Text mining on Family Guys scripts

Simone Grassi

April 2021

KEN4153: Information Retrieval and Text Mining

Faculty of Science and Engineering
Department of Data Science and Knowledge Engineering
Maastricht University
Netherlands

Contents

1 Introduction **3**

2 Spacy Library **4**

3 Implementation **5**
3.1 Preprocessing of the corpus 5
3.2 Named Entity Recognition over the time 6
3.3 Wikification and web searching of the entities 7
3.4 Emotions analysis 7
3.5 Speaker prediction 8
3.6 Topic analysis 8

4 Results visualization and analysis **10**

5 Conclusion **14**

1 Introduction

The goal of this practical assignment, from the course *Information retrieval and Text Mining*, is to select a text corpus and to implement different text mining operations to extract meaningful information.

This project is based on a dataset named *family_guy_dialogue*[1], that contains 1285 sentences from the famous animated series **Family guy**. In particular, each sentence is labeled with the character who pronounced it and the corresponding season. This dataset is very interesting and well organized, but it does not contains enough information for a good information retrieval, for this reason I used also the scripts reported on the web site *trascript.fandom.com*[2]. This site contains all the scripts from each seasons, but they are written by different fans, so they are very chaotic without a common format.

First of all, I created my dataset applying preprocessing techniques on fandom's corpus, as example using regular expression I removed time marks, rumors and actions indications, redundant punctuation marks etc. Some fans marked the sentences with the character who pronounced them, so I extracted these sentences to expand the labeled dataset and removed these annotation from the text to allow a fair character extraction using text mining techniques.

The biggest part of my project is based on the library **Spacy**, because it allows to create pipelines with different processing phases, such as TAG, NER, PARSING, and these modules are among the faster available for Python.

Once I obtained two base datasets, one labeled and one not, I applies different text mining operations to extract meaningful and articulated information.

The first operation implemented is the Named Entity Recognition (NER): for each season and each episode, the following entities are extracted with the corresponding frequency: Event, Person, Proper name, Work of art, Organization, and Geographical place. Given that Family guy is a satirical and provocative series, it is interesting to see the real words references cited and the evolution of the topics covered, as well as comedy, from the 1999 to the 2021. I used also the library *wikipedia* to search on the homonyms search engine the *PERSON* terms extracted with the aim of exploiting the metadata to obtain more specific information, classification and photos too.

The second text mining operation consists in tracing the emotional spectrum of each character, this is interesting because in an animated series each of them are very different and stereotyped.

I tried to look at the sentences only to predict who is the speaker. In a real word scenario, this operation is very complex and unreliable because usually people do not have an unique way of speaking, but it could be possible in a cartoon

[1] https://www.kaggle.com/gokulrajkmv/family-guy-dialog
[2] www.transcripts.fandom.com/wiki/Family_Guy

where each character is strongly characterized, involved in different situations, and interact mainly with different other characters. This processing is the most challenging because of the limited number of labeled data, and the high time that requires creating new ones. The idea is to figure out if this operation is possible with a valuable accuracy and in that case, find the most promising strategy.

In the end, I exploited topic analysis techniques to study the different characters from a further point of view, trying to figure out if it is possible to identify them by the different situations and themes they are usually involved.

All the codes, checkpoints results and materials related to this project are available on **Github**: https://github.com/Simone2498/Text_Mining_UMuniversiteit.

2 Spacy Library

In its largest part, this project relies on the library Spacy, and this is very interesting because provides very effective pre-trained model stacked in pipeline with state-of-the-art speed results and optimized on GPUs.

The performance of the software are clearly reported on their web page [3]. The evaluation is based on **OntoNotes 5.0** benchmark dataset. It is the corpus on which Spacy is trained and one of the biggest with more than 1.445 million words for English, and sentences from telephone conversation, news, blogs and religious texts [4].

pipeline	parser	tagger	NER
en_core_web_trf (Spacy v3)	95.1%	97.8%	89.8%
en_core_web_lg (Spacy v3)	92.0%	97.4%	85.5%
en_core_web_trf (v2)	91.9%	97.2%	85.5%

In this project I used Spacy V3, that includes for the first time Transformer based pipelines bringing the performance to the state-of-the-art results and is optimized for the GPUs.

