University of Pisa

Text Analytics Project

Text Classification of Movies by Genre

Eva Manai
Emanuela Fortuna
Francesco Lanci Lanci
Gabriele Gori
Katherine Abramski

January 2024

Contents

1 Introduction

In this study, we explore the application of different analytical tools and machine learning algorithms for classifying movies into distinct genres based on their titles and descriptions (synopses). We chose this task and labeled dataset, which we borrowed from a Kaggle competition, for a number of reasons. From a practical perspective, we thought that the relative simplicity of the task and dataset would allow us to maximize our learning, since we wanted to focus our time and attention on exploring a variety of methods. We also chose the topic because we found it interesting and relatable. Almost everyone likes movies, and most people select a movie to watch based on information in the title and description, so a movie genre classifier could have real-world applications in several different settings. As stated on the Kaggle challenge page:

From a recommendation system perspective, an effective genre classifier can help build more personalized user recommendations, increasing user engagement on streaming platforms. In the context of box office performance, understanding the relationship between genres and how they are perceived in synopses can provide insight into patterns of commercial success or failure. Furthermore, this challenge can facilitate a deeper comprehension of movie themes and trends in the industry, contributing to cultural and societal studies.

For our analysis, which was a single-label multiclass classification task, we explored the application of multiple different classification methods including: DNNs, CNNs, XGBoost, LSTM, and BERT. We also explored the application of topic modeling, specifically latent dirichlet allocation (LDA), to see if it resulted in topics that aligned with the genre labels of the dataset. From our topic modeling exploration, we found that the topics defined by the LDA do not align very well with the dataset genre labels, and from the text classification exploration we found that the BERT model performed the best with an overall accuracy of 0.41.

The remainder of this report is structured as follows. In Section 2, we provide a brief review of previous work. Then in Section 3 we discuss the methodology we applied in our analysis, including a description of our data (Section 3.1), topic modeling approach (Section 3.2), and classification models (Section 3.3). In Section 4 we discuss the results of our analyses. In Section 5 we discuss the implications of our analysis, and finally in Section 6 we provide a final summary of our work.

2 Background

In recent decades, there has been an enormous increase in the volume of unstructured textual data all around us. The availability of this data has led to a boom in the development of text mining and natural language processing methods that make it possible to gain valuable insights from this textual data. Topic modeling and text classification are two natural language processing tasks that have become very popular in a wide variety of applications.

Topic modeling is an unsupervised task that aims to find topics among groups of documents by extracting latent variables present in large textual datasets [18]. This task can be applied in a variety of settings, and has been applied for grouping movies by genres to explore how genres change over time [16], and to create recommender systems [9, 1]. One of the most popular approaches to topic modeling is latent dirichlet allocation (LDA) [2], a probabilistic model guided by two principles: every document is a mixture of (at most k) topics, and every topic is a mixture of words. The analysis indicates the proportion of each topic present in each document. This approach is very popular for it's flexibility and interpretability, however it performs poorly for documents that do not have a sufficient length (< 50 words) or for finding complex topic relationships [18]. In fact in [1], the authors found that latent semantic analysis (LSA) performed

better than latent dirichlet allocation (LDA) for creating a plot-based recommendation system. Several other topic modeling methods exist that aim to overcome the weaknesses of LDA [18].

Unlike topic modeling, text classification is a supervised task that trains a model on labeled data. While there are a wide variety of machine learning and deep learning methods that may be used for text classification (e.g. logistic regression [3], k-nearest neighbor [7], Naive Bayes [8], SVMs, [17], tree based algorithms [19], DNNs [10], RNNs [14], CNNs [13], and BERT [6]) among others, most methods typically involve the same stages: preprocessing, feature selection, dimension reduction, classifier selection/fitting, and model evaluation [11]. Several previous studies have explored the use of various text classification methods for classifying movies. In [5] the authors apply TF-IDF and SVM to classify movies into four genres (animation, action, drama, and comedy) from Chinese descriptions and they find that the algorithm performs very well. In [4] the authors apply bidirectional LSTM to perform movie genre classification from plot summaries and they find it performs better than logistic regression and RNNs as a baseline. In [12] the authors use a combination of problem transformation techniques, text vectorizers, and machine learning classifiers in a multi-label classification task with 27 movie genres and they achieve astonishing 95 percent accuracy. In [15], the authors classify movies into genres using a multimodal approach that includes trailer video clips, subtitles, synopses, and movie posters. They find that a combination of LSTMs – one trained on synopses and another trained on subtitles – was the best performing model.

These previous studies are interesting examples of investigations that are similar to ours. In the next section, we describe the methods that we chose to explore in our analysis, detailing the model specifications that we used.

