Text Analytics Project Report

Chiara Buongiovanni
c.buongiovanni@studenti.unipi.it
507164

Andrea Mattei
a.mattei3@studenti.unipi.it
546139

Academic Year 2021/2022

Contents

1 Introduction

The goal of this project is to investigate the characteristics of a corpus comprised of Italian narrative texts, published on the web and belonging to the *fanfiction* genre, through the use of the Text Analytics techniques learned during the course.

Fanfictions are stories written by fans of a TV series, movie, book etc., using existing characters and situations to develop new plots. Dedicated websites in many languages exist where users can publish their own pieces of literary work inspired to the original book they are fans of. Our corpus has been extracted through Web Scraping[1] from EFPfanfict.net, a portal active since 2001 which allows users to publish both *fanfictions* and original stories, and to comment on them.

This corpus has been put together in 2019 for a bachelor's thesis, where the goal was to transform the texts into vectors of multi-level linguistic features through the *linguistic profiling* approach (Montemagni, 2013; van Halteren, 2004), in order to detect the ones which significantly discriminate successful from unsuccessful stories with the target audience, taking as a metric the number of positive reviews received. For this reason, the collection was limited to stories based on a single work, the fantasy saga "Harry Potter" by British writer J.K. Rowling, which allowed for the biggest monothematic sample of texts to analyze. Moreover, only fanfictions comprised of just one chapter written before 2018 were selected, in order to eliminate popularity discrepancies when dealing with long-running or very recent stories. The resulting corpus comprises over 26.000 stories.

In that first study, the results of which are detailed in (Mattei, Brunato, dell'Orletta, 2020) [3], the correlation between writing style and success has been investigated. In this project we want instead to consider the full content of the texts at hand, to carry out both supervised and unsupervised machine learning tasks. In particular, this report will present:

- Classification on the Rating of the stories, with language models as well as with simpler learning algorithms (section 2 and 3);
- Classification on the Popularity of the stories (section 4);
- Creation of a network of similar authors with Word2Vec (section 5);
- Sentiment Analysis at the sentence level and clustering with respect to the sentiment structure of the stories (section 6).

[1] The spiders for this task are public and avaiable at https://github.com/AndreMatte97/Fanfiction

2 Rating Classification - Simple Models

2.1 Introduction

Stories from the source website come with a label assigned by their authors (and presumably checked by a moderation staff), representing the presence of explicit language and mature themes. The four possible values are color-coded as *green*, *yellow*, *orange* and *red*. Their distribution in our dataset is as follows:

- *green* - 26104 (69.2% of the total)
- *yellow* - 6927 (18.4%)
- *orange* - 2623 (7.0%)
- *red* - 2039 (5.4%)

The goal of this task is to train a classifier to automatically recognize the appropriate target audience for this kind of narrative texts. In the following sections we will show the various versions of the task carried out, as well as their results.

2.2 Preprocessing and GridSearch

For each version, we split the data into train and test set (70-30 proportion). Then, a pre-trained spaCy models, `it_core_news_sm` for Italian text, is used for preprocessing tasks such as tokenization, lemmatization, stopwords elimination on both sets. Finally, we use CountVectorizer from ScikitLearn's package to generate a vocabulary and term frequencies.

We set up a pipeline object that:

- selects the k best features from the data, using the SelectKBest function, taking chi2 as score, where k is a hyperparameter that we will tune during the fitting process;
- weights the words with TfidfTransformer;
- uses an estimator to perform classification, testing three different kinds of learning algorithm.

Then we define the space of hyperparameters and estimators we want to search through. We try out different values of k for the feature selector SelectKBest, as well as different hyperparameter values for each of our three estimators as shown below:

- k: 10000, 100000, 250000, 500000, 'all';
- MultinomialNB, searching over
  - `alpha`: 1e-3, 1e-2, 0.1, 1.0, 10.0;
  - `fit_prior`: True, False;
- LinearSVC:
  - `C`: 0.01, 0.1, 1, 10, 100;
- LogisticRegression, using `liblinear` as solver
  - `C`: 0.01, 0.1, 1, 10, 100;
  - `penalty`: l1 and l2.

2.3 Rating *green* vs *non-green*

For the first execution of this task, we group together all the stories labeled as *yellow*, *orange* and *red* in a single class, named simply *red*. The motivation behind this decision is twofold: first of all, we imagine a possible real-world application for this kind of classification, useful for detecting which stories are completely safe for children. Additionally, we notice that the website's guidelines for labeling stories as *yellow* or *orange* are quite ambiguous and prone to misinterpretation. Doing so, we end up with a distribution of 69.2% *green* stories and 30.8% "*non-green*" stories: the resulting dataset is therefore quite balanced and as such we feel there is no need to apply sampling techniques beforehand.

For this particular case, the GridSearchCV has selected the MultinomialNB model, with `alpha`=3 and `fit_prior`=False, along with the k=250000 best features chosen by SelectKBest.

Table 1 shows the results obtained. The model seems to recognize the *green* stories quite well (f1 score = 0.8), however it has some difficulties in identifying the *red* ones (f1-score = 0.59), as expected due to the class imbalance.

Rating Classification	Precision	Recall	F1
green	0,82	0,81	0,81
non-green	0,58	0,6	0,59
Accuracy	0,75		

Table 1: Rating Classification (MultinomialNB) grouping *green* vs + *yellow* + *orange* + *red*

Even though the model inspection shows that our Multinomial Naive Bayes classifier is able to correctly identify which features are typical of mature texts, especially if dealing with sexual themes, the performances on the *non-green* class are lower than expected. We can try to explain this behaviour with the ambiguous definition for what constitutes a *yellow* or *orange* fiction: in fact, manually looking at some of them from the former class, we can find a lot of very "safe" stories. We therefore wonder if *yellow* fictions could be diluting the supposedly mature class, hampering the detection of more risque or violent content. For this reason, in the next iteration of this task we group the *yellow* stories with the *green* ones, leaving the *orange* texts with the *red* ones.

