Specialistic monotopic classification

Authors: Gaia Corbetta, Florin Andrei Rusu

Abstract
The goal of this project is to train a model for **classifying text topics**. Our dataset consists of texts in Italian from scraping legal-themed websites and documents. To perform the classification a thorough cleaning of the dataset was necessary. Several approaches for supervised classification have been considered including the use of the **LSTM** (*Long Short-Term Memory)* model, with **TF-IDF** (*term frequency–inverse document frequency*) vectorization. One of the main challenges was to deal with a large number of topics (labels) and texts that are correlated between each other. As a solution we propose two different approaches: grouping topics in **macro topics** and applying **data enrichment** processes. Then the models are evaluated based on their accuracy to predict the correct topic.

Table of contents

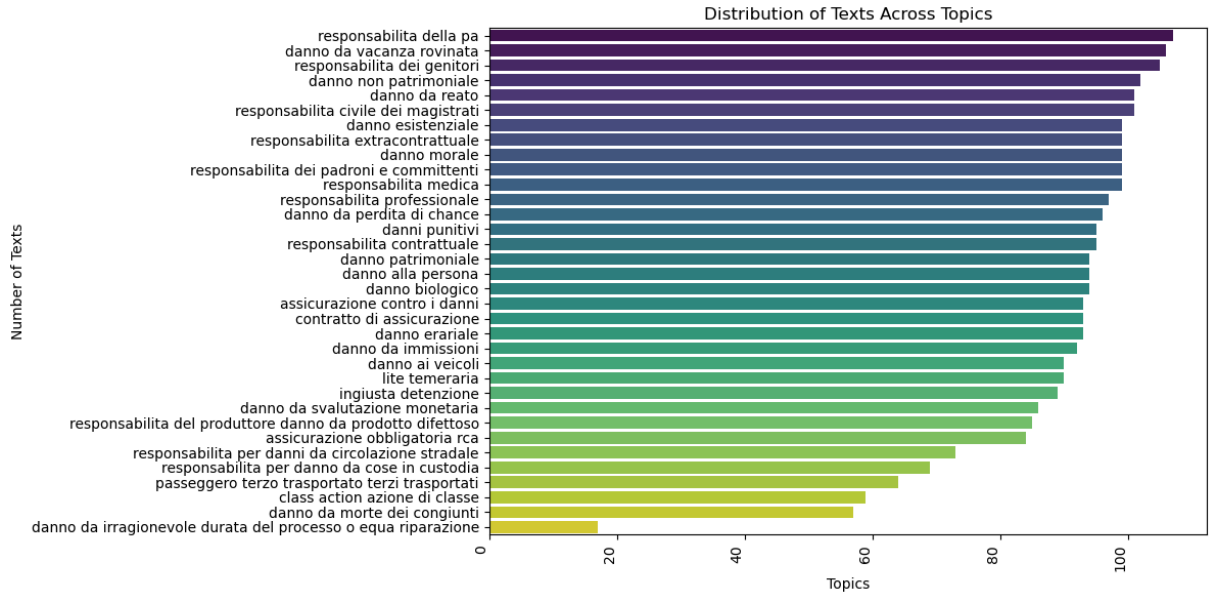
Introduction

The original dataset is in Italian and is structured in two columns (Topic and Text) where each row represents a single sample. It is monotopic, enclosing texts pertaining to the legal field. The text column needs an extensive cleaning procedure to reduce all the noise that is not necessary to our objective and for this purpose the following operations were employed:

- converting all text to lowercase;
- normalize all characters according to ASCII unicode;
- removing URLs and emails;
- removing new lines and unnecessary whitespace;
- removing quotation marks and hyphens of all kinds;
- removing numbers;
- removing exclamation points and anything that is not according to ASCII unicode;
- removing words consisting of more than 20 characters;
- removing words consisting of less than 4 characters;
- removing stopwords;

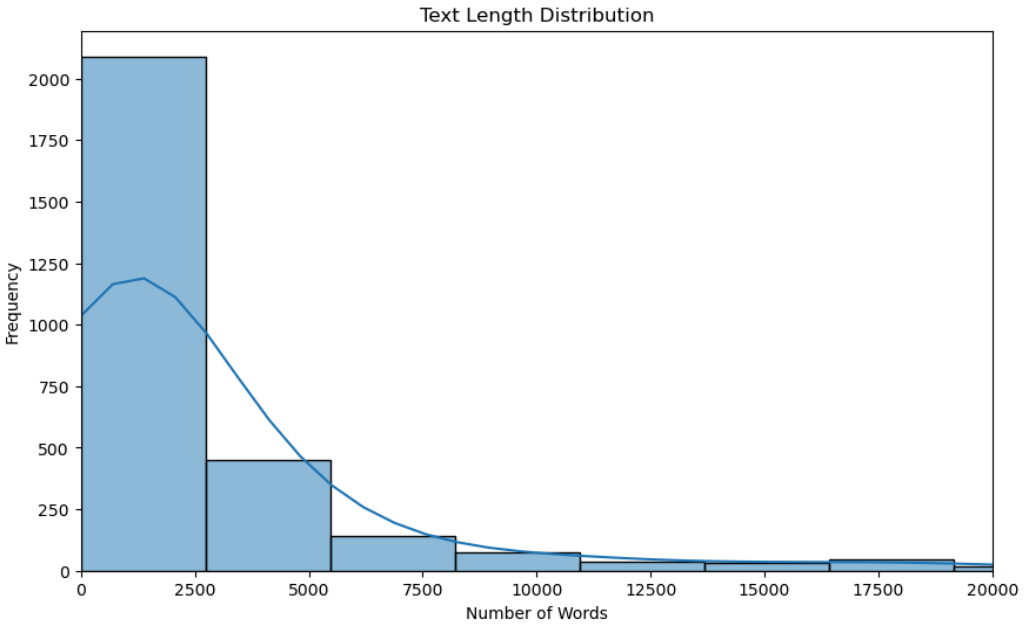
After the cleaning operations we obtained a dataset containing in total 3009 samples for 34 different topics. The dataset is quite balanced with only 2 topics showing an imbalance as can be observed in Figure 1. The most common topic is “Responsabilità della pa” with 107 texts, while the least common topic is “Danno irragionevole durata processo equa ripartizione” with 17 texts. The balance of topics is fundamental since most classification algorithms tend to be biased towards the majority class while training.

FIgure 1 - Topic distribution



Another important aspect is text length, which can be measured as the total number of words in each text. In this regard, the majority of texts have less than 10.000 words within them as can be observed in Figure 2. It is important to note that the texts have quite a good amount of useful words for prediction within making the samples good for classification.

Figure 2 - Text length distribution



The following word cloud was computed to quickly understand the most common words among all texts:

Figure 3 - Word cloud of most frequent words among all texts

The idea is to use Machine Learning and Deep Learning algorithms to perform supervised text classification. This technique is a subset of natural language processing (NLP) and is widely used in various applications due to its ability to organize, filter, and interpret large volumes of text data.

