ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

Corso di Laurea Magistrale in Ingegneria Informatica

TESI DI LAUREA

in
Intelligent Systems

Automatic Juridical Decision-making: a Neural Network approach applied to the European Court of Human Rights

CANDIDATO

Enrico Damini

RELATORE:

Chiar.mo Prof.ssa Michela Milano

CORRELATORE
Chiar.mo Prof. Barry O'Sullivan

TUTOR
Ph.D Andrea Visentin

Anno Accademico 2017/2018

Sessione III

Abstract

In recent years, the potential to speed up legal processes via Machine Learning techniques has increased. Consequently, this study serves to investigate different NLP and ML methods. It does this by building a model that predicts whether or not an article of ECtHR has been violated, based on the texts of the published case law. The most effective model, which consists of a Neural Network (for embedding the corpus) and a SVM (as classifier), reaches a total mean accuracy of 72%.

Negli ultimi anni sempre più tecniche di Machine Learning sono state prese in considerazione allo scopo di velocizzare, laddove sia possibile, il sistema legale. Questo studio investiga differenti metodi di Natural Language Processing e di Machine Learning per costruire un modello in grado di predire se un dato articolo dell' ECtHR sia stato violato o meno basandosi sulle procedure legale pubblicate sull'omonimo sito dal 2014. Il modello finale, la cui pipeline consiste in una Neural Network e una Support Vector Machine, raggiunge un accuracy totale di 72%.

Acknowledgments

First of all, I would like to thank the people who allowed me to do the remarkable experience of writing my thesis abroad, my Professor Michela Milano and Professor Barry O'Sullivan .

A great thanks to Andrea Visentin, my second supervisor, for all of his wisdom and his guidance through this amazing experience.

I would like to thank my colleagues, and also friends, at the Insight Centre of Cork: Diego Carraro (a great Inter supporter), Mesut Kaya, Federico Toffano, Yves Sohege. Your discussion, ideas, and feedback have been absolutely invaluable.

A special thank to Roberto Damini (my father) and Emanuela Secci (my mother), who have played an important role in all my career as they are my first, and only, sponsors. I want also to thank my brother Riccardo for the support he shown during the years, especially during his master.

My friend Prof.ssa Laura Gargioni has always been to my side in the worst and in the best moment of my career; thank you for everything and especially for your corrections.

I can not forget about Alessandro Broggio, one of my closest friend. Thank you for the many Whatsapp calls.

As the last, but not the least, I would also like to thank Mary Elizabeth O'Brien (Maidy). Your love, laughter and support have kept me smiling and inspired. You and Ireland are and always will be in my heart.

Table of Contents

Introduction

The interaction between computers and the application of the Law is widely acknowledged. Scientists believe that information technology can make a significant contribution to the work of judges and lawyers. In recent years, the idea of using Artificial Intelligence in the legal domain has become even more tangible given the considerable progress in Natural Language Processing and in Machine Learning techniques.

Continued advancements might one day allow judges to rely completely on Artificial Intelligence, providing robust tools for the legal system which could potentially avoid the involvement of the court. However, we are far from that today: the majority of studies in this domain endeavour to speed up the legal process rather than replace it. Consequently, judges must be involved in the final decision. The software act as a support in judges decisions. Nonetheless, AI tools can be very helpful, for example, by identifying whether or not an article has been violated.

The current study focuses on the automatic analysis of cases of the European Court of Human Rights (ECtHR) and explores the potential of employing language analysis in the legal domain in order to extract meaningful information for Machine Learning. It also discusses the advantages and disadvantages of various techniques adopted by other studies and identifies possible solutions. The addressed methods are then compared to determine the most efficient and effective practice.

A detail analysis was conducted to identify what factors influence the judicial decision the most.

The present study sought to build a robust and reliable model consisting of several modules that can easily be changed. This should facilitate future studies to achieve a more reliable, time efficient and robust model.

Literature review

Over the last years, the interest among researchers in the legal domain classification has fairly grown, as a result, an increasing number of studies have been conducted. The majority of these studies are based on quantitative methods and, in fact, they have been applied to dataset of case law from many different countries, for instance : Germany [1], Spain [2], France [3], the Netherlands [4], Belgium [5] and United States which are leaders since they have a longer tradition in this terms of data collection [6].

International Courts have been taken into account by many researchers, for example, the Court of Justice of the European Union [7] [8] or the International Criminal Court [9] [10].

Other approaches consist in applying apply multi-label classification algorithms [11], employing quantitative network analysis [12] or using topic modelling techniques to automatically extract latent features [13].

Advanced techniques , such as neural networks, are relatively less common in this particular field. The main reason, it is because NNs do not always explain clearly the choices they make for classifying an instance. However, it was claim that NNs can take into account semantic between words rather than just the frequency [14].

An interesting study in the medical sector was proposed by [15]. Their work caught our attention since they aim to predict some common features given a medical record, which is a semi structured text. In this case they use a similar structure to [14] for the embedding the text in a continuous domain and on top of that they train a Convolutional Neural Network which is able to find common pattern and predict the desired outcome.

[16] in 2016 were able to extract features from ECtHR by applying the common method n-grams for the word embedding. Medvedeva, Vols and Wieling have done a similar study [17] with a larger dataset and a more rigorous approach have been applied by [24]. However, none of these recent works (which can be considered the state of the art) consist in using Neural Networks.

The work we are going to present we build upon the results obtained in these papers and implement different advanced techniques in order to compare them.

Description of the problem

Nowadays, we have already entered the big data era, thus an increasingly amount of case law have been published online. It is definitely senseless and naive, rather than impossible, trying to physically analyse this data. In these terms, AI is helping legal researchers to read and systematise all the international and national court decisions.

The problem we are going to address belongs to the Legal Judgment Prediction class of problems. We define it as a binary classification problem. Our goal would be creating a system which is able to predict whether or not a specific article has been violated by analysing each cases available of the ECtHR. The main difficulty is to understand how the models we are going to build can learn features from a text corpus and also how it uses them for the classification task.
Moreover, we want that our model is as transparent as possible in terms of explanations, in order to overcome the problem of not being collaborative, which the NNs normally suffer.

