A.a 2023-2024

TEXT MINING PROJECT REPORT

Eleonora Brambatti (858098), Camilla Fracchia (898235), Marta Privitera (898017)

14^{th} February 2024

Contents

Abstract

The aim of this project is to make an analysis on books reviews taken from *Goodreads* website. In particular, we focused on the text classification and topic modeling tasks.
Starting from the text classification part, we used different type of text representation (Tf-idf, Word2Vec, GloVe) and then we tested different types of classifier (Logistic Regression, Decision Tree, Random Forest, XGBoost and KNN) and different types of neural networks (LSTM, CNN and Hybrid architectures of the two) in order to find the best solution (in efficiency and effectiveness terms) for classifying the reviews as negative or positive.
Then, for the topic modeling part we tried to use different methods as LSA, LDA, Bert, Top2Vec and we evaluated the goodness of the models by using the coherence values.
We obtained good results for the classification with both the representation, in particular the highest values of accuracy and F1 score reached were the one associated to the Logistic regression with TF-IDF representation and to first Baseline Deep Learning model with GloVe Word Embedding, while (unfortunately), for the topic modeling task we obtain quite bad coherence scores for each model, in fact, we obtained the best result with LSA model but with only 0.54 of coherence.

1 INTRODUCTION

The dataset used in this analysis is the Goodreads reviews 'mystery', 'crime', 'thriller' dataset[1]. It comprises reviews of books falling under the three genres mentioned above containing information such as user_id, book_id, review_id, rating, review_text, date_added, date_updated, read_at, started_at, n_votes, and n_comments.
For analytical purposes, only the top 5 columns were considered relevant: timestamp information and interactions on reviews were deemed non-essential for analysis.

The 'rating' column, providing information on the assigned review score, facilitated the dataset's use in a text classification task by determinating the predefined classes. Simultaneously, the different genre reviews enabled the execution of the topic modeling task which will be described in the following paragraphs.

2 DATA PREPARATION

The initial json-formatted file underwent conversion into a dataframe making it simpler the analysis using the Pandas library.
After removing missing data, reviews lacking content were excluded as they held no analytical significance. In fact, the subjective nature of reviews posed challenges in predicting their content, leading to the removal of empty rows.

Considering the sizable corpus (around 2 million records), we conducted stratified sampling from each rating class, reducing it by randomly selecting 20% from each class.
This process resulted in a homogeneous dataset that maintained the original class weights.

2.1 Hadling default class

The '0' rating class was excluded during the exploratory phase due to inconsistencies where some reviews with a '0' rating conveyed positive sentiments. We inferred that '0' represented an automatic

[1]The dataset is available at the following link: https://mengtingwan.github.io/data/goodreads#datasets

evaluation when a user forgot to specify a grade, making it unsuitable for analytical purposes.

```
Just amazing.

I just want to be stephanie plum.
```

Figure 1: Examples of review texts associated with class 0. The positive tinge of the reviews is appreciable, confirming that class 0 is the default class.

To streamline algorithmic processing, the 'Langid' library identified review languages, retaining only those in English.
Approximately 170,000 reviews were then categorized into 'positive' (rating 4-5) and 'negative' (rating 1-3) for binary classification.

3 PRE PROCESSING

A general text pre-processing phase was conducted on the entire dataset to expedite algorithm application. Using regular expressions and specialized textual analysis libraries (such as 'Nltk' and 'Strings') , numbers and reviews resulting in empty strings were removed. Subsequently, text normalization, which has the goal of making text uniform and easier to process, involved converting all text to lowercase and removing links pointing to external sites.

Stop words were removed carefully to preserve semantic integrity, retaining 'no' and 'not' from the English stop words list. This step was crucial because a straightforward negative sentence like 'I did not like this book', with the removal of the stop word 'not', would alter the semantic meaning of the sentence. Even though it was expressing a negative sentiment, omitting this stop word could lead to complications and potential distortions in the algorithm's understanding. For this reason, the simple terms 'no' and 'not' were retained and not removed.

After these steps, punctuation and white spaces were also removed, again with the aim of having the corpus in the most optimal format for translating terms into vectors. Lemmatisation was omitted from the general dataset, as raw data was necessary for embedding techniques like GloVe.
Lemmatisation would be applied only when essential for analysis. Then the 'cleaned' dataset was implied in the following described tasks.

4 TEXT CLASSIFICATION

4.1 Text Classification Dataset Preparation

To prepare the dataset for the text classification task, an additional preprocessing phase has been carried out. Let's walk through the steps taken to set up the dataset for Text Classification:

1. **Label Encoding:** The sentiment labels in the 'label' column of the dataset were transformed into numerical values. 'Positive' was encoded as 1, and 'Negative' as 0. This step was essential for machine learning models to interpret and learn from the labels.

2. **Shuffling Dataset:** To prevent biases in the training and test sets, the dataset was shuffled. Without shuffling, there is a risk of having only one type of label in either the training or test set so shuffling ensured a balanced distribution of positive and negative labels in both sets.

3. **Checking Dataset Balance:** We noticed there was a class imbalance (see fig.2). To further address this problem we decided to undersample the dataset. Undersampling was performed on the positive class in order to prevent the model from being biased towards the majority class. This procedure involved randomly removing instances from the majority class (positive) to match the number of instances in the minority class (negative).

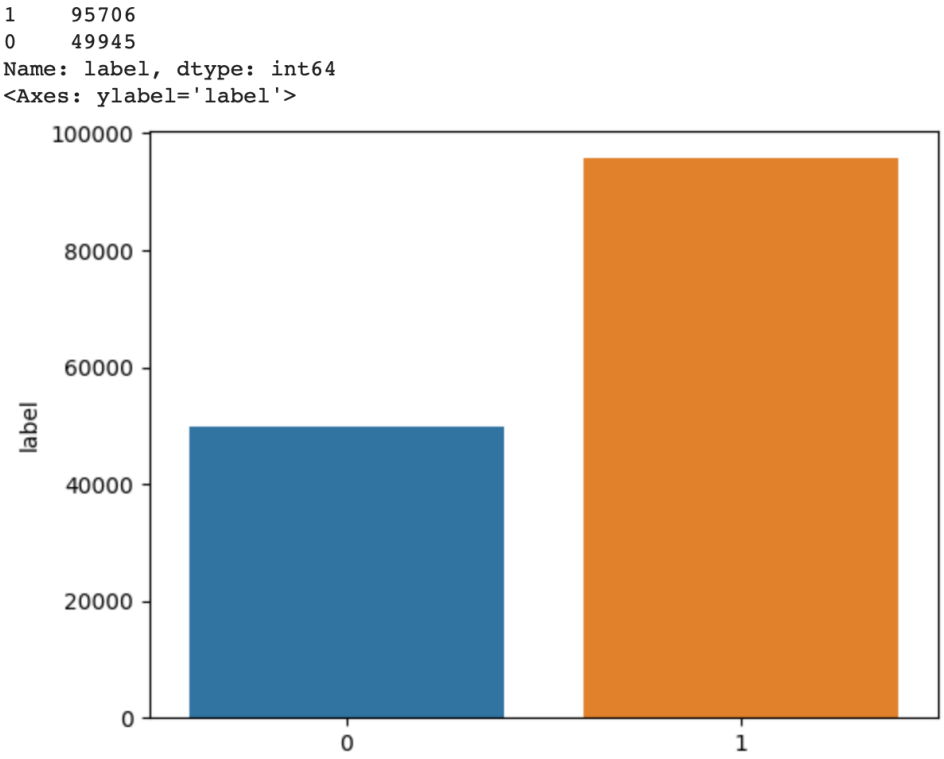


Figure 2: A bar plot was generated to visualize the distribution of positive and negative labels. This step helps in understanding the balance between the two classes: 95706 reviews belong to positive class, 49945 belong to negative class.

4. **Word Count Analysis:** We took a two-step approach to refine the dataset. First, we figured out how many words, on average, were in each review (more or less 57 for positive class and 51 for negative class). Then, we decided to keep only those reviews that have a maximum of 250

words (this limit set based on the fact that most reviews are short, as we can see from fig.3).

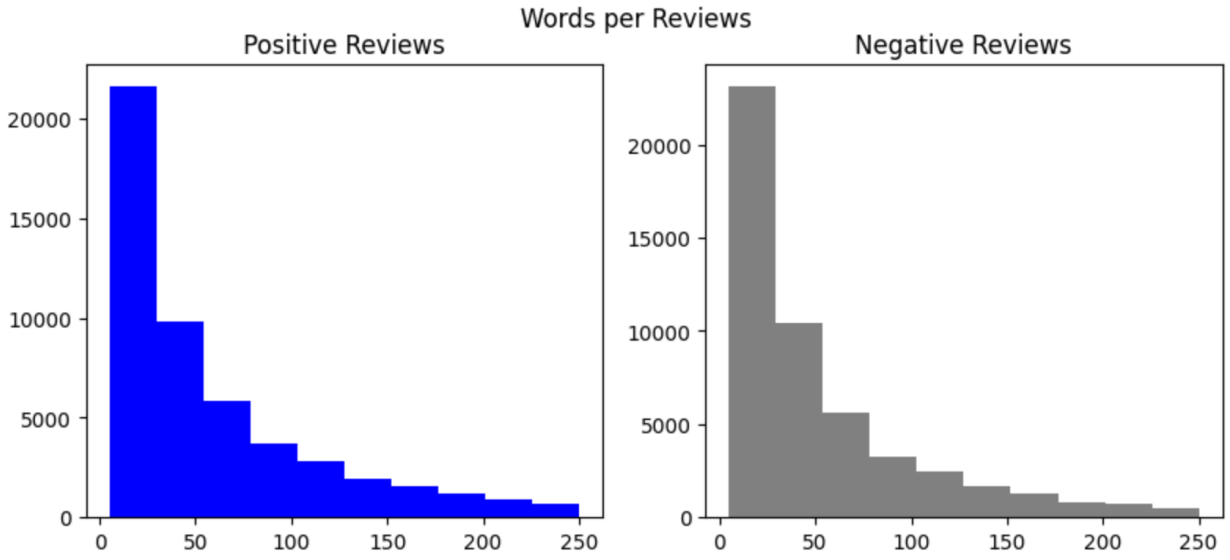


Figure 3: Plotting the number of reviews against the word count per review: the majority of reviews are short in both the positive and negative classes.