For instance, a law firm could digitize old documents and categorize them automatically. By scanning and processing the documents, the entire pipeline can classify them by topic and store them digitally in an efficient manner. Our objective is to train a model that can accurately perform this classification.

Methodologies

TF-IDF Vectorization

TF-IDF[1] (Term Frequency-Inverse Document Frequency) is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents (corpus). It helps to highlight important words in each document while downplaying the more common words that appear frequently across all documents. The TF-IDF score is a combination of two metrics: term frequency (TF) and inverse document frequency (IDF).

$$TF(t,d) = \frac{Number\ of\ times\ term\ t\ appears\ in\ document\ d}{Total\ number\ of\ terms\ in\ document\ d}$$

$$IDF(t) = \frac{Total\ number\ of\ documents}{Number\ of\ documents\ containing\ term\ t}$$

We chose this approach as it helps to emphasize important words in each document while also reducing dimensionality. The numerical representation of text through TF-IDF can improve the performance of text classification algorithms. It is faster and easier to implement than the alternative of using word embeddings that require more computational power and time for training.

TruncatedSVD[2] (Truncated Singular Value Decomposition)

The TruncatedSVD is a similar algorithm to Principal Component Analysis (PCA) for dimensionality reduction. It is a matrix factorization technique used to reduce dimensionality of large and sparse datasets.
It is based on the Singular Value Decomposition where given a matrix A we obtain three different matrices: U, Σ, V^T

$$A = U \Sigma V^T$$

- U: An orthogonal matrix whose columns are the left singular vectors.
- Σ: A diagonal matrix whose entries are the singular values.
- V^T: An orthogonal matrix whose rows are the right singular vectors.

The TruncatedSVD is obtained as follows:

$$A \approx U_k \Sigma_k V_k^T$$

Where U_k, Σ_k and V_k^T are truncated versions of the original U, Σ, V^T with only the top k components.
The main advantage over other techniques like the PCA is that TruncatedSVD is able to work with sparse matrices like those produced by the TF-IDF vectorization.

Our intention is to use TruncatedSVD for further dimensionality reduction. Most Machine Learning algorithms benefit from lower dimensionality since it allows the model to focus on the most important features instead of the noise present in the data.

[1] TF-IDF: https://en.wikipedia.org/wiki/Tf%E2%80%93idf
[2] TruncatedSVD: http://mlwiki.org/index.php/Singular_Value_Decomposition#Truncated_SVD

t-SNE (t-distributed Stochastic Neighbor Embedding)

t-SNE[3] (t-distributed Stochastic Neighbor Embedding) is a machine learning algorithm for dimensionality reduction, particularly useful for visualizing high-dimensional data. It works by converting similarities between data points to joint probabilities and then minimizing the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data.

Unlike linear dimensionality reduction methods like PCA, t-SNE is capable of preserving local structure, often resulting in more meaningful visualizations of complex datasets. The algorithm proceeds in two main steps:

First, it constructs a probability distribution over pairs of high-dimensional objects so that similar objects have a high probability of being picked, while dissimilar points have an extremely small probability of being picked.

Then, it defines a similar probability distribution over the points in the low-dimensional map, and it minimizes the Kullback-Leibler divergence between the two distributions with respect to the locations of the points on the map.

The t-distribution is used in the low-dimensional space to alleviate both crowding problems and optimization problems of the original SNE. This allows t-SNE to be particularly effective at creating a single map that reveals structure at many different scales, which is particularly important for high-dimensional data that lie on several different, but related, low-dimensional manifolds. We decided to implement this algorithm over others because of its capability to project high-dimensional data into a two-dimensional space, allowing for a clear visualization and evaluation of the text distribution.

Lemmatization

Lemmatization[4] is common practice in Natural Language Processing and Machine Learning. The goal of this algorithm is to reduce the word to its root, in our case to the lemma. One of the main advantages of using lemmatization instead of stemming is that the lemmatizer function is able to better understand the context around the word while the stemming only truncates suffixes. The algorithm works in two steps:

- First, lemmatization begins by identifying the part of speech (POS) that each word represents in a text. Common POS tags include noun, verb, adjective, and adverb. For example, "running" can be either a verb ("to run") or a noun (the act of running).
- Then using a language model or dictionary, the lemmatizer maps each word to its base form. This mapping considers the word's context and POS. For instance, "running" as a verb maps to "run," while "better" as an adjective maps to "good."

[3] t-SNE: https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding
[4] Lemmatization: https://ayselaydin.medium.com/2-stemming-lemmatization-in-nlp-text-preprocessing-techniques-adfe4d84ceee

We adopted this technique because it improves the text by reducing complexity, ensuring that different forms of a word are treated as the same word. Instead of alternatives this is the most time consuming procedure, since it requires quite some time to lemmatize long texts.

Data Enrichment

Enriching the data is an important step that can improve the initial data and lead to better model performance. Synonym replacement was used to add variation in the text data and avoid correlation between topics. The primary drawback of synonym replacement compared to other techniques is the potential loss of the original semantic meaning of words. However, synonymous words are usually closely related to the original topic. This variation can enhance the model's ability to generalize effectively.

Tokenization

Tokenization is an important step in natural language processing. It involves breaking down a piece of text into smaller units called "tokens," which can be words, phrases, or even characters, depending on the specific requirements of the task and the tokenizer algorithm used. The primary advantage of tokenization is that it transforms raw text into a more structured and analyzable form, enhancing the performance of subsequent modeling.

LSTM (Long Short-Term Memory)

LSTM refers to "Long Short-Term Memory," a type of recurrent neural network (RNN) that is versatile and can be applied to various sequence-based tasks, including but not limited to classification, regression, and sequence generation.

Recurrent neural networks are a special subcategory of neural networks that contain loops which allow information to be stored, making them very useful when working with sequences. Unfortunately, RNNs don't work well when there are significant gaps between relevant information. As with other neural networks that rely on backpropagation or gradient-based methods for optimization, they often suffer from the vanishing gradient problem when the depth of the network increases significantly[5]: in this type of optimization, the network weights are updated after every iteration proportionally to the gradient value. When the sequence is very long, the gradient value (which represents the partial derivative of the error function with respect to the current weights) can become increasingly small, causing the weights to not be updated efficiently or even completely stopping the training. The opposite problem, the exploding gradient problem, can also occur in RNNs. In this case, the gradient becomes increasingly large, potentially causing numerical instability in the process of updating the network weights.