3 Methods

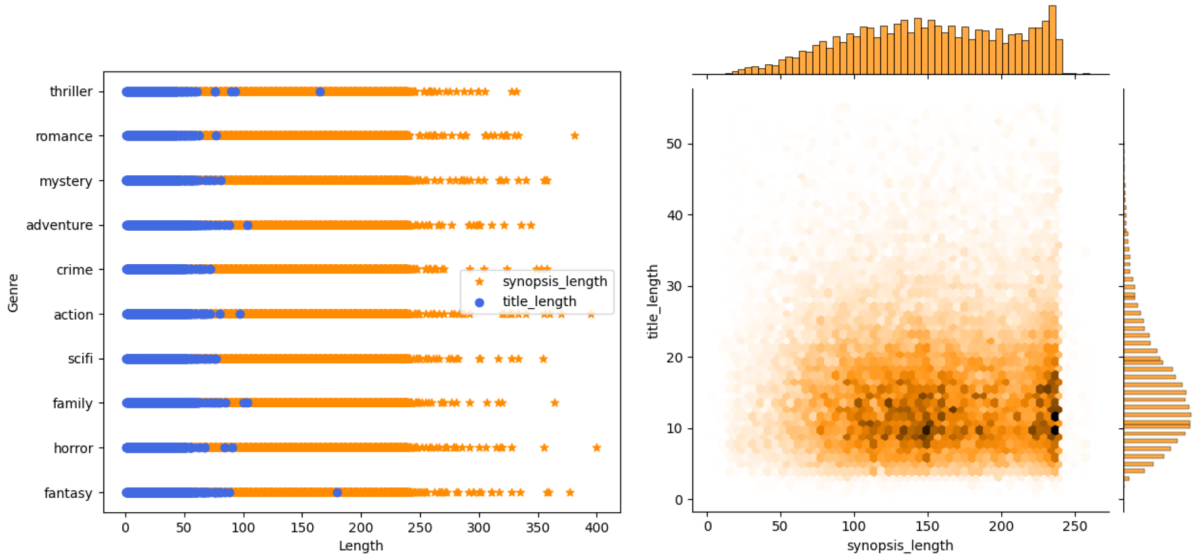
To conduct this study, we followed the Knowledge Discovery Database process, which fit our need to explore and manipulate raw data with the purpose of gaining useful insights. First we started with the data collection phase where we searched over main data repositories for an interesting dataset related to entertainment, as we decided in earlier planning stages of the project. We found the Movie Genre dataset, a good challenge for our purposes, due to the fact that it is not too detailed or enriched with other information such as reviews or rankings. Therefore, its simplicity allows us to test various models on it, only using the titles and movie descriptions as predictors (typically the first information available when a new movie is announced), with the goal of understanding if that information is powerful enough to feed our algorithms in a useful way. Then we proceed with the data preparation phase, in which we define our own set of functions as described in detail in the dedicated paragraph. We merge this phase with the subsequent data cleaning phase to obtain a cleaned dataset on which we can work without too much noise. Then we perform data transformation, to make the structure and the format of the data uniform for later mining with several algorithms, including the creation of embeddings for texts. Then we train our chosen models to exploit our objectives and evaluate them using the classical measures of performance including accuracy, F1-score and recall, aiming to improve these results using hyperparameter tuning for an adequate fit.

3.1 Data

Description

For this study, we chose the Movie Genre Dataset, available at the following link: *https://www.kaggle.com/datasets/guru001/movie-genre-prediction*. The original data are organized into four attributes: *id, movie_name, synopsis* and *genre*, and our main objective is to

infer a movie's genre from its title and description (synopsis). The original files were already split into train and test sets but we noticed that the test data contained only the *action* genre. So, given the large size of the train data, we decided to use it as our entire dataset, upon which we performed the train-test split with a 20/80% ratio, discarding the original test data. In the remainder of the report, we refer to this portion of data as "our dataset", or simply "data". In subsequent steps of our study, we also needed to further split the train data to obtain a validation set, as described in dedicated paragraphs. The *genre* attribute, which we aim to predict, can take 10 different values: *action, adventure, crime, family, fantasy, horror, mystery, romance, scifi, thriller*, and we have nearly a perfectly balanced dataset. The lengths of the synopses across genres do not vary much: the majority of them have from 148 to 153 characters each. The lengths of the titles vary similarly for all genres as well: most titles are shorter than 70 characters, and we notice that longer titles belong to *fantasy* while shorter titles belong to *thriller, horror*, and *romance*, with the exception of some outliers that reach about 180 characters of length, which we kept in the dataset. An overview of title and synopsis lengths can be seen in Figure 1



((a)) Synopsis and title lengths

((b)) Jointplot

Figure 1: Title and Synopsis lengths

Preprocessing

First, we removed unnecessary strings from the *synopsis* (such as "plot kept under wraps" or "See full synopsis") found in movie descriptions using Regex. To do this, we first manually inspected rows to find these unwanted strings that were indicating missing data. We defined a set of words related to this problem, and then we used regexes to find them and remove them. We also attempted to detect the language used in the synopses using the *langDetect* library, with the purpose of deleting non-english records, but unfortunately we noticed that too many records were incorrectly flagged as non-english using this library. Second, we converted all text to lowercase, tokenized the text (split into words), removed one-character words as they were considered insignificant, lemmatized the text, and removed stopwords. We also considered concatenating the *title* field with its *synopsis*, but no further useful information was gained from this. At this point, the text was ready for feature extraction.

Feature selection

To extract the features to use in our text classification models, we applied two approaches: a binary representation which we used as input for the DNNs, and vectorized word indexing, employed as input for all other models.

Binary representation Our first attempt of feature selection was to implement a binary representation of data, exploiting the Gensim *corpus2dense* function. Tokens were cleaned (we removed numbers and single characters) and lemmatized. In this case we used both unigrams and bigrams, excluding stopwords. This resulted in a dictionary of length 11037.

Vectorized word indexing We applied this approach to create dense vector representations. Here, we first created the dictionary (with only the words from the training set) associating each word with an index using the Keras *tokenizer* function. We then converted each word list to a vector of indices corresponding to the words. This led to the padding problem. Since for each model we used different approaches to split the data – for some we created training, validation, and test sets, for others we used k-fold cross-validation – we computed the word list lengths of each x_train list for each model, and used the mean and maximum lengths to choose the optimal length for padding (length = 30) to reach a good compromise between complexity and loss of information.

3.2 Topic modeling

For this task we exploited both gensim and LDAvis libraries. In this unsupervised task, our main goal was to harness the opportunity to find hidden themes/patterns in the data. We consider this part of the project as our exploratory analysis, and we will discuss also some genre characteristics in the results. We conducted LDA with $k = 10$ topics, to see whether the genres relate to them or not. Moreover, we tested also different topic configurations. Before the actual LDA algorithm, we prepared our data. We repeated the same preprocessing done for other tasks, but we also had to prepare the dictionary excluding words that appear in less than 5 documents and in more than 40% of them. This dictionary is fed into the LDA algorithm to compute the topics. The final dictionary has 10533 tokens. *alpha* and *eta* hyperparameters were chosen independently by the algorithm, and the number of iterations was set to 600.

3.3 Text Classification

For our supervised text classification task, we explored the following machine learning algorithms: dense neural networks (DNNs), convolutional neural networks (CNNs), LSTM, BERT, and XGBoost. The details of how we fit these models are described in the what follows.

Dense Neural Networks

Our dense neural networks rely on the binary representation of data (1 = presence, 0 = absence). We have $11,037$ features overall, considering also bigrams. Our validation split is 0.33 in this case. All the major aspects of the net are randomly chosen. Each net is composed of 3 hidden layers, and their dimensions are chosen randomly within the set (32, 64, 128). Every layer has the same number of neurons, except for the first hidden layer, which can have more neurons than the others. Its number of neurons is multiplied by a random factor 1, 2 or 3. The activation function for the hidden layers is ReLU and softmax for the output layer, and finally, the optimizer is Adam.