Background

We are going to introduce the background of this project by explaining, first, how the all process works and, second, all the parts which a case law document consists of.

The European Court of Human Rights is the highest international court of the UE which was established in 1959 by the European Convention on Human Rights. This administrative organ has to guarantee the respect of the list of rights which has been laid down by the Convention itself.

The Court is based in Strasbourg, France and since the foundation, it has expanded significantly, up to count forty-seven states in total, with a population of nearly eight-hundred million.

The Court, which incorporate 47 judges (equal to the numbers of member States), hears applications concerning about civil and political rights, that have been ratified by the Convention and its protocols. The judges are completely independent of their country of origin and do not represent either applicants or States.

The Court investigates complaints, known as applications, presented by individuals or even by States, in order to discover and examine the facts (an incident, allegation, etc.) so as to establish the truth. The Court delivers a judgment finding a violation and, in the end, it concludes whether a member State has breached one or more of these rights and guarantees.

This is an example of the main rights which are defended:

- the right to life;
- the right to a fair hearing in civil and criminal matters;
- the right to respect for private and family life;
- freedom of expression;
- freedom of thought, conscience and religion;
- the right to an effective remedy;
- the right to the peaceful enjoyment of possessions;
- the right to vote and to stand for election.

On the other hand, this is what is strictly prohibited:

- torture and inhuman or degrading treatment or punishment;
- arbitrary and unlawful detention;
- discrimination in the enjoyment of the rights and freedoms set out in the Convention;
- expulsion or denial of entry by a State in respect of its own nationals;
- the death penalty;

- the collective expulsion of aliens.

One of the main characteristics of case law procedures is that the court decisions are manually collected, commented, summarised and filed in the overall legal system. This is the key factor of the NLP analysis in this particular domain, since the data is, more or less, semi-structured.

Data

Case law structure

All the case law have the same structure, except of certain particular cases which has not been take into account. They consist of three main parts, which contain several other sub-parts depending on the considered case law. The sections are the follow: “The procedure”, “The facts” and “The law”. Before these sections a judgment should contain:

- the names of the President and the other judges constituting the Chamber or the Committee concerned, and the name of the Registrar or the Deputy Registrar;
- the dates on which it was adopted and delivered;
- a description of the parties;
- the names of the Agents, advocates or advisers of the parties;

All the sections, as the documents themselves, are vary in terms of number of words; there is not any minimum or maximum length.

The procedure

The procedure text’s chunk represent a summary of what is happened previously the Court handled the case, from when it has been lodged the reclaim to date written on the document.

The facts

This part, that has been drawn up by the Court, consists of all material which is not treated as a question of law. They are not just the facts that happened in the past, but they also refer to the presumed violation of one or more articles.
This section is split in two sub-parts which are the follows:

- **The circumstances of the case** which contains information about the factual background of the case and all the domestic legal history of what happened before the application is handled by the Court. In this sub-section is told about all the possible actions and events that have caused a violation of one of the articles listed in the ECHR. It is needed to stress the fact that, because this section is written by the Court, the events are weighted by judges and this means that some actions compare in the text with explicit judgments, hence some facts could be more relevant than others. Therefore, there could be the possibility of introducing biases. However, this is not likely to happen since both parties normally agree on the events mentioned in this section.
- **Relevant law** consists of all the legal agreements which refers to laws that mostly do not compare in the ECtHR. They must be "relevant" since they could determinate the decision of the Court.

The Law

In this last section the Court declares the decision that has been taken by applying the rules of law. In the sub-sections are explained the reasons and are examined in details each problem raised by the application, so each violation of an article. Normally, there is more sub-sections depending on the number of the raised problems, however, where it is possible, they has been examined all together.

- **Alleged violation of article x**, one for each violated article. It is divided in two other sub-parts which are **"The Parties' Submissions"**, a summary of the main arguments, and **"Merits"**, the main reasons that explain the choice taken by the Court.

- **Operative provisions**, it is the actual outcome of the case, so whether if an article has been violated or not. It also provides all the information about the legal costs and, if any, the punitive measures.

Dataset description

As a starting point, we use the data which is available on the HUDOC website. We use the dataset automatically crawled by [17]. We decided it, not only for practical reasons but also because we want to replicate as close as possible the experiments they conducted (we will talk about that in the next chapters).
They consider altogether the admissible judgments of both the Chamber and the Grand Chamber. In the dataset, it has been considered just the English documents and the download refers to September 11, 2017. It consists of 13 folders which represent each Article which was taken into account for the research, from Article 2 to Article 14 and Article 18. Articles are separated in two different folders (violation or non-violation), in order to discriminate them for future labelling. This information was automatically obtained through the HUDOC website. Some samples could appear in more than one folder, due to the fact that they could consider multiple violations at once.
The full dataset is available at this link[1].

As the dataset was not balanced by nature, it was necessary to apply some changes. First, we counted the number of cases of each class, second, we considered as total number of sample (referring to each article) the double of the smallest class. This is done by randomly excluding the exceeding number of samples in the biggest class. In this way, it is possible to achieve a balanced dataset. As most of the articles have more samples belonging to the violations class, it has been created a folder for future evaluations of the classifier. However, this is not the proper way for testing a classifier, in fact it is not a best practice evaluate a classifier just on one class. We will discuss about this in the next chapters.

[1]https://www.dropbox.com/s/lxpvvqdwby30157/crystal ball data.tar.gz

Furthermore, it has been excluded the 20% of the whole dataset for future experiments.

Since in this study we need as many as possible samples, we merged the dataset, in different ways, in order to be able to conduct some experiments. For the code we used for merging the dataset see the project on github[2].

The table below shows the number of total samples and how many of them belong to the different classes, divided by articles.