2.4 Rating *green* + *yellow* vs *orange* + *red*

By combining the data as explained above, we obtain two very unbalanced classes: 33031 *green* and 4662 *red* stories. In order to address this imbalance, we try out various techniques: in detail,

- Increasing the weight of the minority class;
- Undersampling the majority class, obtaining 1:1 and 2:1 ratios with the minority class;
- Oversampling the minority class, obtaining 1:1 and 1:2 ratios with the majority class.

Table 2 shows the performances of the best estimators identified by GridSearchCV, applied on each rebalancing technique.

The attempt conducted on the unbalanced (Normal) dataset shows an acceptable precision score on the *red* class, flanked by an insufficient recall score. The results obtained by modifying the weight of the classes lead to more balanced precision and recall scores and to the highest F1-score. We started making the minority class weight 9 times the majority class, since their sample size ratio was around 1:9. We then tried changing the ratio, obtaining the best results when setting the minority weight to 20 times the majority one. The results obtained with the resampling techniques are instead more disappointing: undersampling improves Recall while significantly sacrificing Precision, while the inverse happens with oversampling, leading to a significant lower F1-score in both cases.

Rating Classification	*green + yellow*				*orange + red*				
	Precision	Recall	F1	Support	Precision	Recall	F1	Support	Accuracy
Normal	0,93	0,98	0,95		0,74	0,48	0,58		0,91
Class_weight 9:1	0,94	0,96	0,95	23122	0,67	0,58	0,62	3263	0,91
Class_weight 20:1	**0,95**	**0,94**	**0,94**		**0,6**	**0,64**	**0,62**		**0,9**
Over-sampled 1:1	0,92	0,99	0,95	23122	0,84	0,38	0,52	23122	0,91
Over-sampled 2:1	0,92	0,98	0,95	23122	0,73	0,42	0,53	11561	0,91
Under-sampled 1:1	0,96	0,82	0,88	3263	0,38	0,79	0,51	3263	0,81
Under-sampled 2:1	0,95	0,92	0,94	6526	0,55	0,65	0,6	3263	0,89

Table 2: Performances of best estimators found by GridSearchCV on various optimization methods of our imbalanced learning problem: the Rating Classification grouping *green + yellow* vs *orange + red*. Support columns refers to the number of actual occurrences of the class in the training set.

To select the best model, we imagined a real situation in which the task is to discriminate stories suitable for a young audience from those with more controversial content, aimed at an adult audience. From this point of view, the possibility that a child could read a red story seemed much less desirable to us than flagging a story that was actually "harmless", which in the worst scenario would lead to a complaint from the author an the need for a manual review. Based on these considerations, we selected the model with the highest recall score on the *red* class (among those with an acceptable F1-score to not sacrifice precision too much), i.e. a Logistic Regression model, setting class weights to 20:1.

2.5 Multiclass

Next, we want to test the ability of a model to classify documents keeping all available labels separated. After finding an optimal configuration using GridSearchCV, we performed another search, this time randomized, on hyperparameter values lying within a certain range from the ones discovered in the first phase, thus selecting a Logistic Regression with C equal to 90, *l2* as penalty and taking as input all the features available.

Rating Classification - multiclass				
Class	Precision	Recall	F1	Support
green	0,77	0,96	0,85	7831
yellow	0,41	0,17	0,24	2078
orange	0,36	0,05	0,09	787
red	0,79	0,63	0,7	612
Accuracy		0,74		

Table 3: Rating Classifier (LogisticRegression) - Multiclass version

	Green	Yellow	Orange	Red
Green	7555	250	17	9
Yellow	1672	355	34	17
Orange	475	194	40	78
Red	139	67	20	386

Table 4: Confusion Matrix for Multiclass Rating Classifier

The performances (shown in table 3) clearly show that the model is very robust on the *green* and *red* classes, while its performance falls on the other two labels. Analyzing the confusion matrix obtained from the classification of the test set (shown in **??**, it appears that most of the *yellow* (1672 out of 2078) and *orange* (475 out of 787) stories are classified as *green*.The other *orange* stories are classified as *yellow* (194) or *red* (78), yielding the lowest number of true positives (only 40).

The cause of this debacle could be identified in the scarce presence of these categories in the dataset: this hypothesis, however, is not confirmed if we look at the satisfactory performances obtained on the *red* class, the least represented of all. We therefore conclude that the yellow and orange categories are the worst characterized.

2.6 Binary - *green* vs *red*

Encouraged by the performance obtained on the *red* and *green* classes, we want to test the model's performance by totally excluding the *yellow* and *orange* stories from the analysis.

In table 5, the performances shown in the first row (Normal) refer to the results obtained on the unbalanced dataset, using a simple LinearSVC with default parameters, using all the available features. The predictions of the model

turn out to be very precise on both classes. However, since we would to increase the Recall on the red class and the classes are again unbalanced, we use the class weight optimization method, the best method among those tested in 2.4, this time finding the best results when bringing the weight of the minority class in a ratio of 2:1 with the majority one. The best model found is a Logistic Regression classifier, with C equal to 100, l1 as penalty, taking all features available as input, thanks to which we obtain a balance between precision (-0.1) and recall (+0.1) of the *red* class, reaching a +0.03 F1-score with respect to the base classifier.

2.7 Conclusions

As expected, the red stories are predicted quite well, despite the low number of examples in the total dataset, which unfortunately does not allow to reach optimal recall values. However, this metric can be increased up to 0.79, even if at the expense of some points in precision. From the inspection of the typical features of red stories, we see that the models have no difficulty in identifying stories with sexual or violent content. This leads us to think that the model does not recognize less explicit stories from the lexical point of view but not from the content, developing the typical themes of "red" stories using metaphors and similes. Finally, to investigate the reasons behind the choices of the model, we searched in the dataset for fan-fictions classified as *green* but predicted as *red*: in several cases, we agreed that it was the label assigned to the fan-fiction by the author that was inappropriate, while the one assigned by the model correct. We concluded that also within the training set there could have been similar errors, which will have somewhat confused the model.

