POLITECNICO DI MILANO

Facoltà di Ingegneria

Ingegneria Industriale e dell'Informazione

Dipartimento di Elettronica, Informazione e Bioingegneria

Corso di Laurea Magistrale in

Ingegneria Informatica

Mood prediction of movies using multi-label text classification

Relatore:

PROF. PIER LUCA LANZI

Correlatore:

DR. NICOLA PADOVANO

Tesi di Laurea di:

ANDREA DI GIOSAFFATTE
Matr. 852589

MATTEO IMBERTI
Matr. 863662

Anno Accademico 2017-2018

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The goal of this work is the study and design of a framework for the prediction of multimedia contents *metadata*. Metadata are information which describe a product specifying its distinctive properties: for example, the movie genre, the setting or the story's pace are all attributes which define the characteristics of a movie. The process of compilation of that information is often performed manually by domain experts, who rely exclusively on their experience in the field. With this thesis we aim to show that, using movie plots, it is possible to develop a machine learning architecture able to make complex predictions (multi-label) of the values of metadata like the *mood* (i.e. the effect induced by a content in the viewer). This thesis starts with the study of the state of the art, focusing on the multi-label classification of textual contents. We then present the designed architecture, describing the mathematical modeling of input text, the identified classification algorithms and the rebalancing methods used to harmonise the distribution of target classes. Subsequently, we describe the experimental analysis performed on the available data, discussing the results and highlighting strengths and weaknesses of each approach. Finally, we present some possible future developments of this work by proposing algorithmic and architectural improvements.

SOMMARIO

L'obiettivo di questo lavoro è lo studio e la progettazione di un framework per la predizione dei *metadati* di contenuti multimediali. I metadati sono informazioni che descrivono un prodotto specificandone le proprietà distintive: ad esempio, il genere cinematografico, l'ambientazione o il ritmo del racconto sono tutti attributi che tratteggiano le caratteristiche di un film. Il processo di compilazione di queste informazioni è spesso svolto in maniera manuale da un gruppo di esperti che si basano esclusivamente sulla loro esperienza nel settore. Con questo lavoro di tesi si vuole mostrare che, utilizzando la trama di un film, si riesce a sviluppare un efficiente sistema di machine learning in grado di compiere predizioni complesse (multi-label) dei valori di metadati come il *mood* (l'insieme degli effetti indotti dall'opera nello spettatore). L'elaborato di tesi parte dallo studio dello stato dell'arte e in particolare della classificazione multi-label di contenuti testuali. Si presenta, in seguito, l'architettura ideata, descrivendo le tecniche di modellizzazione matematica dei testi, gli algoritmi di classificazione identificati e i metodi di ribilanciamento della distribuzione delle classi da predire. Successivamente vengono mostrati gli esperimenti sui dati a disposizione discutendo i risultati ed evidenziando i punti di forza e di miglioramento di ciascun approccio. Infine vengono esposti i possibili sviluppi futuri proponendo innovazioni algoritmiche e architetturali al lavoro presente.

1 INTRODUCTION

The fiercely competitive world of the industry of media service providers is leading the major players of the field to invest more and more in the development of recommendation systems to target the final user of their on-demand platforms. Recommendation systems can rely on information which describes the content to make suggestions to users and thus the quality of such information plays a major role in the effectiveness of recommendations. The data which provides such information is called *metadata*.
One of the major Italian players in the field of communication and multimedia distribution is the Mediaset Group. Multimedia production and distribution play a significant role inside the company, and the interests of the company are both on traditional mediums (e. g. television, radio) and on the front of online multimedia streaming. In Mediaset Group, in particular within the Strategic Marketing, the role of metadata has relevance in different areas in addition to recommendation systems. Metadata is used as support for the editorial sector of the company, providing strategic insights to the production of multimedia contents, and is applied to the marketing sector, allowing for more precise advertising. Given the importance of metadata, Mediaset Strategic Marketing has designed a complex structure for the description of multimedia contents. The retrieval of metadata relies on human experts which manually fill a form following some guidelines, in a process called *tagging*. The submitted forms are then revisioned by supervisors who approve or debate possible adjustments. We worked within the Mediaset DataLab with the purpose of speeding up the tagging process, designing a machine learning architecture with the purpose of providing suggestion to domain experts in the correct filling of the metadata collection form. We were asked to study this possibility focusing on the prediction of the *mood*, one of the wide range of metadata defined by Mediaset Strategic Marketing, which describes the underlying state of mind arising by watching a content. This metadata constituted an interesting case of study since the number of *moods* associated with each multimedia content can vary from one to two and so, was this task was interesting in order to study simpler problems in the future.

1.1 MOVIE CLASSIFICATION FROM TEXT

The mood metadata is used in different categories of multimedia contents, we were asked to concentrate our work on those of the movie category. Despite the fact that, typically, movie classification tasks are based on audio and video features, a minority has shown how text data can be used to provide meaningful results. Text sources used for the text-based classification of movie genres are usually a dialogues transcript, obtained directly from the video source or extracted using OCR and speech recognition software. The text transcripts obtained were not suited to capture the situation of a scene since they are composed by dialogues. We overviewed those methods and tried to overcome the limitations of the input selected for text classification of movies. To perform mood prediction of movies using multi-label text classification, we started from a dataset where having a movie plot and a set of prediction targets coming from a predefined set of twenty possibilities. We analyzed the dataset starting from its structure. We detected the presence of some imbalancement for the *mood* distribution in the dataset, and we highlighted the correlation between *moods*. Finally, we selected the information that could constitute a domain-specific feature to improve the text classification.

1.2 MOOD PREDICTION OF MOVIES USING MULTI-LABEL TEXT CLASSIFICATION

To overcome the limitation of movie textual transcripts, we chose movie summaries as the main source of input for the *mood* prediction of movies using multi-label text classification. In the dataset, we were already provided with the plot, which we considered inadequate to describe sufficiently a movie. For that reason, we scraped the web for summaries using the links to Wikipedia[1] and IMDb[2] movie pages in the dataset. We applied various techniques to build features vectors from the gathered summaries, after a preprocessing phase, using the *Bag-of-Words*, *TF-IDF*, *Doc2Vec*, a vectorization based on *LDA Topic Modelling*, named *Topic2Vec*, and a custom vectorization we called *Tf-Idf-Weighted Word2Vec*, based on *Tf-Idf* scores and the pre-trained Google *Word2Vec* model. We then formulated a feature vector representation of the cast of a movie, namely *Cast2Vec*, with the purpose of improving the quality of text-based classification using a domain-specific feature. For the purpose of multi-label classification, we used problem transformation techniques which allowed us to trace back binary classification problems for each *mood*, to a multi-label setting. The binary classifier methods we used were *Multinomial Naive Bayes* [23], *Support Vector Machines* [12], *Logistic Regression* [17], *Random Forest* [5] and *XGBoost* [11]. The first technique we adopted was *Binary Relelevance* [60], which we used as a benchmark to discard some of the poor performing vectorization techniques. We then applied rebalancing algorithms, namely *Random Undersampling* [19], *Random Oversampling*, *Edited Nearest Neighbours* [55] and *Synthetic Minority Oversampling Technique* [10] to Binary Relevance and assessed the value those methods brought to the classification performance. To verify the importance of *moods* correlation we highlighted in our dataset, we then moved toward a model able to exploit such information: *Classifier Chains* [39]. Using the same base classification methods as for Binary Relevance and the input vectorization which resulted being the most effective, we built an *Ensemble of Classifier Chains* for each classification method and assessed the utility of exploiting correlation information. Finaly we used the *Stacking* [56] ensemble technique to combine the classifiers using a meta-learner. We then used the *Stacking Aggregation* [14] technique to exploit the label correlation, comparing the prediction performance to the classical Stacking with and without the use of the rebalancing techniques which resulted successfully.

1 Wikipedia: https://en.wikipedia.org/
2 IMDb: https://www.imdb.com/

1.3 RESULTS

Our work laid the foundations for metadata automatic classification, showing how movie summaries can be useful to the prediction of the *mood* of movies. Using Binary Relevance approach as a benchmark, we highlighted the superiority of Doc2Vec, Topic2Vec and TF-IDF-Weighted-Word2Vec over classical TF-IDF and Bag-of-Words in our classification task. Similarly we tested the selected balancing techniques which proved to be unsuccessful with the exception of ENN. Combining different vectorization we also underlined how domain specific features like Cast2Vec could improve the quality of predictions based on text vectorizations. Testing approaches which were able to capture label correlation, like ECC and Stacking Aggregation, we showed how those methods generally improve with respect to their Binary Relevance counterparts. In particular the most effective classification method was the ECC with XGBoost as base classifier.

1.4 STRUCTURE

The thesis is structured as follows:
In **chapter 2** we overview the analysis of the literature, with a focus on the problem of multi-label classification and some of the approaches used to solve it.
In **chapter 3** we discuss our scenario, our task and its difference with respect to previous problems, and the analysis of the dataset.
In **chapter 4** we present our classification architecture, showing the steps and approaches we adopted.
In **chapter 5** we show the results of the different approaches and techniques and their comparison.
In **chapter 6** we discuss the results of the work and lay down some possibilities for future developments.

2 STATE OF THE ART

In this chapter, we introduce text classification reviewing all the major methods used to achieve good classification performances. We define the concept of classification, focusing on the multi-label case. Then, we describe the two main steps of text classification: the first one is text vectorization, needed to have an input representation valid for the classification task, while the second one is classification. We present the classification methods employed for text classification in general, putting particular emphasis on those related with multi-label data, also by showing common evaluation metrics used by them. Finally, we describe some strategies used when inputs are not well-balanced, that is a frequent problem when working with multi-labeled data.

2.1 CLASSIFICATION

Machine learning is a field of artificial intelligence that uses statistical techniques to give computer systems the ability to "learn" (i. e. progressively improve performance on a specific task) from data, without being explicitly programmed. *Classification* is a particular kind of machine learning task, named as *Supervised Learning*, in which an algorithm is presented with example inputs and their desired outputs, and the goal is to learn a general rule that maps inputs to outputs.
Given a training dataset $\mathcal{D}$ of the form (x_i, y_i), where $x_i \in R^n$ is the i-th example and $y_i \in \mathcal{L} = \{1, ..., K\}$ is the i-th class label, classification aim at finding a learning model $\mathcal{H}$ such that $\mathcal{H}(x_i) = y_i + \varepsilon$, where ε is a general catch-all error for what is missed with the identified model [18]. Two types of classification exist: single-label classification and multi-label classification.
Single-label classification concerns with learning from a set of examples that are associated with a single-label from a set of disjoint labels $\mathcal{L}$, with $|\mathcal{L}| \geqslant 1$. If $|\mathcal{L}| = 2$, then the learning problem is called a binary classification problem, while if $|\mathcal{L}| > 2$, then it is called a multiclass classification problem [52].
Multi-label classification, on the other hand, concerns with learning from a set of examples that are not associated with a single-label, but with a set of labels $Y \subseteq \mathcal{L}$. Nowadays, multi-label methods are increasingly required by modern applications: text classification [22] is one of the main challenges related with multi-labeled data, but many other fields appear as strictly related with these methods, such as music classification [27] and protein function classification [9].

2.2 TEXT CLASSIFICATION

Text classification is the activity of labeling natural language texts with relevant classes from a predefined set. This is typically a multi-label task, since text documents usually belong to more than one conceptual class. As formally stated in [48], given a description $d \in \mathcal{X}$ of a document, where $\mathcal{X}$ is the document space, and given a fixed set of classes

$\mathcal{C} = \{c_1, c_2, ..., c_n\}$, goal of text classification is to learn a classification function $\mathcal{H}$ that maps documents to classes:

$$\mathcal{H} : \mathcal{X} \rightarrow P(\mathcal{C})$$

where P represents the powerset operator.
The first step of every text classification task is *text vectorization*: each document is converted from a textual representation to a numerical one. The main idea behind this step is to obtain numeric vectors able to express differences in the semantic of documents, or at least to highlight the most important document keywords. The second step performed in text classification is the classification itself, which can be divided into two sub-steps: the training phase, in which each vectorized document belonging to the training dataset, together with its labels, is given to a classification algorithm (named as *classifier*), which tries to approximate a classification function able to map every vector to its labels, typically by minimizing a cost function, i. e. a function that measures how wrong the model is in terms of its ability to estimate the relationship between documents and labels; the prediction phase, in which new unseen documents, after being vectorized, are classified using the previously trained classifier, i. e. are associated to a predicted set of labels.

2.3 TEXT VECTORIZATION

In the field of text classification, one of the first steps needed in order to make the classification possible is the so-called *text vectorization*, i. e. the transformation of a text, seen as a collection of words, into a numeric vector. In this way we obtain a representation of documents compatible with all the main classifiers used to perform the classification task.
Prior to vectorization, a data preprocessing phase is needed in order to obtain artifacts which are improved in quality and thus much more useful for performing the classification task.
In the following we present an overview of data preprocessing techniques for text classification, followed by a presentation of the most known models for performing text vectorization.

2.3.1 *Data Preprocessing*

Every textual document is basically composed of words and special characters, such as numbers and punctuation characters. Among all the words and characters in a text, some are unuseful for the purpose of classification and can be viewed as noise, since they do not bring added value in terms of helping the discrimination process with new information. Thus, a data preprocessing step is usually performed before the vectorization of any document, in order to obtain noise-free artifacts which have a higher discriminative power. Moreover, the majority of text vectorization techniques need the documents represented as a list of tokens, i. e. a list in which every element is a single word or a single special character: other preprocessing steps are thus needed. Here is a list of the main data preprocessing steps performed in text classification [54]:

TOKENIZATION The step which splits longer strings of text into smaller pieces, or tokens. Larger chunks of text can be tokenized into sentences, sentences can be tokenized into words, etc. It should be clear that tokenization is not a trivial process, since it requires to make decisions like, for example, which token boundaries to consider (e. g. "full stop", "comma", "dash", etc.) and what to do when those boundaries are reached (e. g. "sugar-free" can be tokenized as "sugarfree", but also as two different tokens: "sugar" and "free").
Further processing is generally performed after a piece of text has been appropriately tokenized.

STEMMING The process of eliminating affixes (suffixed, prefixes, infixes, circumfixes) from a word in order to obtain a word stem. For example, using stemming, the words "fishing", "fished", and "fisher" are reduced to the stem "fish". The stem need not be a word, however: in the Porter algorithm [35], for example, "argue", "argued", "argues", "arguing", and "argus" reduce to the stem "argu".

LEMMATIZATION The process of grouping together the inflected forms of a word so they can be analyzed as a single item, identified by the word's lemma. In particular, in computational linguistics, lemmatization is the algorithmic process of determining the lemma of a word based on its intended meaning. For example, the words "good", "better" and "best" are lemmatized to "good" since all the words have a similar meaning.

PUNCTUATION REMOVAL The step in which all the punctuation characters are removed, since they do not bring any useful information. This step is usually performed during tokenization.

NUMBER SUBSTITUTION OR REMOVAL The step in which all the numbers are converted to their textual representation or removed (when they do not bring any useful information to the discrimination task).

STOP WORDS REMOVAL The step in which are removed all the stop words, i. e. the words that contribute little to overall meaning, given that they are generally the most common words in a language (e. g. "the", "and", "a", etc.).

2.3.2 *Bag Of Words*

The simplest model used to vectorize texts is *Bag of Words* (*BoW*). This model analyses all the documents in a collection, finds all the words in it and builds a vocabulary $\mathcal{V}$ of unique words. Then it represent each document of the collection through an integer vector which dimension is equal to $|\mathcal{V}|$, where each element x_i of the vector is equal to the frequency in the document of the *i-th* word of $\mathcal{V}$ [46].

2.3.3 *Term Frequency - Inverse Document Frequency*

One of the main limitations of Bag of Words model is that the resulting vectors are highly pronunced in the direction of the words that are very frequent in a document but not useful in the discrimination process (e. g. *'the'*, *'and'*, *'that'*, ...). In order to overcome this limitation, an extension of the BoW model can be used: the *Term Frequency - Inverse Document Frequency* (*TF-IDF*) model. With TF-IDF, every element x_i of the BoW vector is multiplied by a normalizing factor, that depends on the number of documents in which the word associated with x_i appears. In particular, this normalizing factor, in its classical form is equal to $\log(\frac{\mathcal{N}}{d_i})$, where $\mathcal{N}$ is the cardinality of the documents collection, while d_i is the number of documents in which the *i-th* word of $\mathcal{V}$ appears. Thanks to this normalizing factor, terms like "stop words" are highly penalized, while very discriminative terms, i. e. terms that appears frequently only in few documents, are highly favoured [46].

2.3.4 *Word2Vec*

Both TF-IDF and BoW models see documents as an unordered collection of words, losing information on the documents' semantic. More specifically, it can happen that two documents composed of very different words, i. e. having a very different vector representation, are actually semantically similar. Moreover, both TF-IDF and BoW models generate high-dimensional and sparse vectors: this can sometimes affect the performance of a classifier.
A step forward towards a vectorization able to take into account documents' semantic is given by the *Word2Vec* model, presented in [31]. Using an architecture based on neural networks, this model allows to obtain a dense vector representation of words, so that semantically similar words have similar vectorizations. The intuition on which the architecture lays down is that similar words typically appears in similar contexts. Such intuition can be implemented through two different neural network architectures: the first one, called Continuous Bag of Words (*CBoW*), uses as input a specific number of terms (extracted from each document and named as *context*) before and after a word w_i, and tries to predict w_i; on the contrary the second architecture, called Skip-Gram, starts from a word w_i and tries to predict its context. Of course in both cases, what we're interested in are not the predictions of the network but the weights learned by it, that represents the vectorization of each word.
An improved version of Word2Vec model, both in terms of efficiency and quality, can be found in [30].

2.3.5 *Doc2Vec*

The nature of Word2Vec model doesn't allow to obtain documents vectorization directly, since it is used basically as a word vectorizer; thus, some strategies are needed in order to obtain a document vectorization starting from word vectorizations, such as averaging or concatenating all the vectorized words belonging to the same document.
A model that takes inspiration from Word2Vec architecture and allows to directly obtain a vector representation of documents is the so-called *Doc2Vec* model, presented in [25]. Similarly to Word2Vec, Doc2Vec can be implemented through two different neural network architectures too: the first one, called Paragraph Vectors - Distributed Memory (*PV-DM*), uses as input a context of words plus a token representing the document

from which the context is taken, and tries to predict the next word after the context; on the other hand the second architecture, called Paragraph Vectors - Distributed Bag of Words (*PV-DBoW*), uses as input just the document token, with the objective of predicting random words belonging to the document itself. As in the case of Word2Vec architectures, even with Doc2Vec what we're interested in are the weights learned by the network, that represents the vectorization of each document.

2.3.6 *Topic2Vec*

A completely different approach to text vectorization can be derived starting from the so-called Latent Dirichlet Allocation (*LDA*) model, presented in [3]. Through this generative statistical model, sets of observations can be explained by unobserved groups that explain why some parts of the data are similar. In the case of texts, where the observations are words collected into documents, this model posits that each document is a mixture of a small number of topics and that each word's presence is attributable to one of the document's topics. In this way, each document can be transformed in a vector, whose size is equal to the number of topics, representing its probability distribution over the topics.

2.4 CLASSIFICATION METHODS

After texts have been preprocessed and vectorized, they become ready to be given to a classifier that learns from them trying to reduce the misprediction error. The term "classifier" may refer to a single learning algorithm, as well as to multiple learning algorithms combined in some smart way.
In the following we present a taxonomy of the main algorithms, trained in a supervised manner, involved in text classification.

2.4.1 *Linear Classifiers*

The goal of classification is to use an object's characteristics to identify which class it belongs to. An object's characteristics are also known as feature values and are typically presented to the machine learning algorithm in a vector called a *feature vector*. A linear classifier achieves its goals by making a classification decision based on the value of a linear combination of the feature values. In particular, if the input feature vector to the classifier is a numeric vector **x**, then the output score is:

$$y = f(\mathbf{w} \cdot \mathbf{x}) = f\left(\sum_j w_j x_j\right) \tag{2.1}$$

where **w** is a real vector of weights and f is a function that converts the dot product of the two vectors into the desired output. The weight vector **w** is learned from a set of labeled training samples. Often f is a simple function that maps all values above a certain threshold to a specific class and all other values to another class. A more complex f might give the probability that an item belongs to a certain class. A linear classifier is often used in situations where the speed of classification is an issue, since it

is often the fastest classifier, especially when **x** is sparse.
In the field of text classification, linear classifiers such as *Logistic Regression* [17], *Multinomial Naive Bayes* [23] and *Support Vector Machines* [12] are really used, since they work very well when inputs have a large number of dimensions, which is a typical characteristic of vectorized documents.

2.4.2 *Decision Trees*

Another way of performing classification is the usage of a *decision tree* classifier, also called *classification tree*. With this classifier, a predictive model is built through a tree structure, in which leaves represent class labels and branches represent conjunctions of features that lead to those class labels. More specifically, each interior node corresponds to one of the input variables; there are edges to children for each of the possible values (or intervals) of that input variable. Each leaf represents a class label or class label distribution given the values of the input variables represented by the path from the root to the leaf.
In order to build a good performance classification tree, at each node one input variable is chosen to split training examples into distinct classes as much as possible. The selection of the best input variable upon which splitting, is performed through a purity measure such as the one used in *ID3* [37], called information gain, which increases with the average purity of the subsets that a split produces.
Beyond ID3, other algorithms are commonly used to generate decision trees: see *C4.5* [36] and *CART* [6].

2.4.3 *Ensemble Methods*

Supervised learning algorithms are most commonly described as performing the task of searching through a hypothesis space to find a suitable hypothesis that will make good predictions with a particular problem. Even if the hypothesis space contains hypotheses that are very well-suited for a particular problem, it may be very difficult to find a good one. Ensemble methods combine multiple hypotheses to form a (hopefully) better hypothesis, i. e. use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone.
In the following we present some of the most effective ensemble methods: *Bagging*, *Boosting*, *Stacking*.