[3] https://spacy.io/usage/facts-figures
[4] https://catalog.ldc.upenn.edu/LDC2013T19

3 Implementation

This project is articulated in 6 main files, each of them takes care of one type of text mining operation. To work step by step and to obtain intermediate results in a format directly readable, each step works is articulated in functions that execute an operation and save the result. For this reason the passages from one processing to the following one is sometimes not efficient and require the reading of files, each of these intermediate results are available as attachment of this paper.

3.1 Preprocessing of the corpus

The pre-processing functions are grouped in the file *dataset_creator.py*, the corpus downloaded from *transcripts.fandom.com* are structured with different pattern, according with the author who transcripted the episode, and at first each season was stored in a TXT file.

The preprocessing pipeline is composed by the following steps:

1. remove the special characters, for example sometimes the songs are delimited by two musical notes, these symbols are useless for the following operations and are difficult to manage.
2. some scripts report the time frame during which the sentence is pronounced, it is always in the form "00:00:00,000 –¿ 00:00:00,000" so it is possible to remove them using a regular expression
3. some episodes divide the three acts with a line in the form "Act 1", so it is removed with a regular expression.
4. in the scripts are reported also the actions indicators for the characters and the rumors descriptions, they are respectively grouped by rounded and square parenthesis, so it is possible to remove them with regular expressions too.
5. Some authors start the sentences with the name of the character who is speaking, delimited by :, for this reason it is easy to exploit a regular expression to extract the entire sentence and its label, increasing in this way the labeled examples in the *Kaggle dataset*. In the end, I removed the character's marks to use text mining techniques to try to extract them in the following phases.
6. As a final step, all the redundant characters, such as multiple new line, lines with only punctuation marks, multiple - (that sometimes starts the lines without a particular meaning) are removed.

In the end, the labeled dataset is merged with the Kaggle's ones, to obtain 10939 examples. With the purpose of making more effective the following operations, a coreference resolution is essential in this phase, I made this with the module

neuralcoref based on Spacy. So the outputs of the preprocessing phase are two text files:

1. *texts_dataset_coref* contains all the corpus in the form of a map {season, episodes }, episodes is an ordered array where each element is a script of the corresponding episode.
2. *labeled_dataset_coref* contains a matrix with examples as row and three columns: character-label, sentence, and season's number.

3.2 Named Entity Recognition over the time

The goal of this analysis is to find the entities frequency over the episodes (and so over the time) to show the evolution of the treated themes and the comedy during the years.

Spacy was born with a pretrained pipeline with modules such as Parser, Lemmizator, Named Entity Recognitor (NER), and Part of Speech tagging (POS), so I exploited them to extract the following entities: events (EVENT), countries and states (GPE), organizations (ORG), people (PERSON), films and other art works (WORK_OF_ART); more two POS categories: adjectives (ADJ) and verbs (VERB).

At the same time, I constructed a global map with all the occurrences group together without considering the episode.

To avoid redundant and duplicated keys, caused by tokenization errors or entities named in different ways, (such as Peter Griffin, Peter, Griffin Peter etc.) a normalization phase is required. Through the analysis of the problem the main errors to solve are:

1. multi word entities
2. Missing surname or name/surname inversion
3. Numbers and special characters
4. Lemmization of the verbs
5. genitive
6. disambiguation between main character's names and famous people and extras ones[5]

I tried different approaches and in the end the best performances were achieved using an hybrid approach: **Monge-Elkan** with **Levenshtein** as local similarity, it is able to face all the problems before.

[5]solved noting that only the main characters are named with name or surname only, conversely the extra are always named with name and surname ae. Tom Cruise will never be named as Tom, as opposed to Tom Tucker

The output of this step is a 2D list, for each episode is stored a list of all the entities and the corresponding frequency. This data structure is stored in json format in a file with the name of the entity. The idea is to visualize these information as a bar graph that evolve in time and a word cloud.

3.3 Wikification and web searching of the entities

The previous processing phase resulted in a set of entities assigned to a macro class, such as persons, events. I can go deeper in these classes mining more specific information, such as actors/actress references during all the seasons.

To do this, I use the python library wikipedia. For each entity I searched if existed a Wikipedia page, and in this case I could extract all the categories used to classify it and searched inside them if the entity is defined, as example, as an actor/actress. This approach revealed to be very powerful, not only to identify and normalize entities belonging to particular classes or with specific characteristics, but also to extract additional information. As example in my project, for each actor/actress I searched and download a small photo that will be used in the following visualization phase.