[5] Basodi, S., Ji, C., Zhang, H., & Pan, Y. (2020). Gradient Amplification: An Efficient Way to Train Deep Neural Networks. *Big Data Mining and Analytics, 3*(3), 196-207. Retrieved from https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9142152

LSTM models[6] can handle both problems better than standard recurrent neural networks thanks to their gate structures. The nodes in the model are called memory cells, each with an internal state that is updated thanks to three gates:

1. A forget gate that allows the cell to determine which information to discard from the cell state, using a sigmoid layer.
2. An input gate that decides which new information to add to the cell state, also using a sigmoid layer.
3. An update step that creates a vector of new candidate values that could be added to the state, typically using a tanh layer.
4. An output gate that decides what parts of the cell state to output and further processes and filters the information before transmitting it to connected cells.

To sum up, the input gate and update step work together to determine what new information to retain. The sigmoid layer from the input gate is multiplied by the candidate values provided by the tanh layer. This way, the new information is weighted by how much we want to "remember."

Thanks to their structure, the model is able to efficiently learn long-term dependencies, making it particularly useful for tasks involving sequences with long-range interactions.

LSTMs are highly versatile, capable of handling a wide range of applications from natural language processing to time series forecasting, making them a valuable tool in many domains. However, LSTMs are computationally more complex than simpler models, requiring more resources and time to train, which can be a significant drawback in scenarios with limited computational capacity or tight time constraints.

An alternative is to use BERT[7] (Bidirectional Encoder Representations from Transformers) which is a recent architecture published by researchers at Google AI Language. The model architecture presented makes use of a Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text. It can be used on a wide range of tasks which in our specific case is supervised text classification. The main advantage of using BERT is its ability to understand context inside the text. This gives this model a powerful capability in learning the structure inside the data and therefore predicting with a contextual meaning of the choice. On the other hand the main disadvantages are the extensive computational resources and time required for the training. In order to effectively use this approach high performance hardware is required and this is the main reason we opted not to implement it.

---

[6] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735-1780. Retrieved from https://www.bioinf.jku.at/publications/older/2604.pdf

[7] BERT: https://arxiv.org/pdf/1810.04805

Experiments

TF-IDF Vectorization and Truncated SVD

Both TF-IDF[8] and Truncated SVD[9] have been implemented in python through the scikit-learn package. The goal of the TF-IDF algorithm is to reduce dimensionality in our data. We chose a maximum number of features equal to 10000, meaning that only the first ten thousand most common words are preserved for further analysis. Then the truncated SVD has been implemented. It further reduces the dimensionality in our case to only 300 TF-IDF vector dimensions, balancing between computational efficiency and information retention.

LSTM Modeling

The basic LSTM model was implemented from Keras[10] through the Tensorflow package in Python. The data was split in training (80%) and test (20%) with stratified sampling based on the topics before implementing the model.

The first Layer of the LSTM is added to the model with 128 units (neurons). This choice is a common standard practice and offers a balanced approach between complexity and computational efficiency. The dropout is set to 0.2 in order to prevent overfitting, meaning that 20% of the input units will be randomly set to 0 during training. In the same way the recurrent dropout is set to 0.2.

The Dense layer is added with 34 units, which in our case is the number of singular topics the model needs to make predictions on. We chose the softmax activation function for the output layer since it converts the raw output into probabilities that sum to 1, making it a suitable function for predicting categorical labels.

The model is compiled with an Adam optimizer, which is a popular choice due to its adaptive learning rate and efficient computation. It combines both the advantages of the AdaGrad and the RMSProp optimizers. As a loss function the categorical cross entropy was used as it is particularly suitable for multi-class classification tasks by quantifying the dissimilarity between the predicted probabilities for the labels and the true categorical labels.
As validation, accuracy is a straightforward metric for this kind of classification task, reporting the proportion of correctly predicted topics.

The model is fitted over 10 epochs, meaning that the model will go through the training dataset for 10 times. Choosing a bigger number of epochs could lead to overfitting, and we concluded that this is a good number of iterations in order to achieve an optimal result, as the models do not particularly overfit the data, but do not seem to underfit it either. The batch size for training is set to 16 which is a common choice in these types of models. The weights

[8] TF-IDF Vectorization: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
[9] TruncatedSVD: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html
[10] Keras: https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM

are computed based on the proportion of texts per topic in order to rebalance eventual class imbalances in an effective way.

t-SNE visualization

The t-SNE plot with only 2 components of the TF-IDF features presents a two-dimensional representation of text distribution across various legal topics. To perform this task we used the TSNE[11] function from the scikit-learn library in Python. To reproduce the same results in every iteration we implemented the random state equal to 42.

Lemmatization

The lemmatization algorithm was implemented with the Spacy[12] library and the pretrained "*it_core_news_sm*" lemmatizer for the Italian language that has to be downloaded separately. The lemmatization process has to be implemented after the basic cleaning of texts but absolutely before removing stopwords. The stopwords are necessary for the model to understand the context and better perform the lemmatization process.

Synonym replacement

A different approach for removing correlation between the texts belonging to different topics is to try synonym replacement as a data enrichment process which has been implemented through the Italian "wordnet" dictionary present inside the NLTK[13] library. It has been applied only to the training dataset in order to avoid data leakage. As a first step we group all the texts by topic and then identify the 100 most common words. Then through the WordNet italian dictionary the most common words are replaced with a random synonym ensuring that it is different from the original word. The new texts, per each topic, have an enhanced diversity in the words that comprise them, allowing better generalization by the model.

Tokenization

Tokenization has been performed using the "*Punkt Sentence Tokenizer*"[14] from the NLTK library. Before tokenization, the texts need to be cleaned and preprocessed. This included removing punctuation, numbers, special characters, and stopwords, lemmatization and text enrichment with synonym replacement. It is applied both on train and test data ensuring that both are tokenized in the same manner. The main advantage of this process is that tokenized text provides a structured format that is easier for machine learning algorithms to process.

[11] TSNE: https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html
[12] Spacy: https://spacy.io/models/it
[13] NLTK: https://www.nltk.org/howto/wordnet.html
[14] Punkt tokenizer: https://www.nltk.org/api/nltk.tokenize.punkt.html