Bagging

Bootstrap aggregating, also called *Bagging* [4], is an ensemble method designed to improve the stability and accuracy of machine learning algorithms. Given a training set $\mathcal{D}$ of size n, bagging generates m new training sets of size n' (not necessarily equal to n) by sampling from $\mathcal{D}$ with replacement, i. e. reinserting every instance in $\mathcal{D}$ after having been sampled. Then m identical models are trained using the previously created m datasets and finally combined through a majority voting based on their outputs (for

classification). Beyond improving the accuracy, using bagging usually reduces variance and helps avoid overfitting.
An extension of the bagging method can be found looking at one of the most famous ensemble learning methods: the so-called *Random Forests* [5], widely used in many learning tasks. Basically, in the case of classification, this method operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (for classification) of the individual trees. Each individual tree is built using a random subset of the available features when splitting a node; then it is trained on a random subset of the original training set and finally combined with other trees, in the prediction phase, through a majority voting mechanism.

Boosting

Boosting [47] incrementally builds an ensemble of models by training each model with a dataset where the weights of instances are adjusted according to the errors of previous predictions. The main idea is forcing the models to focus on the instances which are hard. Finally, the models are combined through a majority voting mechanism (for classification), which assigns more importance to those learners whose performance was better. A very famous boosting algorithm is *XGBoost*, which has recently been dominating applied machine learning competitions, pushing the limits of computing power for boosted trees algorithms [11].

Stacking

Stacked Generalization (also called *Stacking*) [56] combines multiple predictive models to generate a new model. While Bagging and Boosting are used to combine models of the same type, Stacking typically combines models of different types. In particular, as shown in **Figure** 2.1[1], each of the models (named as *base models*) is trained on a complete training set, then a stacked model (named also as *meta-learner*) is trained on the outputs of the base models as features. Most of the time the stacked model outperforms each of the individual models due to its smoothing nature and ability to highlight each base model that performs best and discredit each base model that performs poorly. For this reason, Stacking is most effective when the base models are significantly different. As regards the choice of a meta-learner, an arbitrary model can be theoretically used, although, in practice, a Logistic Regression model is often used as the combiner [51].

2.5 MULTI-LABEL METHODS

All the existing methods for multi-label classification can be grouped into two main categories: *Problem Transformation Methods* and *Algorithm Adaptation Methods* [52]. Problem transformation methods are those methods that transform the multi-label classification problem either into one or more single-label classification or regression problems (i. e. adapt your data to the algorithm): known examples, not presented in this work, are

1 Image taken from `http://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/`

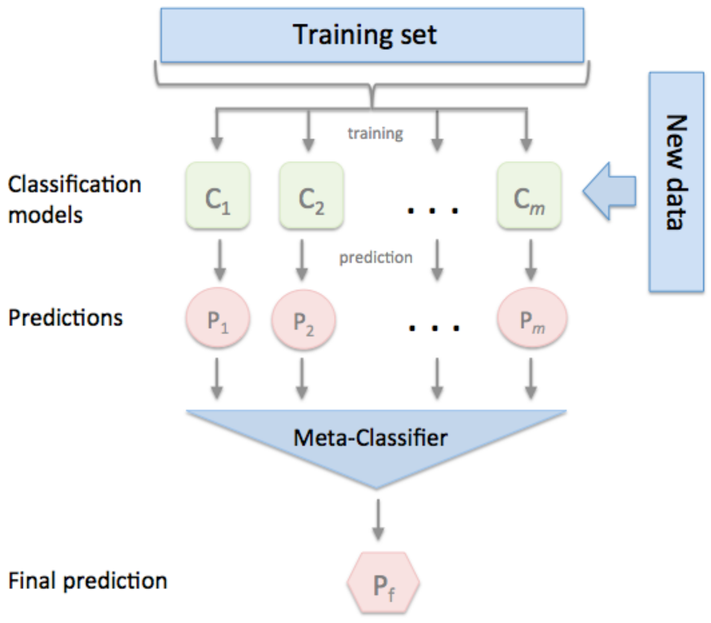

Figure 2.1: How stacking works. Every classification model is trained on the complete training set. Its predictions are used as input feature for a meta-classifier, along with the predictions of other classification models, in order to output a final prediction based on the "advices" given by the base classifiers.

the *Label Powerset* method, where each different combination of labels becomes a new single-class label in a multi-class problem; the *Calibrated Label Ranking* method [13], that uses binary classifiers to discriminate between any possible label pair; the *RAkEL* method [53], which tries to overcome the main drawbacks of Label Powerset.
Algorithm adaptation methods, on the other hand, are those methods that extend specific learning algorithms in order to handle multi-label data directly (i. e. adapt your algorithm to the data): known examples, not presented in this work, are *MLKNN* [58] and *BPMLL* [59], which are multi-label extension respectively of *K-Nearest Neighbours* and *Neural Networks* methods.
In the following we present two of the most known transformation methods for multi-label classification, named as *Binary Relevance* and *Classifier Chains*.

2.5.1 *Binary Relevance*

The most famous approach used to perform multi-label classification and belonging to the category of problem transformation methods is by far the Binary Relevance (*BR*) method [60]. Given a number of labels k, this techniques consists in training k classifier, each for one label, taking the training instances in which the labels appear as positive and all the others as negative. Therefore this method ends with k binary models, each of which is able to say if a given instance has a specific label or not. So, given an unseen sample, the combined model predicts all the labels for which the respective classifiers predict a positive result.
The Binary Relevance method present two main drawbacks: it does not consider label correlation information and it increases the imbalance between labels. As regards label

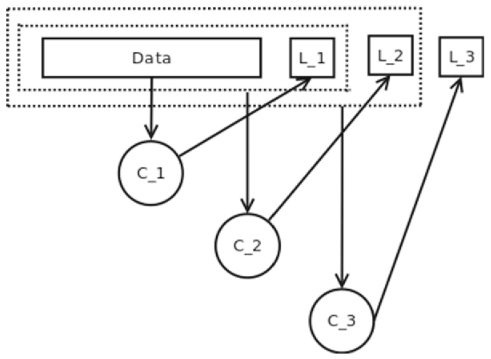

Figure 2.2: How classifier chains works. A first classifier is trained just on the input data and then each next classifier is trained on the input data plus all the previous classifiers' predictions in the chain.

correlation information, the Binary Relevance method builds a set of classifiers (one for each label) which are completely independent, so the prediction made by one of them does not influence how the others make their work. This means that labels are assumed to be fully independent, which might not be the case: indeed, there are use cases where the presence of a certain label could determine whether another one is also more likely to be present or not. Regarding the increasing of imbalance between labels, the problem is that Binary Relevance takes as positive only the instances in which a certain label appears and as negative all other samples, changing the original label distribution and therefore affecting the representation of the considered label (of course, the situation is even worse if the label is already a minority label).

2.5.2 *Classifier Chains*

In order to overcome the main drawbacks related with the Binary Relevance method (see section 2.5.1), a problem transformation method called Classifier Chains (*CC*) has been proposed in [39]. Through this algorithm, it is possible to combine the computational efficiency of the Binary Relevance method while still being able to take the label dependencies into account for classification. What this model does, is to learn k classifiers (where k is the number of labels of the problem) as in Binary Relevance method, with the difference that all the classifiers are arranged in a chain. In particular, as shown in **Figure** 2.2 [42], each binary classifier is trained using as input features the whole training set and all the predictions of those classifiers that were before it in the chain. Hence, the inter-label dependency is preserved but the result can vary for a different order of chains. This is due to the fact that each classifier receives only the predictions from previous classifiers in the chain: therefore, it can exploit label correlation only for those labels whose binary classifier is in a previous position in the chain. In order to solve this problem and increase accuracy, it is possible to use an ensemble of classifier chains (*ECC*), in which several CC classifiers are trained with random order of chains (i. e. . random order of labels) on a random subset of data set. Labels of a new instance are predicted by each classifier separately. After that, the total number of predictions or "votes" is counted for each label: the label is accepted if it was predicted by a percentage of classifiers that is bigger than some threshold value.

2.6 MULTI-LABEL METRICS

In order to measure the classification performances of a trained classifier, is typically necessary to feed it with an evaluation data set, composed of new unseen instances, and extract some performance metrics based on the goodness of its predictions.
Multi-label classification requires different metrics than those used in traditional single-label classification. Let $\mathcal{D}$ be a multi-label evaluation data set, consisting of $|\mathcal{D}|$ multi-label examples (x_i, Y_i), $i = 1, ..., |\mathcal{D}|$. Let $\mathcal{H}$ be a multi-label classifier and $\mathcal{Z}_i = \mathcal{H}(x_i)$ be the set of labels predicted by $\mathcal{H}$ for example x_i. We define *Exact Match Accuracy (EMA)* the metric given by:

$$\mathrm{EMA} = \frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|}[\![\mathcal{Z}_i = Y_i]\!]$$

where $[\![\cdot]\!]$ function is equal to 1 if its argument is true, 0 otherwise. Intuitively, *EMA* can be regarded as a multi-label counterpart of the traditional accuracy metric, since it evaluates the fraction of correctly classified examples, and tends to be overly strict especially when the size of label space is large. While EMA works by evaluating the learning system's performance on each test example separately, returning the mean value across the test set, other evaluation metrics work by evaluating the learning system's performance on each class label separately, and then returning the averaged value across all class labels. In particular, given the j-th class label y_j, four basic quantities characterizing the binary classification performance on this label can be defined based on $\mathcal{H}(\cdot)$:

$$\begin{aligned}
\mathrm{TP}_j &= |\{\mathbf{x}_i \mid y_j \in Y_i \wedge y_j \in \mathcal{H}(\mathbf{x}_i), 1 \leqslant i \leqslant |\mathcal{D}|\}| \\
\mathrm{FP}_j &= |\{\mathbf{x}_i \mid y_j \notin Y_i \wedge y_j \in \mathcal{H}(\mathbf{x}_i), 1 \leqslant i \leqslant |\mathcal{D}|\}| \\
\mathrm{TN}_j &= |\{\mathbf{x}_i \mid y_j \notin Y_i \wedge y_j \notin \mathcal{H}(\mathbf{x}_i), 1 \leqslant i \leqslant |\mathcal{D}|\}| \\
\mathrm{FN}_j &= |\{\mathbf{x}_i \mid y_j \in Y_i \wedge y_j \notin \mathcal{H}(\mathbf{x}_i), 1 \leqslant i \leqslant |\mathcal{D}|\}|
\end{aligned}$$

In other words, TP_j, FP_j, TN_j and FN_j represent the number of true positive, false positive, true negative, and false negative test examples with respect to y_j. Based on the above four quantities, most of the conventional multi-label metrics can be derived accordingly:

$$\mathrm{PRECISION}_{\mathrm{micro}} = \frac{\sum_{j=1}^{|\mathcal{L}|}\mathrm{TP}_j}{\sum_{j=1}^{|\mathcal{L}|}\mathrm{TP}_j + \mathrm{FP}_j} \qquad \mathrm{PRECISION}_{\mathrm{macro}} = \frac{1}{|\mathcal{L}|}\sum_{j=1}^{|\mathcal{L}|}\frac{\mathrm{TP}_j}{\mathrm{TP}_j + \mathrm{FP}_j}$$

$$\mathrm{RECALL}_{\mathrm{micro}} = \frac{\sum_{j=1}^{|\mathcal{L}|}\mathrm{TP}_j}{\sum_{j=1}^{|\mathcal{L}|}\mathrm{TP}_j + \mathrm{FN}_j} \qquad \mathrm{RECALL}_{\mathrm{macro}} = \frac{1}{|\mathcal{L}|}\sum_{j=1}^{|\mathcal{L}|}\frac{\mathrm{TP}_j}{\mathrm{TP}_j + \mathrm{FN}_j}$$

$$F1_{\text{micro}} = \frac{\sum_{j=1}^{|\mathcal{L}|} 2TP_j}{\sum_{j=1}^{|\mathcal{L}|} 2TP_j + FP_j + FN_j} \qquad F1_{\text{macro}} = \frac{1}{|\mathcal{L}|} \sum_{j=1}^{|\mathcal{L}|} \frac{2TP_j}{2TP_j + FP_j + FN_j}$$

In these metrics, *micro* stands for micro-averaged and *macro* for macro-average: conceptually speaking, micro-averaging assumes "equal weights" for example while macro-averaging assumes "equal weights" for labels.

2.7 imbalanced inputs

A problem that can arise in various classification tasks is the so-called *imbalanced class distribution* problem. This is a scenario where the number of observations belonging to one class is significantly lower than those belonging to the other classes. In this situation, the predictive model developed using conventional machine learning algorithms could be biased and inaccurate. This happens because machine learning algorithms are usually designed to improve accuracy by reducing the error. Thus, they do not usually take into account the class distribution or balance of classes.
The two most common techniques used to deal with class imbalance are:

under-sampling Strategy that tries to rebalance data by removing samples from the majority class. Besides the obvious strategy of removing samples at random (*Random Under-Sampling*), different undersampling techniques have been proposed [19], typically using advanced methods to choose which samples remove, as in the case of *Edited Nearest Neighbour* [55], which removes samples whose class label differs from the label of at least half of its k nearest neighbors.

over-sampling Strategy that tries to rebalance data by generating new samples in the classes which are under-represented. Generated samples can be just random duplicates from the minority class (*Random Over-Sampling*), or new synthetic instances as is the case using *SMOTE* [10], which creates entries that are interpolations of the minority class.

3 THE SCENARIO

In this chapter we illustrate the scenario of our work, presenting the task we were assigned. We then overview previous studies in solving similar problems and describe our approach and how it differs from those. Finally, we present the structure of the data and the process of its analysis.

3.1 MEDIASET METADATA

The Mediaset Group[1] is one of the major Italian players in the field of communication and multimedia distribution. Multimedia production and distribution play a significant role inside the company, and the interests of the company are both on traditional mediums (e. g. television, radio) and on the front of online multimedia streaming. The field of online multimedia streaming is highly competitive and, to improve the quality of their service, Mediaset uses recommender systems that rely on metadata to provide relevant recommendations to the user. Within the company, the role of metadata has relevance in different sectors in addition to recommendation systems. Metadata is used as support for the editorial sector of the company, providing strategic insights to the production of multimedia contents, and to the marketing division, allowing for more effective advertising. Given the importance of metadata, Mediaset Strategic Marketing has designed an articulate structure for the description of multimedia contents. The metadata range is quite large and covers information like the pathemic genre, the city or area in which the multimedia content takes place, or descriptions designed to support the marketing division.

3.2 THE SCENARIO

In our scenario, the current process of assigning some of the metadata to multimedia contents is performed by human experts who watch the entire content and compile a form designed for the retrieval of metadata information. This process slows down the process of extraction of metadata for new contents. Our task was to verify whether it was possible to speed up the process of tagging by providing automatic suggestions during the process of tagging of metadata for movies. We were asked to focus our attention on a metadata called *mood* which is an information that describes the underlying state of mind which arises from watching a content, chosen between twenty possible values (see appendix A).

1 Company's website: `https://www.mediaset.it/`

3.3 PREVIOUS STUDIES

The problem of automatic classification of multimedia contents is popular in the machine learning field. Among the studies which covered this topic, various were interested in movies. A typical objective for movies was their classification into classical movie genres. Genre and *mood* present some correlation since movies belonging to the same genre may be linked to some specifics sensation in the viewer which can be traced to one or more *mood*. For this reason, we overviewed previous works in these research segments. The task of multimedia classification can be divided, depending on the nature of the features extracted for the content, into visual-based, audio-based or text-based approaches [7]. The most used approach is the video-based due to the fact that human receive much of their information of the world through vision. Video features are typically formed by a frame or a collection of frames. Features can be extracted from videos with different techniques like color and motion analysis, techniques for detecting shots (i. e. scenes which can be segmented to represent an action that is going on in the video like "two people talking") and object detection inside a video. Audio-based approaches use audio features to approximate the human perception of sound. Audio features can lead to three layers of audio understanding: low-level acoustics, such as the average frequency for a frame, mid-level sound objects, such as the audio signature of the sound a ball makes while bouncing, and high-level scene classes, such as background music playing in certain types of video scenes. That information in a movie can provide an understanding of a scene: for example, a specific kind of background music may play during an action scene, and a different kind, in a romantic setting. Audio features can also come from speech sequences obtained from automatic speech recognition algorithms [43]. In the paper *Movie Genre Classification By Exploiting Audio-visual Features Of Previews* [38], audio and video features extracted from movie previews were combined to classify movies into a small set of genres, initially discriminating between action and non-action movies by estimating the visual disturbance and average shot length, and using audio and color information to further classify movies. Finally in the text-based approach, the least popular approach, text features are extracted from the movie in the form of subtitles [8] or OCR software [16] which retrieve text directly from the video source. The text is then transformed in a feature vector using various vectorization techniques like the *Bag-of-Words* model, on top of which a classification architecture is built.

3.4 OUR APPROACH

For our task, following the requirements of our stakeholder, we used the text-based approach and thus we fall in the last category of section 3.3. The described studies which relied on text-based features presented some issues. In fact the use of OCR are prone to error in the text extraction and, moreover, the text features of subtitles are in the end only composed by dialogues, which are not suited to capture much of what is happening in a scene and so, in the entire movie. For this reasons we differentiate in the selection of text inputs, by using movie summaries, which are more appropriate to describe the sequence of events throughout the whole movie, which is a requirement for capturing the essence of the *mood* metadata. Some of the possible *moods* are in fact relative to situations which may happen in various instants of the movie (e. g. the *mood* "Surprising" is associated to the presence of a shocking event which typically happens in the last bits of a movie). Since in each movie, more than one *mood* may emerge, our task belongs to the family of multi-label classification.

3.5 DATA EXPLORATION

To investigate the possibility of implementing an automatic *mood* classification system, we were provided with a dataset. The dataset contains 6480 entries, each of which provides the following information regarding a specific multimedia content:

- ▷ **ID** → Unique identifier of the content.
- ▷ **TITLE** → Content title in the italian language.
- ▷ **CONTENT_TYPE** → Category of the content. Can assume three different categorical values: "Movie", "TV Series" and "TV Show".
- ▷ **PLOT** → Short description of the content, written in the italian language. For movies and TV series it coincides with content plot, while for TV shows it just coincides with show description.
- ▷ **PRODUCTION_YEAR** → Year in which the content has been produced.
- ▷ **WIKI_LINK** → URL to the italian Wikipedia page of the content.
- ▷ **IMDB_LINK** → URL to the IMDb page of the content.
- ▷ **MOOD** → Set of *moods* characterizing the content. Coincides with one or two categorical values belonging to a superset of 20 *moods*. Examples of *moods* are *Evasive*, *Disturbing*, *Poignant* (see appendix A for a complete description of *moods*).
- ▷ **CASTING** → Information about the cast partecipating in the movie, TV serie or TV show, expressed in a XML representation.

3.5.1 *Moods Distribution*

Our goal was to predict was the *mood* using the other inputs. We were also asked to face the problem only for movie instances, i. e. those entries for which the field **CONTENT_TYPE** is equal to "Movie". The number of useful entries was therefore reduced to 3940.
We analyzed the *moods* distribution among all the examples and noticed a considerable imbalance in *moods* occurrences. As can be seen in **Figure** 3.1, an high discrepancy is present between the most frequent *moods* (e. g. "Breathtaking", appearing 855 times) and the less frequent ones (e. g. "Comfortable", appearing only 62 times). The *mood* named "Angry" is not present at all, since it appears only in entries whose field **CONTENT_TYPE** is equal to "TV Show".

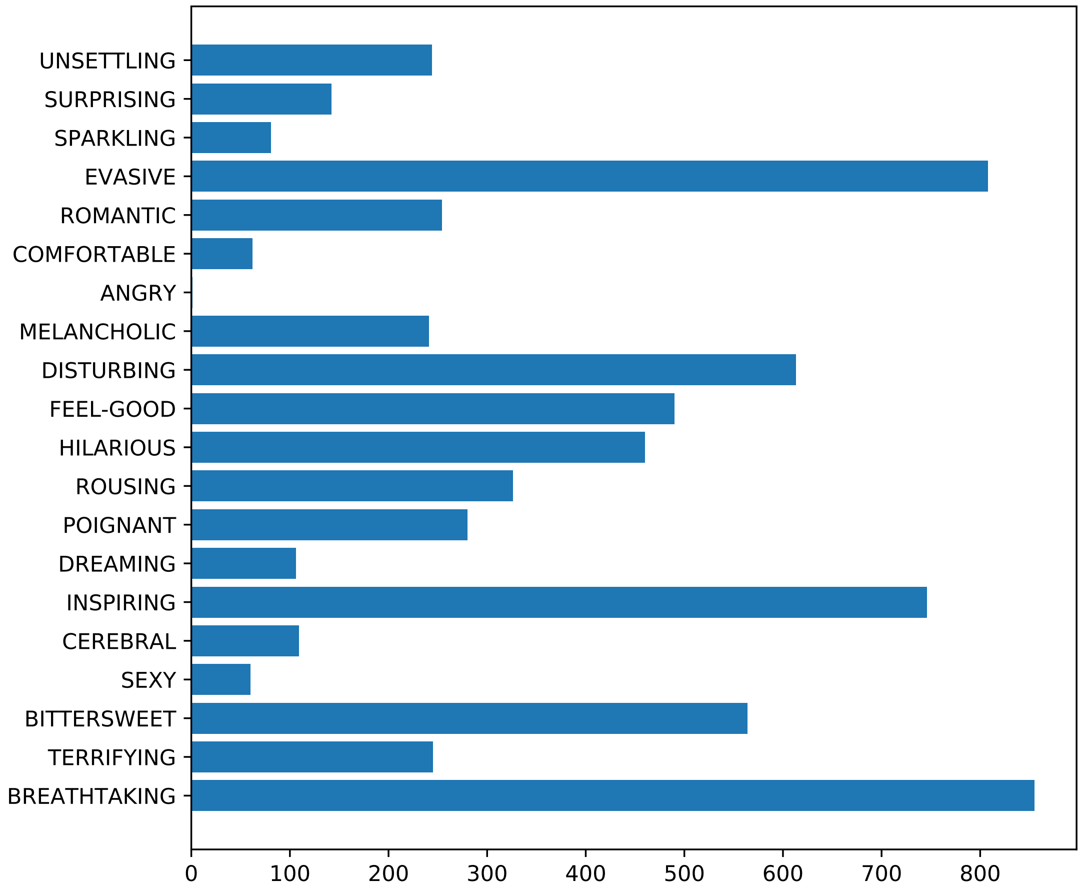

Figure 3.1: *Moods* occurrence distribution

To further analyze the *moods* and to try to have a better insight into the imbalance problem, we computed for each *mood* the so called imbalance ratio, presented in [57]. Given the number of positive and negative examples for a specific *mood*, i. e. the number of examples for which the *mood* is or is not present, the imbalance ratio can be computed as the ratio between the maximum and the minimum of such values. As can be seen in **Figure** 3.2, for less frequent *moods* the imbalance ratio goes up to 65, while for most frequent *moods* is below 5. The mean imbalance ratio has been computed too, and it is approximately equal to 21.