This decision is grounded in practical reasons. Longer reviews can cause problems when we turn the text into numbers (vectorization). Some algorithms struggle with handling very long pieces of text efficiently and cannot ensure comparability with shorter texts. Also, models like Word2Vec or GloVe might find it hard to grasp meaningful connections in really long texts. So, setting the threshold at 250 words was a practical move. It helped us balance the computational load and how well the model works. After preprocessing the dataset, we used different types of textual representations for text classification.

4.2 TF-IDF Text Representation

The first form of text representation that we employed was TF-IDF. This representation takes into account both the frequency of a term in a document (Term Frequency, TF) and its rarity across the entire corpus (Inverse Document Frequency, IDF). The resulting TF-IDF score reflects the importance of a word in a document relative to a collection of documents (corpus).
In particular we used the $TfidfVectorizer$ to transform the text data into numerical features and perform TF-IDF representation. The vectorizer was fit on the training set (X_train) and then applied to both the training and test sets, resulting in X_train_tfidf and X_test_tfidf.

Model Training and Evaluation:

Four classification models were trained and evaluated using the TF-IDF transformed features. For each model (Logistic Regression, Decision Tree, Random Forest, and XGBoost), the accuracy and a detailed classification report were printed (see fig. 4 and 5).
Logistic Regression outperformed the Decision Tree model and it slightly surpassed both Random Forest and XGBoost performances. Random Forest and XGBoost demonstrated comparable performance, achieving accuracies around 73%. Importantly, the models exhibited reasonably balanced precision and recall, indicating their effectiveness in classifying both positive and negative reviews.

Random Forest Accuracy: 0.732
Random Forest Classification Report:

	precision	recall	f1-score	support
0	0.73	0.75	0.74	10035
1	0.74	0.72	0.73	9943
accuracy			0.73	19978
macro avg	0.73	0.73	0.73	19978
weighted avg	0.73	0.73	0.73	19978

XGBoost Accuracy: 0.733
XGBoost Classification Report:

	precision	recall	f1-score	support
0	0.72	0.76	0.74	10035
1	0.75	0.70	0.72	9943
accuracy			0.73	19978
macro avg	0.73	0.73	0.73	19978
weighted avg	0.73	0.73	0.73	19978

Figure 4

Logistic Regression Accuracy: 0.755
Logistic Regression Classification Report:

	precision	recall	f1-score	support
0	0.76	0.75	0.75	10035
1	0.75	0.76	0.76	9943
accuracy			0.76	19978
macro avg	0.76	0.76	0.76	19978
weighted avg	0.76	0.76	0.76	19978

Decision Tree Accuracy: 0.628
Decision Tree Classification Report:

	precision	recall	f1-score	support
0	0.63	0.63	0.63	10035
1	0.63	0.62	0.63	9943
accuracy			0.63	19978
macro avg	0.63	0.63	0.63	19978
weighted avg	0.63	0.63	0.63	19978

Figure 5

Overall, TF-IDF proved to be a valuable tool in text representation, contributing to the success of machine learning models in the classification of reviews. Moreover, the limitation of features imposed by TF-IDF assisted in managing data dimensionality, mitigating the risk of overfitting.

4.3 Word Embedding Text Representation

The second type of text representation we employed is 'Word Embedding'. This is a technique that captures the semantic meaning of words by converting them into dense vectors of real numbers. In essence, each word is represented as a point in a multidimensional space. The distance and direction between points reflect the semantic relationships among the words. Using word embedding for text representation proves valuable in text classification as it enables the capturing of intricate semantic relationships between words.

4.3.1 Word2Vec

The first word embedding model employed is 'Word2Vec'. For its application, we followed the subsequent steps:

1. **Tokenization:** To apply Word2Vec function to our data we firstly tokenized documents. This step breaks down each sentence into individual words, preparing the data for model training.

2. **Training Word2Vec:** We trained the Word2Vec model using the tokenized sentences by setting parameters such as the window size (5), minimum word count (5), and the number of workers (4).

3. **Sentences transformation into vectors:** We defined and then applied to both train and test sets a vectorization function, 'vectorize' which converts sentences into vectors. This function retrieves Word2Vec vectors for words present in a sentence and calculates their mean vector. If a word was absent in the Word2Vec model, a zero vector was assigned. This function was crucial for transforming sentences into numerical representations.

Model Training and Evaluation:
A Logistic Regression model, a K-Nearest Neighbors (KNN) classifier, a Decision Tree classifier, and a Random Forest classifier were trained using the vectorized training data. For each model, the

performance was visualized with a ROC curve (in the code provided), and evaluation metrics were computed to assess accuracy, precision, recall, and F1 score.

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.744	0.743	0.743	0.743
Random Forest Classifier	0.726	0.730	0.712	0.721
KNN Classifier	0.665	0.655	0.689	0.672
Decision Tree Classifier	0.613	0.611	0.613	0.612

Table 1: Performance measures of different models

The Logistic Regression model achieved an accuracy, precision, recall, and F1 score of around 74%, showcasing a balanced performance in classifying sentiments. The KNN classifier instead, with an accuracy of 66.5%, demonstrated slightly lower performance compared to Logistic Regression. While precision and recall were reasonable, there was room for improvement, possibly through hyperparameter tuning or considering different algorithms. The Decision Tree Classifier performed well, achieving an accuracy of 61.3%. However, it exhibited slightly lower precision, recall, and F1 score compared to the Logistic Regression and Random Forest models.
The Random Forest Classifier outperformed Decision Tree and KNN classifiers with an accuracy of 72.6%. It showed a balanced precision and recall, indicating effectiveness in handling both positive and negative reviews.
In conclusion, Logistic Regression and Random Forest showed similar results between TF-IDF and Word2Vec. Decision Tree classifier exhibited slightly lower accuracy with Word2Vec embeddings compared to TF-IDF. This could imply that the geometric relationships learned by Word2Vec might not be as suitable for these models as the feature-based approach of TF-IDF. The performance might be influenced by the fact that we did not remove the stop words "not" and "no". Probably, this choice introduced ambiguity in classifying reviews as positive or negative by our classifier. The presence of such words could alter the overall meaning of a sentence, leading to a misinterpretation by the algorithm.

4.3.2 GloVe

GloVe (Global Vectors for Word Representation) is another widely used word embedding model that represents words in a vector space. Unlike Word2Vec, GloVe considers not only the similarity of words in similar contexts but also incorporates word co-occurrence frequencies in the corpus. We aimed to exploit these GloVe features for our classification problem, following these steps:

- After generating a list of tokenized reviews, the data was split into training and testing sets (80% training, 20% testing).
- The Keras Tokenizer was used to tokenize the training set. Sequences of integers were created, and padding was applied to training and testing sets to ensure consistent length (max_len).
- The word index and the number of unique words were obtained from Keras Tokenizer.
- GloVe word vectors were downloaded and stored in the 'embedding_dict' dictionary. In our case each word was mapped to a specific vector of 100 dimensions. An embedding matrix was created to represent words in the corpus using GloVe vectors.

We used the embedding matrix to train some machine learning models.

Baseline Deep Learning Model (1):

1. A sequential model was created using Keras.
2. An Embedding layer was added to the model, initialized with the pre-trained embedding matrix (GloVe vectors) and set as non-trainable.
3. An LSTM layer with 100 units and 10% dropout was added to prevent overfitting.
4. Then, we added a dense layer as last layer with *sigmoid* as the activation function.
5. The model was compiled using the Adam optimizer with a learning rate of 3e-4, binary crossentropy as loss function, and accuracy as the evaluation metric.

Model Training:

The model was trained for 25 epochs using the training data (*train_padded* previously created) and validation data was used to monitor performance.

Model Evaluation:

The model was used to predict sentiments on the test set (*X_test*). Predictions were rounded to integers for binary classification.

Results:

The model made approximately 15413 correct predictions and 4565 incorrect predictions. The confusion matrix in figure 6 shows the goodness of the model. From the confusion matrix and the associated metrics, we can infer that the model performed reasonably well, with values of metrics hovering around 77% (as we can see by the report provided in the code).

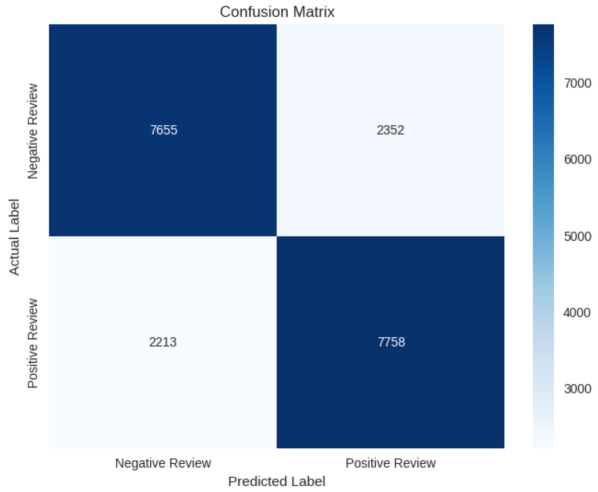


Figure 6: The diagonal elements (top-left to bottom-right) represent correct predictions (true positives and true negatives). Off-diagonal elements represent incorrect predictions (false positives and false negatives).

The goodness of the model is also evident from the accuracy and loss plots in figure 7. Despite a slight overfitting trend after the sixteenth epoch, the gap between training and validation accuracy, as well as between training and validation loss, remained low (on the order of 1e-2 in both cases) at least until the 25th epoch, where we halted the training.

Figure 7: Training and Validation Loss and Accuracy for the first Baseline Machine Learning model.

Baseline Deep Learning Model (2):

The second model, similar to the previous one, has a single modification: the number of nodes in the first layer was increased from 100 to 128 to augment complexity. We initially set the number of epochs to 50 but then we decided to halt the training at the 25th epoch as we indeed observed an instance of overfitting. This decision was made after analyzing the results shown in figure 8.

Figure 8: This plot illustrates the training and validation loss. The overfitting phenomenon appears from the 25th epoch onward.

The confusion matrix in figure 9, indicates a slightly better performance in predicting positive reviews. In fact, from the point of view of recall it is higher (80%) compared to negative reviews (75%). In contrast, the first model (Baseline Machine Learning Model (1)) exhibited fair performance on both classes. Overall, the accuracy for both models is 77%, indicating that this kind of architecture performs quite well in classification task. These results are all displayed in the reporting tables within the provided code.