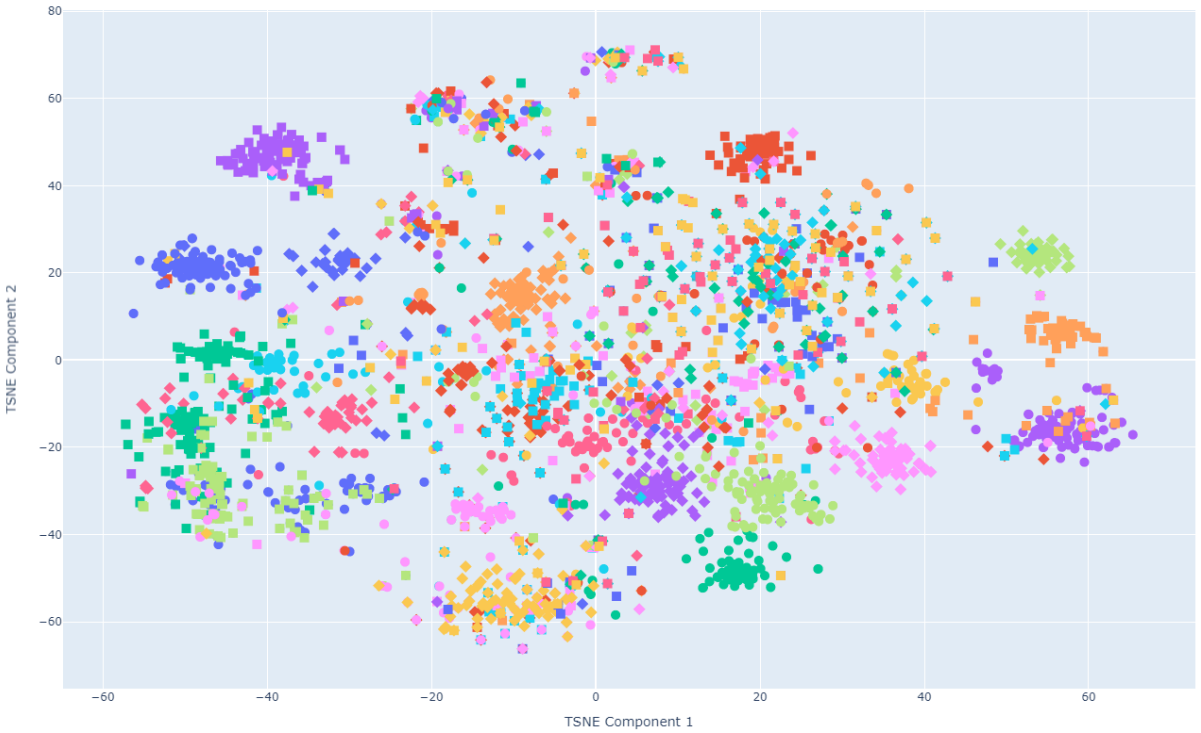
Results

t-SNE results

From the t-SNE plot that was computed, as displayed in Figure 4, we can observe several distinct clusters, indicating that texts within specific topics share similarities, as expected.

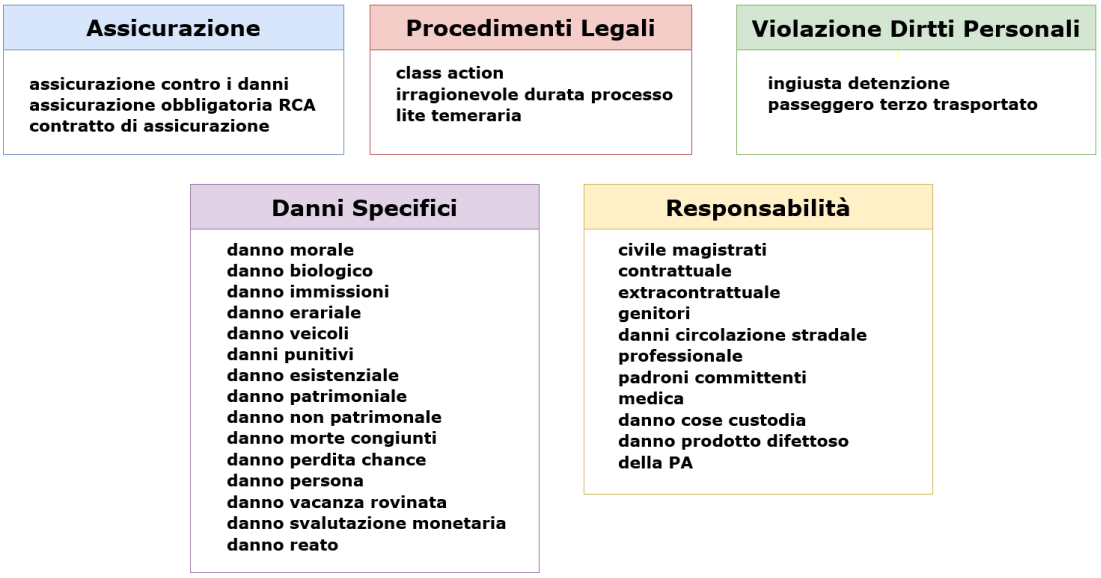
However, there's also significant overlap between clusters, suggesting that some topics have similar linguistic features. This is particularly evident in the central region of the plot, where multiple clusters mix. The overlapping of linguistic features across texts belonging to different topics may induce some bias in the model affecting its classification power.

Figure 4 - Texts t-SNE representation with all 34 topics (legend not present for aesthetic reasons)



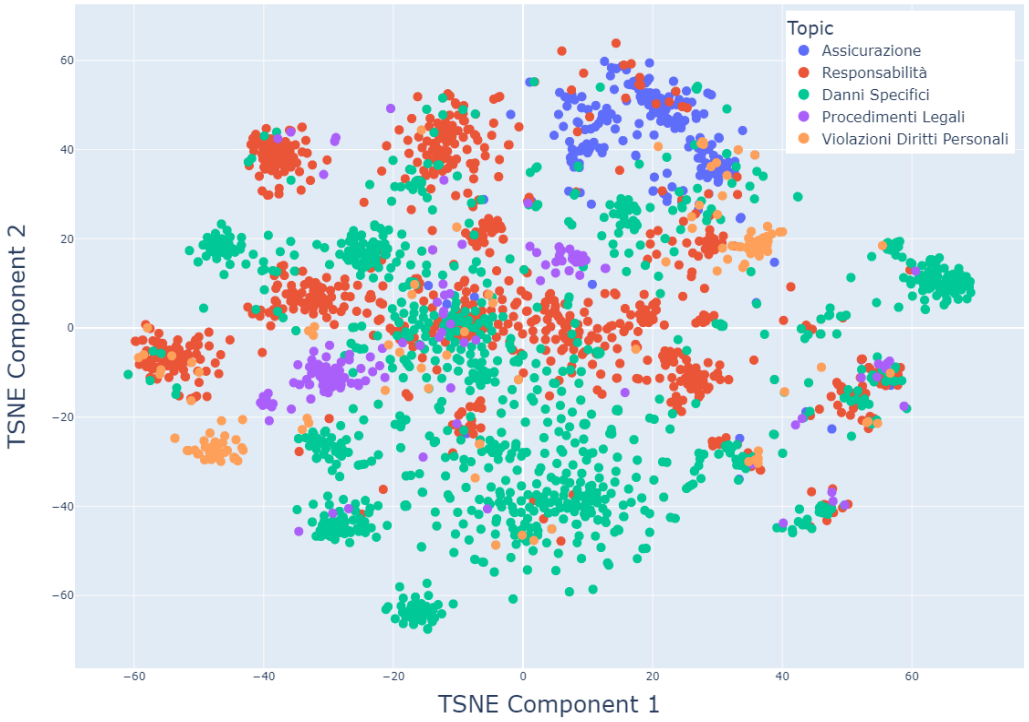
Given the significant overlap between certain topics as observed above, one solution would be to consult a domain expert in legal matters to manually cluster highly correlated topics. However, in the absence of such an expert, we decided to implement grouping based on macro topics. The goal of this strategy is to reduce the correlation between topics by grouping them into broader categories. We opted for five broader macro topics as seen in Figure 5, that might simplify the classification task. This approach naturally leads to loss of information, so the decision to implement it needs to be carefully examined in a real-world setting.