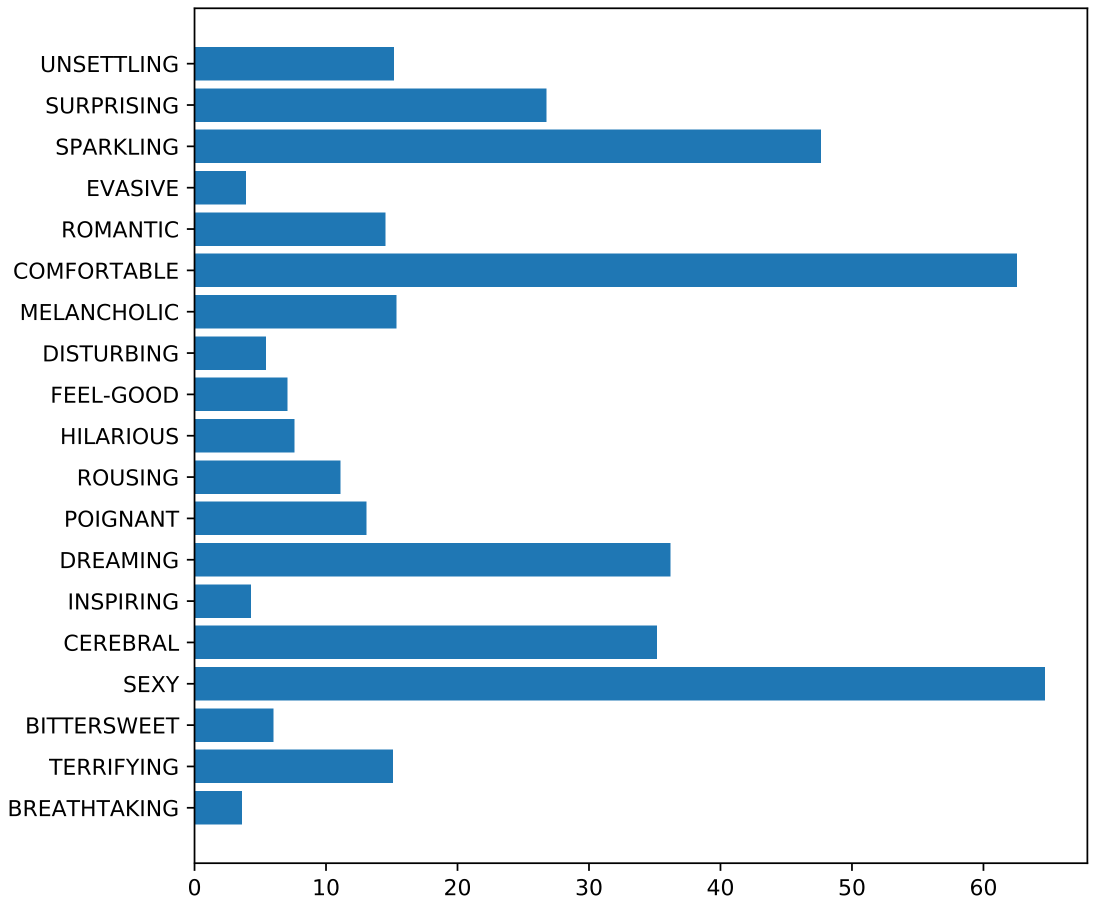

Figure 3.2: *Moods* imbalance ratio.

3.5.2 *Moods Correlation*

Given that most of the examples in the dataset are often associated with two *moods*, we decided to analyze the co-occurrence of each pair of *moods*, in order to possibly discover a correlation between some *moods*. As illustrated by **Figures** 3.3 and 3.4, we discovered some *mood* pairs that are more frequent than others, and how some pairs are instead totally absent.

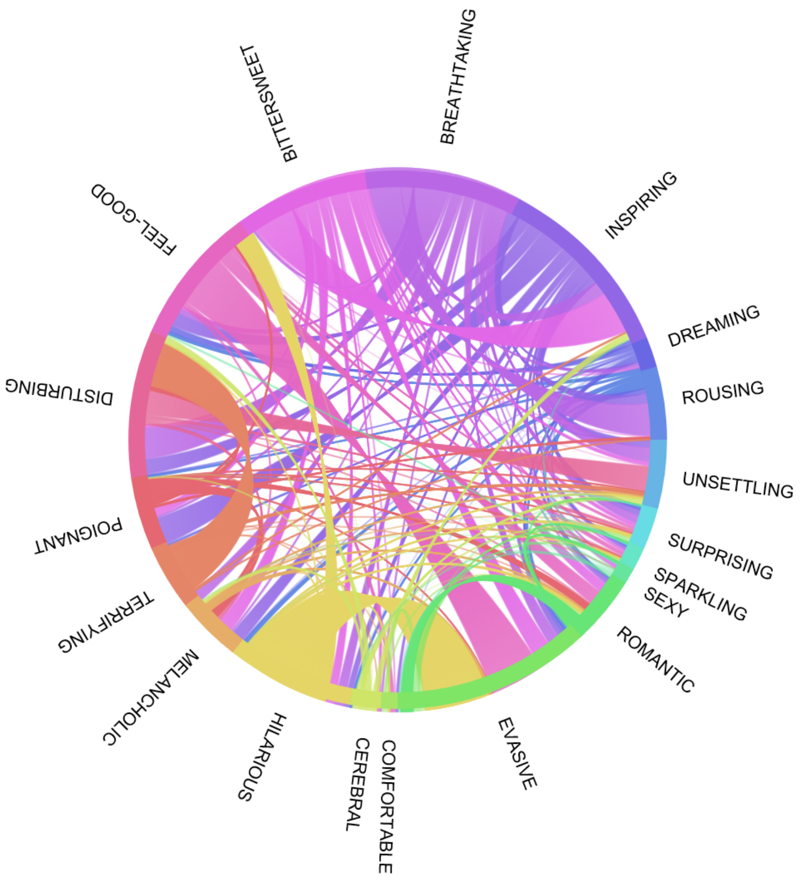

Figure 3.3: Chord diagram showing *moods* co-occurrences.

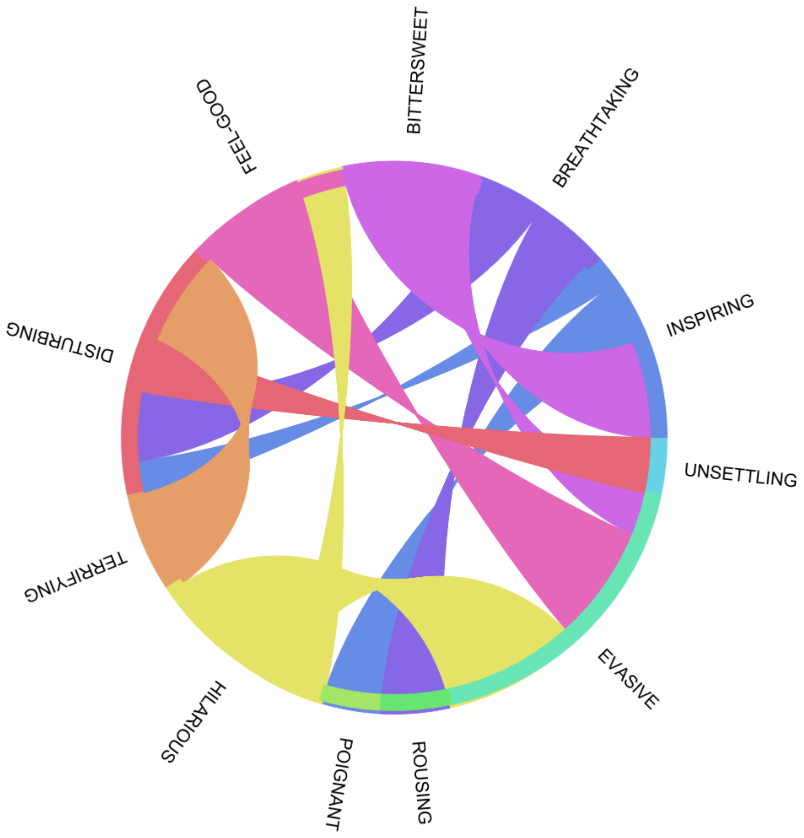

Figure 3.4: Chord diagram showing *moods* co-occurrences greater than 50.

This correlation may be due to the inherent nature of *moods* which, albeit having different meanings, both describes the latent state of mind of a movie (e. g. “Breathtaking” and “Rousing” for action movies), or may be due to the bias of human taggers, who can interpret two different *moods* as *moods* with similar/overlapping meaning, therefore selecting both of them as the correct ones (e. g. “Evasive” and “Feel-good”).
Based on this analysis, we decided to have a better insight into *mood* correlation by computing the Pearson correlation coefficient for each pair of *moods*. Pearson correlation measures the linear correlation between two random variables, that in our case coincide with *moods*. Given a pair of random variables, the correlation coefficient can be computed as the ratio between the covariance of the variables and the product of their standard deviations. Such coefficient has a value between $+1$ and -1, where $+1$ means total positive linear correlation and -1 means total negative linear correlation (0 means no linear correlation).
As can be seen from **Figure** 3.5, the analysis highlights moderate correlation between some *moods*. Most representative examples are probably: (i) “Terrifying” coupled with “Disturbing”, which expresses a marked positive correlation; (ii) “Evasive” coupled with “Inspiring”, which expresses a marked negative correlation.

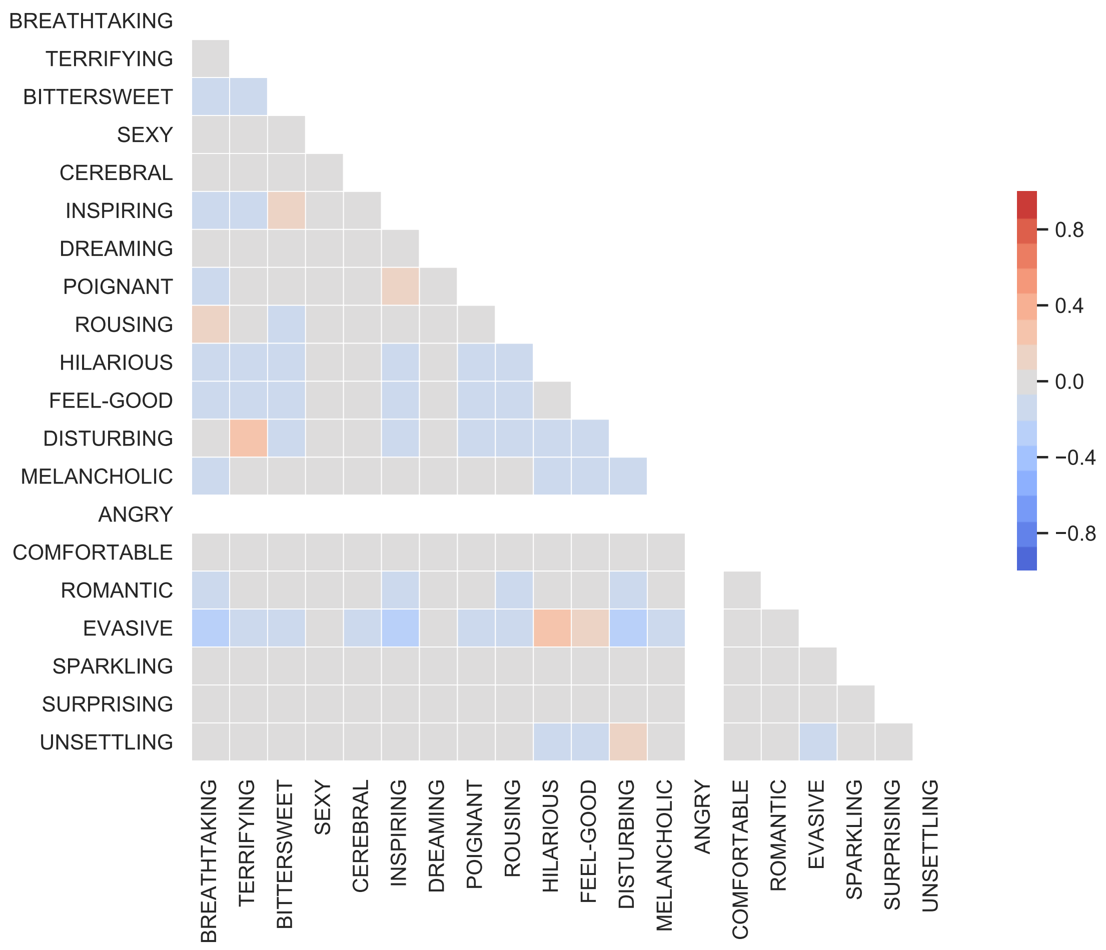

Figure 3.5: Pearson correlation for *moods*.

3.5.3 *Analysis of the others Features*

We analyzed the feature **PLOT** related with movie plots. We discovered that all the movie plots are very short, with an average of around 65 words, and thus very poor in terms of discriminative power they can bring to the mood prediction task.
Analyzing the other features, we noticed that the only one really useful for the *mood* prediction is **CASTING**. This finding is coherent to the study in [45], which shows in fact how domain specific information, used in association with text, can help in improving the performance of text classification methods in a multi-label setting, as is the case with our problem. Therefore, we calculated the contribution that each actor has given to each movie, i. e. the number of times each actor has played a role in movies with a specific *mood*. At the end, we observed that some actors have a strong characterization toward some specific *moods*. **Figures** 3.6, 3.7 and 3.8 highlight this relation for three actors. **Figure** 3.6 refers to *Jack Black* (comedy actor), **Figure** 3.7 refers to *Al Pacino* (action and drama actor) and **Figure** 3.8 refers to *Hugh Grant* (romantic comedy actor), who show a strong contribution respectively toward *moods* "Feel-good", "Breathtaking" and "Romantic".

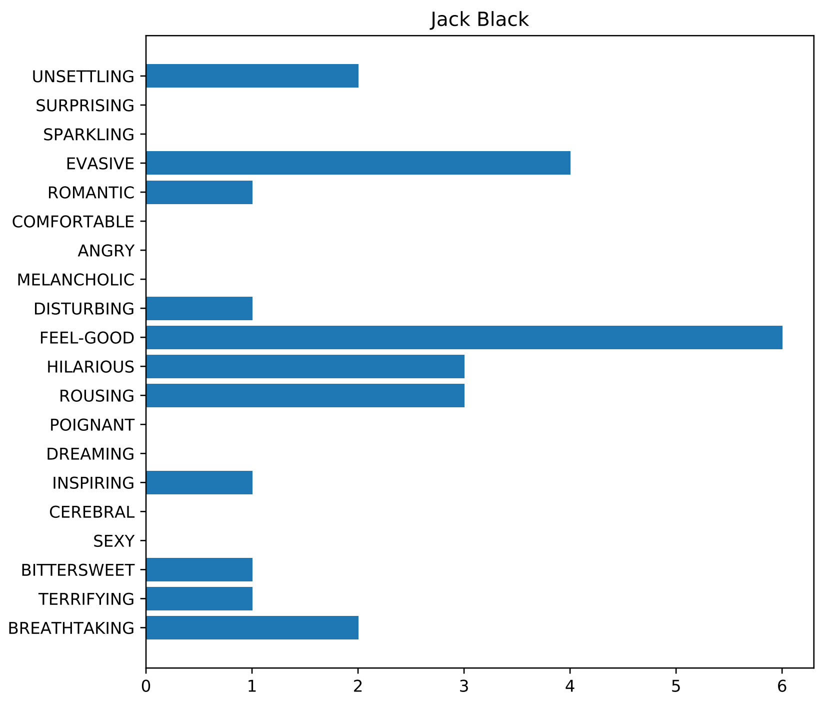

Figure 3.6: Characterization of comedy actor Jack Black.

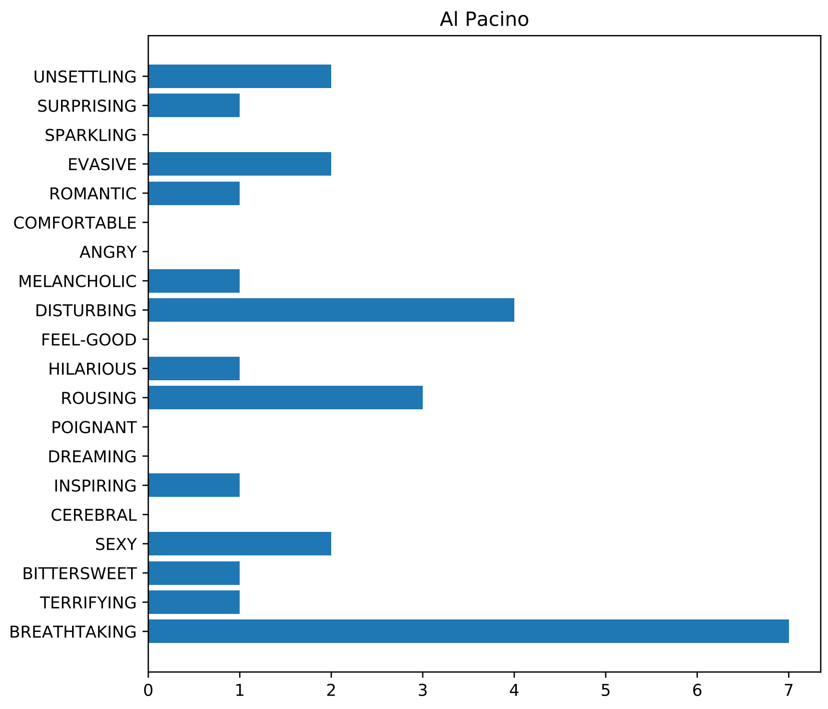

Figure 3.7: Characterization of action and drama actor Al Pacino.

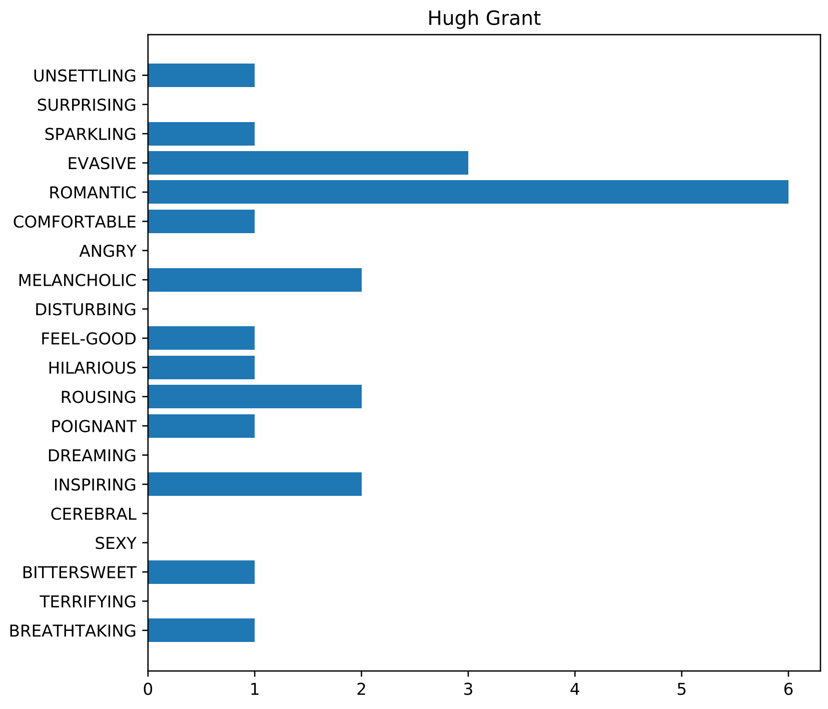

Figure 3.8: Characterization of romantic comedy actor Hugh Grant.

4 OUR ARCHITECTURE

In this chapter we describe the steps performed for the multi-label classification of movies. We start by describing the data preprocessing step, characterized mainly by the extension of textual information present in our dataset. We move on presenting an overview of the tested input vectorization and classification methods, followed by a description of the adopted resampling techniques. We then describe the selection step, in which all the vectorized inputs and resampling techniques were subjected to some statistical tests, in order to select only the relevant ones for subsequent experiments. Finally, we describe the multi-label classification step, presenting all the tested approaches, both those classical and those representing the *state-of-the-art*.

4.1 DATA PREPROCESSING

The first step characterizing our architecture was the preprocessing of the information present in the dataset. As described in section 3.5, we were interested only in classifying *moods* for movies, disregarding TV series and TV shows: this allowed us to delete from the dataset all the entries for which the feature **CONTENT_TYPE** was equal to "TV Serie" or "TV show". From the remaining entries, we decided to remove features **ID**, **TITLE** and **PRODUCTION_YEAR**, since they didn't provide any useful information to the classification problem. Finally, we removed *mood* "Angry" from the set of possible classification outcomes, since it never appeared in movie entries.