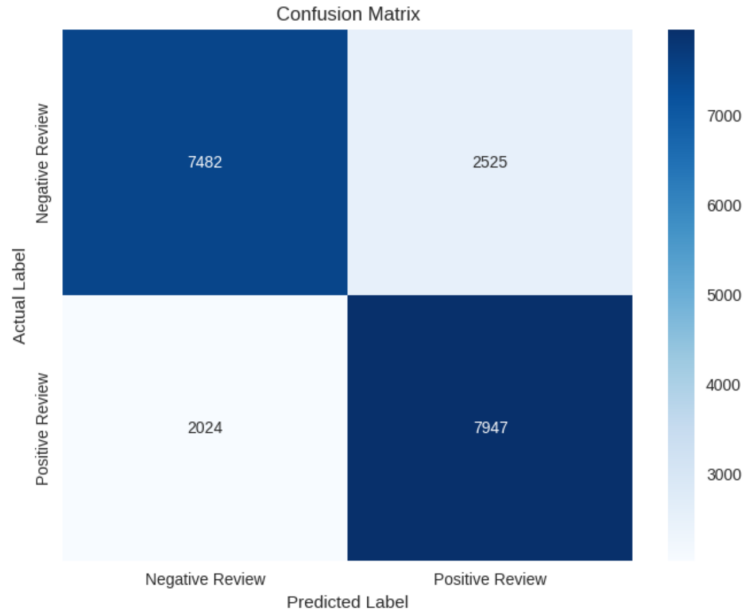


Figure 9: Confusion matrix for the second Baseline Machine Learning model.

LSTM Neural Networks

Exploiting the word embedding matrix we developed various models leveraging the LSTM architecture. We decided to use LSTM (Long Short-Term Memory) layers in neural network because they provide several advantages when dealing with text reviews: they capture complex and long-term relationships present in sequences and they can maintain memory of past information, enhancing the model's ability to understand and interpret the sequential context of the data. The first LSTM model was structured as follows:

1. We set up an Embedding layer which used the embedding matrix as non-trainable weights.
2. We built three LSTM layers, each with 256 units, designed to capture long-term relationships in text sequences.
3. We added Dropout layers that introduced a 20% dropout rate to mitigate the risk of overfitting.
4. We built a dense layer, i.e a single-node layer with *sigmoid* activation for binary classification.
5. We compiled and trained the model. The model was compiled using the Adam optimizer with a specified learning rate and binary cross-entropy loss function. Training involved 50 epochs on the training set (X_train) with validation on the test set (X_test).

Even with a substantial number of layers, the network achieved a peak accuracy of 76% after 43 epochs (results are shown in figure 10).

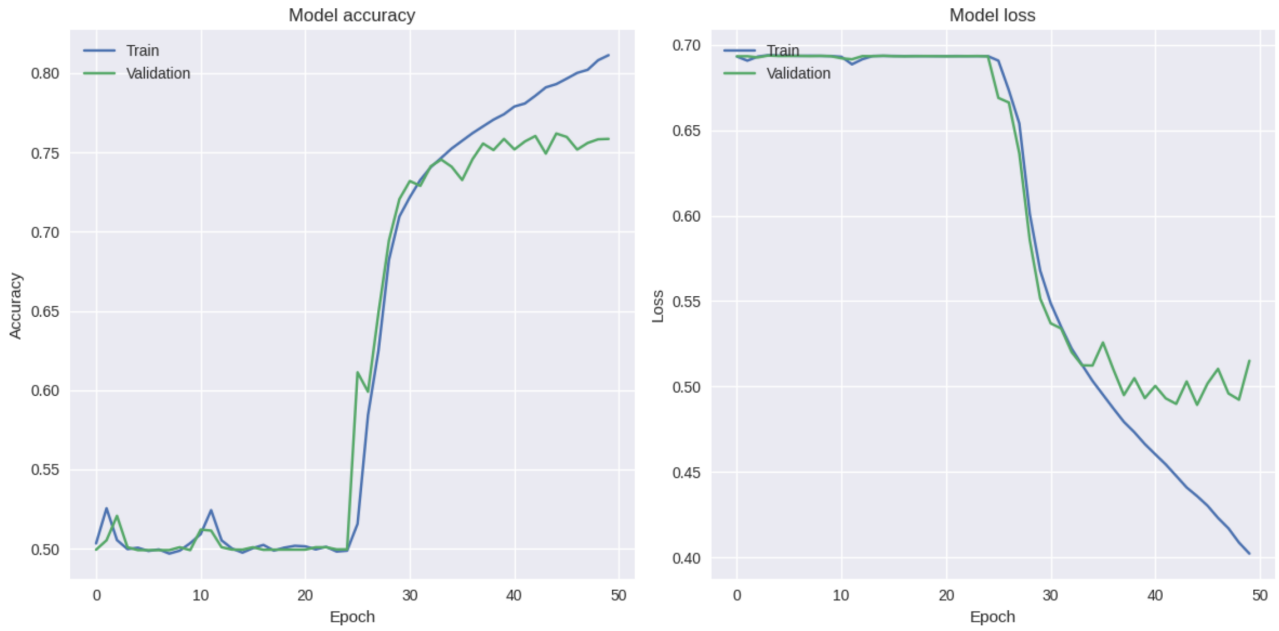


Figure 10: Model Accuracy and Loss for the first LSTM Neural Network.

In an attempt to enhance performance, we decided to augment the model's complexity by introducing two additional dense layers one with 128 nodes and another with 64 nodes. The outcome, however, revealed that despite the significant increase in model complexity, there was no notable improvement in performance. Even at the thirtieth epoch, the accuracy remained around the baseline (fig.11).

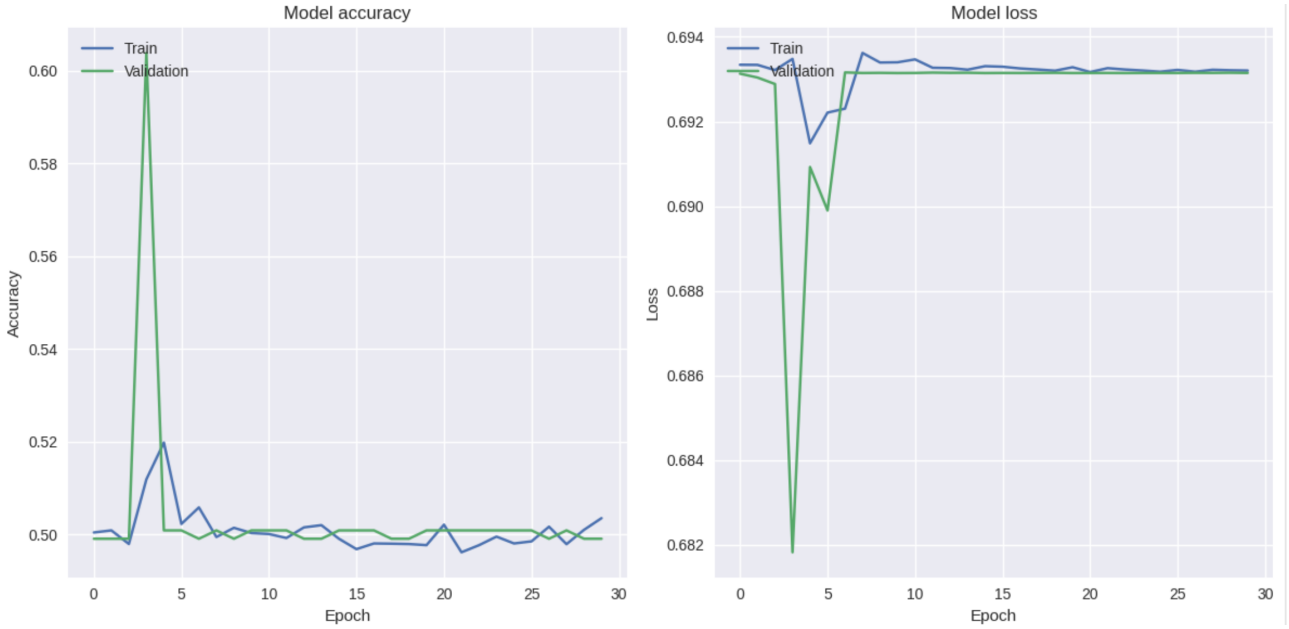


Figure 11: Model Accuracy and Loss for the second LSTM Neural Network.

Then, we opted to reduce the number of layers, exploring an alternative and simpler model configuration:

- We streamlined the structure by removing two of the previous LSTM layers, retaining only a single LSTM layer with 256 units.
- We simplified the model complexity by eliminating the two additional dense layers introduced

earlier (128 nodes and 64 nodes), opting instead for a final dense layer with a single node and *sigmoid* activation for binary classification.

We observed that the training of the model proceeds well until the thirty-fifth epoch, achieving accuracy values close to 75% (see fig.12).

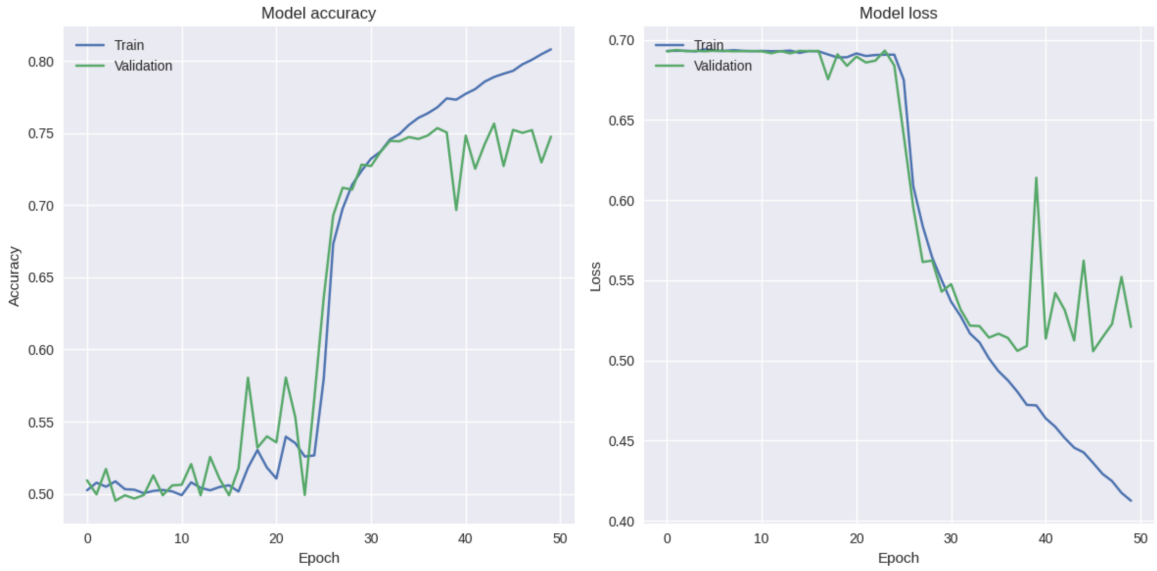


Figure 12: Model Accuracy and Loss for the last LSTM Neural Network.