Articles	Violation	Non-violation	Total
Article 2	398	0	398
Article 3	851	0	851
Article 5	1118	0	1118
Article 6	4092	0	4092
Article 8	496	0	496
Article 10	252	0	252
Article 11	89	0	89
Article 13	1060	0	1060
Article 14	0	44	44

Table 1: number of samples contained in the folder “test_violations” of the dataset crystal_ball

Articles	Violation	Non-violation	Total
Article 2	57	57	114
Article 3	284	284	568
Article 5	150	150	300
Article 6	458	458	916
Article 8	229	229	458
Article 10	106	106	212
Article 11	32	32	64
Article 13	106	106	212
Article 14	144	144	288

[2]https://github.com/daminienrico/NLP_judicial_text_classification

Table 2: number of samples contained in the folder “train” of the dataset crystal_ball

Articles	Violation	Non-violation	Total
Article 2	14	14	28
Article 3	71	71	142
Article 5	38	38	76
Article 6	114	114	228
Article 8	57	57	114
Article 10	27	27	54
Article 11	8	8	16
Article 13	27	27	54
Article 14	36	36	72

Table 3: number of samples contained in the folder “test20” of the dataset crystal_ball

Analysis of previous works

State of the art & Limits

The most popular paper about predicting judicial decision is [16]. Not only did the authors aim to lay the foundations for other other several projects, but also to highlight the possibility of building advanced models which can be a useful tools for both lawyers and judges. In this paper, the authors present a study based on a binary classification problem which consists in building a model able to predict the outcome of the cases of the European Court of Human Rights. They aim to extract textual content from a case and, through an SVM classifier, predict the actual judge decision, which means whether or not an article has been violated or not.

They apply n-grams techniques for representing the text words in a continuous domain. They also use other methods, for example a clustering of the most

common n-grams that they call *topics*. The number of features they use, i.e. the word vectors length, is 2000, which are the most common n-grams in the corpus.
They are finally able to predict a court's decision with an average accuracy of 79%.
However, the dataset used in this study is relatively small, in fact, the total number of cases, which are just three, is 584.
Another very interesting project is [17], which take into account more articles (nine) and consequently have a bigger dataset, 3132 samples. They represent the text words with the same n-grams method of the paper mentioned above, expect that the vector length changes for each article since they take into account all the n-gram features, which means that they have a really high-dimensional vectors. In the end, they reach an overall average of 74%.
It has to be mentioned the fact that they tested the classifier on a dataset containing just one class. They did this because the original dataset was even bigger than 3132 samples, however it was really unbalanced. So they decided to use the remaining slice of unused samples as the test set.
Another observation is about the Data Preprocessing they apply: by reading the code they provided, it came to light that they do not exclude some parts of the text where it is explicitly written the Court's decision. This means that the results they obtained are influenced by biases.
The last, but not least, consideration is about the number of shuffles of the dataset. As we said above, the dataset is not really big and, usually, in this situation is important to adopt an approach which consists in shuffling the dataset more than just one time. The reason, as we will show further in this study, is because a particular shuffle could lead to very good results, while another shuffle of the same data could head to bad results.

Re-implementation of the state of the art model

In order to be able to evaluate the goodness of the models that we are going to introduce in this project, we implemented a version with the same semantic of the state of the art model. Our version differs to [17], in the experiment 1, in

terms of output; we apply 10-fold cross validation and 10 times the shuffling of the dataset, thus, we have a 10x10 matrix for each article.

The dataset we are going to take into account is the merging of the train and test20, since there is no reason we need to exclude the latter.

We apply the exactly same preprocessing of Medvedeva and, thus, use the same iper-parametrisation.

Instead of using the function sklearn.model_selection.cross_val_predict we employe the cross_validate method. We take this decision since it is not appropriate to pass these predictions into an evaluation metric, according to the documentation. The cross_validate function return a dictionary where it is possible to get the test score of each cross validation. In this way, we are able to build the output needed for the bayesian comparison. It is possible to download this version on the github project[3].

The figure below shows the algorithm that has been just explained. As we can notice, there are many points of choice, which, on one hand, it is very useful; especially if we want to find the best possible result. On the other hand, this means that we are overfitting the model with respect to each article.

The points of choice are the following:

- part: the facts, circumstances, procedure, procedure+facts;
- vec: ngram_range, binary=False, lowercase, min_df, norm, stop_words, use_idf;
- the SVM parameter c: [0.1, 1, 5] .

The embedding of the vectors has been done through the function of the sklearn library feature_extraction.text.TfidfVectorizer. We can see that partial of the pre-processing is done at this step and, thus, change for each article. Furthermore, this process is repeated for every shuffle, but, however, it does not interfere much in terms of time-computing.

[3] https://github.com/daminienrico/NLP_judicial_text_classification

Algorithm 7 Re-implementation Medvedeva

$data = extractPart(part, rawText_{articleN})$
$shuffles \leftarrow 10$
$folds \leftarrow 10$
while $i < shuffles$ **do**
    $shuffling(data)$
    $samples, labels = getFrom(data)$
    $pipeline = createPipeline(features(vec), SVM(c))$
    $accuracy[i, j] = crossValidate(pipeline, samples, labels, folds)$
end while

Figure 1 : algorithm of the re-implementation of the Medvedeva model.

Natural Language processing in the legal domain - techniques

Natural language processing is an important field of artificial intelligence and computer science studying the process of how computer programs can understand natural language by analysing text data.
It is surprising that the even the most recent works on juridical decision use simple NPL techniques, generally far from the state of the art. The main papers,[16] and [17], employ approaches which are considered basic compared to the most advanced techniques. The point is that sometimes, especially when there are not so many samples, a naive approach could lead to a better result. We are going to explore many different strategies in order to compare them and find the best NLP approach.
All these approaches follow a common pattern, analysing the raw natural text and produce a embedded representation of the words and documents in a continuous domain, vector space model or term vector model.