3 Rating Classification - Language Models

Next, we would like to train a Language Model-based classifier, using `Bert-base-italian-xxl-cased`[2] as our baseline pre-trained model.

Initially we wanted to repeat most of the versions of this task that have been explored and reported in the previous section. However, the only way we have available for running this type of algorithms is to rely on the GPUs offered by Google Colab which, while quite powerful, still have some troubles working with this amount of data: most of the times, in fact, they would reach the maximum memory capacity, even though we took some precautions to lighten the computational load. Moreover, nearly every time we got the `out of memory` error, Colab forbade us from the service for the next dozen hours, making the overall process quite long even from a real-world time perspective. Therefore, we choose to focus on the version where we discard *yellow* and *green* stories beforehand, which not only proved to be the one with the best results using simpler classifiers, but also allows us not to fill the memory completely, even if just by a smidge.

For the structure of the code we start from the examples seen during the lectures, yet we need to make some changes in order to meet our needs. In particular, first of all we add the possibility of printing not just the accuracy, but the full confusion matrix, with precision, recall and f1 scores at the end of each training epoch: since our classes are very unbalanced, accuracy alone is not very informative. For the same reason, it is also inadequate to be used as a loss metric for training the model: we therefore replace it with the Cross Entropy Loss, which also gives us the possibility of changing class weights. Finally, we allow ourselves to monitor the loss

[2]https://huggingface.co/dbmdz/bert-base-italian-xxl-cased

Rating Classification	*green*			*red*			
	Precision	Recall	F1	Precision	Recall	F1	Accuracy
Normal	0.97	1.00	0.99	0.97	0.65	0.78	0.97
Class_weight 2:1	0.98	0.99	0.99	0.87	0.75	0.81	0.97

Table 5: Results of Logistic Regression classifier on *green* vs *red* classification

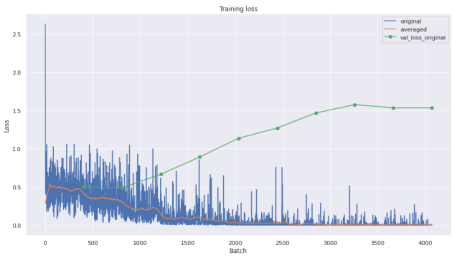


Figure 1: Training (blue and orange) and Validation Loss (green) for our first BERT Rating Classification Model (10 epochs)

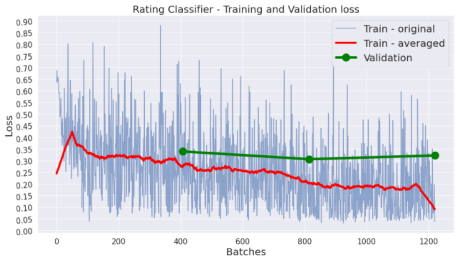


Figure 2: Final training session for our BERT Rating Classification Model (3 epochs)

value at each epoch not only on the training set, but on the validation set as well.

We then separate the data into train and test set, again with a 70-30 split, and create a validation set taking 10% of the train set. In the training phase we tokenize only train and validation, postponing the test set preprocessing for later, in order not to load useless data on our limited GPU memory. The model is passed only 128 tokens, for computational reasons as well: these tokens are taken from from the middle of the stories, since often they open with messages external to the actual narration, like greetings and introductions from the authors.

We then start training our model for 10 epochs, to get an idea of how the training is performed. The validation and train loss trends are shown in figure 1: we can see how the validation loss (in green) starts to increase very early, signaling that some overfitting is happening.

The training is therefore repeated with less epochs (5), modifying the class weights, first setting them with a 2:1 ration in favour of the minority class (best configuration found with the best "simple" model), then with 12:1 (the inverse of the actual label distribution), keeping track of the results on the validation set at each epoch, and comparing them with those obtained during the first training experiment. We notice that the best performances are obtained again with the 2:1 ratio. With 5 epochs, however, there still is a noticeable increase in validation loss at the end: we therefore decide to retrain the model stopping it after 3 epochs, where the validation loss starts going up. Figure 2 shows the validation and training loss trend in this last training session.

	Precision	Recall	F1
green	0.95	0.98	0.97
red	0.72	0.47	0.57
		Accuracy	0.94

Table 6: Results for our BERT Rating Classification Model on test set

The model has then been saved, loaded in another another notebook where we preprocess the test set in the same way we did for the train, and use it to evaluate our classifier. Results are shown in table 6. The performances are quite worse with respect to what we managed to achieve with the Logistic Regression classifier shown at the end of the previous section, and this could be due to a variety of factors: first of all, in general neural networks have shown to have some difficulties in handling unbalanced data with respect to other, simpler classifier; secondly, analyzing only 128 tokens per story, the model is bound to misclassify ones where mature or explicit content lies elsewhere in the text; finally, with more time and resources (namely a GPU with more memory), it could have been more feasible to adequately fine-tune the network structure and the learning parameters.

4 Success Classification

Each story in our dataset comes labeled with the number of received reviews, which can be used as a metric to model its level of success / popularity. The review count is kept separated for positive, neutral and negative feedback, even though the vast majority of the them are favorable (over 99%). The goal of the task described in this section is to train a classifier for detecting whether or not a fanfiction has been or will be popular with the target audience.

First of all, we need to create a definition for "popular" and "unpopular" stories, creating a

categorical target variable selecting thresholds the review count. Considering that the average number of reviews for a story is around 5, and in order to get balanced classes, we label our texts in the following way:

- Unpopular: 2 or less reviews (13.603 stories)
- Popular: 6 or more reviews (11.644 stories)
- Medium: between 3 and 5 reviews (12.446 stories)

The hyperparameter tuning phase for the non-deep classifiers is performed exploring the same space already described in section 2.2. The metric chosen is the F1 score on the Popular class.