The model demonstrated balanced performance with good precision, recall, and F1-score values for both classes (all measurements were above 0.72. See the report in fig. 13).

	precision	recall	f1-score	support
0	0.76	0.72	0.74	10007
1	0.74	0.77	0.75	9971
accuracy			0.75	19978
macro avg	0.75	0.75	0.75	19978
weighted avg	0.75	0.75	0.75	19978

Figure 13: Overview on the third model metrics

However, the fluctuating pattern of accuracy and loss values on the validation set led us to consider that the LSTM layer structure might not be optimal for our dataset even with a simplified architecture. Consequently, we explored alternative architectures in an attempt to build models better suited to our data characteristics.

Convolutional Neural Networks

The convolutional layers use filters or kernels to scan across the input data and learn spatial hierarchies of features. Actually, they are effective at capturing local patterns (which represent combinations of words) in our data. For this reason we built up neural networks exploiting convolutional architecture (Conv1D layers in Keras). We also decided to apply MaxPooling operations in order to reduce the dimensionality of the data. Max pooling involves taking the maximum value from a set of values in

a specific window or region. This operation progressively downsamples the spatial dimensions of the input, retaining the most important information while reducing computational complexity.
The first model implemented using Conv1D and MaxPooling operations set up as follows:

1. An embedding layer was initialized with a pre-trained embedding matrix.
2. Three 1D convolutional layers were stacked. Each convolutional layer had 256 filters with a kernel size of 3 and ReLU activation. The purpose was to capture local patterns in the input sequences.
3. Max pooling was applied after each convolutional layer with a pool size of 3.
4. Dropout layers with a dropout rate of 0.5 were added after the second and third convolutional layers. Dropout helped mitigate the risk of overfitting by randomly dropping units during training.
5. We added a Global Max Pooling layer to obtain the maximum value across the entire sequence. This operation captured the most salient features from the output of the convolutional layers.
6. Two dense layers followed the Global Max Pooling layer. Each dense layer consisted of 256 units with ReLU activation. Dropout with a rate of 0.5 was applied after the first dense layer.
7. Finally, we inserted a dense layer consisted of a single node with *sigmoid* activation function. This layer was used for binary classification, producing an output in the range [0, 1]. Binary cross-entropy was chosen as the loss function.

The model was compiled using the Adam optimizer with a specified learning rate (0.0001). It was trained for 20 epochs on the training set with validation on the test set and *EarlyStopping* was applied to prevent overfitting and improve training efficiency.
The depicted graph in fig.14 which shows results from the model (Model 1) implementation, indicates that the model experienced overfitting tendencies from the first epochs.

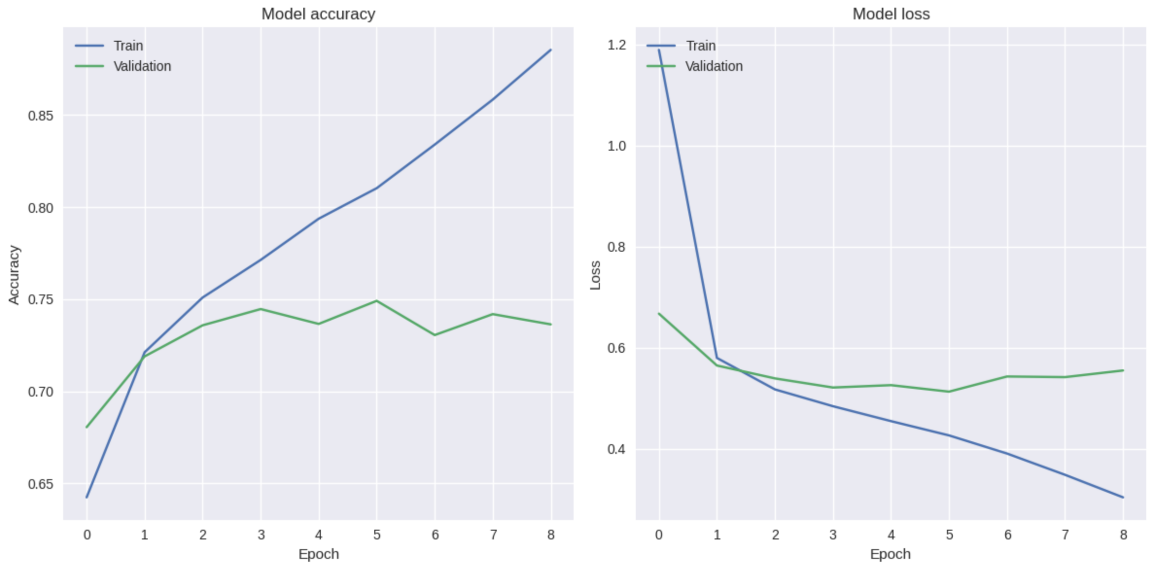


Figure 14: Results from the first implementation of the Convolutional Neural Network indicate that the model is experiencing overfitting. The widening gap between the training and test sets is evident in both accuracy and loss. Notably, the loss begins to rise as early as the fifth epoch, signaling a potential loss of generalization capability and an increase in the model's sensitivity to the training data.

In response to this observation, we implemented corrective measures to mitigate overfitting. Specifically, we reduced the number of nodes in the layers and incorporated kernel regularization into the final two dense layers (Model 2). Then we removed callback dynamic learning rate and the kernel regularizers from the dense layers (Model 3). Finally, we added two kernel regularizers to the first two layers of the neural network (Model 4). The final implementation is depicted in figure 15.

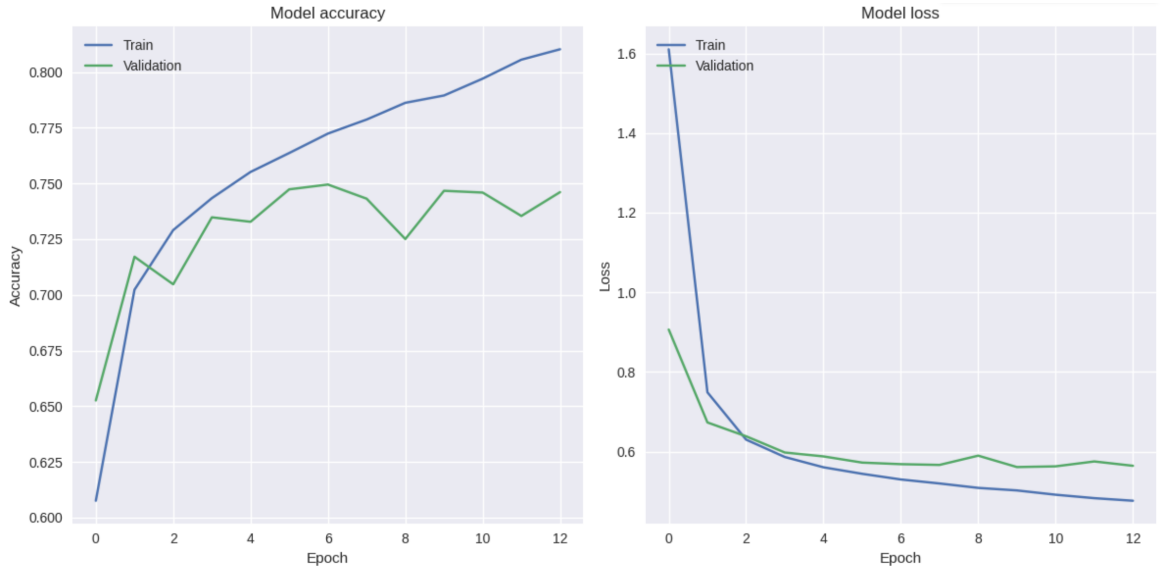


Figure 15: Last implementation of the Convolutional Neural Network. Even though the Loss and Accuracy curves for the validation set exhibit some irregular patterns, it is worth noting that the training and validation curves closely track each other. However, it is important to recognize that we cannot guarantee complete elimination of overfitting, but we have taken steps to mitigate it.

The classification reports for the four convolutional networks are depicted in the following images (fig. 16, 17, 18, 19):

	precision	recall	f1-score	support
0	0.75	0.74	0.75	10007
1	0.74	0.76	0.75	9971
accuracy			0.75	19978
macro avg	0.75	0.75	0.75	19978
weighted avg	0.75	0.75	0.75	19978

Figure 16: Metrics results for the first convolutional network implemented (Model 1)

	precision	recall	f1-score	support
0	0.74	0.75	0.75	10007
1	0.75	0.74	0.74	9971
accuracy			0.75	19978
macro avg	0.75	0.75	0.75	19978
weighted avg	0.75	0.75	0.75	19978

Figure 17: Metrics results for the second convolutional network implemented (Model 2)

	precision	recall	f1-score	support
0	0.72	0.79	0.75	10007
1	0.77	0.70	0.73	9971
accuracy			0.74	19978
macro avg	0.74	0.74	0.74	19978
weighted avg	0.74	0.74	0.74	19978

Figure 18: Metrics results for the third convolutional network implemented (Model 3)

	precision	recall	f1-score	support
0	0.74	0.75	0.75	10007
1	0.75	0.74	0.74	9971
accuracy			0.75	19978
macro avg	0.75	0.75	0.75	19978
weighted avg	0.75	0.75	0.75	19978

Figure 19: Metrics results for the fourth convolutional network implemented (Model 4).

Comparing the classification reports above, we can highlight the following strengths in the models:

- Higher Precision for Class 0 in Model 4: this suggests that when Model 4 predicts negative reviews (Class 0), it tends to be more accurate compared to the other models. It had a lower tendency to incorrectly classify actual negative reviews as events.
- Higher Recall for Class 0 in Model 3: Model 3 excelled in capturing a higher proportion of actual negative reviews (Class 0). It had a lower tendency to miss instances of non-events(negative reviews), indicating strong sensitivity to this class.
- Strong Recall for Class 1 in Model 4: Model 4 performed exceptionally well in recalling instances of postive reviews (Class 1). It had a higher capacity to correctly identify actual events, minimizing false negatives.
- The F1-score, which considers both precision and recall, showed comparable performance across the models. This means that while there were variations in precision and recall for specific classes, the overall balance between precision and recall was similar.

The final implementation (Model 4) generally outperformed others, achieving a good balance between precision, recall, and F1-score for both classes. The accuracy of the model was also quite strong, reaching 75%.