3.4 Emotions analysis

Each character in this series is strongly characterized to appear as comical as possible, moreover they are involved in different situations and have typical problems. For this reason, it is interesting to try drawing an emotional spectrum of the main characters.

I exploit the library **text2emotions**, it is based on a thesaurus approach, where each word is correlated to a score under a model with 5 metrics: Happiness, Angry, Surprise, Sadness, Fear.

This approach is enough robust, considering that the results are calculated as average on all the sentences pronounced by a character, in this way is possible to underline the persistence of a particular emotion. The approach based on a vocabulary is able to manage the morphological negation, but this is not possible for the syntactic one. In particular is necessary to solve the negation before the verb be, changing the polarity of the following adjective, and the one of an action verb such as like or hate. To do this, I used a library named **PyDictionary**, it is able to return the meaning, homonyms and antonyms. I used the entity linking module of Spacy to identify a negation, removed it and lead back to the corresponding verb. Then, according to the type of verb, I replaced the adjective or the verb itself with one with the opposite meaning. According to my tests, this proved to be effective managing the different polarity of the sentences and filter the subjective ones.

The output of this phase is a structure, saved in the file *characters_emotions.txt*, that links to each character its emotions score array.

3.5 Speaker prediction

This is proved to be the hardest operation to do, because of the limited number and the characteristics of the training example. Rather than the merging phase was able to increase the number of examples from 1285 to 10939, a lot of these sentences are short and commonly used.

There are a lot of them that are very common, but there are also a big part that is typical of a specific speaker, as example the characters in some cases has a personal vocabulary or specific words. Moreover the characters usually interact with different others and in different situations, as example Stewie with Rupert and Brian, or Chris who start many sentences with "Mooom". The two approaches tested look at the information from a very different point of view: a **Tranformer** that is able to take into account the order and the relation among the words, and a **SVM with tf-idf** features vectors that consider only the salience of particular words.

Given the premise, the first approach could be overcomplicated and underfit the problem because of the limited number of training examples, instead classifying the sentence embedding I obtained an accuracy of only 34%. As said this number could not be too high because of the commonly used sentences present in the dataset, but this is not an impressive result.

To obtain the **tf-idf** features I used a personal vocabulary, generated during the preprocessing phase and filtered from the stop words, to reduce as minimum as possible the dimension. Then I applied a very easy model, a Support Vector Machine, for a multiclass problem. I tested different hyperparameters and strategy combination without finding a particular effective model, the results are reported in the following section.

3.6 Topic analysis

Despite the fact that the comedy typical of this type of series effects all the characters at the same level, they usually are involved in different situation and so they treat different topics. For this reason the last test I wanted to implement is a topic analysis. The first attempt was to extract the topic from the episodes and try to print a time evolution, this idea was fast stuck because of two main problems: a big amount of common use words (also after a strong filtering), the small changes is the topics go lost into the main ones that remain constant during the time and so take over.

For these reasons, I focused on the extraction of the topics from the characters. The first step was to generate a tf-idf encoding of the labeled dataset filtering the 6 main characters only. Looking at the term frequency encoding it's evident how many common words are present, for these reason I applied a filter removing all the ones over a threshold.

In the end, I applied an NMF with 20 topics, from the library *scikitlearn* followed by a PCA to reduce the dimension to 2 to have a better visualization on a scatter

graph.

4 Results visualization and analysis

The first processing phase was the NER and POS extraction, rather than I extracted two type of data structures: one global and one time dependent, I decided to visualize only the first one because more meaningful for the following analysis. In the figure 1 are represented the entities labeled as PERSON, we

Figure 1: Word cloud visualization with the entities labeled as PERSON

can see that the main characters are the biggest ones and, thanks to the normalization phase, there is no repetition and the name and surname are properly linked.

Figure 2: Word cloud visualization with the entities labeled as PERSON

The figure **??** represent the terms tagged as ADJ (adjective) and VERB by the POS tagger, without any filter the most common terms are overdimensioned and the most characteristic are loosen, so I applied a filter on the frequency and

in this way is possible to see the emergence of themes and language typical of this type of ironic series.

