%.

h.p.	C	class weight	gamma	kernel
value	1	*balanced*	*scale*	*linear*

Table 1: Word embedding parameter set

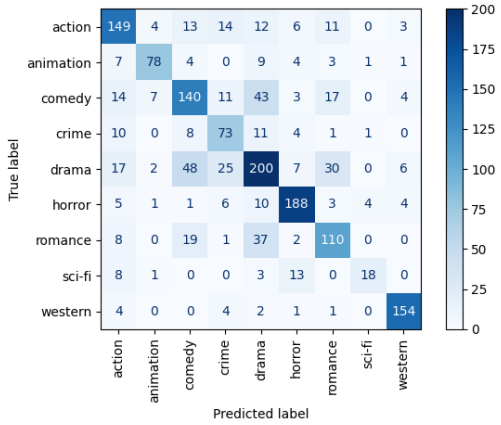

Figure 2: Confusion Matrix

4.2 Sentence Embedding

The cross-validation, combined with the grid search, identified the Support Vector Classifier (SVC) as the best-performing model when using BERT embeddings for the plot data. The optimal hyperparameter set is summarized in Table 2. This configuration resulted in an accuracy of 62.2%.

h.p.	C	class weight	gamma	kernel
value	10	*balanced*	*scale*	*rbf*

Table 2: Sentence embedding parameter set

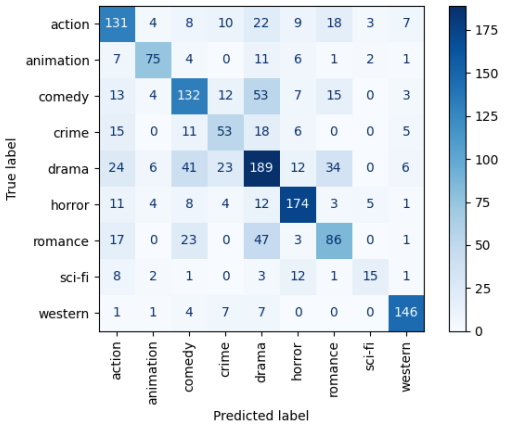

Figure 3: Confusion Matrix

5 Discussion

The presented models did not achieve satisfactory performance. As shown in the confusion matrices (see Figure 2 and Figure 3), the model demonstrated a tendency to over-classify movies as *drama*, indicating an imbalance in its classification performance across genres. We believe that this is partly because some of the misclassified movies could reasonably belong to more than one genre. In addition, we found that in some cases the model predictions were more appropriate than the actual labels. Examples of these cases can be found in Table 3 Therefore, we conclude that if plots were manually labeled, there would likely be significant disagreement between annotators.

Movie	Actual	Pred.	Pos. reason
Hollow Point	Action	Crime	"*FBI agent..*"
Lost Angels	Drama	Crime	"*..police..*"
The Hypnotic Eye	Sci-fi	Horror	"*..gruesomely disfigure..*"

Table 3: Example mispredictions

6 Future Work

The *train.txt* and *test_no_labels.txt* files contain missing values (e.g. 'Unknown' in the director column). To maintain consistency for automatic evaluation we chose not to handle these. Further analysis is required to determine if removal or KNN imputation would yield better results. However, the main gap lies in the lack of useful data. An alternative approach could involve reducing the plots to named entities—such as characters, actors, and places—which may help indicate the genre (e.g., recurring characters or actors in sequels, locations like restaurants, or superhero bases) and include additional features like movie reviews.