N-grams model

The Bag-of-words model is a word representation commonly used in NLP and IR (Information Retrieval). It is the easiest technique and consists in, firstly, defining the length of the embedded vectors and, secondly, mapping the words to their frequency (known as term-frequency) in order to produce a vector as output.
The length of the vectors, corresponding to the number of features, is defined by considering all the words in the processed text. So, for example, if our corpus consists of two sentences “I do love you” and “I do not like Mary”, a vector would have dimension seven since we have totally seven words:

I	do	love	you	not	like	Mary
1	1	1	1	0	0	0
1	1	0	0	1	1	1

As we can imagine this method has some limits, for instance, the vectors length could be very long depending on the corpus we have. This would not be ideal, especially if we want to build a model that can scale. Furthermore, another important problem is that the words ordering information is lost we we transform the sentences into vectors. For the purpose of avoiding this important lost of information it is possible to generalise B.O.W. by considering n-grams. This method tries to preserve the order by considering
together a group of n words. For instance, in the previous example if we consider 2-grams we would have:

I,do	do,love	love,you	do,not	not,like	like,Mary
1	1	1	0	0	0
1	0	0	1	1	1

In this way it is possible to partially take into account the order. We can also imagine to consider more than n-grams at the same time: in this case if we consider 1-grams and 2-grams we would have vectors of length 13.
It is easy to understand that this model does not scale well if we have to deal with a large corpus, however, it tends to perform well otherwise.
A way for improving scalability is to apply the hashing trick model, which maps words directly to indices through an hashing function.

Using Neural Networks

One of the most famous slogan of the well known British linguist JR Firth was: “You shall know a word by the company it keeps”. Exactly the same notion that Wittgenstein proposed; he suggested that the right way to think about the meaning of words is understanding their uses in text. Essentially, if you could predict which textual context the world would appear in, then you understand the meaning of the world.
The main issue of all the previous models, which are similar to the n-grams model, is that they can not take into account the semantic sensitivity and the order in which terms appear. Moreover, they assume terms are statistically

independent and long documents have poor similarity values, in other words a large dimensionality.
To overcome to all these problems is necessary to use different methods, such as employing neural network based techniques. Even if neural networks are not as intuitive as the simpler models we presented above, they represent a really powerful tool that can help on this task. There are plenty of NNs that can be employed, from the easiest one hidden layer NN to more complicated approaches such as Long Short-memory NNs, Recurrent NNs or Convolutional NNs.

Doc2vec model

As a starting point, we decided to use a well known model, a distributed representations of sentences and documents: Doc2Vec, which can represent documents as fixed length low dimensional vectors. The real benefits is that it is possible to consider some semantic meanings between words [18]. Before going into details, it is better first to introduce the model that Doc2Vec generalises: Word2vec.

Word2vec

Not only does this model aim to improve accuracy, but also to reduce computational cost. In our case, due to the fact that the dimension of the crystal ball dataset is not that high, even if we will not increase accuracy of prediction, we would be able to reduce the dimension of each documents representation up to 30000% respect to the model built by [17] and to not decrease the accuracy.
Word2vec is a simple and fast to train model that we can run over billions of words of text in order to produce exceedingly good word representations. Basically, the goal of Word2vec is to change the representations of words, in order to minimise the loss function:

$$loss = 1 - p(context \mid w_t)$$

where $p(context \mid w_t)$ is the probability that the context occurs given the word w_t. There are two main implementations depending on the algorithm used:

CBOW (continuous bag of word) architecture and SG (Continuous Skip-gram Model). The aim of the first one is to learn word representations in order to predict a target word given a set of words. Each word is mapped to a vector which is described by a column in a matrix W, which is indexed by considering the position of the word in the vocabulary. The goal is to use this vectors as features by concatenating, or averaging, them in order to predict the next word in a sentence. More formally the objective of this model is to maximise the average log probability :

$$\frac{1}{T}\sum_{t=k}^{T-k} logp(w_t \mid w_{t-k}, \ldots, w_{t+k}) \text{ . (1)}$$

Where T is total number of training words and $p(w_t \mid w_{t-k}, \ldots, w_{t+k})$ is the probability the w_t occurs, knowing the k words before and after the considered word w_t. Furthermore, it has been used the logarithm for simply representing probabilities in logarithm space, instead of normal domain [0,1].
The prediction task is done through a multi-class classifier, such as softmax

$$logp(w_t \mid w_{t-k}, \ldots, w_{t+k}) = \frac{e^{y_{w_t}}}{\sum_i e_i^y}$$

and each *y* is computed as

$$y = b + Uh(w_{t-k}, \ldots, w_{t+k}; W) \text{ ,}$$

where *b* and *U* are parameters of the softmax while *h* is a function that concatenate or average word vectors obtained from *W.*

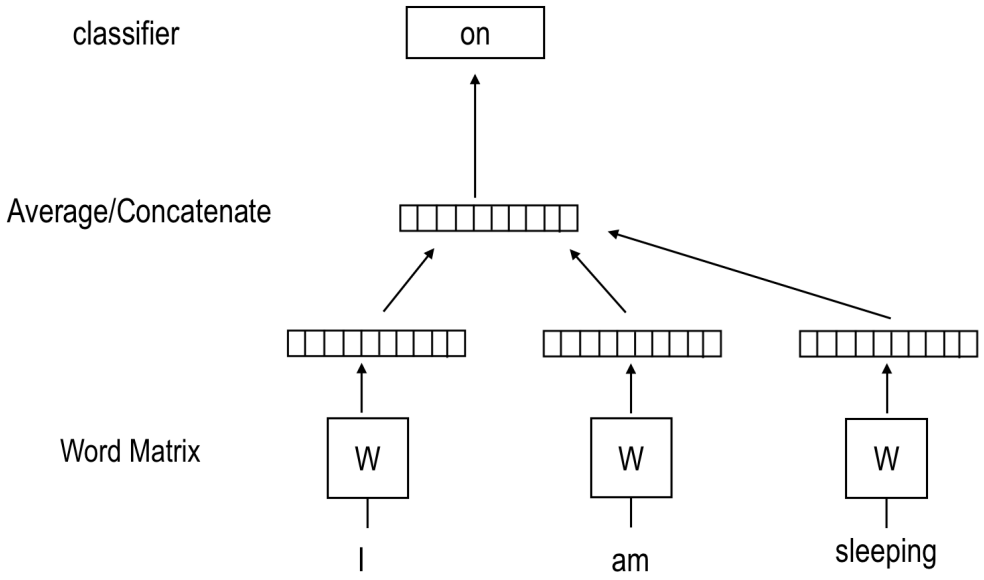


Figure 2 : It is shown the structure used for learning word vectors. The three words are needed for predicting the fourth word "on". As we can see, the matrix W, used for predicting the output, consists of columns which represent the input words.