4.1.1 *Summaries Gathering*

As already stated in section 3.5.3, the movie plots present in the dataset, associated with feature **PLOT**, were too short and information-poor for being useful to the classification task. We therefore decided to retrieve more suitable and complete textual descriptions for each movie, replacing plots with summaries. In particular, using the URLs relative to feature **IMDB_LINK** and **WIKI_LINK**, we were able to scrape related sites looking for summaries. Whereas most of the available tools used in text processing are optimized for the english language, we decided to replace the already present (italian) plots with english summaries. Using the IMDb source allowed us to directly access the english page of the movie, while the Wikipedia source allowed us to only reach the italian page of the movie. To cope with that, we looked for the relative link to the english version of the page and, if not present, we resorted to the expedient of translating the italian summary with Google Translate. Once we gathered the english summaries from both resources, we were able to associate for each movie in the dataset the freshly obtained summaries, keeping only the longest one when both sources supplied a summary. All the entries in the dataset with no textual description related to feature **PLOT** were fixed using the scraping strategies above-mentioned (see section 4.1.1). All the entries for which no URL related to feature **IMDB_LINK** was available were fixed using the

URL related to feature **WIKI_LINK**, and vice versa. Therefore, all the entries with no available URL both for feature **IMDB_LINK** and **WIKI_LINK** were removed from the dataset.

4.2 INPUT VECTORIZATION

The input for our classification task was mainly coming from the summaries we retrieved as explained in section 4.1.1. Those summaries were vectorized using all the methods presented in section 2.3 and a custom vectorization method exploiting Google's pre-trained *Word2Vec* vectors. In addition to the input representation made with summaries, we also exploited the information of the cast provided for each movie to propose a different input representation called *Cast2Vec*.

4.2.1 *Standard Vectorization Methods*

In the following we give a quick overview of the standard vectorization methods we used to convert summaries into numerical vectors.

BOW Represents every text through an integer vector. Each element of the vector is associated with a word belonging to a known vocabulary and indicates the number of occurrences of the associated word in a text. This vectorization generates high-dimensional and sparse vectors, which tends to lose information on the text semantic.

TF-IDF Extension of the BoW model, in which every element of the resulting vector is multiplied by a normalizing factor, that depends on the number of texts in which the word associated with the element appears. This vectorization, as is the case with BoW, generates high-dimensional and sparse vectors, which tends to lose information on the text semantic.

DOC2VEC Represents every text through a dense vector of arbitrary dimension, built by means of an architecture based on neural networks. This vectorization overcomes the main limitations related to BoW and TF-IDF, generating vectors capable of capturing information about the semantic of texts.

TOPIC2VEC Vectorizing model in which every text is transformed in a vector representing its probability distribution over the topics discovered by means of *Latent Dirichlet Allocation* (LDA). Some of the topics discovered from our summaries by LDA, together with the five words most characterizing each topic, are presented in **Table** 4.1.

Topic	Characterizing words				
CRIME	Police	Murder	Case	Druge	Prison
WAR	Fight	Battle	Soldier	King	Army
GAME	Team	Win	Play	Final	Match
YOUTH	School	Girl	Boy	Student	Year
SCI-FI	Destroy	Alien	Machine	World	Space

Table 4.1: Some of the topics found by LDA.

4.2.2 *Tf-Idf-Weighted Word2Vec*

In addition to the standard text vectorization methods, we experimented a custom vectorization method named as *Tf-Idf-Weighted Word2Vec*. This method combines the pre-trained Word2Vec model of Google (see section 2.3.4) with the TF-IDF score of each word in a text. In particular, it computes a weighted average of the Google's pre-trained Word2Vec related with each word in a text, using as weights the TF-IDF score of each word.
The following equation shows how this vectorization method works for a single text:

$$\mathrm{Tf-Idf-W2V_{doc}} = \frac{\sum\limits_{\mathrm{word} \in \mathrm{doc}} \mathrm{W2V_{word}} * \mathrm{TFIDF_{word}}}{\sum\limits_{\mathrm{word} \in \mathrm{doc}} \mathrm{TFIDF_{word}}}$$

4.2.3 *Cast2Vec*

To exploit the information related with actors contained in the **CAST** field of our dataset (see section 3.5.3), we designed a new custom input representation of movies, different from those based on summaries, called *Cast2Vec*. Cast2Vec represents each movie through a 19-dimensional vector (i. e. one dimension for each *mood*), that is build taking into account the *moods* associated to all the known movies in which each member of the cast appeared.
The following pseudo-code shows how this vectorization method works for a single movie.

Cast2Vec Algorithm: how Cast2Vec works

```
Data: ds := Dataset with moods and cast information
Data: movie
Result: Cast2Vec vector

contributes_list = [];
for actor in movie.cast do
    actor_contrib = 0;
    for other_movie in ds\movie do
        if actor in other_movie.cast then
            actor_contrib += other_movie.moods_vector;
        end
        contributes_list.append(actor_contrib);
    end
end

return cast2vec = sum(contributes_list) / max(contributes_list);
```

4.2.4 *Text Preprocessing*

Just prior to input vectorization, we needed to apply some preprocessing operations on summaries, in order to obtain noise-free artifacts with a better discriminative power. All the preprocessing operations described in section 2.3.1 were applied. In addition to them, we also performed the following steps:

HIGH-FREQUENCY TERMS REMOVAL High-frequency terms were removed since they could deviate the quality of those vectorization methods which give higher values to terms appearing often in texts (e. g. BoW).

LOW-FREQUENCY TERMS REMOVAL Low-frequency terms were removed because of the low discriminative power, which makes them similar to noise.

PROPER NAMES REMOVAL Proper names were removed since they could deviate the quality of those vectorization methods which give higher values to terms with a relevant intra-document frequency and a low inter-document frequency (e. g. TF-IDF).

4.3 SINGLE-LABEL CLASSIFIERS

Among all the existing methods for multi-label classification described in section 2.5, we decided to use those falling under the category of problem transformation methods. These methods work by transforming the multi-label classification problem into more single-label classification problems: therefore, we selected six single-label classifiers to be used in subsequent experiments, choosing among the most widely-used ones in text classification problems (see section 2.4).

4.3.1 *Linear Classifiers*

Among the chosen classifiers, three were linear models. In the following we provide a description of each of them:

NAIVE BAYES Probabilistic classifier based on the application of the Bayes' theorem. It works making a strong independence assumption between the features (i. e. between the elements of the input vector). In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Each feature contributes independently to the membership probability of a sample to a class. Naive Bayes is still a popular baseline classification methods for text classification.

SUPPORT VECTOR MACHINE Non-probabilistic classifier presented for the first time in [12]. The main principle of this model is to determine the hyperplane in the feature space which can best separate the different classes. More precisely, as shown in **Figure** 4.1, the best hyperplane is the separator plane which has the maximum margin, i. e. the maximum distance between the plane and the nearest data points which belong to different classes.
The *SVM* implementation we used was based on the *Sequential Minimal Optimization* algorithm for training, presented in [34].

LOGISTIC REGRESSION Model-based classification method used for binary classification. In this method, the log-odds of a given class is computed as a linear combination of one or more independent variables (called *predictors*). The predictors are used to compute the probability by means of a sigmoid function. Training of the model consists in the optimization of the predictors' coefficients.

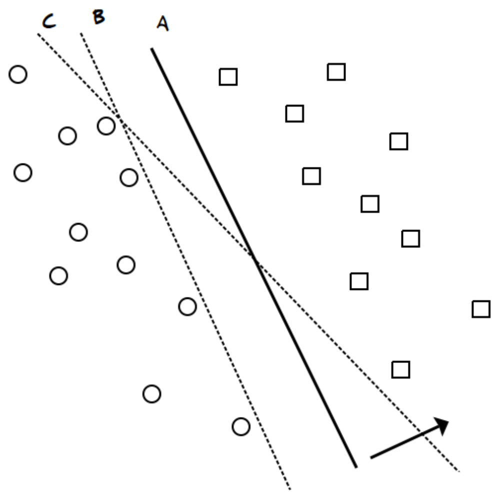

Figure 4.1: The hyperplane A represent the plane with maximum margin of separation, thus it's the best separating hyperplane.

4.3.2 *Ensemble Classifiers*

In addition to the 3 above-mentioned linear models, we decided to use also three ensemble models. In the following we provide a description of each of them:

RANDOM FOREST Ensemble learning method for classification based on the construction of decision trees (see section 2.4.2). Random Forest can be seen as an improvement over trees bagging, which is simply an application of bootstrap aggregation over decision trees models, in the sense that at each candidate split in the training process, a different subset of features is selected.

EXTREME GRADIENT BOOSTING eXtreme Gradient Boosting (*XGBoost*) is a scalable and accurate implementation of gradient boosting machines developed to push the limits of computing power for boosted trees algorithm. Built with this purpose, it is engineered to exploit every bit of memory and hardware resources for tree boosting algorithms. Gradient boosting also comprises an ensemble method that sequentially adds predictors and corrects previous models, but instead of assigning different weights to the classifiers after every iteration, this method fits the new model to new residuals of the previous prediction and then minimizes the loss when adding the latest prediction.

STACKING Ensemble learning method that allows to combine models of different types, differently from Random Forest and XGBoost. In particular, it exploits a two-level architecture: a first level made up of *base learners* (i. e. different kinds of models) and a second level made up of a single *meta-learner*, whose objective is to combine base learners' predictions. Many implementation of Stacking exist, slightly different from each other. The implementation we decided to use can be summarised through the following steps:

Stacking Algorithm: how Stacking works

```
Data: train_set
Data: test_set
Data: base_classifiers
Data: meta_classifier

split train_set into ten disjoint folds;
for clf in base_classifiers do
    for fld in folds do
        train clf on train_set\fld;
        test clf on fld;
        test clf on test_set;
    end
    create a clf_train_set combining folds predictions;
    create a clf_test_set averaging test_set predictions;
end
create a stacking_train_set concatenating all clf_train_sets;
create a stacking_test_set concatenating all clf_test_sets;
train meta_classifier on stacking_train_set;
test meta_classifier on stacking_test_set;
return stacking_test_set predictions;
```

4.4 RESAMPLING STRATEGIES

As highlighted in section 3.5.1, we were provided with a dataset that suffers from imbalance problem. To overcome this issue, we decided to submit our inputs, when possible, to some resampling strategies before the classification process, looking for improvements in the quality of predictions. Those strategies are only applicable to single-label classification problems, and can be traced to two main groups: *Under-sampling* strategies, which try to rebalance data by removing samples from the majority class, and *Over-sampling* strategies, that try to rebalance data by generating new samples in the classes which are under-represented (see section 2.7).

In the following we present a description of the balancing strategies we chose to use in our experiments:

RANDOM UNDER-SAMPLING Samples belonging to the majority class are selected randomly and removed from the dataset. This strategy can be harmful if the number of examples in the dataset is already low.

RANDOM OVER-SAMPLING Produces synthetic copies of examples belonging to the minority class. This equates to give more importance to examples of the minority class, reducing the generalization capacity of the models.

ENN The *Edited Nearest Neighbours* algorithm edits the dataset by removing the examples belonging to the majority class which are considered "outliers" with the nearest-neighbor rule. For each candidate of the majority class, the algorithm selects the group of its nearest neighbors (the data points which are closer in the feature space to it) and compare their class with that of the selected candidate: if the class of the majority of those neighbors coincides with that of the candidate, the candidate is kept, otherwise, the candidate is considered an outlier and removed from the dataset (as shown in **Figure** 4.2).

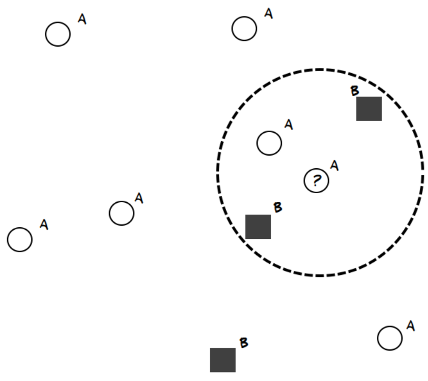

Figure 4.2: The example of the majority class in the figure would be removed since most of its neighbours belongs to class B while its class is A.

SMOTE The *Synthetic Minority Over-Sampling Technique* bases its functioning on the production of new synthetic examples different from the one already present in the dataset for the minority class. To create a new example, SMOTE algorithm selects a data point for which it computes its nearest neighbors of the same class. The algorithm then selects the neighbors and computes the interpolation between those and the original data point, producing new synthetic examples lying on the line which joins the neighbors to the selected data point (as shown in **Figure** 4.3).

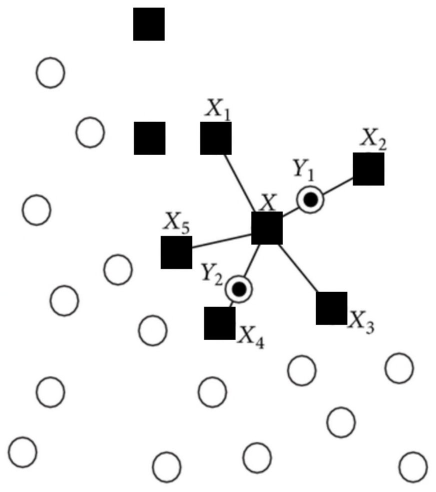

Figure 4.3: Illustration of how the syntethic examples Y_1 and Y_2 are produced starting from X and its neighbours X_i.

4.5 BINARY RELEVANCE APPROACH

The first approach we adopted to solve our multi-label task was *Binary Relevance*. As explained in section 2.5.1, this technique deals with the problem by decomposing it into single different independent binary classification tasks, each of which is responsible for the prediction of a single-label against all the others. In our case, the trained binary classifiers were 19, one for each *mood*.

4.5.1 *Input and Balancing Selection*

Since the number of chosen vectorization models was high, we decided to start using the Binary Relevance approach in order to perform a selection process, with the goal of identifying and discarding the worst vectorization models. Therefore, relying on the assumption that the better the input, the better the classification performance, we decided to evaluate each vectorization model using its vectorizations as input for a variety of classification tasks. In particular, we trained and evaluated, for each vectorization model and without using any resampling strategy, three different multi-label classifiers, each of which used a different linear classifier (see section 4.3.1) as base classifier for the Binary Relevance approach. The idea behind this step was to

try to find some vectorization models that were worse than the others, irrespective of the binary classifier chosen for the Binary Relevance approach. Therefore, based on the classification performance, we were able to identify the worst vectorization models and thus to submit them to some statistical tests, trying to highlight some significant difference with respect to other vectorization models and drop them out from subsequent analysis.
The statistical tests we performed are named as *Student's t-test* and *Mann–Whitney U test*. Student's t-test provides an exact parametric test for the equality of the means of two normal populations [1]. Mann–Whitney U test instead, provides a non-parametric test for examing differences between two independent populations on a continuous scale. It represents a non-parametric alternative to the Student's t-test, with the main difference that Student's t-test compares mean values between two normally distributed populations, while Mann–Whitney U test compares their median and can be used even when populations are not normally distributed [32].
Similarly to the vectorization model selection, we also performed some steps to have an insight on the chosen resampling strategies. The way of assessing the quality of each strategy was identical to the one used with vectorization models, with the only difference that a multi-label classifier (based always on linear binary classifiers) was built for each resampling strategy and for each vectorization model selected in the previous selection. Therefore, based on classification performance, also for resampling strategies we were able, with the help of statistical tests, to select and keep only the best strategies.

4.5.2 *Binary Relevance with Combined Input*

To benefit from the different aspects each vectorization was able to capture, we decided to generate new vectorized inputs combining all the previously selected inputs. In particular, new inputs were obtained by concatenation, using all the possible combinations given by two or three vectorized inputs.
Starting from new combined input, we extended the Binary Relevance analysis to most of the single-label classifiers (see section 4.3), trying all the selected resampling strategies and all the selected vectorized inputs, either alone or in combination. The only exception was represented by the Stacking classifier: given that our client had some constraints on the use of computational resources, Stacking turned out to be computationally expensive. Therefore, it was left out from the analysis and used for subsequent experiments performed with a small number of vectorized inputs.
Working independently on different labels, the Binary Relevance approach wasn't able to exploit the information of the correlation between *moods*, but still constituted a first baseline in the analysis.

4.5.3 *Combined Input Selection*

The performance obtained using the Binary Relevance approach were exploited to have an insight on the discriminative power of the combined vectorized inputs too. Similarly to the steps performed in section 4.5.1, we in fact used the classification performance related to each combined input to perform statistical tests and select some of the best

combinations. Most of those combinations represented the new vectorized inputs for all subsequent experiments.
Using only some of the best combined inputs instead of all the possible ones, testing the Stacking model, presented in section 4.3.2, became computationally feasible. Therefore, we decided to test its performance through the Binary Relevance approach, using as base classifiers some of the best previously tested ones, both linear and non-linear. Thanks to the independence of each Stacking model, all the selected resampling strategies could be tested, trying thus to improve the performance of previous approaches and handle the imbalanced problem at the same time.

4.6 ENSEMBLE OF CLASSIFIER CHAIN

Given the fact that Binary Relevance was not capable to deal with label correlation, we looked towards *Classifier Chains*. As explained in section 2.5.2, Classifier Chain models can take into account the label dependencies when performing multi-label classification. Since the quality of the prediction of a Classifier Chain is sensitive to the ordering of the nodes and an optimal ordering may exist, but since testing all the possible ordering comes at a high computation cost, we used an *Ensemble of Classifier Chains* method to cope with that. We realized different Classifier Chains (i.e. 20), each built with a random ordering of the nodes (i.e. ordering of the labels), all trained with the same dataset. Labels of a new instance were predicted by each chain separately. After that, the total number of predictions or "votes" were counted for each label: the label was considered as relevant if it was predicted as relevant by the majority of the classifiers. Almost all the chosen classifiers were tested with this approach, using as input all the selected combined vectorizations (see section 4.5.3). Again, the only exception was represented by the Stacking classifier which, being computationally expensive, was left out from this analysis.
Because of their peculiar training mechanism, Classifier Chains didn't allow to be tested in combination with selected resampling strategy: therefore, this approach lacked the capacity of counter-acting the imbalance problem, although a very recent extension resilient to class imbalance has been proposed [28].

4.7 STACKING AGGREGATION

Both Binary Relevance and Classifier Chains approaches weren't able to take into account the label correlation and handle the imbalance problem at the same time. In order to overcome these limitations, we performed a last step in our architecture experimenting the so-called *Stacking Aggregation* approach. As presented in [14], Stacking Aggregation is a smart way to use a Binary Relevance approach in order to overcome the label independence problem, basically by applying the Stacking paradigm in the context of multi-label classification. During the learning phase, this method builds a stack of two groups of classifiers. The first one is formed by the same binary classifiers yielded by Binary Relevance method. In a second level, also called *meta-level*, another group of binary models (one for each label again) is learned, but these classifiers consider an augmented feature space that includes the binary outputs of all models of the first level. The idea is to learn the relationships between labels in the meta-level step. In

the testing phase, the final predictions are the outputs of the meta-level classifiers, using the outputs of first-level classifiers exclusively to obtain the values of the augmented feature space.
Thanks to this powerful model, based on Binary Relevance, we were able to train each first-level binary classifier independently: this allowed us to apply all the selected resampling strategies and thus counter-act the imbalance problem, while still taking into account the label correlation.

4.8 IMPLEMENTATION

All the steps performed in our architecture (see a general scheme in **Figure** 4.4) were carried out by means of different kinds of tools, specific for natural language processing and multi-label classification. Since our client required us to use *Python* as programming language, all the used tools consist in Python modules and packages.
As regards input vectorization, a description of the implementations we adopted in our architecture is presented in the following:

▷ **Bow** and **Tf-Idf**: → *Scikit-learn* implementation. Scikit-learn [33] is a free software for machine learning, that allows to use in a simple way tools for data mining and data analysis.

▷ **Doc2Vec** → *Gensim* implementation. Gensim [41] is a free software that allows to realize unsupervised semantic modelling from plain text.

▷ **Topic2Vec** → *Mallet* implementation. Mallet [29] is a Java-based package for statistical natural language processing, document classification, topic modeling and other machine learning applications to text. We used a Python wrapper of Mallet provided by Gensim.

As regards input resampling, all the presented strategies come from package *imbalanced-learn* [26], which provides many off-the-shelf implementations of known balancing techniques.
Regarding the statistical tests, both t-test and U-test were performed using the *SciPy* library [21], in particular the module *stats*, that contains a large number of statistical functions.
All the tested classification algorithms, except XGBoost, come from *Meka* toolkit [40], which provides an open source implementation of methods for single-label and multi-label learning and evaluation. It is a Java-based software, but package *scikit-multilearn* [50] provides a Python wrapper of it. As regards XGBoost, its implementation comes instead from the standard *xgboost* package [11].
Finally, all the adopted performance metrics come from Scikit-learn module *metrics*, which provides many off-the-shelf implementations of known classification metrics.