4.1 Simple Models - Multiclass

The best model found by the GridSearch for the three-class classification is a MultinomialNB with alpha equal to 0.001, taking as input the 10.000 best features. Its results, shown in table 7, are however far from satisfactory, with just the recall on the Unpopular class going above 0.5. We assume that the presence of stories with middling success, which might present characteristics from both popular and unpopular fictions, could be leading astray the model, as hinted by the abysmal performances on the Medium class. For this reason, we repeat the task eliminating from the data all texts belonging to such class, making it binary.

4.2 Simple Models - Binary

Dropping stories from the Medium class allows the classifier to reach much better performances, as shown in table 8. The best model found by the Gridsearch (number (1) in the table) is again a MultinomialNB with alpha equal to 0.001, this time taking as input the top 500.000 features. Doing some testing with other types of learning algorithms, we find that LinearSVC in general trades off some points in Recall for the Popular class, while gaining around the same in Precision but being much better with respect to the Recall on unsuccessful texts. Due to natural oscillations in performances when going from cross validation to testing, in our particular case a LinearSVC taking all the features as input (2) beats the model chosen by the Gridsearch in F1 score for successful stories, as well.

Trying to inspect the most important features for these two models, we find them to be of difficult interpretation, in particular for the MultinomialNB, where most are bi- and trilemmas that do not seem to belong to a particular semantic sphere. LinearSVC identifies as important nearly just single lemmas instead: crafting an explanation out of them remains nonetheless quite hard. In the words contributing the most to a Popular classification we noticed the presence of many terms that are usually not part of a fantasy fiction, but are more likely part of messages from the writers to their readers, which are often present at the beginning or the end of a story. Examples are: *fic*, *commentare*, *dedicare*, *shot*, *recensione*, *contest*, *drabble*, *Ringrazio*, etc. This could signify that the presence of such *writer's corners* might really be useful in making the readers feel more engaged, and in turn increasing the chances of getting reviewed. There might also be some kind of network dynamics involved in the review process, where readers are more likely to leave a comment for stories written by users they are familiar with.

Popularity Classification - Multiclass				
Class	Precision	Recall	F1	Support
Popular	0.46	0.45	0.45	3493
Medium	0.37	0.23	0.28	3743
Unpopular	0.47	0.64	0.54	4081
		Accuracy	0.44	

Table 7: Popularity Classifier (MultinomialNB) - Multiclass version

4.3 Language Models - Binary

As we did for the previous classification task, we also try to solve it with a more sophisticated classifier, making use of language models. The model used is again `Bert-base-italian-xxl-cased`. Due to the great performance discrepancy highlighted between the multiclass and binary version of the task, we choose to focus only on the latter. Additionally, working with just two thirds of the data allows us not to worry about the frequent computational issues we faced when trying to train the Rating neural networks.

Since we are wondering if authors' premises might actually be important for predicting pop-

Figure 3: Training and Validation Loss for our Language Model Popularity classifier.

ularity, this time we decide not to take tokens from the middle of the story, but to start from the beginning instead. We maintain the same parameters and overall network structure as before, obviously without changing weights this time, and launch a first training session 5 epochs long. Looking at the train and validation losses (shown in figure 3), we notice again a tendency to overfit early, signaled by the quick and steep increase in validation loss. For this reason, we choose to stop training at 3 epochs.

Testing such model ((3) in table 8) shows slightly lower performances on the Popular class and much lower ones on the Unpopular class with respect to the LinearSVC classifier. Given that this time the task is not even unbalanced, we feel that there might a bigger margin for improvement here, with more time and resources to fine-tune the model and its parameters.

Out of curiosity, we conclude repeating the training process, this time taking as input 128 tokens from the middle of the stories. Indeed, the results (4) are quite worse than those obtained looking at the incipits.

5 Doc2Vec Embeddings

5.1 Introduction

Since all the stories are labeled with their respective author's nickname, we want to try creating a sort of author clustering, grouping together writers who have produced similar texts. To do so, we train a Doc2Vec model through the Python implementation found in the *gensim*[3] library.

First of all, the documents are passed to the `simple_preprocess` function, which tokenizes and groups them based on their authors. Then this data is passed to the Doc2Vec function, with a vector size of 100 and a window of 10. For the `min_count` we started with 5 and increased it gradually, after inspecting the outcome of the task and noticing that some "outlier" words, mainly english ones that are found in the stories in the form of quotes, errors and non-standard spellings, tended to confuse the model. In the end, we settle for a `min_count` equal to 100, trying to strike a reasonable balance between statistical soundness and information preservation.

5.2 Word Embeddings

Before conducting our analysis on author similarity, we decide to check on the word embeddings, asking the model the most similar words to some peculiar ones from the shared semantic domain of the corpus, i.e. recurring terms in the *Harry Potter* universe. Table 9 shows some examples, along with the similarity scores given by the model. We can see that the similarities found seem to be quite accurate: the embeddings are able to group together names of spells, of characters of different kinds, as well

[3]https://pypi.org/project/gensim/

Popularity Classification - Binary							
	Popular			Unpopular			
	Precision	Recall	F1	Precision	Recall	F1	Accuracy
(1) MultinomialNB	0.62	0.73	0.67	0.73	0.62	0.67	0.67
(2) LinearSVC	0.70	0.66	0.68	0.72	0.75	0.74	0.71
(3) BERT - First tokens	0.68	0.64	0.66	0.61	0.64	0.63	0.64
(4) BERT - Middle tokens	0.65	0.61	0.63	0.57	0.61	0.59	0.61

Table 8: Binary Popularity Classification results

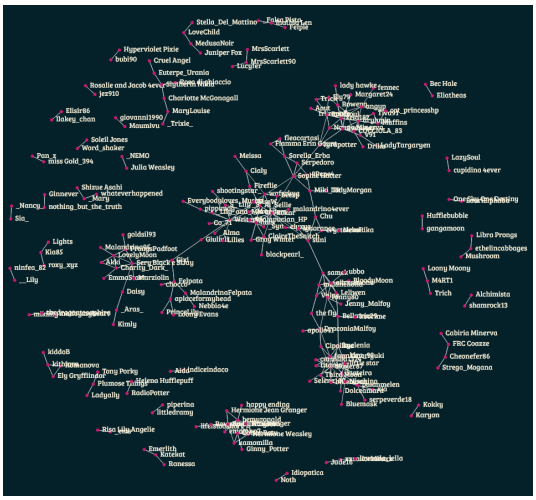


Figure 4: Network of authors by Doc2Vec. $max\_n = 1000$, $min\_sim = 0.72$

as words pertaining concepts like potion making and *Quidditch*.