Figure 5 - Macro topics



We used the same type of representation as before, with t-SNE, in order to see how the new macro topics are distributed in the bidimensional space. From Figure 6 we can observe that some overlapping is still present but clusters are way more separated than before, indicating that this could help the model to achieve a better accuracy in the classification task.

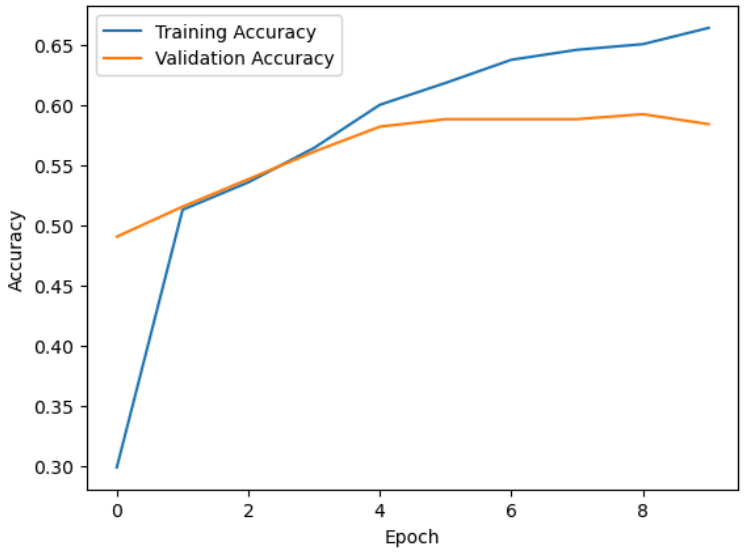
Figure 6 - Texts t-SNE representation when grouped in macro topics



LSTM benchmark

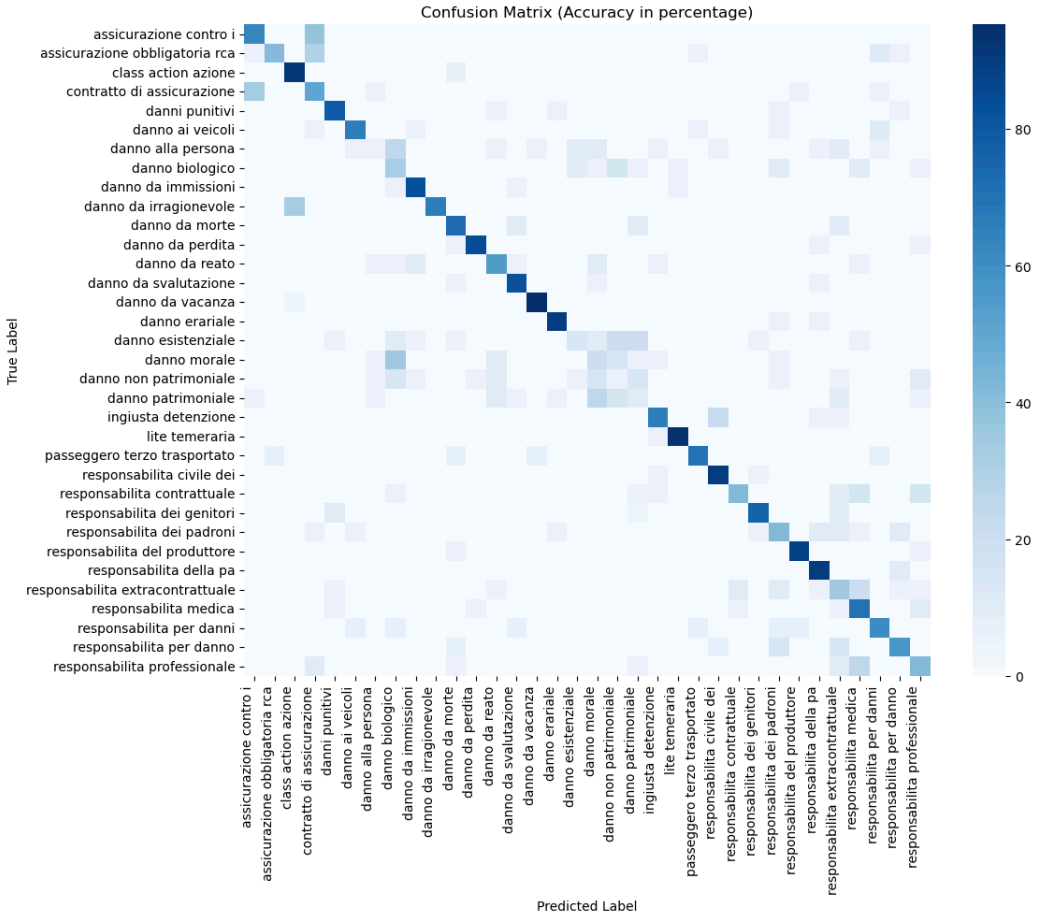
As a benchmark, we used the model without data enrichment and topic grouping. This model achieved a training accuracy of 68,22% after ten iterations, with a validation accuracy of 58,42%. This indicates that the model begins to overfit the data after a certain point.

Figure 7 - LSTM benchmark model accuracy



To better visualize the performance of the model a confusion matrix was computed. The perfect model would predict topics at 100% accuracy meaning that the diagonal of the confusion matrix should be the only highlighted part of the plot. As we can observe in Figure 8 the model predicts some topics incorrectly. This can be attributed to plausible correlation between some specific topics. One example is "Responsabilità medica" and "Responsabilità professionale" where the model is not able to clearly distinguish between the two.