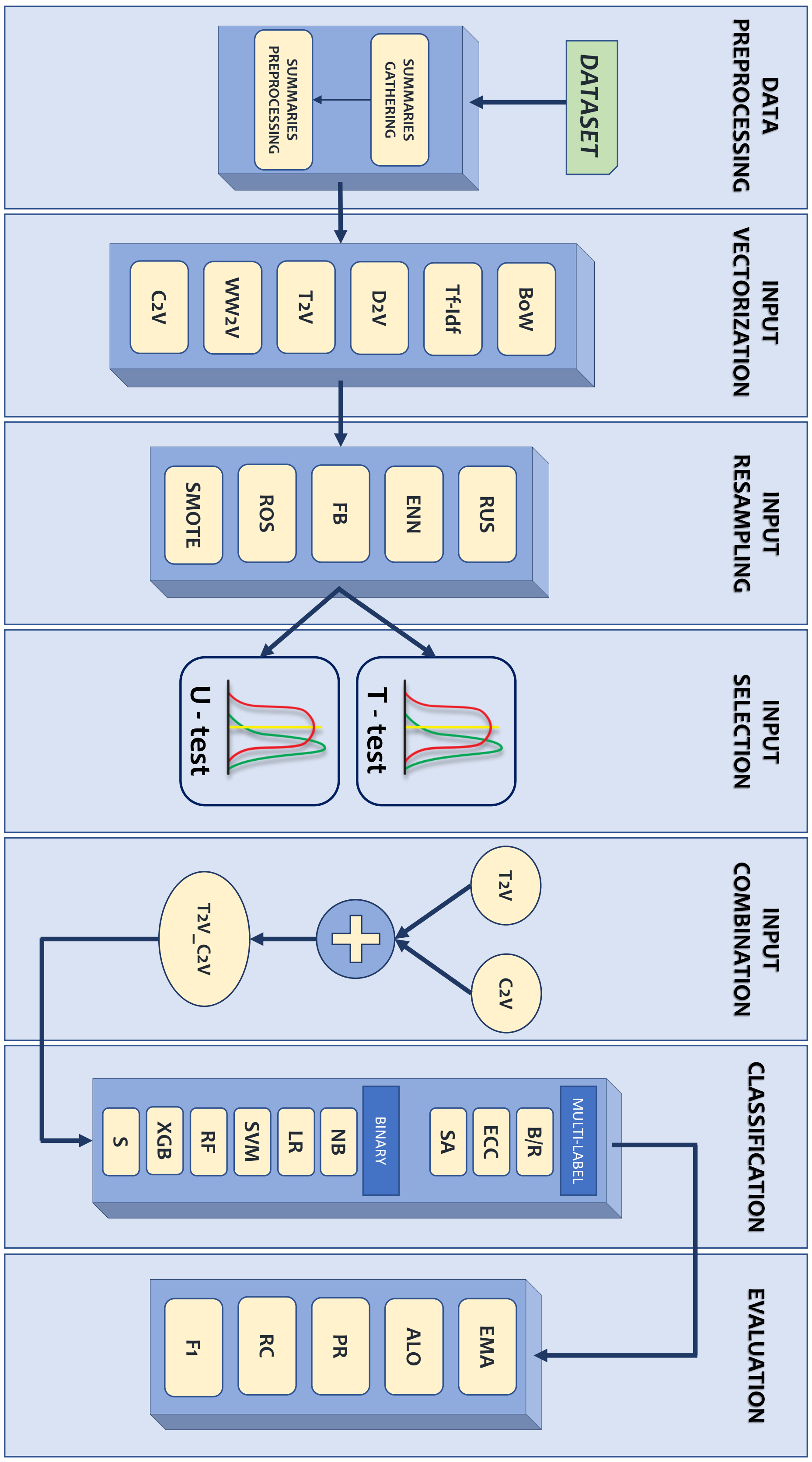

Figure 4.4: Scheme of the presented architecture.

5 EXPERIMENTAL ANALYSIS

In this chapter, we show the results of the experimental analysis done with selected models, highlighting strengths and weaknesses of each one. We describe the chosen model evaluation strategy and the chosen performance metrics. Then, we describe how the hyperparameter selection has been done for selected inputs and models. Finally, we show and comment the achieved performance results of each model.
Whenever necessary, we present the result of some statistical tests performed in order to compare and select different combinations of vectorized inputs and models.

5.1 DATASET PARTITION: TRAIN-VALIDATION

The first step of experimental analysis was the partition of the dataset in two disjoint subsets: the training set, used for training selected models, and the validation set, used for hyperparameter selection of selected models. Validation set was obtained using the *holdout method*, through which a part of the original dataset is set aside and not used in the training phase: for our experiments, we decided to have a validation set equal to 10% of the original dataset. As regards the generation of the partition, we used a stratified approach specific for multi-labeled data, presented in [50], which allows obtaining a well-balanced distribution of label relationship among the partitions.

5.2 MODEL EVALUATION

In order to evaluate selected models, we chose to use a 10-fold cross-validation method, that has been proved to be one of the best choices to get an accurate estimate of performances. In 10-fold cross-validation, the whole training set is partitioned into ten equal size subsets, named as folds. Of the ten subsets, a single subset is retained as the evaluation data for testing the model, while the remaining nine are used as training data for the model. The cross-validation process is then repeated ten times, with each of the ten subsets used exactly once as the evaluation data. The ten results from the folds are then averaged to produce a single estimation of performances.
As regards the generation of the folds, we used the same stratified approach presented in section 5.1.

STRUCTURE OF PREDICTIONS

Given an evaluation set composed of m instances, what a classifier returns as predictions is a matrix with shape $(m \times k)$, where k is the number of labels. Each row of the matrix represents the predictions associated with an evaluation instance, while each column represents all the predictions, for the given evaluation set, related to a particular label. Each cell of the matrix can take two different values: 1, which indicates that the label in the corresponding column is relevant for the instance in the corresponding row; 0, which indicates that the label in the corresponding column is not relevant for the instance in the corresponding row.

5.3 PERFORMANCE METRICS

In order to measure the performances of selected methods, we chose to use most of the metrics presented in section 2.6, which are commonly used in many multi-label classification tasks. Selected metrics can be summarised as follows:

EXACT MATCH ACCURACY (EMA) Evaluates the fraction of perfectly classified examples; can be regarded as a multi-label counterpart of the traditional accuracy metric.

PRECISION (PR) Evaluates the fraction of relevant labels among all the labels predicted as relevant; can be regarded as the ability of the classifier not to label as positive a sample that is negative. There are two kinds of precision: *micro-averaged precision*, that calculates metric globally by counting the total true positives, false negatives and false positives; *macro-averaged precision*, that calculates metric for each label and returns the unweighted mean.

RECALL (RC) Evaluates the fraction of relevant labels, correctly predicted, over the total amount of relevant labels; can be regarded as the ability of the classifier to find all the positive samples. As is the case for precision, even for recall two kinds of measure exist: *micro-averaged recall* and *macro-averaged recall*.

F1 Armonic mean of precision and recall. As is the case for precision and recall, even for F1 two kinds of measure exist: *micro-averaged F1* and *macro-averaged F1*.

In addition to the above-mentioned metrics, we defined another metric called *At Least One (ALO)*, summarised as follows:

AT LEAST ONE (ALO) Evaluates the fraction of examples for which the prediction was correct for at least one label.

This metric is particularly suited for our problem, where each example can be associated with at most two labels and where the main objective is to give a hint to human taggers, who thus, in presence of at least one correct label, are more easily guided in the tagging process.

5.4 VECTORIZERS' HYPERPARAMETER SELECTION

All the experiments were carried out using the vectorized inputs presented in section 4.2, either alone or in combination. The execution of each vectorizer method was often related to the choice of some hyperparameters, that had to be selected properly in order to improve the quality of vectorized documents. Therefore, we decided to select vectorizers' hyperparameters through a grid search approach, which is simply an exhaustive searching through a manually specified subset of the hyperparameter space. Of course, a grid search algorithm must be guided by some performance metric, typically measured by cross-validation. We decided thus to use F1 as the performance

metric and to measure it by stratified 3-fold cross-validation on the validation dataset; as benchmark classifier for the optimization, we adopted a *Binary Relevance* approach based on *Naive Bayes*, since it is a relatively simple algorithm which performs generally well in text classification tasks.
Among all the different vectorizers, there is one in particular which required a hyperparameter selection procedure different from that mentioned above: the *LDA* vectorizer, which finds latent topics in a collection of documents and transforms each document in a vector representing its probability distribution over the topics. The only relevant hyperparameter needed by LDA vectorizer is the number of topics we want to extract from documents: this number is typically chosen by evaluating a coherence measure, which looks at sets of words in generated topics and rates the human interpretability of the topics. Various measures that calculate coherence exist, but the one known as C_v proved to be the most aligned with human interpretability [44]. Therefore, we decided to choose the number of topics for LDA by running the algorithm several times, changing the hyperparameter related to the number of topics and plotting the C_v measure of each run. **Figure** 5.1 show the result of this analysis.

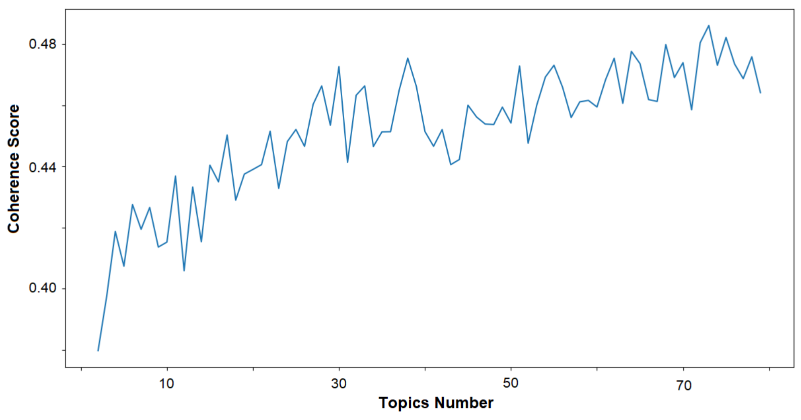

Figure 5.1: Coherence analysis of LDA model.

As is highlighted in the chart, the coherence measure increases with increasing number of topics and then tends to become more stable for a number of topics greater than 30 Whereas having a high number of topics often means to have more topics with overlapping relevant words, we decided to take as LDA hyperparameter the previously highlighted value of 30.

5.5 models' hyperparameter selection

Similarly to the approach described in section 5.4 for vectorizers, even for classification models we optimized the relevant hyperparameters in order to improve the quality of selected classifiers. Again, we decided to use a grid search approach combined with a stratified 3-fold cross-validation on the validation dataset, using F1 as the performance metric. The only exception was the *XGBoost* model, for which we adopted a random search approach, which is a way of performing hyperparameter optimization by selecting them randomly in a manually specified subset of the hyperparameter space. We preferred this approach over grid search, since XGBoost has many relevant hyperparameters which can be optimized and thus, performing an exhaustive search would have been very demanding in terms of computation time.

5.6 ANALYSIS RESULTS

In this section, we show the evaluation results of all performed analysis. In order to make them more readable, we define the following naming conventions for single-label classifiers, vectorized inputs, resampling techniques and metrics:

▷ Single-label Classifiers:

Classifier	**Abbreviation**
Logistic Regression	LR
Naive Bayes	NB
Support Vector Machine	SVM
Random Forest	RF
XGBoost	XGB
Stacking	S

▷ Vectorized Inputs:

Vectorizer	**Abbreviation**
Bag of Words	BoW
Term Freq. - Inverse Document Freq.	TF-IDF
Doc2Vec	D2V
Topic2Vec	T2V
Weighted Word2Vec	WW2V
Cast2Vec	C2V

Each combination of the just mentioned vectorized inputs can be expressed joining names with an underscore character (_). For example, Cast2Vec combined with Doc2Vec can be written as C2V_D2V.

▷ Balancers:

Balancer	**Abbreviation**
Fake Balancer (no resampling)	FB
Random Under Sampler	RUS
Edited Nearest Neighbour	ENN
Random Over Sampler	ROS
Synthetic Minority Over-sampling	SMOTE

▷ Metrics:

Metric	**Abbreviation**
Micro-averaged Precision	PR-mic
Micro-averaged Recall	RC-mic
Macro-averaged F1	F1-mac
Micro-averaged F1	F1-mic
Exact Match Accuracy	EMA
At Least One	ALO

5.6.1 *Statistical Tests for Model Comparison*

Most of the models evaluated in our analysis needed to be compared against other models, in order to possibly highlight significant differences and thus guide subsequent analysis, filtering out irrelevant models. Such comparisons were always carried out by means of two statistical tests, named as *Student's t-test* and *Mann–Whitney U test*, as described in section 4.5.1. Every individual test is performed at a significance level of α, typically equals to 0.05. However, whenever multiple tests are performed, the likelihood of asserting an incorrect conclusion about the outcome of a test increases: therefore, what is typically done is to introduce a correction factor m (named as *Bonferroni correction*) which decrease the significance level by a factor equal to m [2].
In our comparisons, we decided to use as populations the micro-averaged F1 scores obtained from 10-fold cross-validation results, since they are the measures that best summarise the performance of a model. Moreover, whereas no particular assumption on the scores' distribution was made, we decided to perform all the comparisons using both mentioned tests. As regards the Student's t-test, we used the unpaired version, since the folds on which we trained our models were not the same for every model.

5.6.2 *Evaluation of Vectorized Inputs*

The first objective of our analysis was to evaluate the goodness of the vectorized inputs, in terms of the contribution made to the classification problem. To accomplish this task, as explained in section 4.5.1, we used as benchmark the Binary Relevance approach, training and evaluating three different models, one for each linear classifier, in order to have a realistic insight on the discriminative power of each kind of vectorization. No resampling techniques were used in this evaluation.

▷ ‖ BoW | TF-IDF | C2V | TD | WW2V | D2V ‖ *(Vectorized Inputs)*

▷ ‖ NB | LR | SVM ‖ *(Classifiers)*

▷ ‖ FB ‖ *(Fake Balancer: no resampling)*

The evaluation was performed throguh stratified 10-fold cross-validation, as described in section 5.2. Results are shown in **Tables** 5.1, 5.2 and 5.3. The only metrics shown are micro-averaged precision, micro-averaged recall and micro-averaged F1, which are sufficient to highlight the most relevant aspects related to the performances of a multi-label classification problem.

Vectorized Input	PR-mic	RC-mic	F1-mic
BoW	0.285 ± 0.020	0.354 ± 0.011	0.315 ± 0.012
TF-IDF	0.301 ± 0.017	0.366 ± 0.014	0.330 ± 0.010
T2V	0.351 ± 0.018	0.341 ± 0.017	0.346 ± 0.018
D2V	0.357 ± 0.012	0.347 ± 0.011	0.352 ± 0.011
WW2V	0.349 ± 0.016	0.345 ± 0.014	0.347 ± 0.015
C2V	0.342 ± 0.010	0.332 ± 0.009	0.337 ± 0.009

Table 5.1: Vectorized input evaluation: Naive Bayes.

Vectorized Input	PR-mic	RC-mic	F1-mic
BoW	0.306 ± 0.014	0.369 ± 0.016	0.334 ± 0.009
TF-IDF	0.331 ± 0.013	0.382 ± 0.023	0.354 ± 0.011
T2V	0.404 ± 0.017	0.392 ± 0.015	0.398 ± 0.015
D2V	0.432 ± 0.013	0.419 ± 0.012	0.425 ± 0.012
WW2V	0.366 ± 0.015	0.376 ± 0.009	0.371 ± 0.012
C2V	0.350 ± 0.014	0.340 ± 0.013	0.345 ± 0.013

Table 5.2: Vectorized input evaluation: Logistic Regression.

Vectorized Input	PR-mic	RC-mic	F1-mic
BoW	0.266 ± 0.018	0.272 ± 0.009	0.269 ± 0.009
TF-IDF	0.283 ± 0.014	0.287 ± 0.016	0.284 ± 0.009
T2V	0.320 ± 0.015	0.311 ± 0.013	0.315 ± 0.014
D2V	0.391 ± 0.006	0.380 ± 0.006	0.385 ± 0.006
WW2V	0.397 ± 0.011	0.385 ± 0.009	0.391 ± 0.01
C2V	0.285 ± 0.013	0.280 ± 0.016	0.283 ± 0.014

Table 5.3: Vectorized input evaluation: Support Vector Machine.

As can be seen from the results, BoW, TF-TDF and C2V were apparently the inputs that gave the worst classification performances. Given that C2V input was the only one that encoded information not related with movie plots, we decided not to drop it, but instead to use it for subsequent analysis performed with combined inputs, in order to exploit better its discriminative power. As regards BoW and TF-IDF instead, we decided to submit them to statistical tests, in order to compare them with other vectorizers to possibly highlight a significant difference and thus drop them out from subsequent analysis. The statistical tests we performed were those mentioned in section 5.6.1: Student's t-test and Mann–Whitney U test. Since a direct comparison between vectorized inputs was not possible, we used as benchmark all the previously evaluated classifiers. For each classifier, we compared the performances obtained using BoW and TF-IDF against the performances obtained using the other 3 vectorized inputs: a Bonferroni correction factor of 3 was thus applied to the test (i. e. the models are significantly different if p-value $< 0,0167$). **Tables** 5.4, 5.5, and 5.6 show results of performed tests.

Vectorized Input	**p-value t-test BoW**	**p-value U-test BoW**	**p-value t-test TF-IDF**	**p-value U-test TF-IDF**
T2V	0.0004	0.0009	0.0305	0.0156
D2V	0.0000	0.0001	0.0004	0.0011
WW2V	0.0001	0.0005	0.0111	0.0057

Table 5.4: Vectorized input comparison: Naive Bayes.

Vectorized Input	**p-value t-test BoW**	**p-value U-test BoW**	**p-value t-test TF-IDF**	**p-value U-test TF-IDF**
T2V	0.0000	0.0001	0.0000	0.0003
D2V	0.0000	0.0001	0.0000	0.0001
WW2V	0.0000	0.0001	0.0055	0.0070

Table 5.5: Vectorized input comparison: Logistic Regression.

Vectorized Input	**p-value t-test BoW**	**p-value U-test BoW**	**p-value t-test TF-IDF**	**p-value U-test TF-IDF**
T2V	0.0000	0.0001	0.0000	0.0002
D2V	0.0000	0.0001	0.0000	0.0001
WW2V	0.0000	0.0001	0.0000	0.0001

Table 5.6: Vectorized input comparison: Support Vector Machine.

All the performed statistical tests highlighted a significant difference between the two candidate vectorizations (TF-IDF and BoW) and all the other ones, irrespective of the selected linear classifiers: therefore, we decided to filter out BoW and TF-IDF from subsequent analysis.

5.6.3 *Evaluation of Resampling Strategies*

Similarly to the evaluation of the vectorized inputs, we also performed some steps to have an insight on the goodness of resampling strategies. The way of assessing the quality of each strategy was identical to the one used with vectorization models, with the only difference that a multi-label classifier (based always on linear binary classifiers) was built for each resampling strategy and for each vectorization model previously selected.

- ▷ ‖ C2V | TD | WW2V | D2V ‖ *(Vectorized Inputs)*
- ▷ ‖ NB | LR | SVM ‖ *(Classifiers)*
- ▷ ‖ RUS | ENN | ROS | SMOTE ‖ *(Balancers)*

The evaluation was performed throguh stratified 10-fold cross-validation, as described in section 5.2. Results are shown in **Tables** 5.7, 5.8 and 5.9. The only metrics shown are micro-averaged precision, micro-averaged recall and micro-averaged F1, which are sufficient to highlight the most relevant aspects related to the performances of a multi-label classification problem.

Vectorized Input	**PR-mic**	**RC-mic**	**F1-mic**
T2V + RUS	0.146 ± 0.004	0.814 ± 0.017	0.247 ± 0.006
T2V + ENN	0.335 ± 0.013	0.42 ± 0.016	0.373 ± 0.014
T2V + ROS	0.154 ± 0.002	0.774 ± 0.014	0.257 ± 0.003
T2V + SMOTE	0.143 ± 0.003	0.768 ± 0.011	0.241 ± 0.005
D2V + RUS	0.137 ± 0.005	0.764 ± 0.023	0.232 ± 0.008
D2V + ENN	0.313 ± 0.012	0.401 ± 0.018	0.352 ± 0.014
D2V + ROS	0.15 ± 0.003	0.705 ± 0.014	0.247 ± 0.005
D2V + SMOTE	0.131 ± 0.005	0.605 ± 0.021	0.215 ± 0.008
WW2V + RUS	0.146 ± 0.004	0.801 ± 0.017	0.247 ± 0.006
WW2V + ENN	0.319 ± 0.007	0.404 ± 0.01	0.356 ± 0.007
WW2V + ROS	0.199 ± 0.004	0.702 ± 0.013	0.31 ± 0.005
WW2V + SMOTE	0.16 ± 0.006	0.61 ± 0.031	0.254 ± 0.009
C2V + RUS	0.14 ± 0.002	0.783 ± 0.01	0.238 ± 0.004
C2V + ENN	0.329 ± 0.011	0.415 ± 0.014	0.367 ± 0.012
C2V + ROS	0.136 ± 0.003	0.76 ± 0.017	0.231 ± 0.005
C2V + SMOTE	0.137 ± 0.002	0.767 ± 0.014	0.233 ± 0.004

Table 5.7: Resampling strategies evaluation: Naive Bayes.

Vectorized Input	PR-mic	RC-mic	F1-mic
T2V + RUS	0.152 ± 0.003	0.846 ± 0.012	0.257 ± 0.005
T2V + ENN	0.376 ± 0.013	0.472 ± 0.015	0.418 ± 0.014
T2V + ROS	0.154 ± 0.002	0.861 ± 0.009	0.262 ± 0.003
T2V + SMOTE	0.154 ± 0.002	0.857 ± 0.012	0.261 ± 0.004
D2V + RUS	0.151 ± 0.003	0.819 ± 0.019	0.255 ± 0.006
D2V + ENN	0.388 ± 0.009	0.497 ± 0.01	0.435 ± 0.009
D2V + ROS	0.154 ± 0.003	0.857 ± 0.015	0.261 ± 0.005
D2V + SMOTE	0.151 ± 0.002	0.841 ± 0.015	0.256 ± 0.004
WW2V + RUS	0.139 ± 0.003	0.749 ± 0.012	0.234 ± 0.005
WW2V + ENN	0.346 ± 0.018	0.461 ± 0.016	0.395 ± 0.017
WW2V + ROS	0.182 ± 0.005	0.804 ± 0.017	0.297 ± 0.007
WW2V + SMOTE	0.183 ± 0.004	0.802 ± 0.014	0.297 ± 0.005
C2V + RUS	0.143 ± 0.003	0.796 ± 0.014	0.242 ± 0.005
C2V + ENN	0.331 ± 0.011	0.419 ± 0.012	0.37 ± 0.011
C2V + ROS	0.143 ± 0.002	0.799 ± 0.012	0.243 ± 0.004
C2V + SMOTE	0.144 ± 0.003	0.804 ± 0.014	0.244 ± 0.005

Table 5.8: Resampling strategies evaluation: Logistic Regression.