5.3 Doc Embeddings

Proceeding towards the analysis of the document embeddings, we start ordering the authors by their document count. Setting thresholds `max_n`, for the maximum number of authors to include in our analysis, and `min_sim`, for the minimum similarity to consider for linking two authors from the top `max_n`, we plot several network-like visualizations using the d3.js code by our professor Andrea Esuli[4]. An example is shown in figure 4, produced taking from the top 1000 authors by document count, the pairs with similarity higher than 0.72.

5.3.1 Doc embeddings semantics

Unfortunately, writers' nicknames are not explicative of the content they produced, so the quality of the results is not easily comprehensible at a glance. As a first inspection tool, we try using the `most_similar` function from Word2Vec, passing to it the Doc2Vec vectors of some authors, to check on the words found by the model to be most similar to those documents. As mentioned in the introduction, however, this returns mostly English words, errors and typos, that for the most part seem to not be present neither in the analyzed author's stories, nor in those by the users linked to them. This behaviour keeps occurring even when increasing the `min_count` for the model: to get only Italian words from this method we would need to increase it by several hundreds, at which point too many interesting words would be deleted from the data.

After observing this even when trying on different kinds of documents (e.g. the restaurant reviews data seen during the lectures), and searching with no avail for any built-in explainability methods in the *gensim* library, we decide to implement ourselves a simple method to inspect the similarities found by Doc2Vec. The idea is the following:

- Concatenate stories by their author to create a single document;
- Tokenize and lemmatize them;
- For each author, compute TD-IDF score for each word, in order to get the most representative words for their document;
- Take the top n words (10, in our case) of each document and compare them to the ones of the documents linked to it;
- For each link, print and store the shared representative words, if any.

For ease of interpretation, this method is tested on a smaller network, shown in figure 5, built considering only the authors with similarity greater or equal than 0.80 from the top 3000 by number of stories written.

With this method, we find at least a shared

[4] https://github.com/aesuli/word2vec_exploration

Words									
Expelliarmus		***Harry***		***Sirius***		***Pozione***		***Cercatore***	
Protego	0.86	Ron	0.65	Remus	0.82	Boccetta	0.73	Portiere	0.81
Stupeficium	0.77	Hermione	0.61	Peter	0.79	Fiala	0.72	Capitano	0.76
Totalus	0.75	Ginny	0.58	James	0.76	Bevanda	0.67	Campione	0.74
Sectumsempra	0.73	Hagrid	0.52	Felpato	0.74	Preparazione	0.65	Giocatore	0.72
Petrificus	0.71	Draco	0.48	Ramoso	0.74	Ciotola	0.64	Cacciatore	0.70

Table 9: Word embeddings

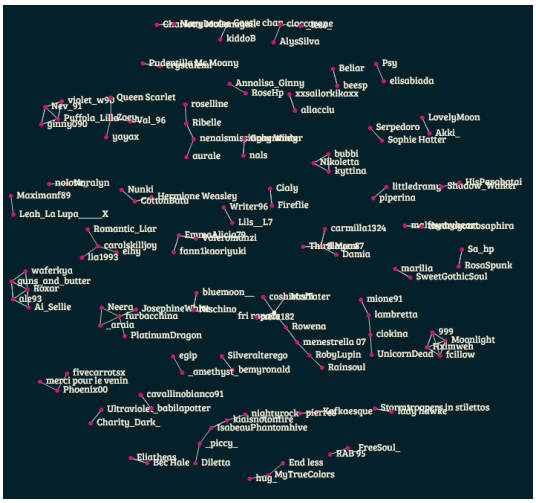


Figure 5: Smaller network of authors by Doc2Vec, used for interpretation. $max\_n = 3000$, $min\_sim = 0.80$

representative word for 91 of the 108 links in the test networks. The vast majority of them are character names: this could have been expected, since *fanfictions* are by nature character-focused stories, whose main selling point is the possibility of seeing favourite characters interact in different ways than those seen in the source material. It is therefore quite normal to see that some of the most important words in distinguishing an author's corpus from another are the character they choose to write about.

Figure 6 shows a wordcloud with all the shared words found in the links between authors, with the size representing how many times such words appear in a link. As we can see from the cloud, TD-IDF identifies as important also some words like "*amministrazione*", "*segnala*", and "*provvederà*": these are found in documents for authors which have had some stories removed by the moderators of the source website, and replaced with a standard message. This kind of outliers should be eliminated from the corpus before another round of analysis.

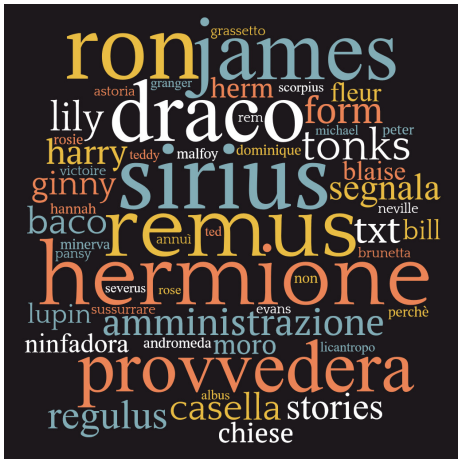


Figure 6: Wordcloud

We are aware that this method for inspecting the similarities is an heavy simplification with respect to what Doc2Vec is actually capable of doing. To get a broader, albeit anecdotal, look at its results, we manually compare some authors with just 1 published story that have been found similar with a high score. From this rough inspection, we can see that Doc2Vec seems in fact able to identify documents with similar topics, mood or structure. Table 10 shows some examples we found.