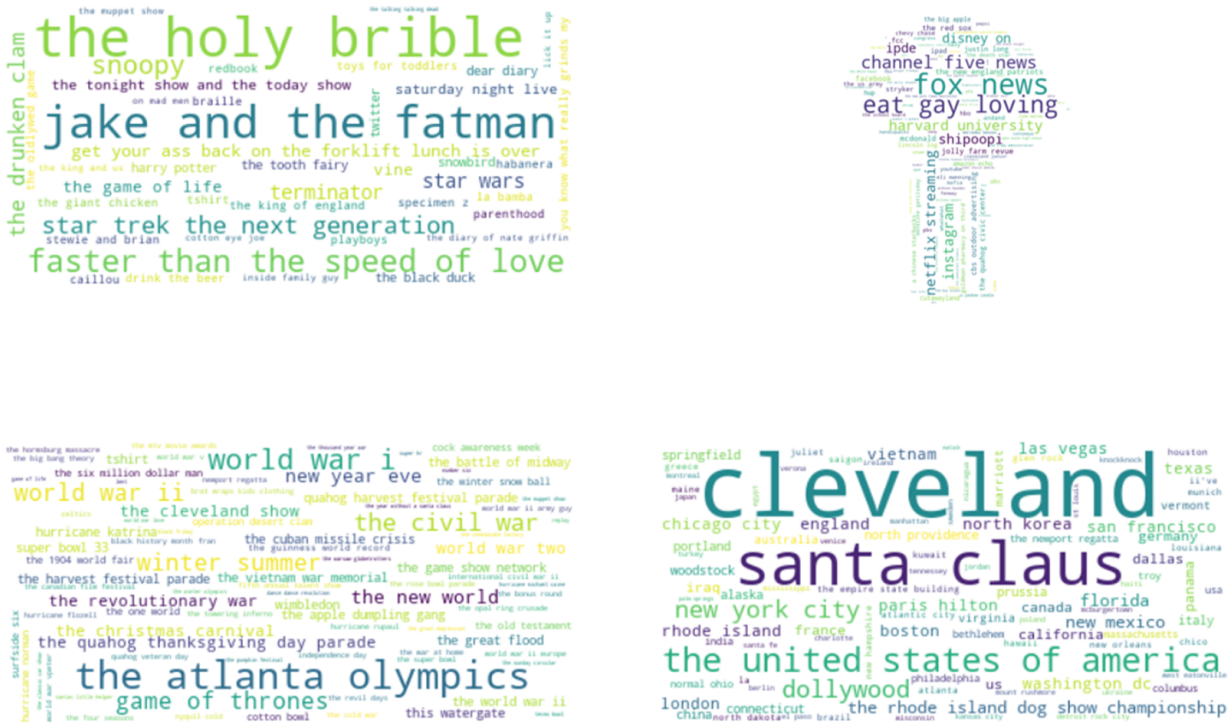

Figure 3: Word cloud visualization of (in order from top-left): WORK-OF-ART, ORG (organizations), EVENT, GPE (geographical places)

The figure 3 resume the results of the entities labeled as Work of art, Organizations, Events, Geographical places; Family Guy is famous for its citation at real word elements especially the ones related to the pop culture such as Start Trek or Star Wars, but also organizations such as Fox News or BBC that are criticized and joked.

The *wikification* phase gives us very interesting results, first of all using the pages' name was possible to normalize in a very effective way the actor names[6] that usually are always written as name and surname but with different format. Secondly, through the exploitation the Wikipeda metadata is possible to obtain many additional information useful for further processing phases, so in this test I was able to identify and extract 580 actors/actress cited during all the seasons, save and process one photo for each of them and a partial result is showed in the following image.

[6] validation made by hand on the 580 actors extracted

Figure 4: Sample of the photos automatically extracted by the wikification procedure, exploiting the meta information available on Wikipedia is possible to obtain many meaningful information

The emotional mining phase returned an array that represent the scores in a model with 5 emotions: Happiness, Angry, Surprise, Sad, Fear, the results are then visualized with a radial graph in the figure 5. The first thing that catches the eye is the general tendency to exceed in the direction of the Surprise, this could be brought back to two main causes:

1. The composition of the emotion thesaurus for the class Surprise, because it contains a lot of words that are not affected by negations, so all the occurrences belong to the same category, conversely to Happy/Sad.
2. Exaggerated or stupid events and action represent the greatest part of the series, so the characters manly express emotions belonging to the macro category of the surprise

The emotional analysis reflect very well the character traits of each component of the family, as example: Lois and Peter tend to have more happy and carefree moments, Meg is usually bullied so the main emotions expressed are Fear and Sadness, Stewie is involved in many adventures so expresses mainly Fear but also happiness, Brian is the stereotype of a drunk and cynical man so its emotional spectre is very limited. Another funny main character in Family Guy is God, that has to face the Peter's disaster so he express a lot of times Fear and Sadness, but never Angry.