Vectorized Input	PR-mic	RC-mic	F1-mic
T2V + RUS	0.153 ± 0.004	0.852 ± 0.011	0.259 ± 0.006
T2V + ENN	0.362 ± 0.011	0.454 ± 0.012	0.403 ± 0.011
T2V + ROS	0.154 ± 0.003	0.86 ± 0.018	0.261 ± 0.006
T2V + SMOTE	0.154 ± 0.003	0.856 ± 0.013	0.26 ± 0.005
D2V + RUS	0.153 ± 0.002	0.856 ± 0.01	0.26 ± 0.004
D2V + ENN	0.381 ± 0.01	0.488 ± 0.015	0.428 ± 0.012
D2V + ROS	0.154 ± 0.003	0.858 ± 0.01	0.261 ± 0.004
D2V + SMOTE	0.151 ± 0.003	0.842 ± 0.014	0.256 ± 0.005
WW2V + RUS	0.155 ± 0.003	0.847 ± 0.016	0.263 ± 0.005
WW2V + ENN	0.381 ± 0.011	0.476 ± 0.012	0.423 ± 0.011
WW2V + ROS	0.155 ± 0.004	0.84 ± 0.016	0.262 ± 0.006
WW2V + SMOTE	0.156 ± 0.002	0.842 ± 0.009	0.263 ± 0.004
C2V + RUS	0.143 ± 0.003	0.798 ± 0.016	0.243 ± 0.005
C2V + ENN	0.321 ± 0.013	0.409 ± 0.02	0.36 ± 0.015
C2V + ROS	0.144 ± 0.002	0.801 ± 0.012	0.244 ± 0.004
C2V + SMOTE	0.144 ± 0.003	0.803 ± 0.017	0.244 ± 0.005

Table 5.9: Resampling strategies: Support Vector Machine.

As can be seen from the results, almost all the chosen resampling strategies performed very badly compared to not resampling at all, irrespective of the selected linear classifier used in the Binary Relevance approach. The only exception was represented by the *ENN* strategy, which allowed to obtain an F1 score comparable to the one obtained with no resampling strategy. Therefore, we decided to filter out from subsequent analysis all the resampling strategies, except ENN. No statistical tests were performed, since the poor quality of the strategies was absolutely clear.

5.6.4 *Binary Relevance with Combined Input*

After the selection process, we decided to test the Binary Relevance approach with all the chosen single-label classifier (except *Stacking*), using all the selected vectorized input, either alone or in combination, and all the selected resampling strategies, as described in section 4.5.2.

▷

C2V	TD	WW2V	D2V	Combined

(Vectorized Inputs)

▷

NB	LR	SVM	RF	XGB

(Classifiers)

▷

FB	ENN

(Balancers)

The evaluation was performed throguh stratified 10-fold cross-validation, as described in section 5.2. Results are shown in the following **Tables**:

- ▷ **Table** 5.10 is related to Naive Bayes + Fake Balancer.
- ▷ **Table** 5.11 is related to Naive Bayes + ENN.
- ▷ **Table** 5.12 is related to Logistic Regression + Fake Balancer.
- ▷ **Table** 5.13 is related to Logistic Regression + ENN.
- ▷ **Table** 5.14 is related to Support Vector Machine + Fake Balancer.
- ▷ **Table** 5.15 is related to Support Vector Machine + ENN.
- ▷ **Table** 5.16 is related to Random Forest + Fake Balancer.
- ▷ **Table** 5.17 is related to Random Forest + ENN.
- ▷ **Table** 5.18 is related to XGBoost + Fake Balancer.
- ▷ **Table** 5.19 is related to XGBoost + ENN.

Vectorized Input	ALO	EMA	PR-mic	RC-mic	F1-mic	F1-mac
C2V	0.451 ± 0.009	0.066 ± 0.008	0.342 ± 0.01	0.332 ± 0.009	0.337 ± 0.009	0.24 ± 0.156
T2V	0.481 ± 0.027	0.065 ± 0.016	0.351 ± 0.018	0.341 ± 0.017	0.346 ± 0.018	0.26 ± 0.16
D2V	0.488 ± 0.016	0.068 ± 0.014	0.357 ± 0.012	0.347 ± 0.011	0.352 ± 0.011	0.257 ± 0.163
WW2V	0.477 ± 0.018	0.082 ± 0.012	0.349 ± 0.016	0.345 ± 0.014	0.347 ± 0.015	0.256 ± 0.17
D2V_C2V	0.512 ± 0.018	0.078 ± 0.008	0.39 ± 0.014	0.379 ± 0.013	0.384 ± 0.013	0.288 ± 0.167
D2V_WW2V	0.494 ± 0.012	0.076 ± 0.013	0.367 ± 0.011	0.357 ± 0.01	0.362 ± 0.01	0.273 ± 0.167
T2V_C2V	0.537 ± 0.02	0.08 ± 0.013	0.41 ± 0.013	0.399 ± 0.011	0.405 ± 0.012	0.306 ± 0.177
T2V_D2V	0.507 ± 0.019	0.072 ± 0.009	0.372 ± 0.017	0.362 ± 0.016	0.367 ± 0.016	0.274 ± 0.164
T2V_WW2V	0.483 ± 0.02	0.081 ± 0.015	0.358 ± 0.016	0.352 ± 0.016	0.355 ± 0.016	0.268 ± 0.167
WW2V_C2V	0.508 ± 0.014	0.085 ± 0.009	0.382 ± 0.015	0.374 ± 0.01	0.378 ± 0.012	0.285 ± 0.174
T2V_C2V_D2V	0.53 ± 0.03	0.076 ± 0.009	0.404 ± 0.023	0.392 ± 0.02	0.398 ± 0.022	0.302 ± 0.165
T2V_WW2V_D2V	0.497 ± 0.025	0.078 ± 0.013	0.368 ± 0.015	0.359 ± 0.012	0.363 ± 0.013	0.274 ± 0.169
WW2V_C2V_D2V	0.523 ± 0.016	0.083 ± 0.01	0.391 ± 0.01	0.381 ± 0.011	0.386 ± 0.01	0.289 ± 0.178
WW2V_C2V_T2V	0.519 ± 0.011	0.085 ± 0.014	0.387 ± 0.013	0.378 ± 0.01	0.383 ± 0.011	0.288 ± 0.175

Table 5.10: Binary Relevance: Naive Bayes + Fake Balancer.

Vectorized Input	ALO	EMA	PR-mic	RC-mic	F1-mic	F1-mac
C2V	0.547 ± 0.016	0.065 ± 0.009	0.329 ± 0.011	0.415 ± 0.014	0.367 ± 0.012	0.26 ± 0.161
T2V	0.58 ± 0.017	0.062 ± 0.013	0.335 ± 0.013	0.42 ± 0.016	0.373 ± 0.014	0.276 ± 0.168
D2V	0.558 ± 0.02	0.056 ± 0.011	0.313 ± 0.012	0.401 ± 0.018	0.352 ± 0.014	0.259 ± 0.159
WW2V	0.55 ± 0.013	0.062 ± 0.009	0.319 ± 0.007	0.404 ± 0.01	0.356 ± 0.007	0.264 ± 0.165
D2V_C2V	0.589 ± 0.012	0.058 ± 0.011	0.35 ± 0.009	0.447 ± 0.011	0.392 ± 0.01	0.291 ± 0.168
D2V_WW2V	0.58 ± 0.025	0.067 ± 0.013	0.33 ± 0.015	0.427 ± 0.015	0.372 ± 0.015	0.283 ± 0.165
T2V_C2V	0.618 ± 0.013	0.078 ± 0.006	0.386 ± 0.008	0.473 ± 0.011	0.426 ± 0.009	0.318 ± 0.183
T2V_D2V	0.583 ± 0.021	0.06 ± 0.014	0.333 ± 0.012	0.426 ± 0.017	0.374 ± 0.014	0.282 ± 0.164
T2V_WW2V	0.563 ± 0.02	0.073 ± 0.013	0.328 ± 0.011	0.413 ± 0.012	0.366 ± 0.011	0.273 ± 0.171
WW2V_C2V	0.59 ± 0.017	0.075 ± 0.012	0.36 ± 0.013	0.444 ± 0.014	0.398 ± 0.013	0.299 ± 0.173
T2V_C2V_D2V	0.6 ± 0.022	0.062 ± 0.011	0.355 ± 0.012	0.454 ± 0.017	0.399 ± 0.014	0.303 ± 0.166
T2V_WW2V_D2V	0.581 ± 0.014	0.064 ± 0.014	0.327 ± 0.008	0.427 ± 0.011	0.37 ± 0.008	0.279 ± 0.168
WW2V_C2V_D2V	0.6 ± 0.012	0.065 ± 0.011	0.348 ± 0.009	0.45 ± 0.012	0.392 ± 0.01	0.295 ± 0.174
WW2V_C2V_T2V	0.592 ± 0.015	0.08 ± 0.01	0.363 ± 0.008	0.446 ± 0.008	0.4 ± 0.008	0.299 ± 0.179

Table 5.11: Binary Relevance: Naive Bayes + ENN.

Vectorized Input	ALO	EMA	PR-mic	RC-mic	F1-mic	F1-mac
C2V	0.461 ± 0.015	0.075 ± 0.009	0.35 ± 0.014	0.34 ± 0.013	0.345 ± 0.013	0.238 ± 0.169
T2V	0.531 ± 0.022	0.084 ± 0.011	0.404 ± 0.017	0.392 ± 0.015	0.398 ± 0.015	0.307 ± 0.173
D2V	0.576 ± 0.018	0.104 ± 0.012	0.432 ± 0.013	0.419 ± 0.012	0.425 ± 0.012	0.327 ± 0.18
WW2V	0.545 ± 0.014	0.097 ± 0.01	0.366 ± 0.015	0.376 ± 0.009	0.371 ± 0.012	0.272 ± 0.168
D2V_C2V	0.616 ± 0.017	0.131 ± 0.014	0.459 ± 0.012	0.446 ± 0.012	0.452 ± 0.012	0.346 ± 0.188
D2V_WW2V	0.56 ± 0.018	0.107 ± 0.011	0.39 ± 0.014	0.385 ± 0.011	0.388 ± 0.012	0.28 ± 0.172
T2V_C2V	0.594 ± 0.018	0.124 ± 0.016	0.452 ± 0.015	0.439 ± 0.014	0.445 ± 0.014	0.338 ± 0.193
T2V_D2V	0.583 ± 0.028	0.116 ± 0.011	0.435 ± 0.02	0.422 ± 0.019	0.428 ± 0.019	0.329 ± 0.177
T2V_WW2V	0.567 ± 0.015	0.098 ± 0.014	0.375 ± 0.009	0.389 ± 0.012	0.382 ± 0.01	0.282 ± 0.17
WW2V_C2V	0.578 ± 0.021	0.111 ± 0.011	0.395 ± 0.019	0.405 ± 0.013	0.4 ± 0.016	0.294 ± 0.178
T2V_C2V_D2V	0.622 ± 0.019	0.131 ± 0.012	0.462 ± 0.016	0.449 ± 0.015	0.455 ± 0.015	0.346 ± 0.19
T2V_WW2V_D2V	0.557 ± 0.022	0.105 ± 0.014	0.387 ± 0.01	0.381 ± 0.01	0.384 ± 0.01	0.278 ± 0.165
WW2V_C2V_D2V	0.577 ± 0.016	0.117 ± 0.016	0.407 ± 0.011	0.398 ± 0.011	0.403 ± 0.011	0.287 ± 0.177
WW2V_C2V_T2V	0.588 ± 0.02	0.117 ± 0.016	0.404 ± 0.014	0.409 ± 0.017	0.407 ± 0.015	0.297 ± 0.179

Table 5.12: Binary Relevance: Logistic Regression + Fake Balancer.

Vectorized Input	ALO	EMA	PR-mic	RC-mic	F1-mic	F1-mac
C2V	0.56 ± 0.014	0.079 ± 0.008	0.331 ± 0.011	0.419 ± 0.012	0.37 ± 0.011	0.253 ± 0.172
T2V	0.63 ± 0.022	0.087 ± 0.01	0.376 ± 0.013	0.472 ± 0.015	0.418 ± 0.014	0.316 ± 0.18
D2V	0.669 ± 0.018	0.088 ± 0.009	0.388 ± 0.009	0.497 ± 0.01	0.435 ± 0.009	0.333 ± 0.173
WW2V	0.643 ± 0.016	0.084 ± 0.015	0.346 ± 0.018	0.461 ± 0.016	0.395 ± 0.017	0.289 ± 0.173
D2V_C2V	0.707 ± 0.017	0.103 ± 0.014	0.412 ± 0.013	0.527 ± 0.014	0.462 ± 0.013	0.347 ± 0.194
D2V_WW2V	0.655 ± 0.016	0.082 ± 0.014	0.353 ± 0.013	0.466 ± 0.016	0.402 ± 0.013	0.288 ± 0.166
T2V_C2V	0.686 ± 0.02	0.111 ± 0.016	0.422 ± 0.013	0.518 ± 0.015	0.465 ± 0.014	0.355 ± 0.194
T2V_D2V	0.683 ± 0.017	0.096 ± 0.012	0.394 ± 0.01	0.505 ± 0.015	0.442 ± 0.011	0.333 ± 0.185
T2V_WW2V	0.66 ± 0.023	0.094 ± 0.01	0.365 ± 0.014	0.471 ± 0.01	0.411 ± 0.012	0.306 ± 0.17
WW2V_C2V	0.671 ± 0.016	0.099 ± 0.018	0.374 ± 0.013	0.484 ± 0.018	0.422 ± 0.015	0.306 ± 0.187
T2V_C2V_D2V	0.711 ± 0.023	0.111 ± 0.012	0.417 ± 0.016	0.534 ± 0.018	0.468 ± 0.017	0.354 ± 0.191
T2V_WW2V_D2V	0.649 ± 0.026	0.078 ± 0.015	0.343 ± 0.009	0.462 ± 0.018	0.393 ± 0.012	0.286 ± 0.171
WW2V_C2V_D2V	0.669 ± 0.019	0.091 ± 0.01	0.363 ± 0.013	0.485 ± 0.014	0.415 ± 0.013	0.297 ± 0.183
WW2V_C2V_T2V	0.684 ± 0.019	0.109 ± 0.015	0.385 ± 0.013	0.494 ± 0.013	0.433 ± 0.013	0.312 ± 0.19

Table 5.13: Binary Relevance: Logistic Regression + ENN.

Vectorized Input	ALO	EMA	PR-mic	RC-mic	F1-mic	F1-mac
C2V	0.4 ± 0.019	0.055 ± 0.017	0.285 ± 0.013	0.28 ± 0.016	0.283 ± 0.014	0.179 ± 0.18
T2V	0.448 ± 0.015	0.057 ± 0.012	0.32 ± 0.015	0.311 ± 0.013	0.315 ± 0.014	0.215 ± 0.207
D2V	0.541 ± 0.012	0.086 ± 0.012	0.391 ± 0.006	0.38 ± 0.006	0.385 ± 0.006	0.287 ± 0.187
WW2V	0.553 ± 0.016	0.116 ± 0.016	0.397 ± 0.011	0.385 ± 0.009	0.391 ± 0.01	0.289 ± 0.19
D2V_C2V	0.593 ± 0.029	0.127 ± 0.012	0.434 ± 0.022	0.422 ± 0.021	0.428 ± 0.021	0.308 ± 0.223
D2V_WW2V	0.592 ± 0.015	0.12 ± 0.011	0.422 ± 0.011	0.409 ± 0.011	0.415 ± 0.011	0.318 ± 0.18
T2V_C2V	0.551 ± 0.025	0.102 ± 0.015	0.401 ± 0.017	0.39 ± 0.016	0.395 ± 0.016	0.273 ± 0.231
T2V_D2V	0.564 ± 0.024	0.1 ± 0.009	0.41 ± 0.016	0.398 ± 0.012	0.404 ± 0.014	0.301 ± 0.2
T2V_WW2V	0.583 ± 0.024	0.121 ± 0.015	0.416 ± 0.018	0.404 ± 0.016	0.41 ± 0.017	0.302 ± 0.196
WW2V_C2V	0.602 ± 0.017	0.13 ± 0.015	0.436 ± 0.012	0.423 ± 0.014	0.429 ± 0.013	0.316 ± 0.204
T2V_C2V_D2V	0.606 ± 0.017	0.128 ± 0.014	0.446 ± 0.013	0.433 ± 0.014	0.44 ± 0.013	0.327 ± 0.215
T2V_WW2V_D2V	0.596 ± 0.02	0.129 ± 0.015	0.429 ± 0.015	0.417 ± 0.012	0.423 ± 0.013	0.323 ± 0.184
WW2V_C2V_D2V	0.625 ± 0.024	0.14 ± 0.014	0.455 ± 0.017	0.442 ± 0.014	0.448 ± 0.016	0.345 ± 0.19
WW2V_C2V_T2V	0.611 ± 0.013	0.135 ± 0.014	0.444 ± 0.009	0.431 ± 0.009	0.437 ± 0.009	0.326 ± 0.203

Table 5.14: Binary Relevance: Support Vector Machine + Fake Balancer.

Vectorized Input	ALO	EMA	PR-mic	RC-mic	F1-mic	F1-mac
C2V	0.562 ± 0.018	0.079 ± 0.018	0.321 ± 0.013	0.409 ± 0.02	0.36 ± 0.015	0.23 ± 0.194
T2V	0.628 ± 0.015	0.081 ± 0.011	0.362 ± 0.011	0.454 ± 0.012	0.403 ± 0.011	0.284 ± 0.204
D2V	0.661 ± 0.016	0.083 ± 0.01	0.381 ± 0.01	0.488 ± 0.015	0.428 ± 0.012	0.315 ± 0.191
WW2V	0.664 ± 0.018	0.1 ± 0.011	0.381 ± 0.011	0.476 ± 0.012	0.423 ± 0.011	0.31 ± 0.188
D2V_C2V	0.71 ± 0.02	0.113 ± 0.016	0.414 ± 0.01	0.53 ± 0.014	0.465 ± 0.012	0.341 ± 0.211
D2V_WW2V	0.7 ± 0.018	0.109 ± 0.015	0.398 ± 0.008	0.511 ± 0.01	0.448 ± 0.008	0.343 ± 0.182
T2V_C2V	0.681 ± 0.017	0.11 ± 0.013	0.413 ± 0.009	0.506 ± 0.01	0.455 ± 0.009	0.326 ± 0.221
T2V_D2V	0.682 ± 0.02	0.091 ± 0.015	0.395 ± 0.014	0.506 ± 0.021	0.444 ± 0.016	0.336 ± 0.194
T2V_WW2V	0.68 ± 0.029	0.111 ± 0.012	0.399 ± 0.017	0.496 ± 0.022	0.442 ± 0.019	0.336 ± 0.189
WW2V_C2V	0.706 ± 0.02	0.121 ± 0.015	0.425 ± 0.014	0.519 ± 0.013	0.467 ± 0.013	0.345 ± 0.211
T2V_C2V_D2V	0.716 ± 0.015	0.111 ± 0.01	0.419 ± 0.008	0.536 ± 0.01	0.47 ± 0.008	0.352 ± 0.202
T2V_WW2V_D2V	0.706 ± 0.013	0.109 ± 0.013	0.402 ± 0.011	0.516 ± 0.012	0.452 ± 0.011	0.354 ± 0.178
WW2V_C2V_D2V	0.729 ± 0.014	0.119 ± 0.024	0.422 ± 0.009	0.54 ± 0.012	0.474 ± 0.01	0.363 ± 0.189
WW2V_C2V_T2V	0.712 ± 0.013	0.131 ± 0.019	0.431 ± 0.014	0.525 ± 0.015	0.473 ± 0.014	0.357 ± 0.203

Table 5.15: Binary Relevance: Support Vector Machine + ENN.

Vectorized Input	ALO	EMA	PR-mic	RC-mic	F1-mic	F1-mac
C2V	0.496 ± 0.016	0.108 ± 0.015	0.364 ± 0.016	0.361 ± 0.015	0.362 ± 0.015	0.264 ± 0.16
T2V	0.572 ± 0.018	0.112 ± 0.018	0.405 ± 0.017	0.406 ± 0.014	0.405 ± 0.015	0.308 ± 0.184
D2V	0.574 ± 0.012	0.101 ± 0.008	0.409 ± 0.013	0.415 ± 0.013	0.412 ± 0.013	0.319 ± 0.176
WW2V	0.57 ± 0.019	0.113 ± 0.01	0.402 ± 0.014	0.406 ± 0.014	0.404 ± 0.014	0.314 ± 0.178
D2V_C2V	0.616 ± 0.024	0.123 ± 0.016	0.453 ± 0.018	0.456 ± 0.018	0.455 ± 0.018	0.355 ± 0.19
D2V_WW2V	0.606 ± 0.022	0.108 ± 0.014	0.43 ± 0.015	0.437 ± 0.013	0.433 ± 0.014	0.342 ± 0.178
T2V_C2V	0.628 ± 0.012	0.137 ± 0.015	0.462 ± 0.01	0.46 ± 0.01	0.461 ± 0.009	0.361 ± 0.183
T2V_D2V	0.606 ± 0.022	0.119 ± 0.019	0.432 ± 0.016	0.437 ± 0.017	0.434 ± 0.016	0.34 ± 0.18
T2V_WW2V	0.592 ± 0.019	0.116 ± 0.014	0.414 ± 0.013	0.42 ± 0.013	0.417 ± 0.013	0.329 ± 0.176
WW2V_C2V	0.617 ± 0.016	0.13 ± 0.013	0.446 ± 0.016	0.451 ± 0.016	0.448 ± 0.015	0.348 ± 0.192
T2V_C2V_D2V	0.628 ± 0.017	0.129 ± 0.016	0.465 ± 0.012	0.466 ± 0.01	0.466 ± 0.01	0.368 ± 0.185
T2V_WW2V_D2V	0.613 ± 0.019	0.117 ± 0.016	0.437 ± 0.014	0.442 ± 0.011	0.439 ± 0.012	0.351 ± 0.169
WW2V_C2V_D2V	0.63 ± 0.023	0.129 ± 0.014	0.456 ± 0.02	0.46 ± 0.018	0.458 ± 0.019	0.358 ± 0.186
WW2V_C2V_T2V	0.627 ± 0.018	0.132 ± 0.014	0.454 ± 0.014	0.457 ± 0.012	0.456 ± 0.013	0.357 ± 0.187

Table 5.16: Binary Relevance: Random Forest + Fake Balancer.