6 Sentiment Analysis

In this section we describe the methods used to extract and analyze sentiment information in texts. Furthermore, we show and discuss the results obtained by measuring the degree of association between sentiment and the two target variables used in classification, Popularity and Rating.

6.1 Introduction

First of all, we carried out tests with three different Sentiment analysis models for Italian, in order to identify the best performing one on our domain. In particular, we tested:

- VADER Sentiment Analysis Multilanguage[5], which uses the Google Translate API to translate texts into English and therefore proceed with the sentiment analysis. This option was discarded due to the fact that the API returned the "Error 429: Too Many Requests" after just a few sentences;
- FEEL-IT[6]: Emotion and Sentiment Classification for the Italian Language, an open-source Python library in which you can use a model trained on a novel benchmark corpus of Italian Twitter posts annotated

[5]https://github.com/brunneis/vader-multi
[6]https://pypi.org/project/feel-it/

Story 1 (Excerpt)	Story 2 (Excerpt)	Sim	Mood
[...] Non avrei mai creduto di poter piangere così. Non l'ho mai fatto. Tu non lo avresti voluto. Devo mostrarmi forte. Come lo sono sempre stato. Ma i ricordi. I ricordi bruciano più del fuoco.	*Ehi, amico mio. Dove sei? Mi hai lasciato solo, ci hai lasciati soli, non abbiamo più niente, adesso. A noi per essere felici bastava poter stare insieme, poter rimanere l'uno con l'altro. Ehi, amico mio. Perché te ne sei andato? [...]*	0.93	Grieving
[...] Ti lascio nella busta un piccolo album con tutte le foto della piccola Roxanne e alcune foto di Lily la nostra piccola Cercatrice .. Ti amo.. Tuo e SEMPRE tuo Teddy	*Sappi che ti amerò per sempre mio adorato. Mi manchi così tanto... Senza di te sono perso. Ricordo con sofferenza tutti i bei momenti passati insieme a te. Ti penso sempre e ogni notte sei nei miei sogni. [...] Il tuo Voldy.*	0.91	Love Letter
-Mi scusi, ma vado di fretta -, Albus si avvicinò lentamente al quadro. -Sempre di fretta voi ragazzi!-, biasciò l'uomo nel quadro. -Hai i suoi occhi, sei imparentato con i Potter?- -Esatto signore!- Albus si gonfiò d'orgoglio. -Sono Albus, il figlio di Harry Potter!-	*-Al- -Dimmi- -Allora ... che ne pensi?- -Beh, per il bene superiore...- Rispose il giovane, seduto su un piccolo sgabello di legno- [...] Voleva sentire soltanto una parola, con cui avrebbe accettato la sua proposta. -Allora, Albus?- -Direi ... direi che si può fare, Gellert. Si può fare-*	0.89	Dialogue

Table 10: Doc2Vec similarity examples

with basic emotions, for inferring both sentiments and emotions from Italian text;

- Italian BERT Sentiment model[7], trained starting from an instance of `bert-base-italian-cased`, and fine-tuned on an Italian dataset of tweets.

We tested both FEEL-IT and BERT on few randomly chosen test sentences and manually compared the results. We found the results from BERT to be a bit more representative of the sentiment contained in them, especially when dealing with neutral sentences, therefore we decide to carry on the analysis using it.

After choosing the model, in order to alleviate the computational load, we selected a subset of fanfictions, discriminated according to their length (in sentences). The study of the distribution of the stories in relation to their number of sentences has highlighted a consistent set of fanfictions between 20 and 25 phrases in length, finally selected as starting corpus, for a total of 2682 stories. For each sentence we took the first 512 tokens.

6.2 Method

Here we describe the approach used to perform the sentiment classification of a document (a fanfiction). Each document has been split in phrases using NLTK sentence splitter. For each sentence, we compute its probability of having positive (P_p), negative (P_n) or neutral (P_0) sentiment. These values are then aggregated to create the following measures for each document:

- three values, given by the sums of P_p, P_n and P_0 of its sentences, divided by the total number of sentences, which describe the probability that the average sentiment of the story is respectively positive ($\bar{P}_p$), negative ($\bar{P}_n$) or neutral ($\bar{P}_0$), ;
- SentimentLabel, a label depending on whether the highest computed average refers to positive (POS), negative (NEG) or neutral (NEU) sentiment probability. Given that most stories had a majority of neutral sentences, we only consider the maximum between the POS and NEG averages;
- a time series, in which each sentence of the story is represented by a single score obtained by the maximum of the three probabilities (P_p, P_n and P_0) given by the model: it assumes a negative or positive sign if the greatest probability of the three refers respectively to negative or positive sentiment, while it takes value 0 if the probability that the sentence is neutral is the greatest;

[7] https://huggingface.co/neuraly/bert-base-italian-cased-sentiment

- SentimentScore, an overall polarity strength of its sentences calculated by adding the sentences scores stored in the time series, normalized on the number of sentences with a non-neutral score (!= 0). A value greater than 0 corresponds to a story with overall positive sentiment and viceversa. We find this way of categorizing stories by their overall sentiment to be consistent with SentimentLabel (spearman correlation equal to 0.68 with a *p_value* of 0.0), so we deem the two interchangeable in our subsequent analysis.