Figure 8 - Confusion Matrix of the benchmark model



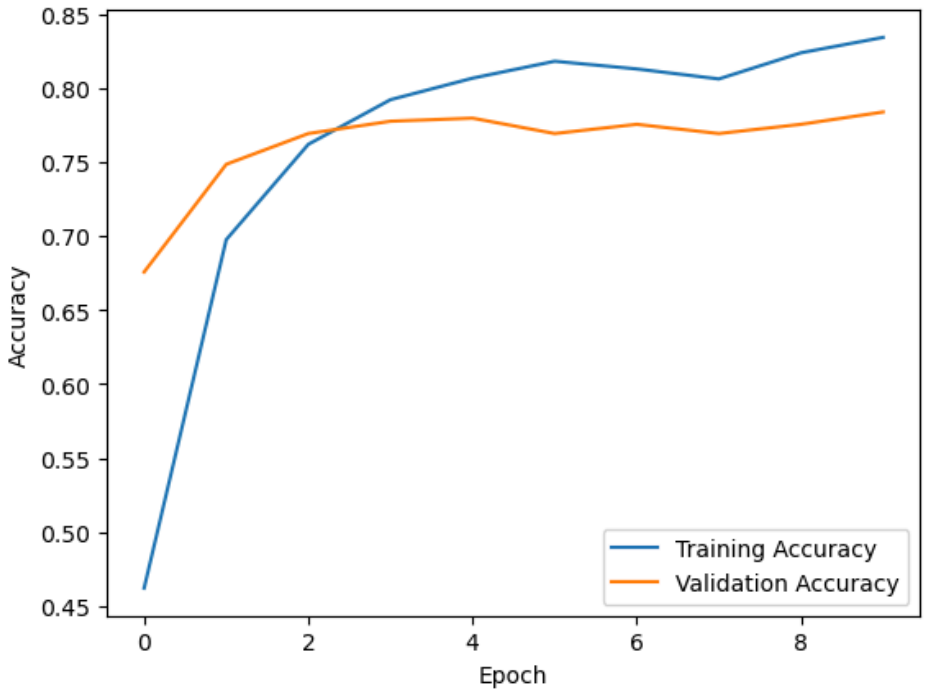
As easily seeable the model struggles to predict certain classes, the five most problematic ones are the following:

1. Actual class 'Assicurazione contro i danni' misclassified as 'Responsabilità contrattuale': 7 times (36.84%)
2. Actual class 'Danno da reato' misclassified as 'Responsabilità civile dei magistrati': 7 times (35.00%)
3. Actual class 'Responsabilità contrattuale' misclassified as 'Assicurazione contro i danni': 6 times (33.33%)
4. Actual class 'Responsabilità dei genitori' misclassified as 'Assicurazione contro i danni': 1 times (33.33%)
5. Actual class 'Assicurazione contro i danni' misclassified as 'Responsabilità contrattuale': 5 times (29.41%)

Where the percentages are computed by dividing the number of instances of each prediction by the total number of instances for the corresponding actual class and then multiplying the result by 100.
Considering the high percentage of misclassified instances between similar classes, summarizing the original topics in five macro topics can lead to better performances.

We implemented the grouping of the 34 initial topics into five macro topics and performed the training of the LSTM model again to see if there was an improvement. As expected the model is able to achieve better accuracy as shown in Figure 9:

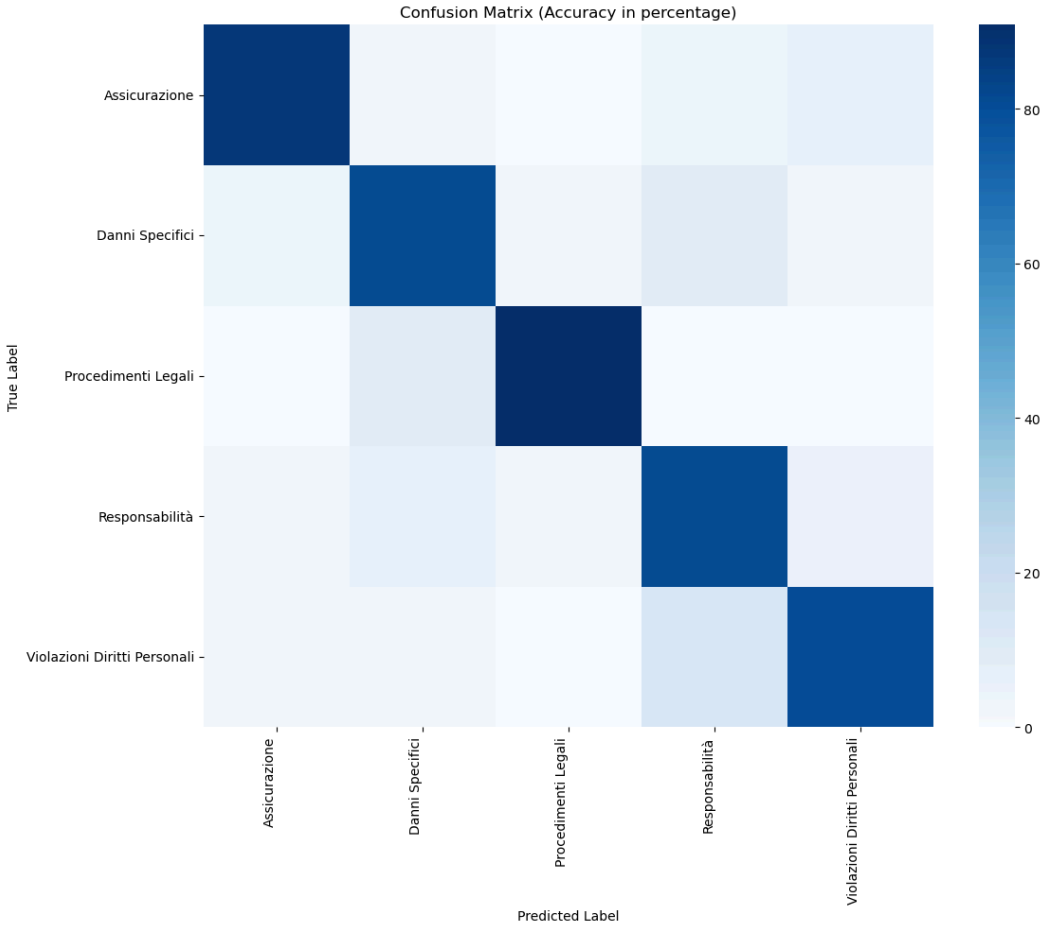
Figure 9 - Model Training and Validation with Macro Topics



The LSTM model was now able to reach 83,69% training accuracy and 78,38% validation accuracy after 10 training iterations. As seen in the previous graph, training accuracy keeps

increasing faster than validation accuracy. This could be due to the model overfitting on the training data.
The confusion matrix was also computed to better visualize the performance of the model.