Vectorized Input	ALO	EMA	PR-mic	RC-mic	F1-mic	F1-mac
C2V	0.587 ± 0.03	0.1 ± 0.017	0.34 ± 0.019	0.437 ± 0.02	0.382 ± 0.019	0.271 ± 0.167
T2V	0.659 ± 0.017	0.098 ± 0.011	0.378 ± 0.012	0.482 ± 0.015	0.424 ± 0.013	0.323 ± 0.179
D2V	0.679 ± 0.014	0.09 ± 0.011	0.381 ± 0.011	0.505 ± 0.015	0.434 ± 0.012	0.334 ± 0.179
WW2V	0.648 ± 0.023	0.092 ± 0.011	0.369 ± 0.014	0.475 ± 0.018	0.416 ± 0.016	0.319 ± 0.174
D2V_C2V	0.705 ± 0.019	0.096 ± 0.009	0.409 ± 0.01	0.536 ± 0.014	0.464 ± 0.011	0.362 ± 0.183
D2V_WW2V	0.684 ± 0.014	0.096 ± 0.009	0.387 ± 0.006	0.51 ± 0.011	0.44 ± 0.008	0.344 ± 0.177
T2V_C2V	0.699 ± 0.027	0.11 ± 0.015	0.422 ± 0.011	0.526 ± 0.018	0.469 ± 0.014	0.362 ± 0.186
T2V_D2V	0.694 ± 0.016	0.101 ± 0.019	0.394 ± 0.011	0.517 ± 0.013	0.447 ± 0.011	0.342 ± 0.185
T2V_WW2V	0.673 ± 0.016	0.101 ± 0.013	0.385 ± 0.014	0.493 ± 0.02	0.432 ± 0.016	0.337 ± 0.182
WW2V_C2V	0.696 ± 0.014	0.102 ± 0.013	0.418 ± 0.009	0.524 ± 0.012	0.465 ± 0.01	0.356 ± 0.193
T2V_C2V_D2V	0.723 ± 0.015	0.109 ± 0.011	0.42 ± 0.012	0.55 ± 0.016	0.476 ± 0.013	0.37 ± 0.185
T2V_WW2V_D2V	0.699 ± 0.022	0.097 ± 0.011	0.396 ± 0.013	0.522 ± 0.015	0.45 ± 0.013	0.354 ± 0.173
WW2V_C2V_D2V	0.714 ± 0.012	0.102 ± 0.016	0.41 ± 0.009	0.539 ± 0.011	0.466 ± 0.009	0.359 ± 0.191
WW2V_C2V_T2V	0.697 ± 0.018	0.108 ± 0.012	0.421 ± 0.009	0.526 ± 0.012	0.468 ± 0.01	0.363 ± 0.19

Table 5.17: Binary Relevance: Random Forest + ENN.

Vectorized Input	ALO	EMA	PR-mic	RC-mic	F1-mic	F1-mac
C2V	0.612 ± 0.029	0.077 ± 0.012	0.297 ± 0.013	0.453 ± 0.021	0.359 ± 0.014	0.238 ± 0.175
T2V	0.647 ± 0.018	0.103 ± 0.014	0.387 ± 0.016	0.46 ± 0.014	0.42 ± 0.015	0.286 ± 0.206
D2V	0.564 ± 0.017	0.145 ± 0.013	0.483 ± 0.009	0.388 ± 0.019	0.43 ± 0.015	0.276 ± 0.214
WW2V	0.517 ± 0.02	0.138 ± 0.016	0.507 ± 0.017	0.351 ± 0.015	0.415 ± 0.015	0.275 ± 0.2
D2V_C2V	0.555 ± 0.001	0.154 ± 0.01	0.516 ± 0.003	0.384 ± 0.002	0.44 ± 0.002	0.282 ± 0.227
D2V_WW2V	0.567 ± 0.017	0.147 ± 0.01	0.513 ± 0.013	0.388 ± 0.012	0.442 ± 0.011	0.295 ± 0.204
T2V_C2V	0.676 ± 0.023	0.136 ± 0.013	0.445 ± 0.016	0.492 ± 0.016	0.467 ± 0.014	0.327 ± 0.219
T2V_D2V	0.584 ± 0.023	0.144 ± 0.01	0.488 ± 0.015	0.401 ± 0.018	0.44 ± 0.016	0.284 ± 0.224
T2V_WW2V	0.561 ± 0.024	0.154 ± 0.016	0.505 ± 0.018	0.388 ± 0.018	0.439 ± 0.017	0.297 ± 0.207
WW2V_C2V	0.593 ± 0.018	0.163 ± 0.011	0.539 ± 0.013	0.419 ± 0.014	0.471 ± 0.011	0.324 ± 0.219
T2V_C2V_D2V	0.638 ± 0.022	0.165 ± 0.013	0.516 ± 0.018	0.449 ± 0.02	0.48 ± 0.017	0.324 ± 0.228
T2V_WW2V_D2V	0.57 ± 0.02	0.153 ± 0.013	0.517 ± 0.019	0.393 ± 0.017	0.447 ± 0.016	0.296 ± 0.219
WW2V_C2V_D2V	0.62 ± 0.018	0.172 ± 0.012	0.541 ± 0.015	0.434 ± 0.014	0.482 ± 0.012	0.322 ± 0.228
WW2V_C2V_T2V	0.609 ± 0.02	0.167 ± 0.011	0.533 ± 0.02	0.428 ± 0.014	0.475 ± 0.015	0.322 ± 0.228

Table 5.18: Binary Relevance: XGBoost + Fake Balancer.

Vectorized Input	ALO	EMA	PR-mic	RC-mic	F1-mic	F1-mac
C2V	0.726 ± 0.028	0.046 ± 0.01	0.279 ± 0.013	0.562 ± 0.021	0.373 ± 0.015	0.252 ± 0.172
T2V	0.78 ± 0.019	0.066 ± 0.01	0.339 ± 0.005	0.586 ± 0.012	0.429 ± 0.005	0.306 ± 0.191
D2V	0.731 ± 0.019	0.098 ± 0.011	0.393 ± 0.01	0.533 ± 0.009	0.452 ± 0.008	0.303 ± 0.209
WW2V	0.713 ± 0.024	0.099 ± 0.011	0.402 ± 0.009	0.519 ± 0.016	0.453 ± 0.01	0.318 ± 0.196
D2V_C2V	0.725 ± 0.009	0.118 ± 0.014	0.427 ± 0.014	0.538 ± 0.0	0.476 ± 0.009	0.312 ± 0.228
D2V_WW2V	0.735 ± 0.02	0.122 ± 0.012	0.423 ± 0.012	0.54 ± 0.023	0.474 ± 0.015	0.326 ± 0.211
T2V_C2V	0.801 ± 0.024	0.082 ± 0.011	0.383 ± 0.015	0.619 ± 0.017	0.473 ± 0.016	0.339 ± 0.213
T2V_D2V	0.746 ± 0.012	0.102 ± 0.015	0.4 ± 0.009	0.552 ± 0.008	0.464 ± 0.007	0.318 ± 0.218
T2V_WW2V	0.745 ± 0.013	0.106 ± 0.014	0.404 ± 0.009	0.553 ± 0.01	0.467 ± 0.01	0.323 ± 0.21
WW2V_C2V	0.753 ± 0.02	0.11 ± 0.014	0.429 ± 0.011	0.572 ± 0.015	0.49 ± 0.011	0.345 ± 0.211
T2V_C2V_D2V	0.784 ± 0.012	0.114 ± 0.008	0.428 ± 0.008	0.592 ± 0.013	0.497 ± 0.008	0.35 ± 0.22
T2V_WW2V_D2V	0.754 ± 0.016	0.119 ± 0.02	0.422 ± 0.01	0.552 ± 0.012	0.478 ± 0.009	0.331 ± 0.215
WW2V_C2V_D2V	0.76 ± 0.017	0.124 ± 0.009	0.444 ± 0.013	0.57 ± 0.017	0.499 ± 0.013	0.35 ± 0.216
WW2V_C2V_T2V	0.785 ± 0.01	0.123 ± 0.015	0.435 ± 0.009	0.596 ± 0.012	0.503 ± 0.009	0.358 ± 0.215

Table 5.19: Binary Relevance: XGBoost + ENN.

As highlighted by the results, the chosen resampling strategy induced an improvement of F1 scores but also a worsening of EMA scores. This is mainly due to the fact that every classifier trained on an undersampled dataset loses potentially useful information relative to the majority class, which thus becomes harder to be correctly predicted. This results in an improvement of the Recall and in a worsening of the Precision (and thus of EMA), since more labels are predicted as relevant, especially for the minority class, but more predictions turn out to be wrong. In our case, the improvement of the Recall was higher than the worsening of the Precision: this explains the general improvement of the F1 score.
As regards the tested inputs, it turned out that combined vectorization models outperformed the "basic" ones. In particular, combinations generated from the concatenation of three vectorizations turned out to be the best for the classification task. It is interesting to note that all the combinations including the Cast2Vec vectorization gave significantly better results than those not including it: this confirmed the high discriminative power of Cast2Vec model.
Based on the considerations just mentioned, we decided to perform a further selection process on the vectorization models to be used in subsequent analysis, as described in section 4.5.3. In particular, we decided to discard all the "basic" inputs and all the inputs generated from the concatenation of two vectorizations. The only exception was the Topic2Vec_Cast2Vec combination, which was not discarded thanks to its high performance in spite of its low dimensions. Among the combinations not already discarded, we identified the one named Topic2Vec_WeightedWord2Vec_Doc2Vec as the worst one in terms of classification performance, probably because of the absence of the Cast2Vec vectorization in its components. Therefore, we decided to submit it to statistical tests, in order to compare it with other combinations to possibly highlight a significant difference and thus drop it out from subsequent analysis. As in section 5.6.2, the statistical tests we performed were Student's t-test and Mann–Whitney U test. Since a direct comparison between vectorized inputs was not possible, we used as the benchmark all the previously evaluated classifiers. For each classifier, we compared the performances obtained (without resampling) using Topic2Vec_WeightedWord2Vec_Doc2Vec against the performances obtained (without resampling) using the other 3 combined vectorized inputs: a Bonferroni correction factor of 3 was thus applied to the test (i. e. the models are significantly different if p-value $< 0,0167$).
Tables 5.20, 5.21, 5.22, 5.23 and 5.24 show results of performed tests.

Vectorized Input	**p-value t-test T2V_WW2V_D2V**	**p-value U-test T2V_WW2V_D2V**
T2V_C2V_D2V	0.0007	0.0007
WW2V_C2V_D2V	0.0009	0.0007
WW2V_C2V_T2V	0.0037	0.0057

Table 5.20: Combined input comparison: Naive Bayes.

Vectorized Input	p-value t-test T2V_WW2V_D2V	p-value U-test T2V_WW2V_D2V
T2V_C2V_D2V	0.0000	0.0001
WW2V_C2V_D2V	0.0010	0.0011
WW2V_C2V_T2V	0.0010	0.0011

Table 5.21: Combined input comparison: Logistic Regression.

Vectorized Input	p-value t-test T2V_WW2V_D2V	p-value U-test T2V_WW2V_D2V
T2V_C2V_D2V	0.0124	0.0156
WW2V_C2V_D2V	0.0013	0.0018
WW2V_C2V_T2V	0.0126	0.0046

Table 5.22: Combined input comparison: Support Vector Machine.

Vectorized Input	p-value t-test T2V_WW2V_D2V	p-value U-test T2V_WW2V_D2V
T2V_C2V_D2V	0.0001	0.0005
WW2V_C2V_D2V	0.0193	0.0106
WW2V_C2V_T2V	0.0117	0.0106

Table 5.23: Combined input comparison: Random Forest.

Vectorized Input	p-value t-test T2V_WW2V_D2V	p-value U-test T2V_WW2V_D2V
T2V_C2V_D2V	0.0004	0.0004
WW2V_C2V_D2V	0.0001	0.0003
WW2V_C2V_T2V	0.0010	0.0011

Table 5.24: Combined input comparison: XGBoost.

All the performed statistical tests highlighted a significant difference between the candidate vectorization (Topic2Vec_WeightedWord2Vec_Doc2Vec) and all the other ones, irrespective of the selected classifiers.
Therefore, we decided to filter out Topic2Vec_WeightedWord2Vec_Doc2Vec from subsequent analysis.

5.6.5 *Binary Relevance with Stacking*

As stated in section 4.5.2, testing the Binary Relevance approach with Stacking was computationally expensive, because of some constraints that our client had on the usage of computational resources. Therefore, we decided to test it using only some of the best combined inputs, found in the previous step, and all the selected resampling strategies. As regards the chosen base classifiers for Stacking, we decided to choose some combinations with three or four base classifiers, keeping in mind that Stacking is most effective when base classifiers are significantly different.

▷ | T2V_C2V | T2V_C2V_D2V | *(Vectorized Inputs)*

▷ | LR | *(Stacking Classifier)*

▷ | RF + SVM + LR | RF + SVM + XGB | RF + NB + SVM + XGB | *(Base Classifiers)*

▷ | FB | ENN | *(Balancers)*

The evaluation was performed throguh stratified 10-fold cross-validation, as described in section 5.2. Results are shown in **Tables** 5.25, 5.26 and 5.27.

Vectorized Input	ALO	EMA	PR-mic	RC-mic	F1-mic	F1-mac
T2V_C2V [FB]	0.618 ± 0.012	0.133 ± 0.01	0.466 ± 0.011	0.455 ± 0.009	0.46 ± 0.01	0.356 ± 0.191
T2V_C2V [ENN]	0.705 ± 0.01	0.119 ± 0.007	0.431 ± 0.003	0.534 ± 0.008	0.477 ± 0.005	0.369 ± 0.193
T2V_C2V_D2V [FB]	0.647 ± 0.017	0.144 ± 0.011	0.488 ± 0.01	0.476 ± 0.009	0.482 ± 0.009	0.378 ± 0.193
T2V_C2V_D2V [ENN]	0.735 ± 0.022	0.121 ± 0.014	0.435 ± 0.008	0.561 ± 0.013	0.49 ± 0.009	0.378 ± 0.195

Table 5.25: Binary Relevance: Stacking [RF + SVM + LR].

Vectorized Input	ALO	EMA	PR-mic	RC-mic	F1-mic	F1-mac
T2V_C2V [FB]	0.632 ± 0.02	0.136 ± 0.016	0.473 ± 0.017	0.461 ± 0.016	0.467 ± 0.016	0.364 ± 0.191
T2V_C2V [ENN]	0.71 ± 0.022	0.121 ± 0.011	0.434 ± 0.014	0.536 ± 0.014	0.48 ± 0.014	0.372 ± 0.19
T2V_C2V_D2V [FB]	0.648 ± 0.023	0.143 ± 0.02	0.488 ± 0.013	0.478 ± 0.016	0.483 ± 0.015	0.377 ± 0.196
T2V_C2V_D2V [ENN]	0.742 ± 0.013	0.124 ± 0.012	0.44 ± 0.011	0.565 ± 0.012	0.495 ± 0.011	0.383 ± 0.198

Table 5.26: Binary Relevance: Stacking [RF + SVM + XGB].

Vectorized Input	ALO	EMA	PR-mic	RC-mic	F1-mic	F1-mac
T2V_C2V [FB]	0.628 ± 0.017	0.137 ± 0.008	0.474 ± 0.008	0.461 ± 0.01	0.468 ± 0.009	0.363 ± 0.193
T2V_C2V [ENN]	0.703 ± 0.021	0.12 ± 0.016	0.435 ± 0.014	0.534 ± 0.016	0.479 ± 0.015	0.37 ± 0.192
T2V_C2V_D2V [FB]	0.648 ± 0.018	0.147 ± 0.014	0.49 ± 0.011	0.477 ± 0.014	0.483 ± 0.012	0.373 ± 0.202
T2V_C2V_D2V [ENN]	0.74 ± 0.01	0.122 ± 0.014	0.439 ± 0.006	0.563 ± 0.007	0.494 ± 0.006	0.384 ± 0.193

Table 5.27: Binary Relevance: Stacking [RF + NB + SVM + XGB].

As can be seen from results, this approach did not improve the classification performance: the obtained scores were in fact perfectly comparable with those obtained with previously Binary Relevance approaches. This was probably due to the incapacity of base classifiers to capture and learn different facets of the given data: indeed, as stated in section 2.4.3, Stacking is most effective when the base models are significantly different.

5.6.6 *Ensemble of Classifier Chain*

The main drawback of the Binary Relevance approach was the inability of taking into account the correlation between labels: therefore, as described in section 4.6, we decided to deal with the problem testing the *Ensemble of Classifier Chain* approach. Almost all the chosen classifiers were used with this approach, using as input all the selected combined vectorizations (see section 5.6.4) without any resampling strategy (see section 4.6). Again, the only exception was represented by the Stacking classifier which, being computationally expensive, was left out from this analysis.

▷ T2V_C2V | T2V_C2V_D2V | WW2V_C2V_D2V | WW2V_C2V_T2V *(Vectorized Inputs)*

▷ NB | LR | SVM | RF | XGB *(Classifiers)*

▷ FB *(Balancers)*

The evaluation was performed throguh stratified 10-fold cross-validation, as described in section 5.2. Results are shown in **Tables** 5.28, 5.29, 5.30, 5.31 and 5.32.

Vectorized Input	ALO	EMA	PR-mic	RC-mic	F1-mic	F1-mac
T2V_C2V	0.571 ± 0.021	0.09 ± 0.016	0.424 ± 0.017	0.431 ± 0.017	0.427 ± 0.017	0.281 ± 0.208
T2V_C2V_D2V	0.57 ± 0.011	0.076 ± 0.017	0.403 ± 0.013	0.423 ± 0.013	0.413 ± 0.011	0.288 ± 0.193
WW2V_C2V_D2V	0.546 ± 0.024	0.065 ± 0.011	0.378 ± 0.016	0.412 ± 0.023	0.394 ± 0.015	0.278 ± 0.172
WW2V_C2V_T2V	0.553 ± 0.009	0.079 ± 0.011	0.395 ± 0.011	0.415 ± 0.008	0.405 ± 0.009	0.287 ± 0.181

Table 5.28: Ensemble of Classifier Chain: Naive Bayes.

Vectorized Input	ALO	EMA	PR-mic	RC-mic	F1-mic	F1-mac
T2V_C2V	0.685 ± 0.028	0.171 ± 0.017	0.459 ± 0.018	0.484 ± 0.023	0.471 ± 0.015	0.324 ± 0.214
T2V_C2V_D2V	0.68 ± 0.018	0.138 ± 0.012	0.429 ± 0.02	0.481 ± 0.018	0.453 ± 0.013	0.338 ± 0.194
WW2V_C2V_D2V	0.611 ± 0.019	0.116 ± 0.013	0.405 ± 0.02	0.429 ± 0.014	0.416 ± 0.013	0.321 ± 0.175
WW2V_C2V_T2V	0.613 ± 0.018	0.121 ± 0.01	0.403 ± 0.011	0.427 ± 0.017	0.415 ± 0.012	0.324 ± 0.172

Table 5.29: Ensemble of Classifier Chain: Logistic Regression.

Vectorized Input	ALO	EMA	PR-mic	RC-mic	F1-mic	F1-mac
T2V_C2V	0.689 ± 0.02	0.178 ± 0.015	0.458 ± 0.018	0.482 ± 0.016	0.47 ± 0.015	0.293 ± 0.247
T2V_C2V_D2V	0.703 ± 0.026	0.173 ± 0.019	0.481 ± 0.022	0.498 ± 0.018	0.489 ± 0.019	0.357 ± 0.207
WW2V_C2V_D2V	0.685 ± 0.014	0.166 ± 0.016	0.468 ± 0.014	0.487 ± 0.013	0.477 ± 0.011	0.374 ± 0.187
WW2V_C2V_T2V	0.69 ± 0.022	0.175 ± 0.019	0.471 ± 0.019	0.489 ± 0.02	0.48 ± 0.018	0.372 ± 0.196

Table 5.30: Ensemble of Classifier Chain: Support Vector Machine.

Vectorized Input	ALO	EMA	PR-mic	RC-mic	F1-mic	F1-mac
T2V_C2V	0.425 ± 0.016	0.152 ± 0.009	0.618 ± 0.021	0.286 ± 0.011	0.391 ± 0.014	0.204 ± 0.235
T2V_C2V_D2V	0.308 ± 0.028	0.117 ± 0.012	0.713 ± 0.037	0.199 ± 0.017	0.311 ± 0.023	0.149 ± 0.205
WW2V_C2V_D2V	0.198 ± 0.018	0.087 ± 0.009	0.782 ± 0.025	0.123 ± 0.01	0.212 ± 0.015	0.098 ± 0.156
WW2V_C2V_T2V	0.227 ± 0.012	0.095 ± 0.009	0.769 ± 0.028	0.143 ± 0.006	0.24 ± 0.009	0.114 ± 0.17

Table 5.31: Ensemble of Classifier Chain: Random Forest.

Vectorized Input	ALO	EMA	PR-mic	RC-mic	F1-mic	F1-mac
T2V_C2V	0.678 ± 0.015	0.169 ± 0.012	0.474 ± 0.012	0.482 ± 0.013	0.478 ± 0.011	0.321 ± 0.231
T2V_C2V_D2V	0.677 ± 0.018	0.196 ± 0.012	0.527 ± 0.014	0.476 ± 0.011	0.5 ± 0.011	0.327 ± 0.246
WW2V_C2V_D2V	0.656 ± 0.015	0.203 ± 0.017	0.554 ± 0.02	0.46 ± 0.01	0.503 ± 0.012	0.328 ± 0.239
WW2V_C2V_T2V	0.654 ± 0.023	0.192 ± 0.012	0.541 ± 0.012	0.456 ± 0.015	0.495 ± 0.013	0.329 ± 0.236

Table 5.32: Ensemble of Classifier Chain: XGBoost.