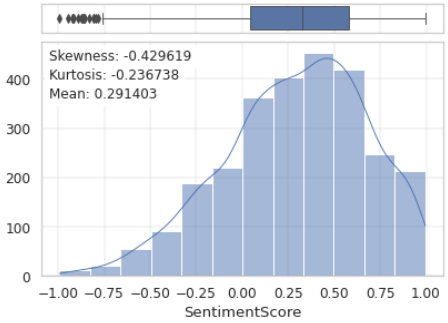


Figure 8: Distribution of SentimentScore and its respective kernel density estimate line. Top panel: boxplot of the attribute.

6.3 Analysis Results

Once we extracted the necessary information, we carried out studies on the distributions of the newly generated attributes. As shown by the distribution of the three probabilities, $\bar{P}_p$, $\bar{P}_n$ and $\bar{P}_0$ (fig. 7), the probability with which the model tends to classify sentences as neutral is much higher than for the other two classes.

In particular, the mean value of $\bar{P}_n$ is the lowest, with a very pronounced asymmetry to the right.

Analyzing the polarity strength of the texts (fig. 8), it is evident that the texts have an average positive sentiment score (0.29), with a rather accentuated left asymmetry ($skewness = -0.42$).

These considerations are also confirmed by the proportion of rows with SentimentLabel equal to POS (79% of the entire corpus) and NEG (21%): a ratio of almost 4:1.

We now want to investigate a possible correlation between sentiment and the target variables, Rating and Popularity. First, let's analyze the distribution of both the SentimentScore and the SentimentLabel in relation to the Rating variable. The bar graph in figure 9a shows that the percentages of fanfictions classified as positive or negative do not vary much according to their rating. The only exception seems to concern the "red" ones, whose probability of being classified as positive is about 5 percentage points higher than the others. Furthermore, referring to the scatterplot in figure 9b, we note that there is a very small number of red points with a negative SentimentScore.

As for the Popularity attribute, the situation is similar: the percentage distribution of SentimentLabel in relation to it (fig. 9c) does not show substantial differences.

In order to obtain a more rigorous measure of the relationship between these attributes, we calculated the Pearson correlation coefficient, which confirms the absence of correlation of the sentiment score with both the Rating attribute (−0.05) and the number of reviews (−0.04).

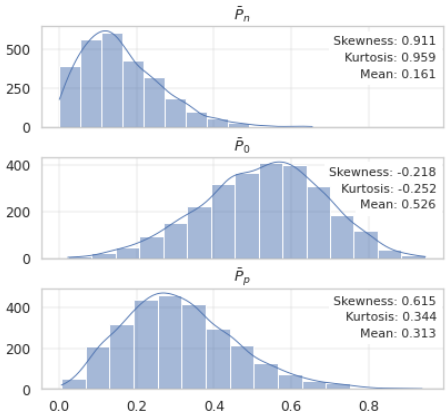


Figure 7: Distribution of $\bar{P}_p$, $\bar{P}_0$ and $\bar{P}_n$.

6.4 Sentiment Series Clustering

As illustrated in paragraph 6.2, for each fanfiction we have extracted a time series in which each data point describes the sentiment score of a sentence. We therefore want to study the trends of the time series in order to identify any recurring "emotional structures" in the stories, such as dramatic incipits or happy endings, and to try to cluster them according to these structures.

In the preprocessing phase, we carry out a smoothing task by applying on each time series

a moving average on a window of 3 time stamps. We then replace the null values created by this process at the beginning of the time series with the first available value, and those eventually present at the end due to the difference in length of documents (which goes from 20 to 25 sentences) with the last.

Furthermore, to approximate the data points to one decimal digit, we used the Symbolic Aggregate Approximation (SAX) method, implemented in the `tslearn` Python library, selecting 21 as the number of symbols (10 for the positive values, 10 for the negatives and 1 for 0), and keeping the number of segments equal to the number of "time stamps", 25, in order not to further reduce the length of the series.

The algorithm chosen for the clustering task is the K-means for time-series data, implemented in the `tslearn` Python package, using three different metrics: Euclidean, DTW and DTW with the Sakoe-Chiba Band as global constraint.

At this stage, we focused on choosing the optimal number of clusters by computing the silhouette score and the Sum of Squared Error (SSE) with the "Elbow-method", using both Euclidean and Dynamic Time Warping (DTW) as distance metrics. The results obtained (figure 10) show that although the range of the silhouette score obtained using the Euclidean distance is slightly higher, the range of values of the SSE obtained using the DTW as a metric is significantly lower. In both cases, however, the curves described by the two scores not only have no real "elbow".

Given the ambiguity of the results, we test two different partitions, in 4 and 5 clusters respectively, choosing to keep the number of clusters low in order to facilitate subsequent analysis.

6.4.1 *K-means* with Euclidean metric

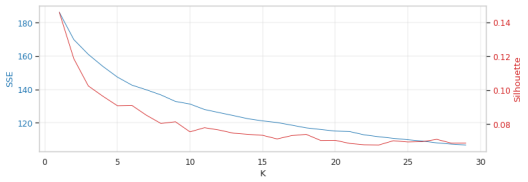
The first attempts at clustering was carried out using the Euclidean metric. As anticipated by the study of the Elbow-method graph, for both partitions carried out, we obtained an SSE of 161 with 4 clusters, 152 with 5 clusters, and a silhouette score equal to 0.1. In both cases, the size of the clusters obtained is generally balanced. By carrying out an analysis on the centroids, we can identify more or less distinctly some characterizing trends. In the clustering obtained with 5 partitions, shown in figure 11, we see the fanfiction divided according to the following trends:

- (Cluster 0) medium-high sentiment at the beginning and at the end, with a slight decrease in the central part;
- (Cluster 1) positive sentiment at the beginning, gradually decreasing throughout the story;
- (Cluster 2) positive sentiment for the entire length of the story;
- (Cluster 3) slightly negative sentiment for the entire length of the story;
- (Cluster 4) incipit and body characterized by medium-low sentiment, with significantly higher values at the end.