Figure 10 - Confusion matrix of the model with Macro Topics



Now the three highest instances of misclassification are the following:

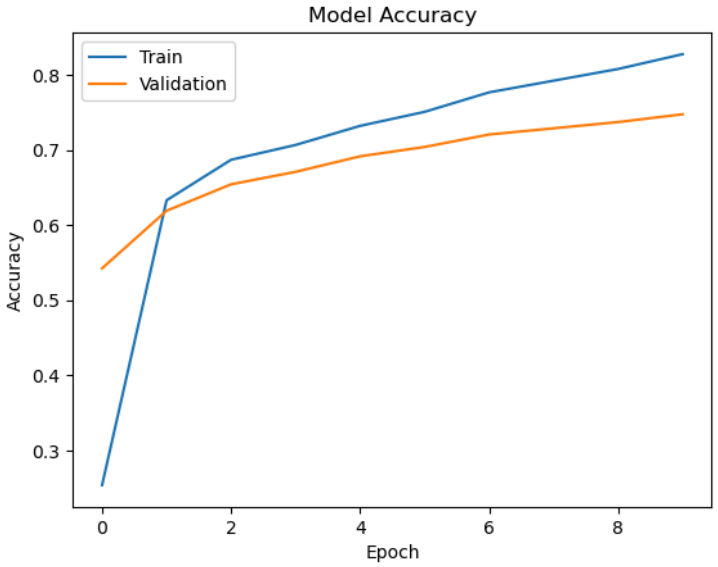
1. Actual class 'Responsabilità' misclassified as 'Responsabilità': 4 times (12.90%)
2. Actual class 'Assicurazione' misclassified as 'Responsabilità': 27 times (9.64%)
3. Actual class 'Assicurazione' misclassified as 'Assicurazione': 3 times (9.09%)

Grouping the topics seems to lead to a better performance with lower error rates.

LSTM with data enrichment

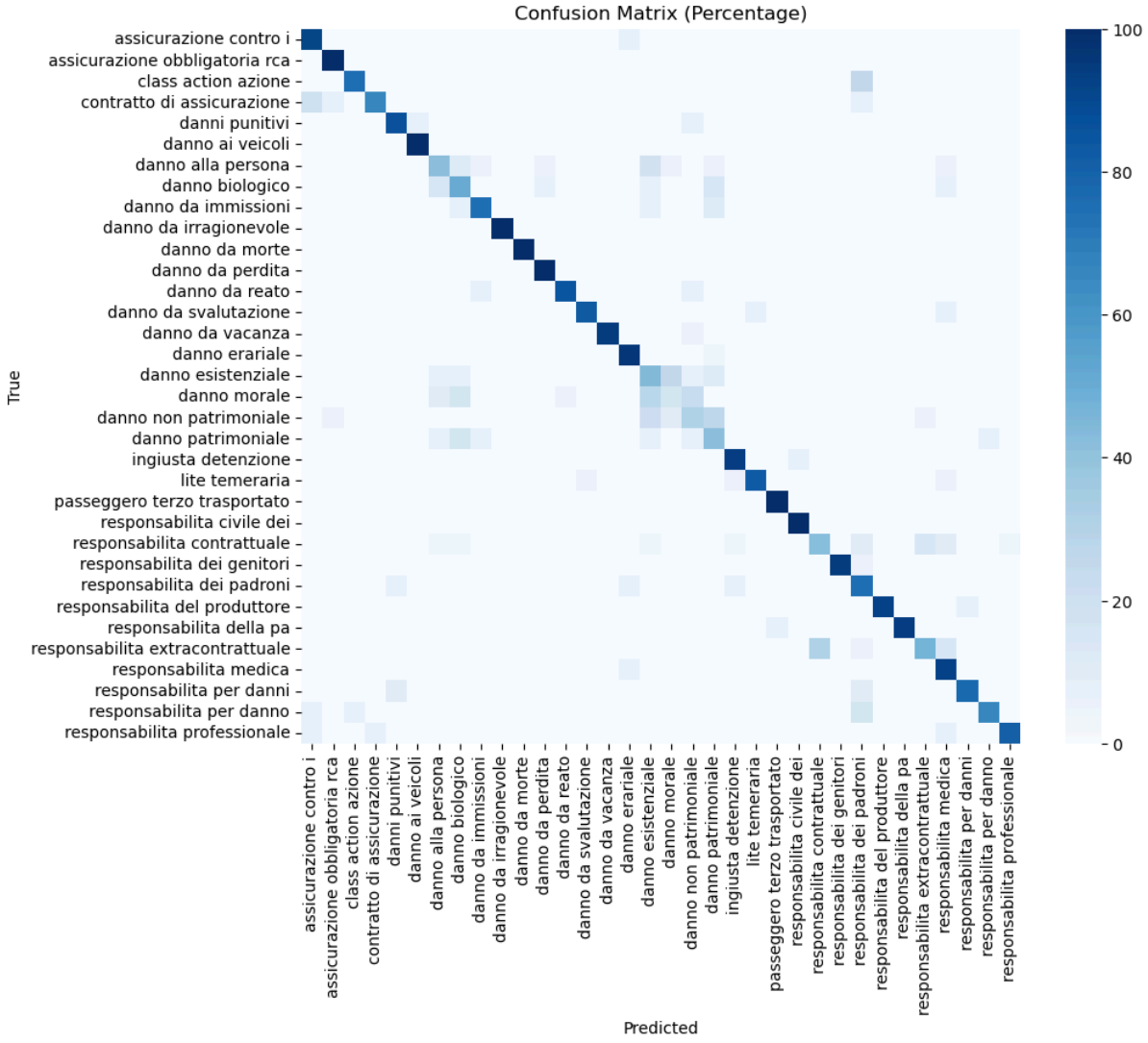
Our second solution to avoid collinearity between topics is to use data enrichment. Lemmatization and tokenization were implemented as well to further improve the data quality. Over 10 epochs the model is able to achieve a training accuracy of 84,03% and a validation accuracy of 74,74% (Figure 11). The enrichment process has a positive effect on the model performance improving classification accuracy of almost 20%. In this case the dropout was increased to 0.3 as the model tends to overfit the data.

Figure 11 - LSTM with enriched data and initial 34 topics



In this case the confusion matrix changes as displayed in the next figure:

Figure 12 - Confusion Matrix after enrichment



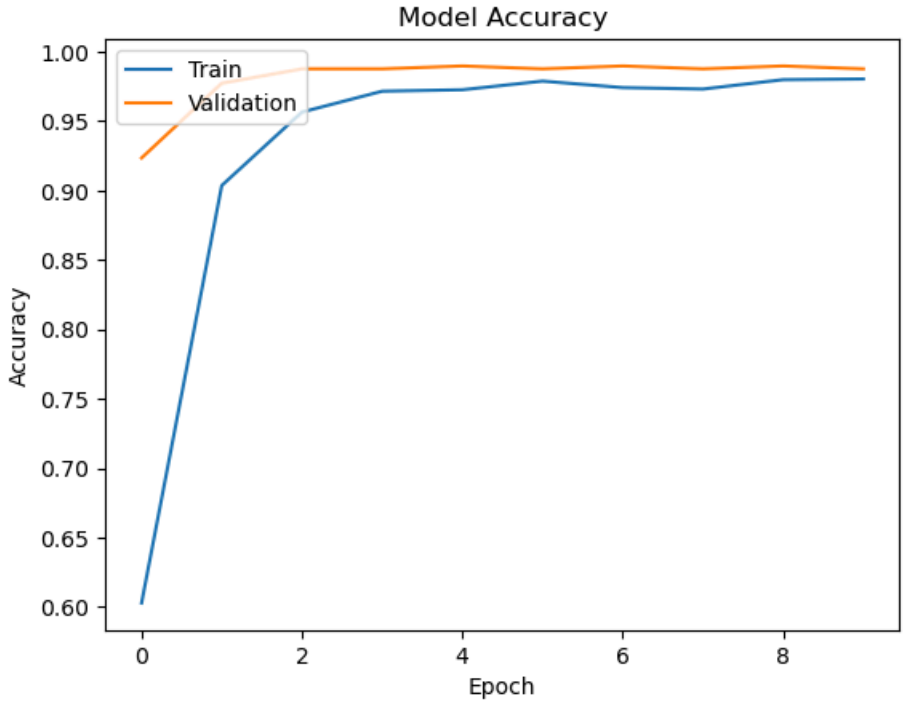
The most problematic classes to predict for the model are the following:

1. Actual class 'Danni punitivi' misclassified as 'Responsabilità civile dei magistrati': 6 times (31.58%)
2. Actual class 'Danno da reato' misclassified as 'Responsabilità dei padroni e committenti': 5 times (27.78%)
3. Actual class 'Assicurazione contro i danni' misclassified as ‘Danno da immissioni': 5 times (26.32%)
4. Actual class 'Assicurazione contro i danni' misclassified as 'Danno da svalutazione monetaria': 1 times (25.00%)
5. Actual class 'Responsabilità dei padroni e committenti' misclassified as 'Danno da reato': 4 times (25.00%)

It is very clear that there has been a significant improvement in the percentage of wrong predictions in comparison to the benchmark model.

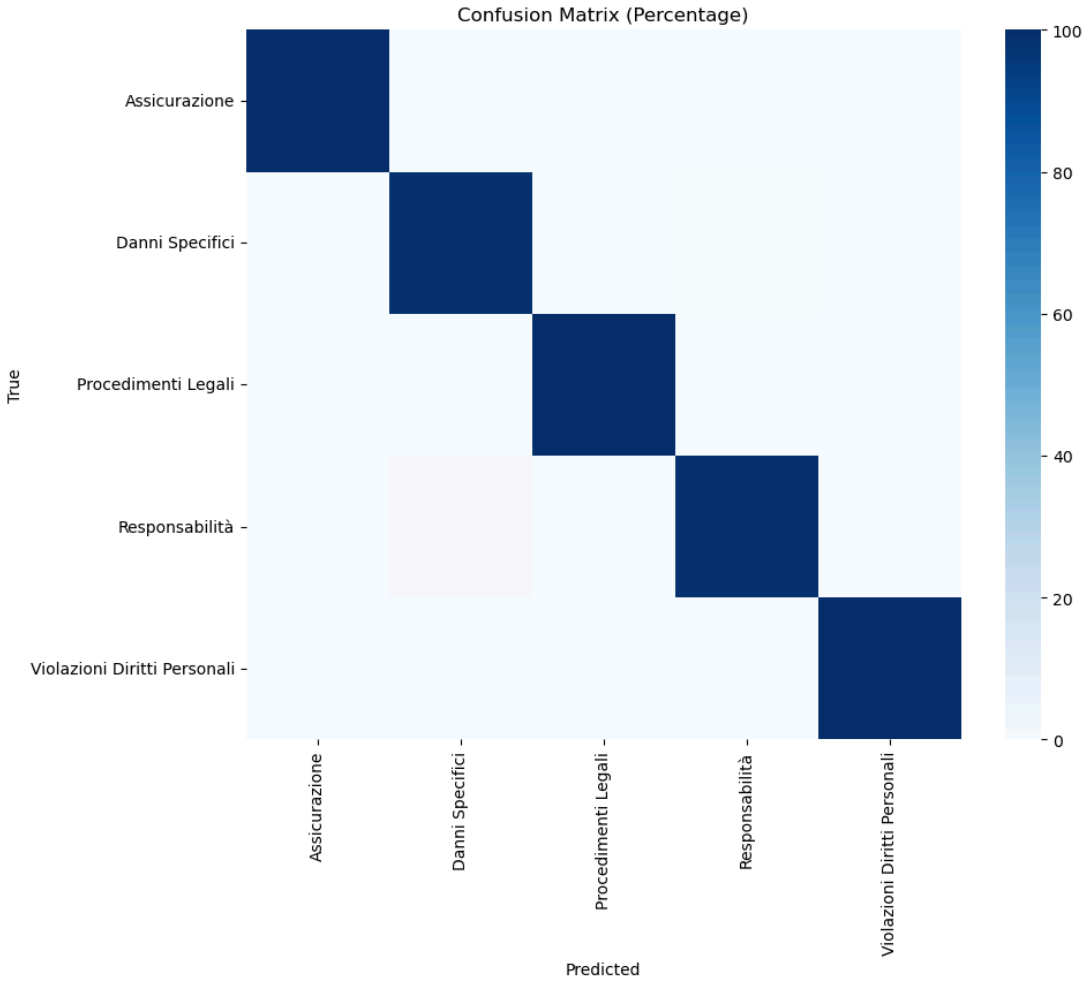
When combining the two techniques we were able to achieve an accuracy of 98,76% in validation and 97,26% in training, in this case the dropout rate was decreased to 0.1 to avoid the model underfitting the data (Figure 13). This is the best result we obtained so far making it a valid solution to the classification problem.

Figure 13 - LSTM with enriched data and macro topics



Lastly, the confusion matrix was also computed, as shown in the next figure:

Figure 14 - Confusion matrix of the model with Macro Topics after enrichment



The confusion matrix almost resembles the confusion matrix that would be obtained with a perfect performance.
In this case the class with the highest error rate are:

1. Actual class 'Responsabilità' misclassified as 'Assicurazione': 2 times (1.25%)
2. Actual class 'Assicurazione' misclassified as 'Responsabilità': 2 times (0.89%)
3. Actual class 'Responsabilità' misclassified as 'Assicurazione': 1 times (0.62%)

This confirms that the model rarely makes wrong predictions.
We opted to show both implementations in order to let a possible client choose the best solution for their specific case considering all the pros and cons in our different approaches.

Conclusions

The benchmark LSTM model without any data quality improvement and all the initial topics achieves an accuracy of 58% in the classification task which is fairly good but there is a big margin of improvement. Grouping topics into broader categories removes the correlation between texts belonging to different topics, allowing the model to improve the accuracy to almost 80%. Better results are obtained when the initial texts are processed with

lemmatization, synonym replacement and tokenization. This approach grants a performance of 76% accuracy with all the topics. When combined with grouping in macro topics the model is able to achieve the best result with more than 97% accuracy in validation. Further improvements can be implemented through fine tuning the macro topics grouping with the help of a legal domain expert. Moreover with better hardware performance more sophisticated models like BERT could be implemented and tested.