As can be seen from the results, the performance obtained using Random Forest as "base classifier" were much worse than those obtained using the same classifier in the Binary Relevance approach. The reasons behind this bad result are difficult to understand and need further study, but are probably related to the splitting strategy adopted by the classifier during the generation of trees (see section 2.4.3), which tends to favor splitting on the predictions made by previous classifiers in the chain, loosing thus potentially useful information from input data.
As regards other models, some results highlighted an improvement in the Binary Relevance approach, while others showed little or no change. In order to asses the difference between every Classifier Chain model and the respective Binary Relevance model, we decided to perform some statistical tests, comparing only models related to the same combined input. As in section 5.6.2, the statistical tests we performed were Student's t-test and Mann–Whitney U test. Since for each Classifier Chain model we tested 4 different inputs, a Bonferroni correction factor of 4 was thus applied to the test (i. e. the models are significantly different if p-value $< 0,0125$).
Tables 5.33, 5.34, 5.35 and 5.36 show results of performed tests.

Vectorized Input	p-value t-test BR vs ECC	p-value U-test BR vs ECC
T2V_C2V	0.0038	0.0023
T2V_C2V_D2V	0.0801	0.0378
WW2V_C2V_D2V	0.2100	0.0702
WW2V_C2V_T2V	0.0002	0.0009

Table 5.33: ECC vs BR comparison: Naive Bayes.

Vectorized Input	p-value t-test BR vs ECC	p-value U-test BR vs ECC
T2V_C2V	0.0016	0.0007
T2V_C2V_D2V	0.7072	0.3957
WW2V_C2V_D2V	0.0305	0.0188
WW2V_C2V_T2V	0.2378	0.1537

Table 5.34: ECC vs BR comparison: Logistic Regression.

Vectorized Input	p-value t-test BR vs ECC	p-value U-test BR vs ECC
T2V_C2V	0.0000	0.0001
T2V_C2V_D2V	0.0000	0.0001
WW2V_C2V_D2V	0.0003	0.0009
WW2V_C2V_T2V	0.0000	0.0007

Table 5.35: ECC vs BR comparison: Support Vector Machine.

Vectorized Input	p-value t-test BR vs ECC	p-value U-test BR vs ECC
T2V_C2V	0.0846	0.0606
T2V_C2V_D2V	0.0095	0.0070
WW2V_C2V_D2V	0.0019	0.0029
WW2V_C2V_T2V	0.0080	0.0070

Table 5.36: ECC vs BR comparison: XGBoost.

As highlighted by the performed statistical tests, the ECC-XGB and the ECC-SVM approach obtained significantly better performance than those obtained with the BR-XGB and BR-SVM approach: this can be interpreted as an evidence of the effectiveness of ECC approach in capturing correlation between labels. Moreover, looking at the previous results of the models, it easy to observe an improvement of the EMA metrics, which clearly confirms the models' ability to take into account label correlation and thus of making more accurate predictions.

5.6.7 *Stacking Aggregation*

The last experiment we performed in our analysis was testing *Stacking Aggregation* approach, which allows overcoming the main drawbacks related with Binary Relevance and Classifier Chains. Indeed, thanks to this approach, it is possible to take into account the label correlation and counter-act the imbalance problem at the same time (see section 4.7). As in the case of Binary Relevance with Stacking, also this approach revealed to be computationally expensive: therefore, we decided to test it using the same combined inputs, the same resampling strategies and the same combination of base classifiers selected for testing Binary Relevance with Stacking (see section 5.6.5).

▷ | T2V_C2V | T2V_C2V_D2V | *(Vectorized Inputs)*

▷ | LR | *(Stacking Classifier)*

▷ | RF + SVM + LR | RF + SVM + XGB | RF + NB + SVM + XGB | *(Base Classifiers)*

▷ | FB | ENN | *(Balancers)*

The evaluation was performed throguh stratified 10-fold cross-validation, as described in section 5.2. Results are shown in **Tables** 5.37, 5.38 and 5.39.

Vectorized Input	ALO	EMA	PR-mic	RC-mic	F1-mic	F1-mac
T2V_C2V [FB]	0.655 ± 0.017	0.148 ± 0.015	0.476 ± 0.017	0.476 ± 0.019	0.476 ± 0.018	0.331 ± 0.219
T2V_C2V [ENN]	0.657 ± 0.026	0.152 ± 0.016	0.477 ± 0.018	0.476 ± 0.019	0.477 ± 0.018	0.32 ± 0.225
T2V_C2V_D2V [FB]	0.669 ± 0.026	0.155 ± 0.017	0.487 ± 0.023	0.486 ± 0.023	0.487 ± 0.023	0.336 ± 0.227
T2V_C2V_D2V [ENN]	0.681 ± 0.017	0.159 ± 0.015	0.49 ± 0.008	0.489 ± 0.012	0.489 ± 0.009	0.329 ± 0.228

Table 5.37: Stacking Aggregation: RF + SVM + LR.

Vectorized Input	ALO	EMA	PR-mic	RC-mic	F1-mic	F1-mac
T2V_C2V [FB]	0.663 ± 0.017	0.16 ± 0.011	0.478 ± 0.012	0.478 ± 0.009	0.478 ± 0.01	0.323 ± 0.226
T2V_C2V [ENN]	0.668 ± 0.015	0.161 ± 0.017	0.482 ± 0.01	0.481 ± 0.01	0.482 ± 0.01	0.318 ± 0.236
T2V_C2V_D2V [FB]	0.681 ± 0.021	0.165 ± 0.02	0.493 ± 0.017	0.494 ± 0.017	0.494 ± 0.016	0.341 ± 0.228
T2V_C2V_D2V [ENN]	0.69 ± 0.017	0.162 ± 0.019	0.498 ± 0.013	0.497 ± 0.011	0.497 ± 0.012	0.335 ± 0.233

Table 5.38: Stacking Aggregation: RF + SVM + XGB.

Vectorized Input	ALO	EMA	PR-mic	RC-mic	F1-mic	F1-mac
T2V_C2V [FB]	0.657 ± 0.018	0.159 ± 0.016	0.472 ± 0.014	0.473 ± 0.015	0.472 ± 0.014	0.319 ± 0.229
T2V_C2V [ENN]	0.657 ± 0.018	0.158 ± 0.011	0.473 ± 0.015	0.473 ± 0.016	0.473 ± 0.015	0.31 ± 0.232
T2V_C2V_D2V [FB]	0.674 ± 0.02	0.16 ± 0.011	0.485 ± 0.011	0.485 ± 0.012	0.485 ± 0.011	0.329 ± 0.231
T2V_C2V_D2V [ENN]	0.687 ± 0.02	0.165 ± 0.018	0.494 ± 0.013	0.494 ± 0.014	0.494 ± 0.013	0.331 ± 0.233

Table 5.39: Stacking Aggregation: RF + NB + SVM + XGB.

As can be seen from results, this approach did not improve the classification performance: the obtained F1 scores were, in fact, comparable with those obtained with previous approaches. However, looking at the EMA metrics, it turns out that the Stacking Aggregation approach is significantly better than the Binary Relevance approach with Stacking used as binary classifier: this can be interpreted as evidence of the effectiveness of the Stacking Aggregation approach, which is able to capture the correlation between labels and thus to make more accurate predictions.

6 RESULTS AND FUTURE DEVELOPMENTS

In this chapter we discuss the results of our work, making some considerations over the different approaches we used and the input vectorizations. Then we propose possible future developments looking at different architectures and target metadata.

6.1 RESULTS

The goal of our thesis was to study the possibility of providing automatic suggestion to human experts in order to speed up the process of tagging. Our work laid the foundations to further explore a way to automate this process, and obtained interesting results in this direction, showing how movie summaries can be useful to the prediction of movie metadata. Using Binary Relevance as a benchmark classification method, we analyzed the different vectorizations methods and the contributions of balancing techniques. Among the different vectorization, TF-IDF and Bag-of-Words resulted inferior to the others text vectorizations in the F1-micro metrics (see section 5.6.2), as confirmed the statistical tests of Student's t-test and Mann–Whitney U test, and thus have been discarded. We then proceeded testing balancing techniques, highlighting the obtained improvements and drops in performance metrics, depending on the used technique: in fact ENN showed a significant increase in the recall metric and F1-micro metric, despite the lowering of the precision, while the other techniques we choose, instead, were not effective, lowering the score of the of all the classification methods we tested. Combining different vectorizations, we observed how those combinations could enhance the classification metrics, especially when a domain-specific feature, as our proposed Cast2Vec, is used with text vectorizations. The best results were obtained by using concatenations of three vectorizations, in the specific two text vectorization and the Cast2Vec vectorization (see section 5.6.4). Furthermore, we addressed the importance of approaches that account for *mood* correlation using ECC and Stacking Aggregation, which were able to produce, on average, a greater number of exactly matching predictions with respect to approaches like Binary Relevance, which don't capture this information. The best models of our thesis come in fact from the ECC approach with XGBoost as a base classifier (see section 5.6.6), with which we obtained F1-micro values of **0.503** and EMA of **0.203**.

6.2 FUTURE DEVELOPMENTS

Our work was centred on investigating the possibility of the automatic prediction of *mood* metadata. By focusing on it, we set the stage for future studies on the different metadata designed by Mediaset Strategic Marketing. In fact, starting from our multi-label architecture, we can assume its reuse for predicting other metadata having similar properties to *moods* (e. g. *demographic target*), but it's also easy to imagine some architectural adaptations to predict metadata with different properties, like the single-label ones.

Possible feature developments could also include improvements to our machine learning architecture. A first improvement could be to modify the implementation of the Ensemble of Classifier Chains model we used, in order to cope with the imbalance problem, as recently described in [28]. Another refinement could be to test the Stacking based approach using more and new combinations of base classifiers, in order to see if they could produce better results.
Another possible development could be to address the classification task testing deep-learning approaches, such as *Convolutional Neural Networks* and *Long Short Term Memory Networks*, which proved their effectiveness in many natural language areas [20, 24, 15].

Given the improvement we obtained by using Cast2Vec vectorization as a domain-specific feature, we could imagine the possibility of identifying more domain-specific information able to produce similar improvements, such as the classical movie genre or the information regarding the technical cast (i. e. movie director, music director, costume designer), which are easily available on the internet. Besides domain-specific features, another improvement over the simple usage of textual information could be the addition of features extracted from audio and video sources. These features could be used in combination with text vectorizations or could completely replace them, changing thus the way of addressing the classification problem. Working with audio data, as presented in [7], it could be possible to analyse different audio frequencies to extract characteristics typical of certain movies, such as explosions, gunfire and romantic music. Working with video data instead, it could be possible, for example, to perform a chromatic analysis of the source as well as applying techniques for shot detection and motion intensity estimation. For studies on this subject, Convolutional Neural Networks should be taken into consideration, thanks to their ability to extract complex features from visual inputs [49].

BIBLIOGRAPHY

[1] Peter Armitage, Geoffrey Berry, and John Nigel Scott Matthews. *Statistical methods in medical research*. John Wiley & Sons, 2008 (cit. on p. 35).

[2] J Martin Bland and Douglas G Altman. "Multiple significance tests: the Bonferroni method." In: *Bmj* 310.6973 (1995), p. 170 (cit. on p. 43).

[3] David M Blei, Andrew Y Ng, and Michael I Jordan. "Latent dirichlet allocation." In: *Journal of machine Learning research* 3.Jan (2003), pp. 993–1022 (cit. on p. 9).

[4] Leo Breiman. "Bagging predictors." In: *Machine learning* 24.2 (1996), pp. 123–140 (cit. on p. 10).

[5] Leo Breiman. "Random forests." In: *Machine learning* 45.1 (2001), pp. 5–32 (cit. on pp. 2, 11).

[6] L Breiman et al. "Classification and Regression Trees." In: (1984) (cit. on p. 10).

[7] Darin Brezeale and Diane J Cook. "Automatic video classification: A survey of the literature." In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.3 (2008), pp. 416–430 (cit. on pp. 18, 66).

[8] Darin Brezeale and Diane J Cook. "Using closed captions and visual features to classify movies by genre." In: *Poster session of the seventh international workshop on Multimedia Data Mining (MDM/KDD2006)*. 2006 (cit. on p. 18).

[9] Ricardo Cerri and André CPLF de Carvalho. "Hierarchical multilabel protein function prediction using local neural networks." In: *Brazilian Symposium on Bioinformatics*. Springer. 2011, pp. 10–17 (cit. on p. 5).

[10] Nitesh V Chawla et al. "SMOTE: synthetic minority over-sampling technique." In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357 (cit. on pp. 2, 15).

[11] Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system." In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM. 2016, pp. 785–794 (cit. on pp. 2, 11, 37).

[12] Corinna Cortes and Vladimir Vapnik. "Support-vector networks." In: *Machine learning* 20.3 (1995), pp. 273–297 (cit. on pp. 2, 10, 31).

[13] Johannes Fürnkranz et al. "Multilabel classification via calibrated label ranking." In: *Machine learning* 73.2 (2008), pp. 133–153 (cit. on p. 12).

[14] Shantanu Godbole and Sunita Sarawagi. "Discriminative methods for multi-labeled classification." In: *Pacific-Asia conference on knowledge discovery and data mining*. Springer. 2004, pp. 22–30 (cit. on pp. 2, 36).

[15] Yoav Goldberg. "A primer on neural network models for natural language processing." In: *Journal of Artificial Intelligence Research* 57 (2016), pp. 345–420 (cit. on p. 66).

[16] Alexander G Hauptmann et al. "Video Classification and Retrieval with the Informedia Digital Video Library System." In: *TREC*. 2002 (cit. on p. 18).

[17] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*. Vol. 398. John Wiley & Sons, 2013 (cit. on pp. 2, 10).

[18] Gareth James et al. *An introduction to statistical learning*. Vol. 112. Springer, 2013 (cit. on p. 5).

[19] Nathalie Japkowicz. "The class imbalance problem: Significance and strategies." In: *Proc. of the Int'l Conf. on Artificial Intelligence*. 2000 (cit. on pp. 2, 15).

[20] Rie Johnson and Tong Zhang. "Effective use of word order for text categorization with convolutional neural networks." In: *arXiv preprint arXiv:1412.1058* (2014) (cit. on p. 66).

[21] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. 2001 (cit. on p. 37).

[22] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. "Multilabel text classification for automated tag suggestion." In: *Proceedings of the ECML/PKDD*. Vol. 18. 2008 (cit. on p. 5).

[23] Ashraf M Kibriya et al. "Multinomial naive bayes for text categorization revisited." In: *Australasian Joint Conference on Artificial Intelligence*. Springer. 2004, pp. 488–499 (cit. on pp. 2, 10).

[24] Siwei Lai et al. "Recurrent Convolutional Neural Networks for Text Classification." In: *AAAI*. Vol. 333. 2015, pp. 2267–2273 (cit. on p. 66).

[25] Quoc Le and Tomas Mikolov. "Distributed representations of sentences and documents." In: *International Conference on Machine Learning*. 2014, pp. 1188–1196 (cit. on p. 8).

[26] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. "Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning." In: *Journal of Machine Learning Research* 18.17 (2017), pp. 1–5 (cit. on p. 37).

[27] Tao Li and Mitsunori Ogihara. "Detecting emotion in music." In: *ISMIR, 4th International Conference on Music Information Retrieval* (2003) (cit. on p. 5).

[28] Bin Liu and Grigorios Tsoumakas. "Making Classifier Chains Resilient to Class Imbalance." In: *arXiv preprint arXiv:1807.11393* (2018) (cit. on pp. 36, 66).

[29] Andrew Kachites McCallum. "MALLET: A Machine Learning for Language Toolkit." 2002 (cit. on p. 37).

[30] Tomas Mikolov et al. "Distributed representations of words and phrases and their compositionality." In: *Advances in neural information processing systems*. 2013, pp. 3111–3119 (cit. on p. 8).

[31] Tomas Mikolov et al. "Efficient estimation of word representations in vector space." In: *arXiv preprint arXiv:1301.3781* (2013) (cit. on p. 8).

[32] Zivorad M Milenovic. "Application of Mann-Whitney U test in research of professional training of primary school teachers." In: *Metodicki obzori* 6.1 (2011), pp. 73–9 (cit. on p. 35).

[33] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 37).

[34] John Platt. “Sequential minimal optimization: A fast algorithm for training support vector machines.” In: (1998) (cit. on p. 31).

[35] Martin F Porter. “An algorithm for suffix stripping.” In: *Program* 14.3 (1980), pp. 130–137 (cit. on p. 7).

[36] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014 (cit. on p. 10).

[37] J Ross Quinlan. “Discovering rules by induction from large collections of examples.” In: *Expert systems in the micro electronics age* (1979) (cit. on p. 10).

[38] Zeeshan Rasheed and Mubarak Shah. “Movie genre classification by exploiting audio-visual features of previews.” In: *Pattern Recognition, 2002. Proceedings. 16th International Conference on*. Vol. 2. IEEE. 2002, pp. 1086–1089 (cit. on p. 18).

[39] Jesse Read et al. “Classifier chains for multi-label classification.” In: *Machine learning* 85.3 (2011), p. 333 (cit. on pp. 2, 13).

[40] Jesse Read et al. “MEKA: A Multi-label/Multi-target Extension to WEKA.” In: *Journal of Machine Learning Research* 17.21 (2016), pp. 1–5 (cit. on p. 37).

[41] Radim Řehůřek and Petr Sojka. “Software Framework for Topic Modelling with Large Corpora.” English. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, May 2010, pp. 45–50 (cit. on p. 37).

[42] Mona Riemenschneider et al. “eccCL: parallelized GPU implementation of Ensemble Classifier Chains.” In: *BMC bioinformatics* 18.1 (2017), p. 371 (cit. on p. 13).

[43] Matthew Roach and John S Mason. “Classification of video genre using audio.” In: *Seventh European Conference on Speech Communication and Technology*. 2001 (cit. on p. 18).

[44] Michael Röder, Andreas Both, and Alexander Hinneburg. “Exploring the space of topic coherence measures.” In: *Proceedings of the eighth ACM international conference on Web search and data mining*. ACM. 2015, pp. 399–408 (cit. on p. 41).

[45] Hitesh Sajnani et al. “Multi-Label Classification of Short Text: A Study on Wikipedia Barnstars.” In: *Analyzing Microtext*. 2011 (cit. on p. 24).

[46] Gerard Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. New York: McGraw Hill Book Co., 1983 (cit. on pp. 7, 8).

[47] Robert E Schapire. “The strength of weak learnability.” In: *Machine learning* 5.2 (1990), pp. 197–227 (cit. on p. 11).

[48] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. *Introduction to information retrieval*. Vol. 39. Cambridge University Press, 2008 (cit. on p. 5).

[49] Gabriel S Simões et al. “Movie genre classification with convolutional neural networks.” In: *Neural Networks (IJCNN), 2016 International Joint Conference on*. IEEE. 2016, pp. 259–266 (cit. on p. 66).

[50] P. Szymański and T. Kajdanowicz. “A scikit-based Python environment for performing multi-label classification.” In: *arXiv e-print arXiv:1702.01460* (Feb. 2017) (cit. on pp. 37, 39).

[51] Kai Ming Ting and Ian H Witten. "Stacked Generalization: when does it work?" In: (1997) (cit. on p. 11).

[52] Grigorios Tsoumakas and Ioannis Katakis. "Multi-label classification: An overview." In: *International Journal of Data Warehousing and Mining (IJDWM)* 3.3 (2007), pp. 1–13 (cit. on pp. 5, 11).

[53] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. "Random k-labelsets for multilabel classification." In: *IEEE Transactions on Knowledge and Data Engineering* 23.7 (2011), pp. 1079–1089 (cit. on p. 12).

[54] S Vijayarani, Ms J Ilamathi, and Ms Nithya. "Preprocessing techniques for text mining-an overview." In: *International Journal of Computer Science & Communication Networks* 5.1 (2015), pp. 7–16 (cit. on p. 6).

[55] Dennis L Wilson. "Asymptotic properties of nearest neighbor rules using edited data." In: *IEEE Transactions on Systems, Man, and Cybernetics* 3 (1972), pp. 408–421 (cit. on pp. 2, 15).

[56] David H Wolpert. "Stacked generalization." In: *Neural networks* 5.2 (1992), pp. 241–259 (cit. on pp. 2, 11).

[57] Min-Ling Zhang, Yu-Kun Li, and Xu-Ying Liu. "Towards Class-Imbalance Aware Multi-Label Learning." In: *IJCAI*. 2015, pp. 4041–4047 (cit. on p. 20).

[58] Min-Ling Zhang and Zhi-Hua Zhou. "ML-KNN: A lazy learning approach to multi-label learning." In: *Pattern recognition* 40.7 (2007), pp. 2038–2048 (cit. on p. 12).

[59] Min-Ling Zhang and Zhi-Hua Zhou. "Multilabel neural networks with applications to functional genomics and text categorization." In: *IEEE transactions on Knowledge and Data Engineering* 18.10 (2006), pp. 1338–1351 (cit. on p. 12).

[60] Min-Ling Zhang et al. "Binary relevance for multi-label learning: an overview." In: *Frontiers of Computer Science* (2018), pp. 1–12 (cit. on pp. 2, 12).

A

APPENDIX: MOOD DESCRIPTION

Mood	Description
Breathtaking	The rhythm and the frenzy leave without breath.
Rousing	The inspiration, the grandeur and the heroism of the story rouse the viewer.
Feel-good	The content communicates a sense of well-being, puts the viewer in a position of strength.
Evasive	The vision is not very demanding, entertains the viewer.
Hilarious	The content capitalizes on making the viewer laugh.
Surprising	The content is capable of generating surprise compared to expectations generated by consolidated narratives (positive sense).
Comfortable	Content that instills tranquility, trust.
Sparkling	Content that enhances aesthetic dimensions, immersing the viewer in it.
Dreaming	Content that activates the oneiric dimension.
Sexy	The content produces sexual excitement.
Terrifying	The content aims to scare, to terrorize.
Disturbing	The content arouses anxiety, anguish.

Mood	**Description**
Unsettling	The content is capable of generating surprise compared to expectations generated by consolidated narratives, managing to create disturbance in the viewer (negative sense).
Poignant	Makes the viewer cry.
Bittersweet	Content not entirely comforting nor disheartening: pleasure veiled with sadness.
Inspiring	Content that gives food for thought and inspiration for one's own life.
Cerebral	Content in need of an intellectual effort to be fully enjoyed.
Melancholic	The content arouses a vague sadness, even sweet or pleased.
Romantic	The content is characterized by the expression of love.
Angry	The content arouses anger and indignation in the viewer.