When the training is completed (normally the NN word vectors are trained by applying stochastic gradient descent and the gradient is obtained via back-propagation), words with similar meaning are mapped to a similar position in the vector space.

The latter model (SG) is used for the opposite task: given a word tries to predict the most likely surrounding words, in a certain window. More formally the objective function to maximise is:

$$J'(\theta) = \prod_{t=1}^{T} \prod_{-m \leq j \leq mj \neq 0} p(w_{t+j} | w_t; \theta) \quad (2)$$

Basically, given a large sequence of words we iterate over each position in the text: for each position we have a window of size *2m, m* words before and *m* words after it, and for each element in the window we have the probability that a word appears in the context of the centred word. As we can see there is just one parameter, *θ.* This means that we can only change this parameter for maximising that probability. Furthermore, *θ* is going to be the vector representation of the words.

In order to evidence the similarity with the first (1) , the (2) can also be seen as:

$$J(\theta) = \frac{1}{T}\sum_{t=k}^{T-k} logp(w_{t-k}, \ldots, w_{t+k} | w_t)$$

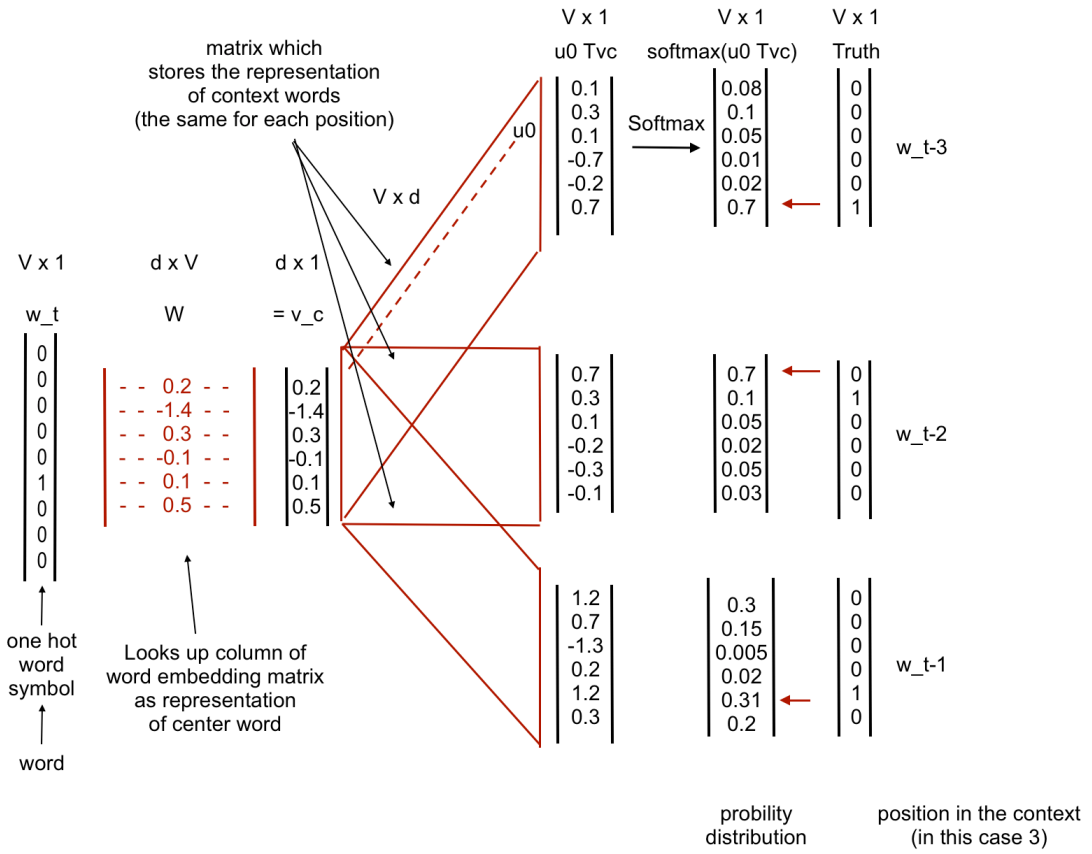


Figure : It shows in details the skip-gram model.

All in all, these NN models have a simple architecture, which consists of one input layer, one hidden layer and one output layer and a softmax activation function for each neuron.

A very attractive and interesting property is that differences between vectors have meaning. This can be highlighted by simply using vector algebra, given the corresponding vectors of King, man and woman :

$$King - man + woman = Queen$$

However, when it comes to capture differences between the same word used in a different context, e.i. different documents, we need to upgrade the current model: we need to include, not just the word vectors, but also paragraph embeddings in order to capture paragraph.

For example, we want to have different representation of the word “nails”:

1. The girl managed to have her nails done.
2. He bought some nails for hanging a painting on the wall.

In the Word2vec model the word “nail” would have exactly the same representation in both sentences, as we are going to see in Doc2vec the have not.

PV-DM

This approach (Distributed Memory Model of Paragraph Vectors) that Mikolov uses in this paper [14] is similar to the word embeddings learning that we have described above. Basically, the paragraph will also be included in the prediction task of the next word (given pieces of context in the paragraph).

In the model, every column of matrix D represents uniquely a paragraph and in the same way, each word is mapped as a unique vector in matrix W. As we can see in figure 3, the vectors are concatenated or averaged so that to predict the next word given a context.