Convolutional Neural Networks

The goal of this part of the project is to implement a convolutional neural network taking advantage of *Word2Vec*. The net takes as input all the synopses in the train set after applying the vectorized word indexing. Exploiting *Word2Vec* we created the embedding matrix. Since each synopsis is composed by less than 30 tokens in most cases (more than 99%), we set the padded input size to 30.

In order to make the model perform as we wanted, we performed a random search involving several parameters for the net (learning rate, L2 kernel regularization, kernel size for the convolution, batch size). The nets always have a similar structure including: an embedding layer; a 1D convolutional layer; ReLU activation; global max-pooling; a dense net with two layers such that the dimension of each of them is randomly selected by the random search; between each layer and the next one (including the layers in the dense net) there is a dropout layer (dropout probability is randomly chosen); the optimizer is Adam. We selected 100 as our optimal vector_size for *word2Vec* embeddings, and validation split is 0.33.

XGBoost

In our exploration, we also want to try a classic machine learning ensemble algorithm that uses what is called "the wisdom of the crowd" by training multiple weak learners and by an "adjust and combine" strategy to reach more precise results: the *eXtreme Gradient Boosting*. First, we preprocess data as we did for the CNN algorithm and encode the *synopses* with *Word2Vec*. Then we perform a randomized search with $cv = 5$, $n\_iter = 10$ to find the optimal parameterization for our case. This tuning allows us to gain about 15 percentage points in accuracy.

LSTM

Here, as for the CNN, the network takes as input all the synopses of the train set after applying the vectorized word indexing, and the embedding matrix created exploiting *Word2Vec* with 100 as vector_size. For model selection, we tested the following hyperparameters by a random search without cross-validation and for a maximum of 15 epochs (for complexity reasons) with early stopping: an embedding layer; an LSTM layer with units between 25 and 150, dropout between 0.2 and 0.5, and recurrent dropout between 0.2 and 0.5; dense layer with 10 units and softmax as activation function; Adam optimizer with a learning rate between 0.00001 and 0.001; batch size between 4, 8, 16, 32. Here the validation split is 20%.

BERT

In this section, to predict genres based on synopses, we used BertForSequenceClassification. Unlike the other models, for the preprocessing we used BertTokenizer and then we did not use a random search, since training the model was time-consuming (about ten minutes per epoch), so we tried the following hyperparameters: 6 epochs; weight decay = 0.01 for all the parameters except biases and layer normalization weights; learning rate = $2e-5$; warmup steps = 0.2 * train length; batch size = 32. Also here the validation split is 20%.

4 Results

4.1 Topic modeling

Average LDA topic coherence is -7.14022 with 10 topics. From here on, we wanted to understand what these 10 topics were telling us. Since this is part of our exploratory analysis, we

went a step further and we also plotted the relationship between genres, in terms of common and frequent words. Figure 2 explains that genres can be more or less defined by their tokens. This

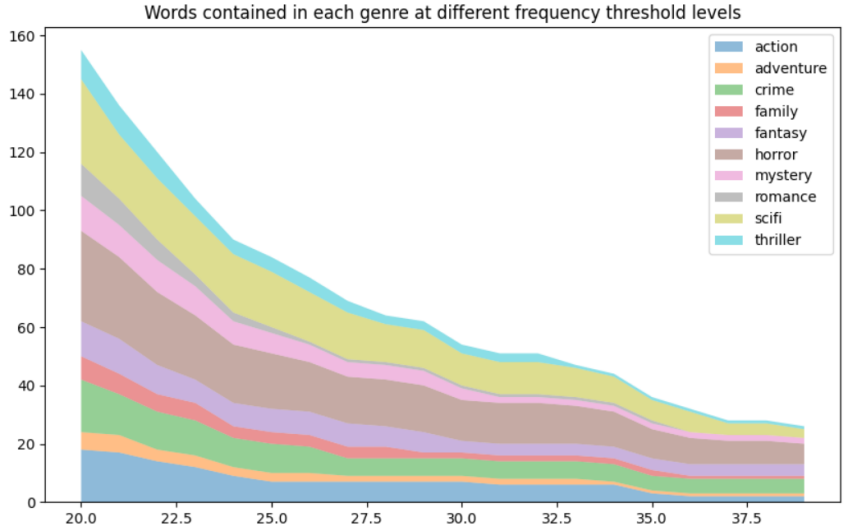
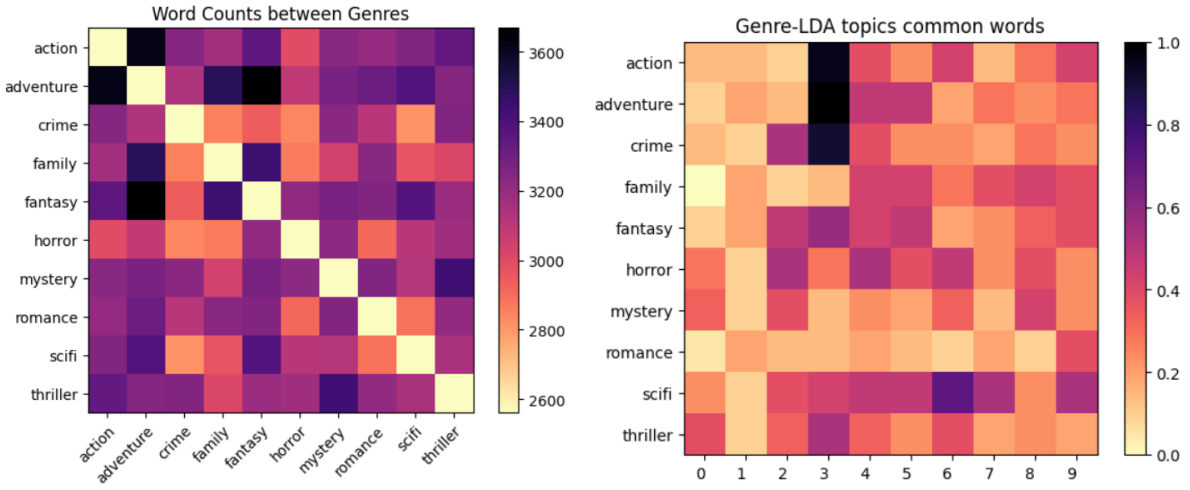