LSTM and Convolutional Neural Networks

Combining both LSTM and convolutional layers, the model can capture both long-term dependencies and spatial patterns simultaneously. This is especially powerful when dealing with complex relationships where understanding both the order of words and local patterns is crucial. Example: "Despite a slow start, the book builds up to a stunning climax." The LSTM can understand the temporal progression, recognizing the change from a "slow start" to a "stunning climax" while the convolutional layer can identify the importance of the phrase "stunning". For all these reasons we have decided to exploit the capabilities of LSTM and convolutional networks, aiming to experiment with hybrid models that harness the strengths of both these architectures. The structure of the first Keras neural network model implemented is as follows:

1. We added an embedding layer that converts sequences of integer-represented words into vectors of a fixed size. This layer is configured as non-trainable (trainable=False) since it relies on a pre-trained embedding.
2. An LSTM layer with 128 units, a dropout rate of 0.2 to reduce overfitting, and a recurrent dropout of 20% were added.
3. A 1D convolutional layer with 128 filters and a kernel size of 5 was added. The activation function was ReLU, introducing non-linearity to the convolution results.
4. A global max pooling 1D layer, which returned the maximum value along the time axis was added. This layer reduced the dimensionality of the convolution output, retaining the most relevant features.
5. We set up two fully connected dense layers. The first layer had 64 units with ReLU activation, while the last layer has 1 unit with sigmoid activation, typical for binary classification.

The model was compiled using the *binary_ crossentropy* loss function, the Adam optimizer, and the *accuracy* metric. The model was then trained using the training data (*train_ padded* and *y_ train*) for 10 epochs, with validation data.

Figure 20: Training and Validation Accuracy associated to the first hybrid LSTM + Convolutional model

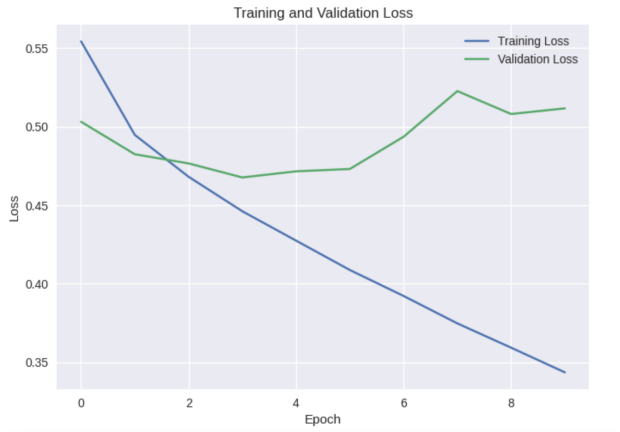


Figure 21: Training and Validation Loss associated to the first hybrid LSTM + Convolutional model

Looking at the graphs above (figure 20 and 21), we noticed that the model was overfitting. To mitigate this effect, we attempted to address the issue by reducing the number of epochs, as illustrated in figure 22 and 23.

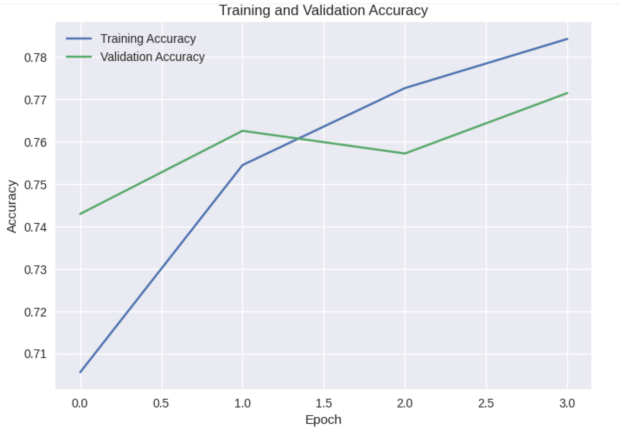


Figure 22: Training and Validation Accuracy associated to the hybrid LSTM + Convolutional model with less epochs

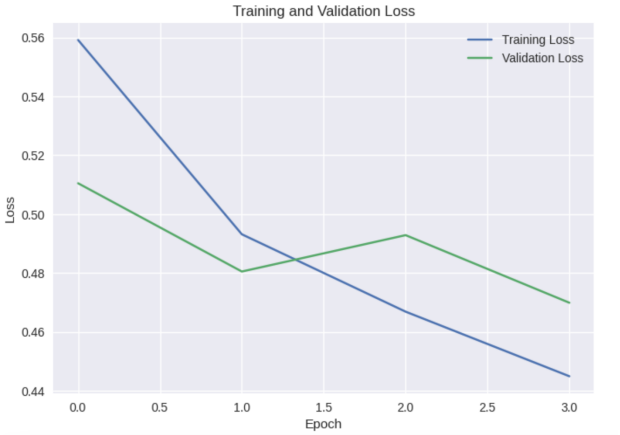


Figure 23: Training and Validation Loss associated to the hybrid LSTM + Convolutional model with less epochs

In this hybrid model, we observe a good level of accuracy, reaching 0.77 on the validation set and 0.78 on the training set. Regarding the Loss, it is recorded at 0.47 for the test set and 0.44 for the training set. Despite these favorable measurements, the adjusted model displayed computational inefficiency, with each epoch demanding several hours for completion. Moreover, with an increase in the number of epochs, the model rapidly succumbed to overfitting. This prompts the consideration that the employed hybrid architecture may not be the most suitable solution for our binary classification problem.
Finally we can affirm that convolutional neural networks deliver superior performance compared to LSTM network models which introduce instability without significantly enhancing performance.

Final results and considerations:

Considering efficiency in terms of computational resources, accuracy (i.e., effectiveness) [2], and F1 score, we identified:

- The best model using the TF-IDF representation as the Logistic Regression model, achieving an accuracy and an average F1 score between the two classes of 76%.
- The best model associated with the Word Embedding text representation as the Baseline Deep Learning model (1), with an accuracy and F1 score of 77%.

Even though Baseline Deep Learning model (2) has identical average accuracy and F1 score values, the preference was given to Baseline Deep Learning model (1). This choice is motivated by the fact that, in this model, the measures are more balanced between the classes. In other words, Baseline Deep Learning model (1) does not exhibit significantly higher precision or recall on one class compared to the other, unlike what is observed in Baseline Deep Learning model (2).

5 TOPIC MODELING

The last task of our analysis is the Topic Modeling that consists in an unsupervised machine learning technique aimed at:

- examining a collection of documents to identify word and phrase patterns within them;
- automatically organizing word clusters and similar expressions that most accurately represent a set of documents.

Each collection of words found corresponds to a cluster. Each cluster is characterized by the words that have a higher probability of occurrence for the given topic. So, each topic will correspond to a specific cluster of words with their probabilities of occurrence. It is possible that the same words characterizes different topic with different probabilities and also that more than one topic are associated with a document.
Before starting we have to do a pair of consideration about our dataset and this type of task:

1. the reviews are on thriller, horror and crime books, so very similar genre, for this reason it is possible that found out very different topic from each other that we can label in specific ways could be difficult;
2. we are going to do this task on reviews, which could not contain relevant information for the task and so it is possible that in some cases it could be difficult to assign one or more topic.

However, we are curious to find out the results of this task on our dataset. We begin our analysis with the preparation of the dataset, in particular:

- we consider only the *review_text* column;
- we shuffled the rows of the column in order to have a random order;
- we applied the lemmatization with *WordNetLemmatizer()*;

[2]Note: In our evaluation, we considered accuracy as a measure of effectiveness, recognizing that there is no imbalanced dataset that necessitates prioritizing F1 score as the primary metric.

- we remove from the reviews the words composed by less than 3 elements;
- we remove the words that occur in more than 10.000 rows.

We applied different Topic Modeling methods and to find the best one, we used three-step comparison between different techniques:

1. we select a subset of observation of the initial dataset and we set the number of topic to find starting from 2 until 15. In this way for each subset we obtain 14 different results based on the number of topic found;
2. we iterate the first step for other different subsets, in particular we used also 15000 observations, 10000 observations and 5000 observations. In this way, for each technique we obtained 56 results and consequently, we select the best result for that method among them;
3. then, we repeat step 1 and 2 for the other methods and at the end we compared the different results obtained from the best model for each different technique and we chose the best one overall.

The methods we implemented are: Latent Semantic Analysis, Latent Dirichlet Allocation, BERTopic and Top2Vec.

5.1 Latent Semantic Analysis

The first method we implemented is the Latent Semantic Analysis (LSA). This technique is based on the idea that similar documents contain similar word frequencies for the certain words.
First of all, the word frequencies are computed and then the Document-Term Matrix (which contains single words on the rows, documents on the columns and the correspondent frequency values in each cell) is created. Then the matrix is decomposed through the Singular Value Decomposition technique and so we obtain the product of three matrix USV^T where:

- matrix U represents the Document-Topic matrix;
- matrix V^T represents the Topic-Terms matrix;
- matrix S is a diagonal matrix with the elements on the diagonal that are considered by the LSA as potential topic.

LSA is one of the simplest techniques and not the optimal one because in this case syntactic and semantic information are ignored.
We used the *LsiModel()* function from the *Gensim* library to implement the LSA model.
This module can work with large corpora (bigger than 20k observations) but in order to compare the results with the other models we consider subset with maximum 20k observations (there are other models that are very slow in processing larger subset). This function implements fast truncated SVD that is a variant of the classical Singular Value Decomposition that retains only the top-k singular values and corresponding vectors, providing a lower-rank approximation of the original matrix.

All the results relative to Latent Semantic Analysis are shown in Appendix A.

5.2 Latent Dirichlet Allocation

The second method we implemented is the Latent Dirichlet Allocation (LDA), that is a generative statistical model. It is based on two assumptions:

1. documents are mixtures of topics;
2. topics are mixtures of words.

In this way documents are categorized by topic, this means that each document can be seen as a combination of the topics and the words are generated from the topics. There is no direct connection between documents and words: topics represent an intermediate layer between documents and words and the connections between documents and topics and between topics and words are found by the algorithm. Both the distributions of topics and words are Dirichlet distributions.

We used three different function to estimate the LDA model:

1. *LatentDirichletAllocation()* from *sklearn* library: this function required a previous conversion of the text data into document-term matrix through *CountVectorizer()* function
2. *LdaModel()* from *Gensim* library: this is an alternative to the previous function that is included in *Gensim*, a library specifically designed for natural language processing and topic modeling;
3. *LdaMulticore()* from *Gensim* library: this is a parallelized version of the previous one that differs from the basic *LdaModel()* because it utilizes multiple CPU cores for faster training.