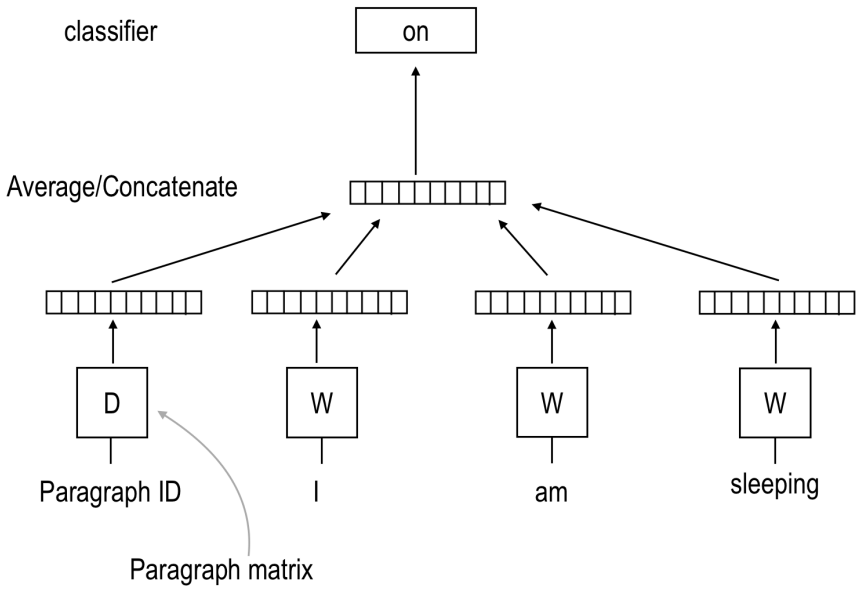


Figure 3 : description of the Distributed Memory Model of Paragraph Vectors.

Essentially, the main and most important change of this model with respect to the previous one is that the h (in equation 1) has been built from both matrix D and W. The name of the model comes from this behaviour: we can think about the paragraph token as another word, which operates as a memory in order to remember the missing words from the context (which is a fixed-length piece of a sliding window over a paragraph).
It is important to underline that the paragraph vector is just shared throughout all the contexts which have been created from the same paragraph; they are not used by all paragraphs together. However, matrix W is shared among all paragraphs so that each single word is represented in the same way for all paragraphs.
For the training phase, it has been employed stochastic gradient descent and the gradient has been got via back-propagation. Each step consists in sampling a fixed-length context from a paragraph randomly chosen, computing the error and use the gradient for updating the parameters in the model.
The gradient descent is also used when it comes the time to predict, it is necessary to carry out an inference step in order to compute the paragraph vector for a new paragraph. During this phase the parameters of the model are fixed.
If we take a closer look at numbers of parameters, we can see that they can be a considerable number as the number of paragraph increases. The model would have a total of parameters equal to $N \times p + M \times q$, where N is the number of paragraphs, M the words in the vocabulary and p and q the dimension of the paragraph and word vectors. However, the updates are typically sparse during training and so it doesn't compromise the efficiency.
Finally, when the training phase is concluded, the paragraph vectors can be seen as features to be feed directly to the most common machine learning methods, such as SVM (support vector machine) or logistic regression.

All in all, the main phases of this model are the follow:

1. the unsupervised learning stage, to get the word vectors W,
2. the inference stage, to get the paragraph vectors D,

3. using the features created, i.e. the embedding vectors, for classifying samples, through a standard classifier.

One of the most important advantages of the Distributed Memory Model of Paragraph Vectors is that they are able to learn from unsupervised data and therefore suitable for tasks that do not have enough labelled data.
This model is substantially better than n-grams model because of the important inherited property that preserves the semantic similarity between words. Secondly, the word order is taken into account in the same way the n-grams model would do with a large n. Even if they both could preserve the same order information about the paragraph, the latter model would generate a very high-dimensional vectors, which means a poor generalisation.

PV-DBOW

A different approach would be to ignore the context in input, i.e. the matrix W in figure 3, and force the model to predict words sampled in the paragraph. This means that, given the paragraph vector, at each step of stochastic gradient descent, a word is randomly chosen from a text window that is currently taken into consideration. In the end, a classification task has been form. This version has been named Distributed Bag of Words version of Paragraph Vector (PV-DBOW) since is the opposite of the previous one.
As we can see from figure 4, this model need to store just the softmax weights, while the PV-DM model has to remember the word vectors too, similarly to the kip-gram model in word vectors.

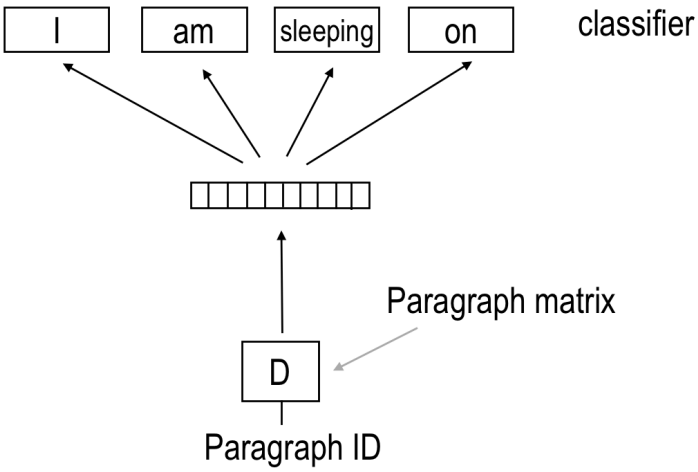


Figure 4 : representation of Distributed Bag of Words version of paragraph vectors.

In conclusion, with the PV-DM model it is possible to achieve the state-of-art performances, but it is also possible to use a combination of them in order to employ this methods across many different tasks.

Machine learning for legal text classification

In this study we explore some different ways for the classification task. In text classification a model that experimentally works well is Support Vector Machine, or SVM. Another possibility is exploiting the Doc2Vec model by measuring the cosine similarity between models and find the 1-nearest neighbour. All of this methods are supervised.