Figure 2: Some genres contain more repeated words than others.

plot is made by computing how many words each genre lexicon contains if we set the token's frequency threshold to a particular value (frequency with respect to the same genre). If we set a threshold of 36, for example, genres such as *family* and *romance* basically disappear. We assume that these genres contain a wider variety of words with fewer repetitions, so we expect that they are more difficult to be detected. Then, we explored how many words have genres in common



((a)) Genres common words **((b))** Genres-LDA topics

Figure 3: Common words between genres and their relationship with topics

with each other. The diagonal in Figure 3 a is filled with 2562, which is just the number of tokens belonging to all genres. *Action* and *adventure* appear to be the most linked ones, but *adventure* also shares tokens with *fantasy*.

It's interesting to notice that *Topic 3* is strongly related to *action*, *adventure* and *crime*, implying that these 3 genres share some characteristics; also *fantasy* is somehow related to it, and it seems to confirm the pattern we can see in Figure (3a). Furthermore, *scifi* is related

to *Topic 6* but also with Topics *4, 5, 7* and *9*. *Topics 0* and *1* contain many words that are not interesting for genre recognition, and they do not seem to point to any genre in particular. Unfortunately, not all LDA topics seem to relate to all the actual genres, but we expected this result since movie genres can be ambiguous. However, some interesting additional information

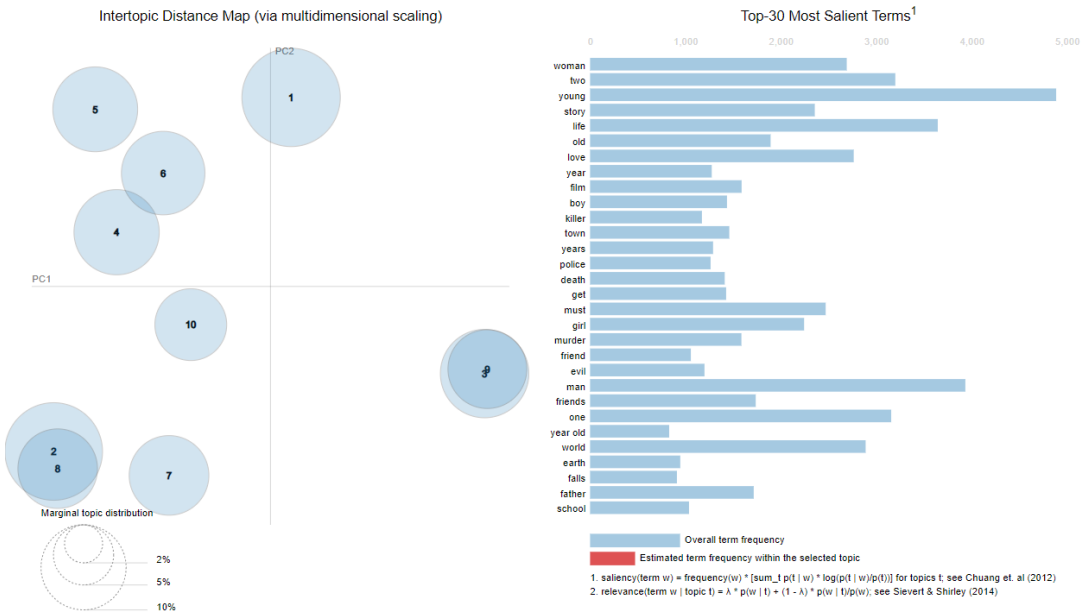


Figure 4: LDAvis with 10 topics

can be found looking at *LDAvis* in Figure 4. First, there are two pairs of genres which are highly overlapped. Second, *LDAvis Topics 3* and *9*, which appear to be the most different from the others, contain a lot of words related to *crime*, *thriller* and *horror* such as: *murder, killer, police, detective, crime, serial, case, mysterious, town, investigate, murdered, wife, serial killer, mystery, murders, cop, small, officer, death, solve*. Important note: *LDAvis* and *Gensim* LDA topics are not the same.

4.2 Text Classification

Dense Neural Networks

Here are the results of the random search. The best model we have found has: *learning rate* = 0.0001; *batch size* = 32; *dropout* = 0.5; *L2 kernel regularization* = 0.0005; *layer sizes* = (64, 32, 32). We observed, during the random search, that many times the net seems to be stuck in a local minimum, the loss function does not show any improvement from 2.3026, and the accuracy is just 10%. This is probably due to the fact that the gradient vanishes with such a significant number of zeros in the input. For this reason, we set an early stopping criteria which monitors the loss function, saving us some computational time. Moreover, we added an adaptive learning rate function that cut the learning rate to one fifth of the previous value if the loss function did not make any progress. Since the input layer of the net is large, we experimented with some "extreme" first hidden layer sizes, which provided some bad results in terms of overfitting.

In Figure 5, we can see our best model ('Model 1') compared to a similar overfitting model with (256, 128, 128) layers sizes, 0.4 dropout and a 10 times higher step size ('Model 2'). Model 1 led to a 32.2037% test accuracy. The latter stops after just 7 epochs, due to loss function decrease. Despite the high discrepancy between validation and training set, overall accuracy is