All the results relative to LDA models are shown in Appendix B.

5.3 BERTopic

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained algorithm used for Topic Modeling task that generates contextual embeddings called BERT embeddings which captures words and documents semantic meaning.
BERTopic technique leverages these contextual embeddings and then, with the application of clustering techniques, (in our case UMAP, Uniform Manifold Approximation and Projection is used) groups similar documents together into topics.

All the results relative to BERTopic models are shown in Appendix C.

5.4 Top2Vec

The last method we implement is Top2Vec. This is a machine learning technique which assumes that semantically similar documents identify an underlying topic. The first step of the algorithm is to create a joint embedding of document and word vectors, in this way we can have a unified representation in a single space. Then, a lower dimension embedding of documents in created. At this point the algorithm identifies dense clusters of documents within the vector space and in particular it finds out which words attracted those documents together. Each dense area identified is a topic and the words that attracted the documents to the dense area are the topic words.
The difference between the previous models implemented is that Top2Vec finds out automatically the number of topics, so we had not to specified them like we have done in the previous cases.

All the results obtained with Top2Vec are shown in Appendix D.

5.5 Evaluation of the models

We star by considering all the models but Top2Vec. For each algorithm we obtain several models characterized by different sizes of subset and number of topics found, in order to evaluate and compare them we use the coherence values. Topic coherence measures give a values about the interpretability of each topic aiming to assess how well the words assigned to a particular topic align semantically.
Each topic is evaluated singularly, a matrix that represents the similarity between words in a given topic is created, in this way it is possible to know and to quantify how often pairs of words appear together in the same context within the documents of that topic. Within the code we specified the measures used as c_v, that is based on a sliding window, one-set segmentation of the top words and an indirect confirmation measure that uses normalized pointwise mutual information and the cosine similarity.
Once the score between pairwise words is computed, an overall coherence of the topic is obtained with an aggregation by averaging the individual scores. At the end, by averaging the coherence score of each topic we obtain the coherence of the model.

We obtained several results, but the highest values of the coherence for each model were obtain with the following combination of subset size and number of topic:

- LSA: 20000 observations and 2 topics;
- LDA with sklearn: 15000 observations and 3 topics;
- LDA with Gensim: 20000 observations and 3 topics;
- LDA with Multicore: 10000 observations and 2 topics;
- BERTopic: 10000 observations and 9 topics.

Due to the fact that we obtained a different selection for each algorithm, in order to decide which can be consider the best model we compare for each combination of best subest and number of topic, the results relative to all the models. We can see in detail in table 2 all the values obtained.
The first thing we can notice is that the values are quite bad and small, and this is and index of a quite bad model found for this topic. So, the considerations that we did at the beginning of paragraph 5 were consistent and appropriate. However, even if these are not optimal results, we can make some considerations.

Subset - n° topics	LSA	LDA with sklearn	LDA with Gensim	LDA with Multicore	BERTopic
20k - 2	**0.542739**	0.464570	0.485323	0.419881	0.405016
15k - 3	0.489735	**0.493352**	0.466433	0.460943	0.421726
20k - 3	0.443380	0.447257	**0.490684**	0.473658	0.332595
10k - 2	0.491949	0.481712	0.440287	**0.479033**	0.416406
10k - 9	0.415634	0.428460	0.405664	0.382402	**0.436607**

Table 2: In this table we can see on the rows the best combination found between subset size and number of topic found in terms of coherence. On the columns we have the models. In each cell there are the values relative to different models. In bold we can see the best values obtained for that model.

From table 2 we can see that the best value overall is the one relative to the Latent Semantic Analysis model, but we can take into account the results obtained from Top2Vec analysis in order to better analyze this situation.
In particular, we had the following results:

1. in this case we have only 4 results, in fact, per each subset, Top2Vec finds out the best number of topics that in our analysis was always 3;
2. the best coherence score is obtained with the subset equal to 5000 observations, but due to the fact that this subset never before was associated with the best result (on the contrary from the table in the Appendices we can see that the worst results were associated to it);
3. to complete the evaluation of Top2Vec model we calculate the cosine similarity scores of the top 50 words to the topic and we found out the highest value of this measures was to one associated with the model estimated on the larger subset.

For the previous considerations we select the second best result that is the one associated to the 20000 subset and now we can consider the values contained in table 2 where the number of topics is 3, so the second and the third rows.
Between these two rows we can see that almost all the values contained in the second row are higher compared to the ones correspondent to the same model but in the third row.
At this point it is difficult to select one model that is the best overall, maybe, taking into account all the consideration, the best choice could be the LDA model with sklearn applied to the dataset of 15000 observations with 3 topics, while considering the "pure" values of coherence, we could assume as the best method applied the LSA.

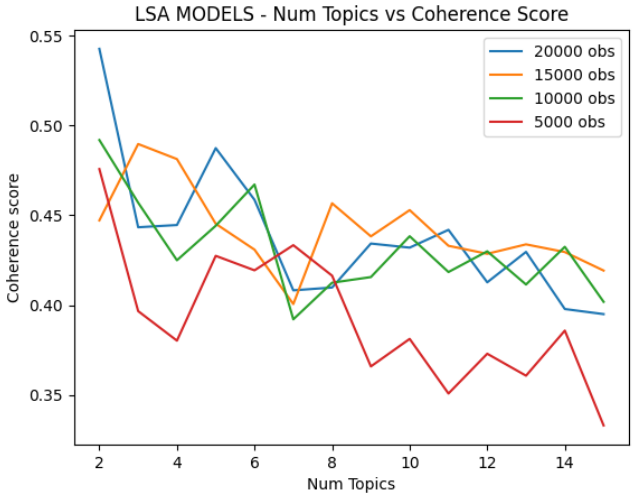
6 CONCLUSION

In conclusion we can say that, starting with the Classification task, we obtained good results in terms of accuracy and F1 score using both TF-IDF representation and Word Embedding. The two best results of the two representations are very similar: 0.76 of Logistic Regression Model vs 0.77 of first Baseline Deep Learning model, in this case the context could not be a crucial information because we are taking into account reviews and not proper text, so, even with the simplest representation we can reach very good results because the documents themselves are not complex.

Unfortunately, the second task didn't give us the same good results: in terms of coherence scores we reached the best result only at 0.54, so it was difficult to find clear topics among our documents and this aspect lead us to a difficult choice of best method used. Probably, our reviews are not the optimal texts on which try methods of topic modeling.

A LSA model results

Appendix A contains all the results relative to the Latent Semantic Analysis models. In figure 24 we can see on the left the graph with the coherence score relative to the four subsets as the number of topic increases. On the right (figure 24b) are shown all the coherence values. From the figure 24a we



(a) Coherence score values as number of topics increases

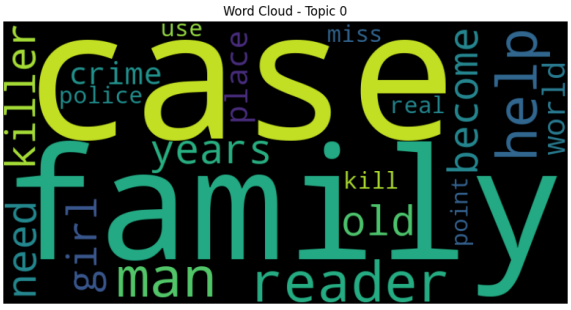
Num Topics	Model 1	Model 2	Model 3	Model 4
2	0.542739	0.447210	0.491949	0.475779
3	0.443380	0.489735	0.457069	0.396672
4	0.444621	0.481345	0.424997	0.380225
5	0.487448	0.445325	0.444212	0.427458
6	0.458659	0.430903	0.467181	0.419389
7	0.408245	0.400617	0.392101	0.433401
8	0.409810	0.456720	0.412494	0.416410
9	0.434311	0.438312	0.415634	0.365868
10	0.432030	0.452930	0.438385	0.381209
11	0.441939	0.433059	0.418443	0.350785
12	0.412716	0.428592	0.429984	0.372940
13	0.429668	0.433848	0.411486	0.360715
14	0.397843	0.429569	0.432480	0.385822
15	0.395006	0.419256	0.401848	0.333005

(b) Coherence score. Model 1: 20k observations, Model 2: 15k observations, 10k observations, 5k observations.

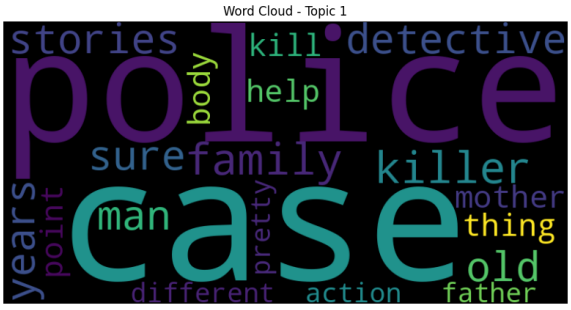
Figure 24: LSA models results: coherence score vs number of topics.

can see that the highest values of coherence is reached for the subset of 20k observations and 2 topics, from figure 24b we can see that this value is equal to 0.542739.

In figure 25 we can see the two word clouds correspondent to the two topics of best LSA model obtained with 20k observations.



(a) First Topic identified with best LSA model



(b) Second Topic identified with best LSA model

Figure 25: Topics identified with best LSA model

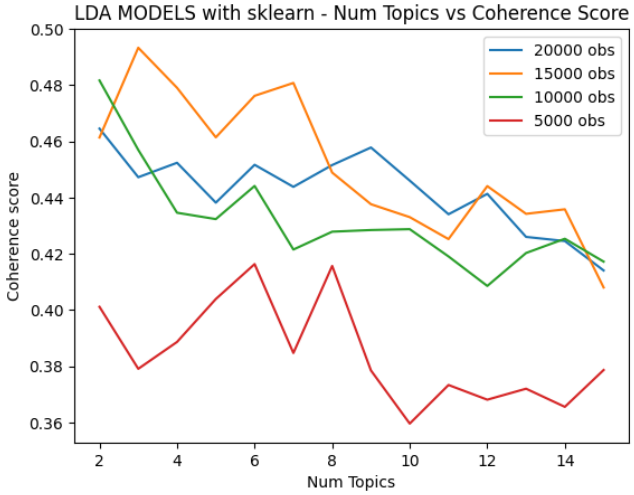
We can see that it is a bit difficult to assign a label to this two topic, of course we can notice that they both contain words linked to the crime, horror and mystery world.