Support Vector Machine

SVM is a supervised method which is able to perform classification by maximising the margin between the two classes through a hyperplane. It is commonly used for tasks such classification, regression and outliers detection.
In an SVM model, the training samples are mapped as points in space and marked as belonging to a specific class. The vectors that belong to the hyperplane that splits the space are called support vectors.
The algorithm tries to that maximises the margin and minimises the misclassifications error. The best result is reached when the nearest training vectors, belonging to different classes, are separated by the hyperplane with the largest distance.
At test time, new samples are mapped in space and the predictions are based on which side of the hyperplane the vectors are.
However, this is just when data are linearly separable. When they are not a hyperplane is not able to separate the two categories. In these cases, SVMs apply the what is called kernel trick, in order to perform non-linear classification. The kernel trick consists in defining the vectors in higher-dimensional space so to find a higher-dimensional hyperplane that is able to separate the vectors in two classes.

More formally, given training vectors $x_i \in \mathbb{R}^p$, *i=1,...,n* in two classes, and a vector $y \in \{1, -1\}^n$, Support Vector Classification solves the primal model:

$$\min_{\alpha} \frac{1}{2}\alpha^T Q\alpha - e^T\alpha$$
$$\text{subject to } y^T\alpha = 0$$
$$0 \leq \alpha_i \leq C, i = 1,...,n$$

where *e* is the vector of all ones, *C > 0* is the upper bound,*Q* is an $n \cdot n$ positive semidefinite matrix, $Q_{ij} \equiv y_i y_j K(x_i, x_j)$, where $K(x_i, x_j) = \phi(x_i)^T\phi(x_j)$ is the kernel. Here training vectors are implicitly mapped into a higher (maybe infinite) dimensional space by the function ϕ .

Consequently, the decision function would be :

$$\text{sgn}(\sum_{i=1}^{n} y_i\alpha_i K(x_i, x) + \rho)$$

See here[4] for further details.

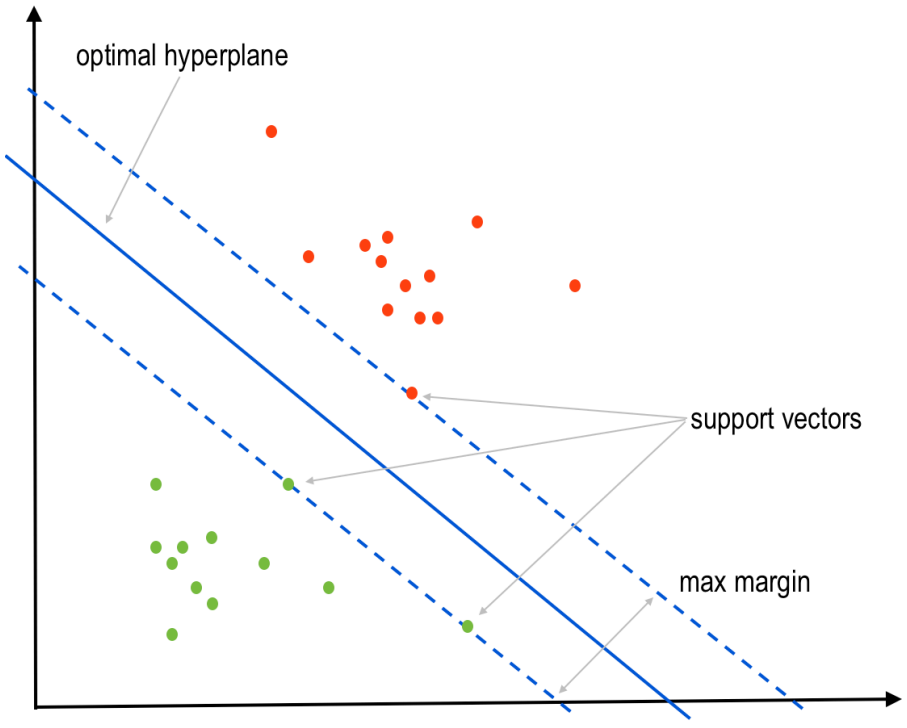


Figure 5 : Support Vector Machine hyperplane example.

[4] http://scikit-learn.org/stable/modules/svm.html#svm-mathematical-formulation

The first main advantage of support vector machines is that it is effective when we have to deal with really high dimensional vectors, even if the number of samples is lower than the number of features. Secondly, it needs to remember just the support vectors and not the all training dataset, thus memory efficient. The complexity of this methods is $O(n_{features} \times n_{samples}^2)$ but it also depends on how the implementation use the cache. Thirdly, it is possible to specify different Kernel functions due to its versatility.
The disadvantages are that it is crucial choosing the right Kernel and it is necessary apply regularisation term when the vectors dimension is much greater then number of samples.

K-nearest neighbours - cosine similarity

The K-nearest neighbours (K-NN) is a classification technique which doesn't require any parameters for the algorithm. As SVM, it is commonly used for classification and regression problems.

In 1-NN, the training samples are mapped in space. When it is needed to perform the classification task, the test sample is mapped in space and K-NN algorithm labels the sample with the same class of the first nearest neighbour. When *K > 1,* it will be assigned the label which is the most common among the K neighbours. The nearest neighbour it is chosen based on some attributes we want to take into account.
It is also possible to adopt a slightly different strategy which consists in weighting the contribution of each neighbour, i.e. depending on the distance.