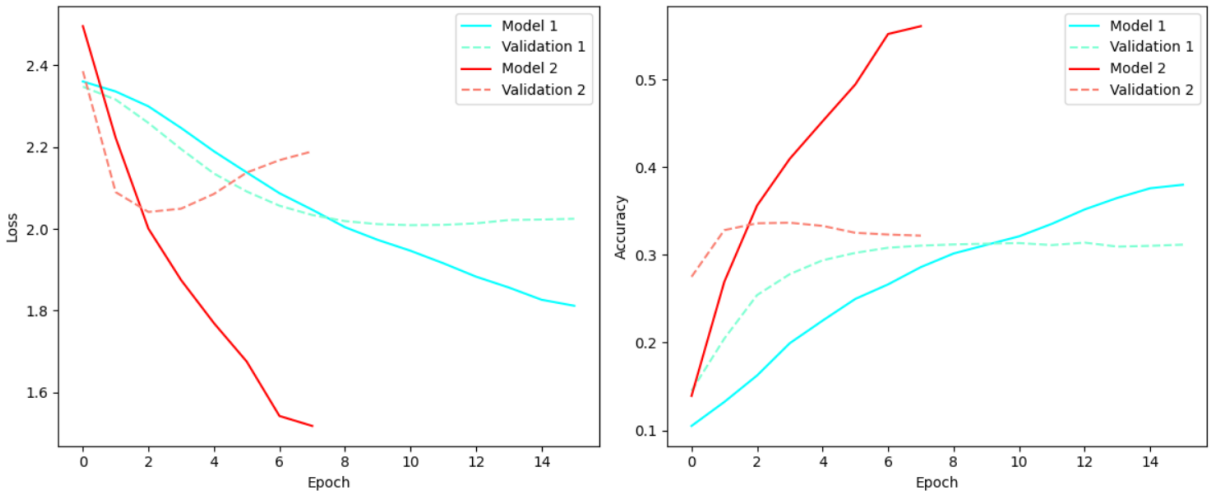
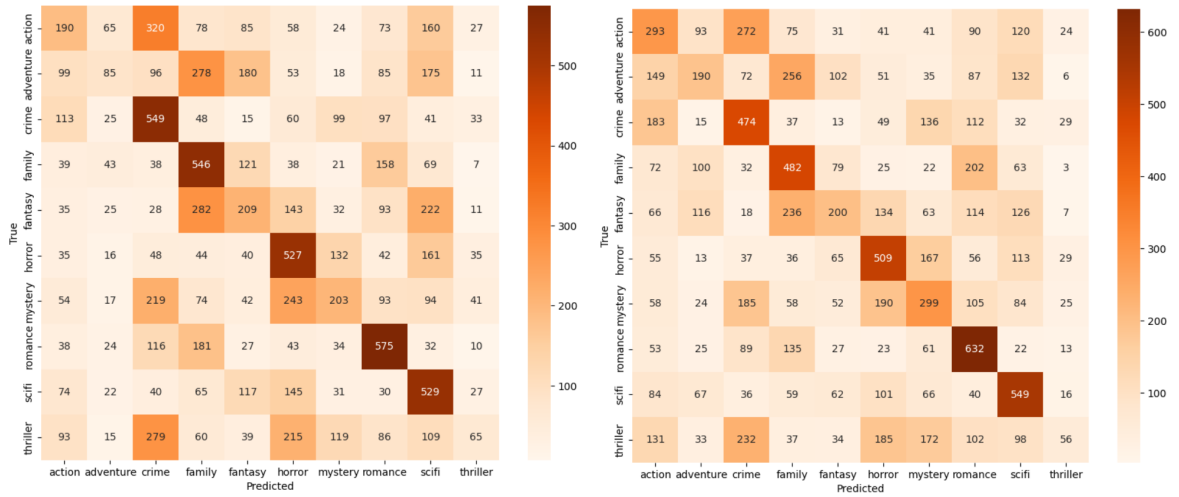


Figure 5: Two DNN models from random search. Model 1 is our best model, while Model 2 is strongly overfitted.



((a)) Model 1 **((b))** Model 2 (overfitting)

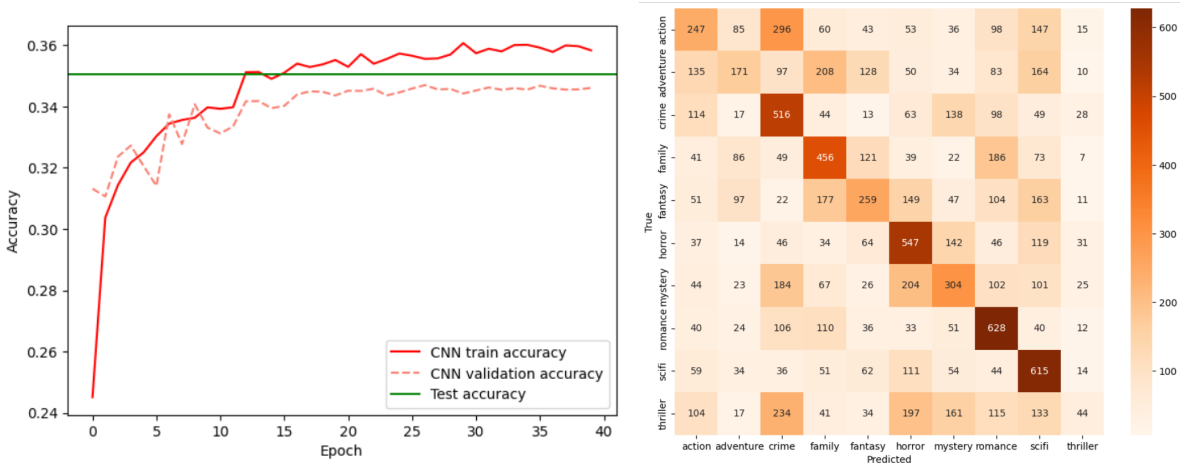
Figure 6: Confusion matrices for Model 1 and Model 2 of DNN

not bad at all on the actual test set, around 34%. (see Figure 6).

As we can see, Model 1 provides robust results for some classes (*romance* has 34% precision and 53% recall). Model 2, on the other hand, provides a better overall accuracy but results are more balanced between classes. Notice that *adventure*, which is a difficult class to detect in general, is correctly identified by model 2 more than twice. This suggests that a more complex model is capable of understanding more complex patterns in the synopses. One important aspect to note is that the dense neural networks take a lot of time to perform just 20 epochs compared to CNNs. The time spent per epoch depends a lot on the net size, but it is estimated between 25 and 30 seconds/epoch for our best model. The risk of overfitting is much higher in this case with respect to other models like CNNs (see following sections). With a slight change in parameters, accuracy can vary enormously. While the accuracy value is still acceptable, the running time and memory usage of such a model prompted us to change model and preprocessing approach.