B LDA model results

Appendix B contains all the results relative to Latent Dirichlet Allocation models.

B.1 LDA model with sklearn results

The first function implemented for the LDA models from *sklearn* library gave the following results, in particular, from figure 26a we can see that the model associated to the subset of 15000 observations is the one with the best coherence score in almost all the cases. From figure 26b we can see that the highest value is in correspondence of 15000 observations subset and number of topic equal to 3.



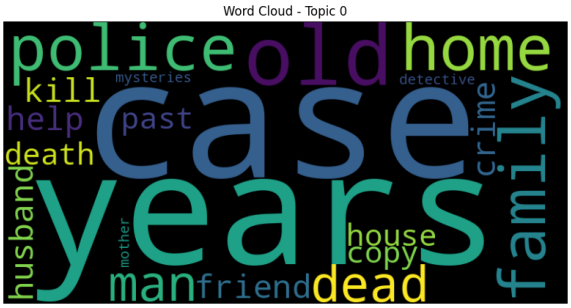
(a) Coherence score values as number of topics increases

Num Topics	Model 1	Model 2	Model 3	Model 4
2	0.464570	0.461402	0.481712	0.401134
3	0.447257	0.493352	0.456977	0.379059
4	0.452395	0.479089	0.434615	0.388632
5	0.438206	0.461458	0.432348	0.403912
6	0.451685	0.476241	0.444154	0.416332
7	0.443829	0.480816	0.421519	0.384698
8	0.451587	0.448908	0.427871	0.415675
9	0.457837	0.437653	0.428460	0.378475
10	0.446054	0.433047	0.428766	0.359559
11	0.434076	0.425228	0.419058	0.373304
12	0.441343	0.444095	0.408558	0.368072
13	0.426013	0.434227	0.420285	0.371982
14	0.424538	0.435839	0.425359	0.365521
15	0.414066	0.408003	0.417219	0.378641

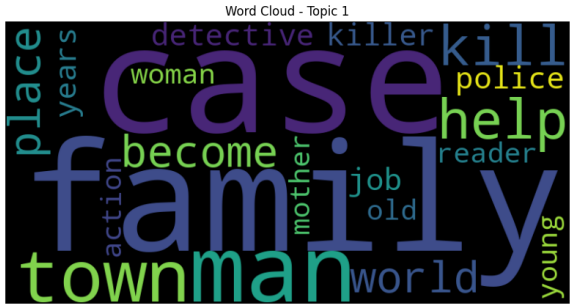
(b) Coherence score. Model 1: 20k observations, Model 2: 15k observations, 10k observations, 5k observations.

Figure 26: LDA models with sklearn results: coherence score vs number of topics.

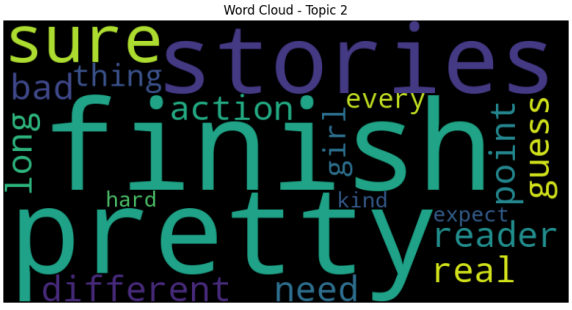
Figure 27 shows the three word cloud associated to the best model. Similar to the case of LSA model, even here the three topics identified seems to be very similar with each other.



(a) First Topic identified with best LDA model with sklearn



(b) Second Topic identified with best LDA model with sklearn

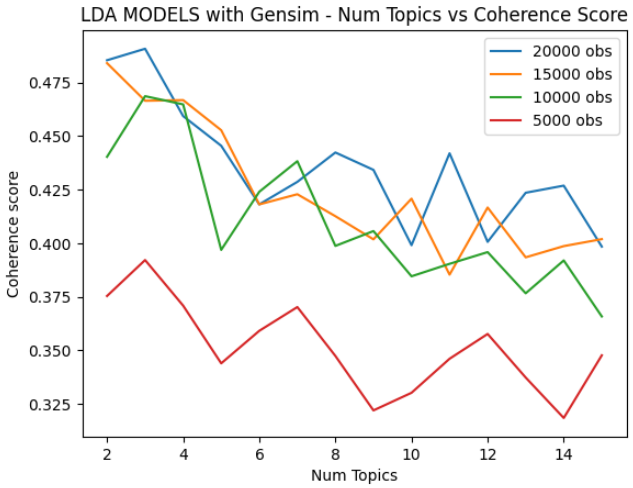


(c) Third Topic identified with best LDA model with sklearn

Figure 27: Topics identified with best LDA model with sklearn

B.2 LDA model with Gensim results

The results obtained with Gensim are very similar to the ones obtained with sklearn. In fact, from the graph in figure 28a we can see that the higher coherence values are associated to the two largest subsets. Even in this case the optimal number of topics found is 3 but from figure 28b we can see that the subset with the highest value is the one with 15000 observations.



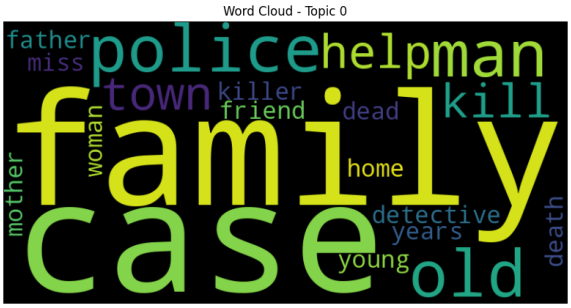
(a) Coherence score values as number of topics increases

Num Topics	Model 1	Model 2	Model 3	Model 4
2	0.485323	0.483976	0.440287	0.375379
3	0.490684	0.466433	0.468577	0.392146
4	0.459272	0.466734	0.464740	0.370878
5	0.445438	0.452700	0.396883	0.343977
6	0.418104	0.418098	0.423981	0.359136
7	0.428607	0.422835	0.438256	0.370226
8	0.442310	0.412593	0.398742	0.347413
9	0.434195	0.401802	0.405664	0.322025
10	0.399070	0.420762	0.384569	0.330239
11	0.441932	0.385340	0.390383	0.346110
12	0.400630	0.416644	0.395846	0.357681
13	0.423525	0.393414	0.376642	0.337351
14	0.426851	0.398641	0.391966	0.318486
15	0.398415	0.401900	0.365828	0.347722

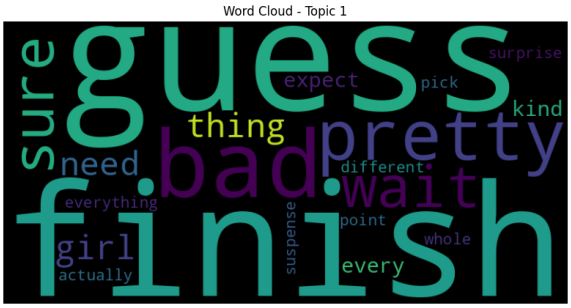
(b) Coherence score. Model 1: 20k observations, Model 2: 15k observations, 10k observations, 5k observations.

Figure 28: LDA models with Gensim results: coherence score vs number of topics.

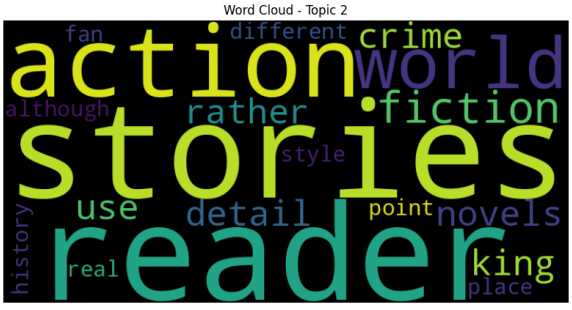
From figure 29 we can see the three word cloud of the model.



(a) First Topic identified with best LDA model with Gensim



(b) Second Topic identified with best LDA model with Gensim

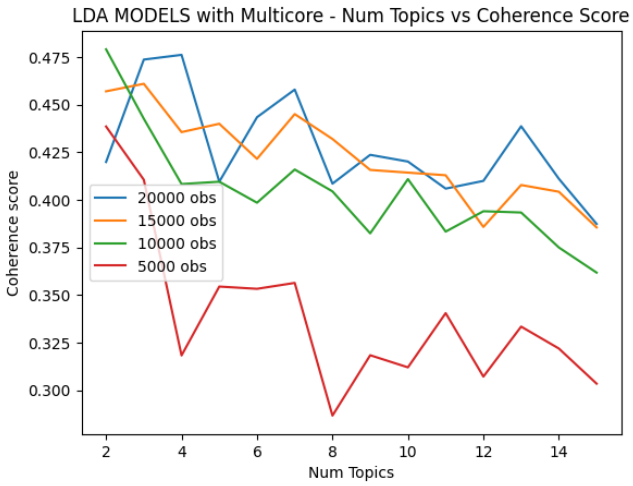


(c) Third Topic identified with best LDA model with Gensim

Figure 29: Topics identified with best LDA model with Gensim.

B.3 LDA model with MultiCore results

Similarly to the previous cases here we can see the graph (figure 30a) of coherence values ad the number of topics increases. From table 30b we can see that the highest value is associated to subset 10000 and number of topics equal to 2.



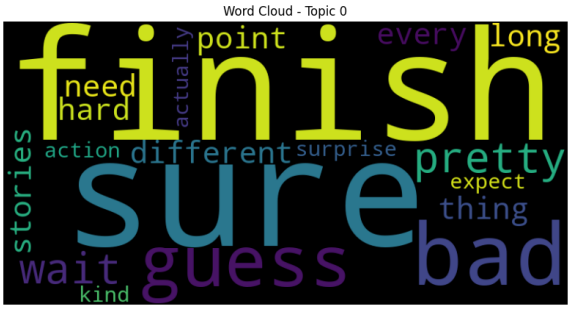
(a) Coherence score values as number of topics increases

Num Topics	Model 1	Model 2	Model 3	Model 4
2	0.419881	0.456978	0.479033	0.438497
3	0.473658	0.460943	0.442583	0.410540
4	0.476137	0.435568	0.408293	0.318293
5	0.409509	0.439944	0.409522	0.354462
6	0.443390	0.421528	0.398489	0.353281
7	0.457895	0.445020	0.415928	0.356376
8	0.408491	0.431949	0.404436	0.286698
9	0.423636	0.415707	0.382402	0.318396
10	0.420102	0.414297	0.410886	0.312035
11	0.405883	0.412910	0.383317	0.340509
12	0.409960	0.385762	0.394011	0.307188
13	0.438637	0.407784	0.393348	0.333439
14	0.411067	0.404252	0.374984	0.321928
15	0.387352	0.385515	0.361784	0.303482

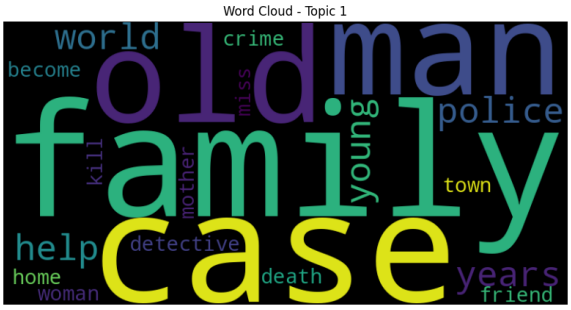
(b) Coherence score. Model 1: 20k observations, Model 2: 15k observations, 10k observations, 5k observations.