In the 4-clusters partition, we find similar trends. It seems that cluster 0 and cluster 3 are merged into a single cluster, with a similar structure to cluster 0 but generally lower due to the presence of the neutral-negative stories from cluster 3.

Figure 9: Distribution of SentimentLabel and SentimentScore with respect to Rating and Popularity.

(a) Euclidean Distance



(b) Dynamic Time Warping

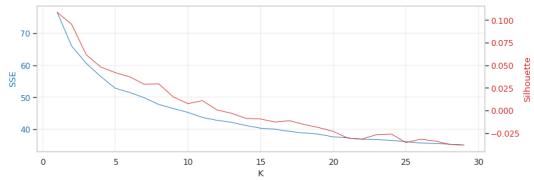


Figure 10: Method with SSE and silhouette score on TimeSeriesKMeans clustering with k from 0 to 30.

6.4.2 *K-means* with Dynamic Time Warping

As expected, the clustering obtained using the DTW as distance metric shows a decrease in the SSE (60.6 with 4 clusters, 56.5 with 5), but also in the Silhouette score (0.06 with 4 clusters, 0.05 with 5). From the analysis of the centroids of both groups of clusters, we are able to distinguish well-characterized sentiment trends, which we can summarize as follows:

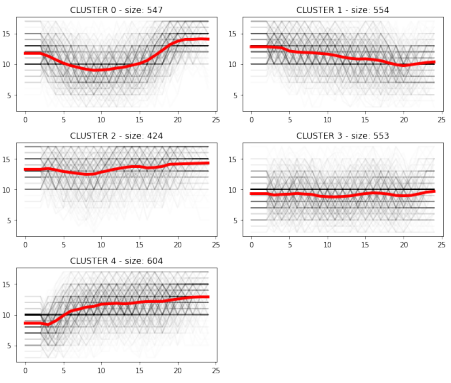


Figure 11: Centroids for the clustering obtained using Euclidean distance. The sentimentScore range is shifted from -1-1 to 0-20 due to the Symbolic Aggregate Approximation.

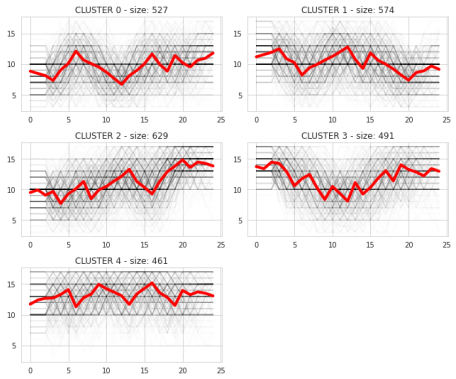


Figure 12: Centroids for the clustering obtained using DTW distance with Sakoe-Chiba band. The sentimentScore range is shifted from -1-1 to 0-20 due to the Symbolic Aggregate Approximation.

- neutral-low sentiment scores, then high, then low and finally high;
- neutral-high sentiment scores, then low, then high and finally low;
- neutral sentiment scores and only finally high;
- high sentiment scores, then neutral-low, and finally high again.
- initial low sentiment scores, then neutral-high, and finally low again.

When we ask the model to identify only four cluster, it does not identify the structure found in cluster 4.

As a last try, we applied the Sakoe-Chiba band as global constraint to the DTW distance metric. While showing an increase in the SSE ($\simeq$ 130), the results obtained (figure 12) in turn confirmed the presence of dynamics already identified by the other models, with particular similarity to the centroids highlighted by the clustering obtained with the DTW as distance metric (the first four centroids in figure 12) and to one of the five centroids of clustering based on Euclidean distance, with positive values almost stationary over time series (cluster 2, fig. 11).

		Rating								Popularity					
cluster	**size**	**verde**	**giallo**	**arancione**	**rosso**	**%verde**	**%giallo**	**%arancione**	**%rosso**	**unpopular**	**medium**	**popular**	**%unpopular**	**%medium**	**%popular**
0	724.0	0.80	0.14	0.04	0.02	27.16	26.00	29.52	23.21	0.41	0.33	0.26	26.71	26.37	28.33
1	687.0	0.76	0.18	0.04	0.02	24.75	30.25	27.62	21.43	0.44	0.32	0.23	27.53	24.18	24.39
2	647.0	0.82	0.12	0.04	0.02	24.94	20.00	21.90	26.79	0.38	0.38	0.24	21.93	27.13	23.64
3	624.0	0.79	0.15	0.04	0.03	23.15	23.75	20.95	28.57	0.42	0.33	0.25	23.83	22.32	23.64

Table 11: Information about the 4 clusters obtained from K-means clustering with DTW. In order, 1) size: cluster size; 2) Rating or Popularity values (such as *verde*, *giallo*, etc; *popular*, etc): fraction of stories with that value in the cluster; 3) Rating or Popularity values preceded by the symbol % (such as *%giallo*, etc): percentage of all the stories with that value that are in the cluster.

6.5 Sentiment Time Series, Rating and Popularity

Moving the analysis on the distribution of the target variables within each cluster, it is evident that the clusters generated are not characterized by a particular category of fanfiction: the distributions of the different classes of the Rating and Popularity variables within each cluster reflect their distribution throughout the entire dataset, with only small non-significant fluctuations.

In table 11, we propose as an example the information related to the 4-means clustering, using the DTW as a distance metric.

7 References

S. Montemagni. 2013. Tecnologie linguistico-computazionali e monitoraggio della lingua italiana. *Studi Italiani di Linguistica Teorica e Applicata* (SILTA), 145–172

H. van Halteren 2004. Linguistic profiling for author recognition and verification. *Proceedings of the Association for Computational Linguistics*, 200–207

A. Mattei, D. Brunato, F. Dell'Orletta. 2020. The Style of a Successful Story: a Computational Study on the Fanfiction Genre. *Proceedings of 7th Italian Conference on Computational Linguistics* (CLiC-it), 1-3