Convolutional Neural Networks

Our best result is a model with: learning rate = 0.001; batch_size = 32; dropout = 0.3; kernel regularization = 0.005; kernel size = 3; filters = 32; NN layers size = (256, 128).



((a)) Accuracy **((b))** Confusion matrix

Figure 7: CNN model performances

With such a model (Figure 7), we effectively prevented overfitting. The gap between validation and training set is not large, and we also have similar performances on the test set (35.4537%). Even though the perfomance is not too high, in absolute terms, we still have a considerable improvement with respect to the trivial classifier and the dense neural networks. In addition, computational time is way lower than the dense counterpart (just 4 minutes for 40 epochs), as is memory usage. An interesting result is that, even though we tried low-sized layers for the dense net like in the previous chapter, it appears that best results are achieved with large nets.

XGBoost

The best run was made with: *reg_lambda* = 1.5; *max_depth* = 6; *learning_rate* = 0.15; *gamma* = 0.3. With this setting, we obtained an overall accuracy of 27%. The worst accuracy was reached for the *thriller* genre with 11% and the best accuracy was obtained for *scifi* with 34%, but only looking at the confusion matrix in Figure 8 it is evident the types of errors that this classifier made. In fact, classes are mostly confused in the following way (citing only the top-3 errors for each class): *action* with *crime*, *thriller* and *adventure*; *adventure* with *fantasy*, *family* and *action*; *crime* with *action*, *thriller* and *mystery*; *family* with *adventure*, *fantasy* and *romance*; *horror* with *mystery*, *thriller*, *scifi*; *mystery* with *horror*, *thriller* and *crime*; *romance* with *family*, *fantasy* and *thriller*; *scifi* with *fantasy*, *adventure*, *action*; *thriller* with *mistery*, *horror* and *crime*.

As seen with dense neural networks, we notice a strong overlap among some sub-groups of classes and also how the identities of the genres (in order) *thriller, adventure, action* are barely recognized and are spread among all other categories.

LSTM

Our best result is a model with: LSTM layer with 100 units, dropout = 0.4, and recurrent dropout = 0.2; learning rate = 0.001; batch size = 16.

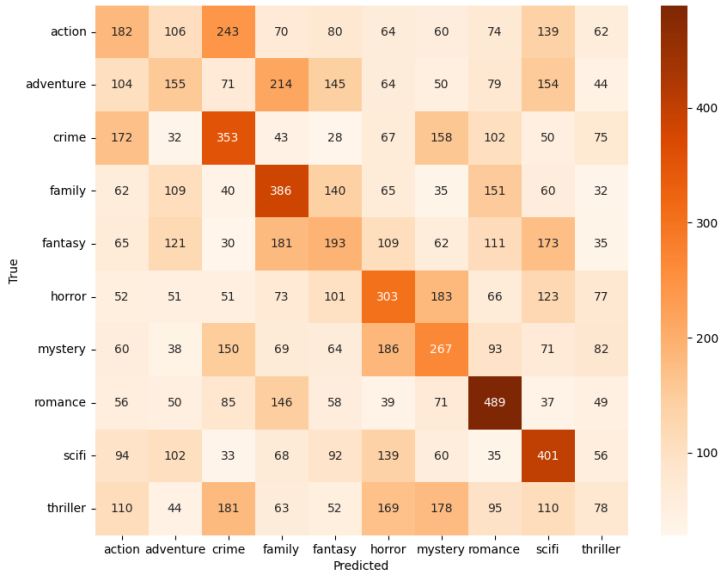


Figure 8: XGB confusion matrix: some strong overlaps occur.