Figure 30: LDA models with Multicore results: coherence score vs number of topics.

In figure 31 we can see the two word clouds based on the two topics.



(a) First Topic identified with best LDA model with Multicore



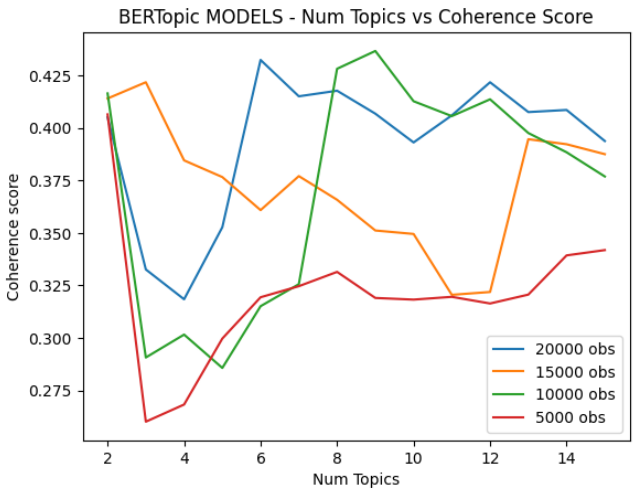
(b) Second Topic identified with best LDA model with Multicore

Figure 31: Topics identified with best LDA model with Multicore.

C BERTopic

In this section are shown all the results relative to BERTopic models.
From figure 32a we can see that models estimated on subsets of 20000 and 10000 show better results in terms of coherence when the number of topics is higher. The highest value of coherence is reached when the subset is equal to 10000 observations and the number of topic found is 9.
From figure 32b we can see that this coherence value is equal to 0.436607.

Comparing this results with the ones in the previous sections, we can notice that this is the smallest and also that BERT is the only one algorithm that gave as best result the one associated with 9 topics.



(a) Coherence score values as number of topics increases

Num Topics	Model 1	Model 2	Model 3	Model 4
2	0.405016	0.413987	0.416406	0.406413
3	0.332595	0.421726	0.290691	0.260130
4	0.318399	0.384528	0.301589	0.268297
5	0.352713	0.376577	0.285716	0.299706
6	0.432341	0.360841	0.315104	0.319364
7	0.415026	0.377039	0.325591	0.324632
8	0.417686	0.365764	0.428129	0.331439
9	0.406759	0.351146	0.436607	0.319034
10	0.393061	0.349499	0.412679	0.318245
11	0.405952	0.320520	0.405596	0.319561
12	0.421712	0.321871	0.413599	0.316396
13	0.407516	0.394608	0.397549	0.320622
14	0.408529	0.392210	0.388380	0.339294
15	0.393735	0.387479	0.376853	0.341814

(b) Coherence score. Model 1: 20k observations, Model 2: 15k observations, 10k observations, 5k observations.

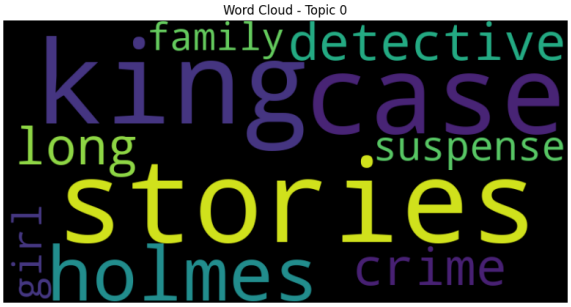
Figure 32: BERTopic models results: coherence score vs number of topics.

Through the function *visualize_barchart()* applied on the best model found above, we can obtain the representation of topics in figure 33. We can notice that only 8 topics are represented in this picture and that's because there is a "topic" classified as -1 that should be ignored which contains all the documents that BERT could not classify in any topic.

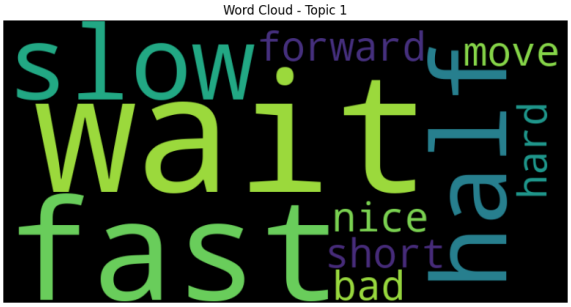
Figure 33: Topic word scores for best BERT model.

From figure 33 we can notice that even in this case is not easy to label the Topics found even if they are more than the ones found with LSA or LDA models. Only Topic 5 is the one that, with the words "laugh", "entertain", "loud", "funny" and "humor", can be easily associate to reviews on funny books.

Figure 34 contains the 8 word clouds relative to the topics of best BERT model.



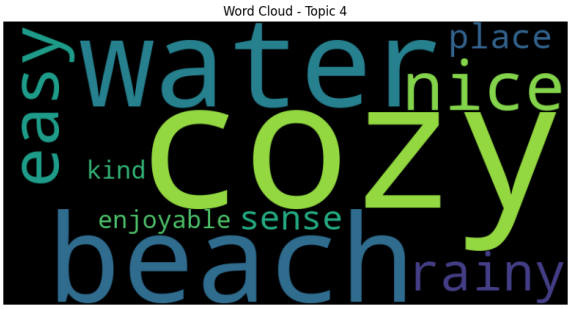
(a) First Topic identified with best BERTopic model



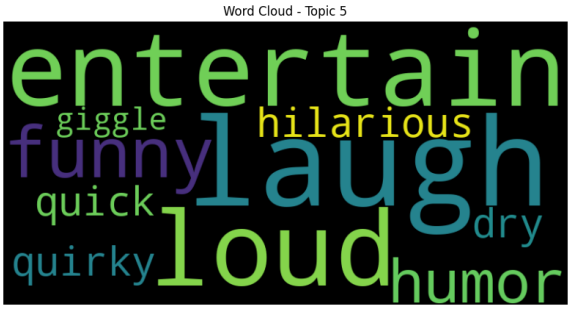
(b) Second Topic identified with best BERTopic model

(c) Third Topic identified with best BERTopic model

(d) Fourth Topic identified with best BERTopic model



(e) Fifth Topic identified with best BERTopic model



(f) Sixth Topic identified with best BERTopic model

(g) Seventh Topic identified with best BERTopic model

(h) Eighth Topic identified with best BERTopic model

Figure 34: Topics identified with best BERTopic model.

D Top2Vec

In this section we can find all the results relative to Top2Vec implementation. In this case we don't have the same number of results as in the previous cases because Top2Vec model finds out automatically the number of topics.

In our case, for each subset the number of topic found was 3 and from figure 35 we can see that the

highest score of coherence is reached when the subset is equal to 5000, the smaller size.

Num Topics	Model 1	Model 2	Model 3	Model 4
3	0.376647	0.327152	0.328603	0.389569

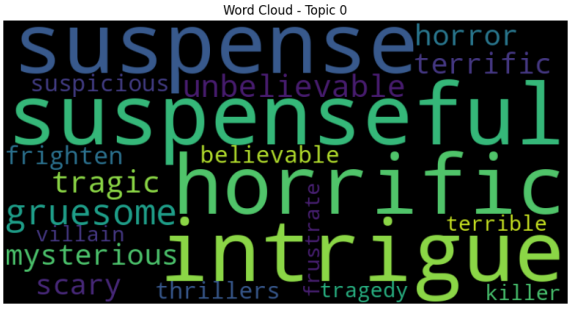
Figure 35: Coherence score. Model 1: 20k observations, Model 2: 15k observations, 10k observations, 5k observations.

From figure 36 we can see the values of cosine similarity. As we anticipated before, the highest values is reached with the model estimated on 20000 observations.

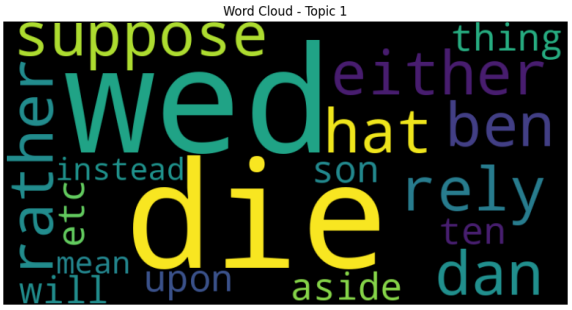
Num Topics	Model 1	Model 2	Model 3	Model 4
3	0.413466	0.409055	0.398718	0.26745

Figure 36: Cosine similarity score. Model 1: 20k observations, Model 2: 15k observations, 10k observations, 5k observations.

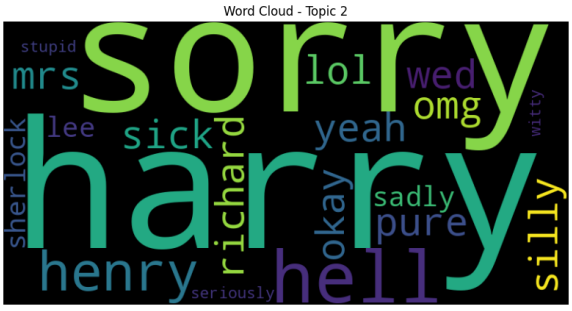
In paragraph 5.5 we have said that we consider the model estimated with 20000 observations (the second best model), because none of the other methods had the best result with 5000 observations. In figure 37 we can find the word clouds relative to the model estimated with 20000 observations.



(a) First Topic identified with best Top2Vec model



(b) Second Topic identified with best Top2Vec model



(c) Third Topic identified with best Top2Vec model

Figure 37: Topics identified with best best Top2Vec model.