The last text mining task I tried to implement was the speaker recognition looking at the style of the sentences pronouncated, this sudenly appear as a very hard problem because of the scarsity and quality of the training data. As

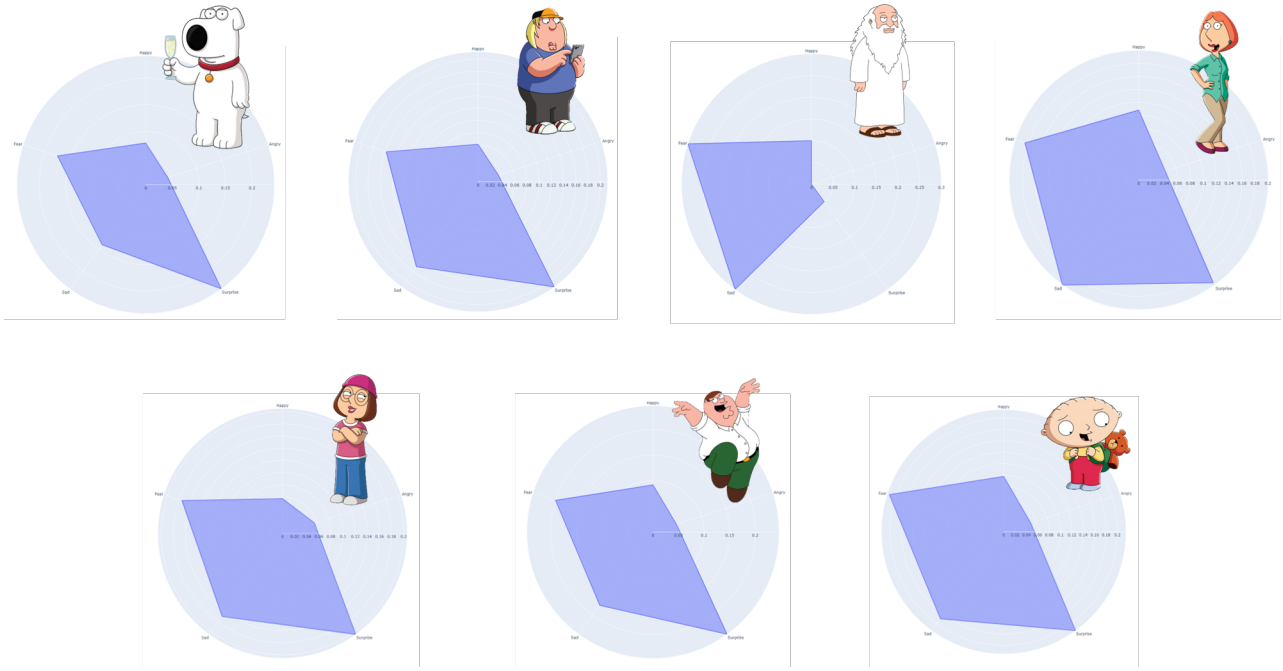

Figure 5: Representation of the emotional spectre expressed by some main characters during the entire tv serie

said in the previous section I tried two different approaches, the first based on the Tranformer available of the Keras webpage, the max accuracy reached during the experiments is 34%. Then i tried to increase the level of abstraction, losing some syntactical information to work with a simpler model, the SVM in combination with a tf-idf representation showed to be able to sligthly overcome the Tranformer results. The following table 4 represents the results of the most promising tests, mode indicates a One-vs-Rest OVR or a One-vs-One OVO approach, weights indicates when I added weights to face an unbalanced dataset. The last row represent an example with a reducted vocabulary that completely ignores the punctuation entry.

mode	kernel	weights	accuracy	precision	recall	F1
ovr	rbf	none	0.348	0.122	0.385	0.180
ovo	rbf	none	0.348	0.122	0.348	0.180
ovo	rbf	w	0.078	0.006	0.078	0.011
ovo	rbf	none	0.349	0.122	0.349	0.180

The not particularly impressive results could be brought back to many causes, as example:

1. A not enough meaningful dataset, the classes are unbalanced and there are too few examples for training a Transformer
2. The speaking order is completely lost, so unlike in a book, in this case is not possible to exploit the speaking order or reference to the speaker
3. The language is very context dependent and many sentences are related to a character's action
4. A lot of sentences are very commonly used, and it is impossible to link with a specific character, this does not effect the quality of the model in an useful prediction, but decrease a lot the value of the accuracy

Despite all these problems, the previous analysis prove how the characters are different and involved in different situations, so this prediction models have still margin of improvement, but they require to return to the dataset and increase quality and quantity of the training examples.