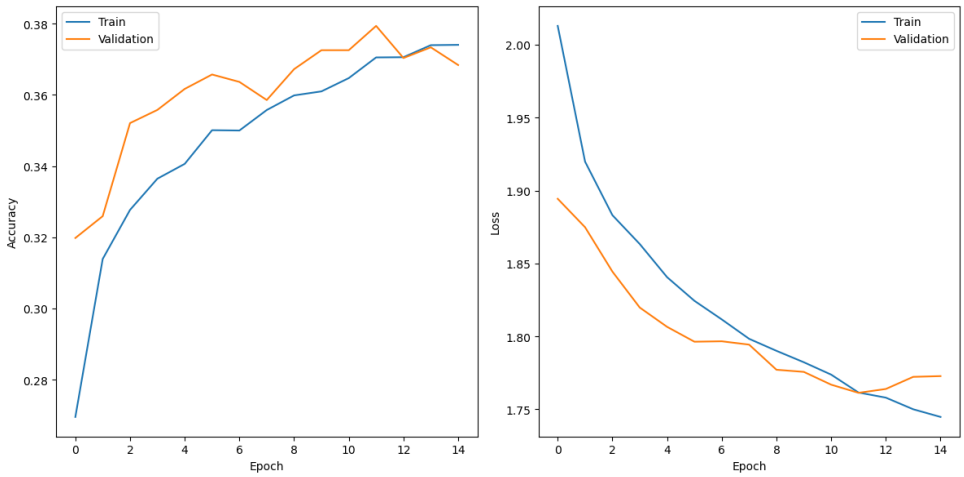


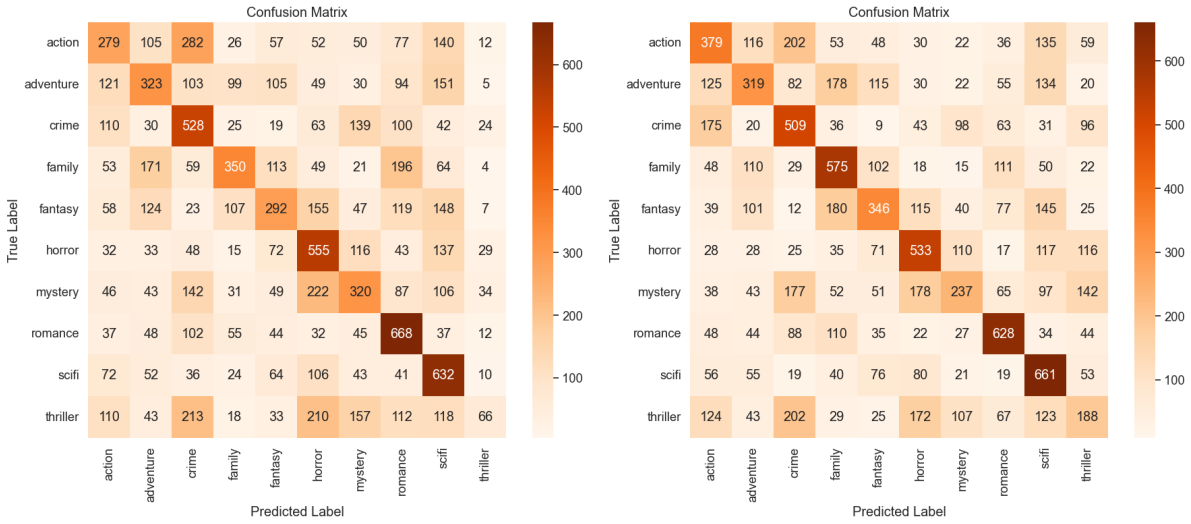
Figure 9: LSTM accuracy and loss

In Figure 9, we can see the trend of accuracy and loss according to epochs, with both curves shown for the training and the validation sets. From these results, we can see that after 11 epochs we have a slight overfitting. Therefore, we retrained the network on the combined training and validation sets, and we tested on the test set, obtaining an accuracy of 37%. The results are shown in Figure 10.

As was also the case with the other models, the class predicted least often was the genre *thriller*, which was confused most often with *crime* and *horror*. The classes predicted most often are *sci-fi* and *romance*. The cell with the most errors (282 errors) is related to *action* movies when the model predicts *crime*.

BERT

During the training with the hyperparameters in Section BERT, we noticed that the loss zigzagged a lot, so we decided to lower the learning rate from $2e-5$ to $1e-5$, not solving the problem entirely, but going from 40% validation accuracy to 41%. Another difference due to

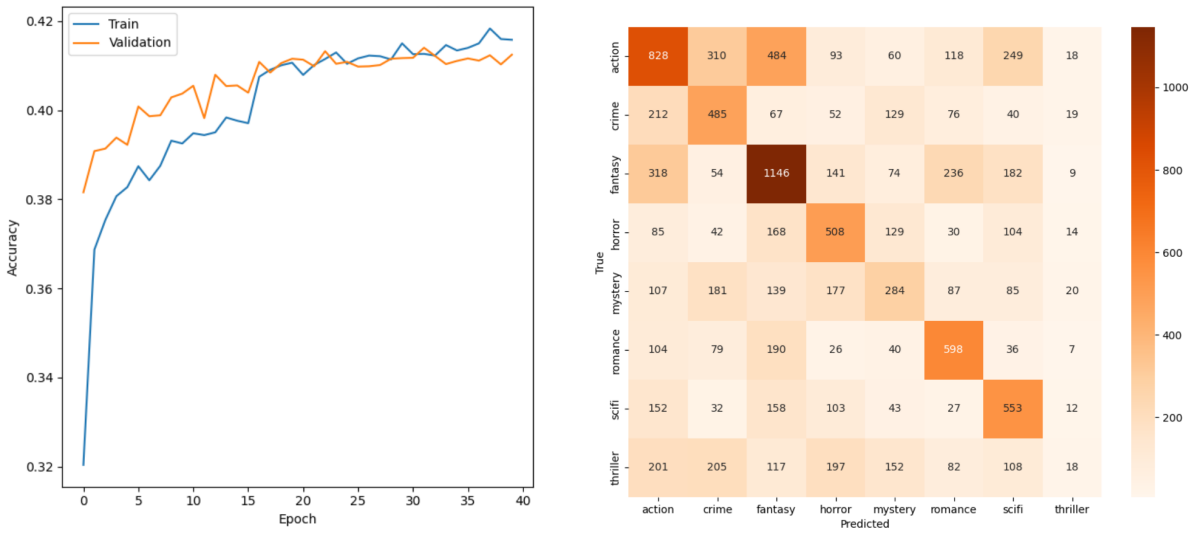


((a)) LSTM ((b)) BERT

Figure 10: LSTM and BERT Confusion matrices

the change of the learning rate was the optimal number of epochs: with a higher learning rate, we achieved the best validation accuracy after only one epoch, rather than after two epochs with a lower learning rate. So we tested the model on the test set, without retraining it on train + validation for complexity reasons, and achieved an accuracy of 41% with the confusion matrix in Figure 10.

As with the other models, the most predicted genres were *sci-fi* and *romance*, while the least predicted was *thriller*, though this time *thriller* was correctly classified 188 times, better than all the other models.



((a)) Accuracy ((b)) Confusion matrix

Figure 11: CNN model performances with 8 classes

8 classes problem

Another attempt that we thought deserved to be made is an 8 classes classification problem. We have discovered that some genres can be ambiguous: they have a lot of tokens in common. The idea is to take two pairs of genres and merge them together. Taking advantage of what we found in Section 4.1 and the confusion matrices, we tried some combinations. We did not want to spend too much computational time on this, so we used the best CNN from Figure 7 to test our new classes, since it was the fastest model. Here we report the most interesting result which comes from merging *adventure* with *action* and *family* with *fantasy*. We are aware that now our problem is no more perfectly balanced as before.
Thriller somehow vanishes in this case, but we observe an increase mean accuracy and recall for all the other genres. Even though classes are now less than before, we now observe a not so good performance for *Mistery*. There is still some confusion between the 2 new classes, but overall test accuracy is 40.9259%.

Summary of results

In Table 1, the summary of evaluation metrics is shown for CNN, XGBoost, and BERT.

	CNN			XGB			BERT		
Class	Precision	Recall	F-score	Precision	Recall	F-score	Precision	Recall	F-score
Action	0.28	0.23	0.25	0.20	0.19	0.19	0.36	0.35	0.35
Adventure	0.30	0.16	0.21	0.20	0.15	0.17	0.36	0.30	0.33
Crime	0.33	0.48	0.39	0.28	0.34	0.31	0.38	0.47	0.42
Family	0.37	0.42	0.39	0.28	0.33	0.31	0.45	0.53	0.49
Fantasy	0.33	0.24	0.28	0.21	0.17	0.19	0.39	0.32	0.35
Horror	0.38	0.51	0.43	0.27	0.31	0.29	0.44	0.49	0.46
Mistery	0.31	0.28	0.29	0.24	0.25	0.24	0.34	0.22	0.27
Romance	0.42	0.58	0.49	0.37	0.45	0.41	0.55	0.58	0.57
Scifi	0.38	0.57	0.46	0.32	0.37	0.34	0.43	0.61	0.51
Thriller	0.22	0.04	0.07	0.17	0.09	0.11	0.25	0.17	0.20
accuracy			0.35			0.27			0.41
macro avg	0.35	0.35	0.33	0.25	0.27	0.26	0.39	0.41	0.39
weighted avg	0.33	0.35	0.33	0.25	0.27	0.26	0.39	0.41	0.39

Table 1: Evaluation metrics for CNN, XGBoost, and BERT.

We find that the best performing model is BERT with an overall accuracy score of 41 percent. We also conclude that the most difficult class to be recognised is *Thriller*, and we can observe this pattern for all the models in 1. On the other hand, *Romance* seems to be the most recognisable one. This fact is surprising for us, since it's the class with less peculiar words in its lexicon. *Crime* and *Action* often are confused with each other, even with 8 classes, and it is probably due to the fact that they share a good number of tokens; also LDA finds this pattern.

5 Discussion

NLP and text classification have proven to be challenging tasks, due to the extreme analogic nature of the raw unstructured data. This difficulty is reflected in our model performances, such that classes are not precisely recognized due to common patterns across synopses in different genres, or simply because these descriptions are written by humans, each one with their own background and bias. Perhaps having more diverse data, possibly non-textual data, could help our models be more precise. This may partially explain the performances achieved by our models, but our accuracy results really are not too bad, considering that they improve with respect to

the trivial classifier that can achieve an accuracy level of n_classes/100 at most by construction. What we find very interesting are the classes that tend to be confused with each other, and this represents the hardest challenge, since it stems from the shared words and characteristics among these genres. However, we are also aware that humans too may have difficulty distinguishing genres, so it is very logical that a machine would do the same. We also note that the preprocessing phase plays a central role in all subsequent steps. We experienced some difficulties in this phase, and this raises the need to consider this fundamental step carefully.

6 Conclusions

In this project, we applied topic modeling and text classification methods in a multiclass single-label classification task to classify movie genres based on titles and descriptions. We believe that we achieved our learning goals of learning about many different text classification techniques as well as LDA. In hindsight, it may have been useful to test also some more traditional algorithms such as SVM or Naive Bayes for comparisons with the neural network approaches, but we are satisfied with the exposure to many different models that this exploration afforded us.

References

[1] S. Bergamaschi and L. Po. Comparing lda and lsa topic models for content-based movie recommendation systems. In *Web Information Systems and Technologies: 10th International Conference, WEBIST 2014, Barcelona, Spain, April 3-5, 2014, Revised Selected Papers 10*, pages 247–263. Springer, 2015.

[2] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[3] D. R. Cox and E. J. Snell. *Analysis of binary data*, volume 32. CRC press, 1989.

[4] A. M. Ertugrul and P. Karagoz. Movie genre classification from plot summaries using bidirectional lstm. In *2018 IEEE 12th International Conference on Semantic Computing (ICSC)*, pages 248–251. IEEE, 2018.

[5] N. Fei and Y. Zhang. Movie genre classification using tf-idf and svm. In *Proceedings of the 2019 7th international conference on information technology: IoT and smart city*, pages 131–136, 2019.

[6] S. González-Carvajal and E. C. Garrido-Merchán. Comparing bert against traditional machine learning text classification. *arXiv preprint arXiv:2005.13012*, 2020.

[7] S. Jiang, G. Pang, M. Wu, and L. Kuang. An improved k-nearest-neighbor algorithm for text categorization. *Expert Systems with Applications*, 39(1):1503–1509, 2012.

[8] S.-B. Kim, K.-S. Han, H.-C. Rim, and S. H. Myaeng. Some effective techniques for naive bayes text classification. *IEEE transactions on knowledge and data engineering*, 18(11):1457–1466, 2006.

[9] M. Kordabadi, A. Nazari, and M. Mansoorizadeh. A movie recommender system based on topic modeling using machine learning methods. *International Journal of Web Research*, 5(2):19–28, 2022.

[10] K. Kowsari, D. E. Brown, M. Heidarysafa, K. J. Meimandi, M. S. Gerber, and L. E. Barnes. Hdltex: Hierarchical deep learning for text classification. In *2017 16th IEEE international conference on machine learning and applications (ICMLA)*, pages 364–371. IEEE, 2017.

[11] K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown. Text classification algorithms: A survey. *Information*, 10(4):150, 2019.

[12] S. Kumar, N. Kumar, A. Dev, and S. Naorem. Movie genre classification using binary relevance, label powerset, and machine learning classifiers. *Multimedia Tools and Applications*, 82(1):945–968, 2023.

[13] S. Lai, L. Xu, K. Liu, and J. Zhao. Recurrent convolutional neural networks for text classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29, 2015.

[14] D. P. Mandic and J. Chambers. *Recurrent neural networks for prediction: learning algorithms, architectures and stability*. John Wiley & Sons, Inc., 2001.

[15] R. B. Mangolin, R. M. Pereira, A. S. Britto Jr, C. N. Silla Jr, V. D. Feltrim, D. Bertolini, and Y. M. Costa. A multimodal approach for multi-label movie genre classification. *Multimedia Tools and Applications*, 81(14):19071–19096, 2022.

[16] P. Matthews and K. Glitre. Genre analysis of movies using a topic model of plot summaries. *Journal of the Association for Information Science and Technology*, 72(12):1511–1527, 2021.

[17] V. Vapnik and A. Y. Chervonenkis. A class of algorithms for pattern recognition learning. *Avtomat. i Telemekh*, 25(6):937–945, 1964.

[18] I. Vayansky and S. A. Kumar. A review of topic modeling methods. *Information Systems*, 94:101582, 2020.

[19] B. Xu, X. Guo, Y. Ye, and J. Cheng. An improved random forest classifier for text categorization. *J. Comput.*, 7(12):2913–2920, 2012.