In the end, figure 6 show the results from the topic analysis for two main characters: Lois and Stewie. I chose them because they are very different in the way they speak and act, and this is reflected in the scatter representation obtained after a NMF with 20 topics and a PCA to reduce the dimensional to 2D. Despite many sentences that can be pronounced by both of them, it is possible to identify two well defined clusters: a yellow one for Lois and a red one for Stewie. Given the problem that the previous classifications raise and the effectiveness of the topic extraction for these two characters I tried to apply an SVM using the 20 topics as features. As the following table shows, the results have drastically overcomed the previous method explored.

method	accuracy	precision	recall	F1
SVM with 20 topics	0.605	0.769	0.605	0.537
SVM with tf-idf	0.506	0.257	0.501	0.341

5 Conclusion

In this project I started from two different dataset, one of which unstructured and disorganized, with a sequence of masks based on regular expression I was able to clear them and extract labeled examples.

I exploited the tokenizer, lemmatizer, POS tagger and NER classifier based on pretrained Transformer from the library Spacy v3. I solved the coreferences and mined meaningful information such as references to organization, real word people, elements of the pop culture etc. I noticed how the normalization of these entities using an edit based and multi word approach such as Monge-Elkan with Levenshtein drastically reduce the typing errors and is able to link name and surname (in a context where there are not characters with the same name and the real word reference are done using both name and surname). With a very easy visualization such as word clouds, I was able to identify the main topics and

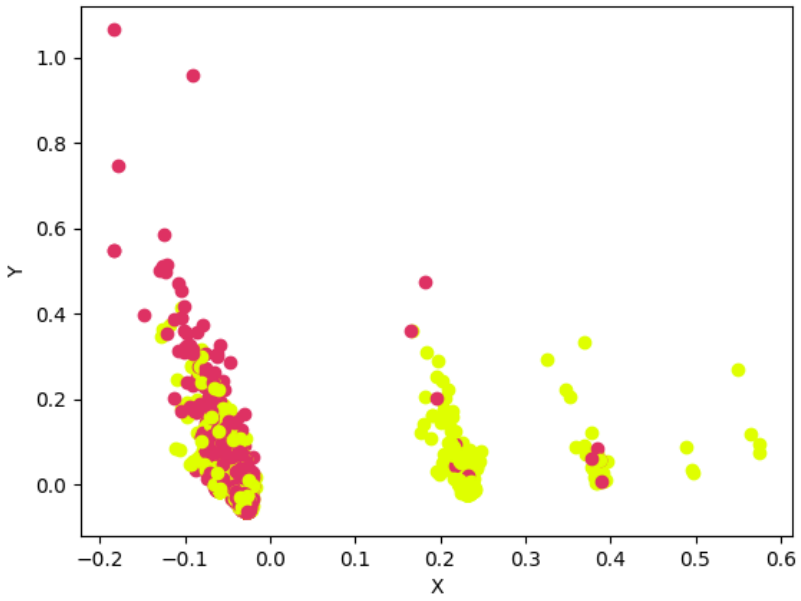

Figure 6: Representation of all Lois (yellow) and Stewie (red) sentences after the topic extraction and the PCA

comedy tracts of the series, as well as the main characters. Then I exploited Wikipedia to extract more information not originally present in the corpus, such as identify, normalize and download a photo for each actor/actress cited in Family Guy.

Given that all the characters have a different behavior and emotional spectre I applied emotional mining techniques to represent each of them in a model with 5 emotion and visualize in a scatter graph.

I used NFM to extract the main topics, as first related to the episodes, but after understanding that the variation were to low to be noticed I focused on the characters analysis with very good results rather than the big amount of sentences impossible to link with a specific speaker.

The most difficult task was the speaker's prediction, I tried two conceptually different approaches: a Transformer based able to consider the word order and context, and a SVM with a tf-idf encoding that needs less examples to be trained. Both results in similar accuracy stucked around 35%, and this is attributable to a shortage and low quality of the training examples as well as the presence of many commonly used senteces.

The good results of the topic analysis was able to provide more meaningful features for an effective classification, that on a one-vs-one problem brings the accuracy to 60%.