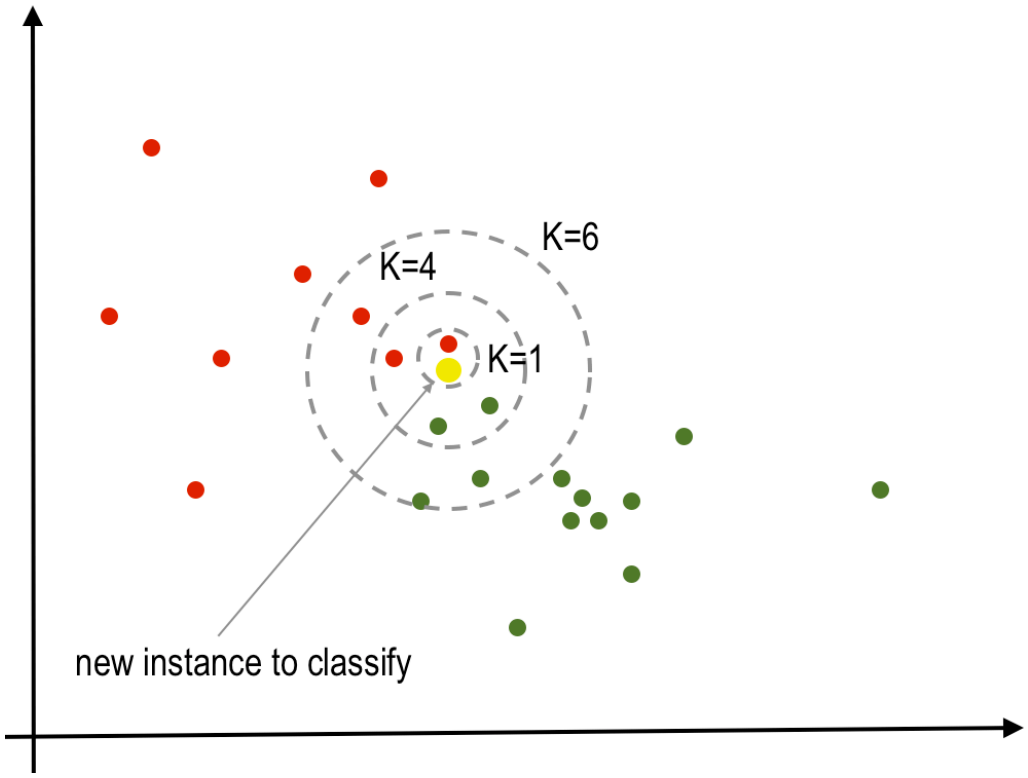


Figure 6 : K-NN example.

Given a new sample to classify, there are many possibilities for calculating the distance between the new sample and training samples. A common distance metric is Euclidian distance, just for continues variables, or Hamming distance for discrete variables.

Another possibility for checking the similarity in vector space, the one we are going to use in this study, is cosine similarity. Sometimes, especially when we need to compare two texts, it is better to adopt this approach as if we used the Euclidian distance, it would be affected by the magnitude of the vectors. Cosine similarity measuring the inner product between two-zero vectors, basically it takes into account the vectors orientation rather than the magnitude. The cosine is 1 when the angle between the vectors is 0 radians and less than 1 for any value in (0,π]. Thus, vectors with the same orientation would be considered identically. On the other hand, when their angle is π radians difference their cosine similarity is 0. The same identical procedure is applied for higher dimensional spaces.

A document would be defined by a vector of a given dimension, depending on what technique we use for embedding the text, where each dimension correspond to a word or a n-gram. Cosine similarity gives useful and

meaningful information about how two texts are similar in terms of what they contain.
More formally:

$$similarity = \frac{A \cdot B}{||A|| ||B||}$$

where A and B are the two text vectors under comparison. The main advantage of cosine similarity is the low time complexity, even for sparse vectors, which is quadratic and thus really suitable for real word applications.

Model

Hardware & Environment

Since most of the experiments we are going to introduce in the next chapters are high time-consuming, we find reasonable set up a server and let the scripts run on it. However, for practical reasons, not every experiment has been done on the server.
The server has the following implementation details:

- Operating system: Ubuntu 18.04.1 LTS
- Processor: 2,2 GHz Intel Xeon E312xx Octacore
- Memory (RAM): 32 GB

While the other computer, a MacBook Pro (Early 2015), on which we have done some of the experiments has the following features:

- Operating system: macOS Sierra 10.12.6
- Processor: 2,7 GHz Intel Core i5
- Memory (RAM): 8 GB 1867 MHz DDR3

Since we track the computational times, it is also important to stress the fact that if we want to boost our performance we need to install some libraries such as BLAS and LAPACK, in order to let Numpy work faster. They are libraries, originally written in FORTRAN and then in C++, that can provide faster computation in doing linear algebra operations involving matrices and vectors

All the programs have been coded in Python 3 adopting Jupyter Notebook as environment. We have chosen this platform because it adapts well to our workflow by speeding up all the routine where it was possible. For example, all the IO operations such as reading datasets from a csv file have been previously loaded in memory, so to not to repeat the process every time.
We also created a file python where we put all the utility functions used among Jupyter notebooks. In this way, we can just import what we need, so to do not write any repeated code.

All the project could be found at this link[5] on github. Every notebook starts with the name of the model applied and a summary of what strategy have been adopted.
In addition to the notebooks, it is possible to find the preprocessed datasets for reproducibility purposes.

Structure

Each of the models we are going to present has the same main structure and pipeline. Firstly, we need to convert the raw text of the original dataset in a format suitable for the Machine learning process. We are going to address this problem and explain the details in the next section Preprocessing.
Secondly, after having created the new dataset, we need to train the first component of our model, which is a Neural Network. This phase is needed for creating the embedding vectors. On top of this, we generate a new dataset in a continues domain.
Thirdly, we train the second component, which is a supervised classifier, by feeding it with the latter generated dataset.
In the end, we test the accuracy of the just trained classifier.

Preprocessing

The Data Preprocessing is a really important and crucial aspect of Machine Learning and many data scientists have struggled to give meaning to the preprocessing of data; we could look at it as the foundation of our work. In fact, it is extremely important to take the right choices during this phase, because they could lead us to better or worst results and, therefore, to wrong conclusions.
In the same way a nurse prepares a patient for surgery, we prepare raw data for further processing. It basically consists of many phases depending on the quality of the data we have to deal with. The goal is to transform raw data into a format which is suitable for the data mining technique we are going to adopt.

[5] github.com/daminienrico/NLP_judicial_text_classification

In general, there are some best practice to follow in text classification, however, we must explore all the possibilities for better understanding how our system works, in order to find the best tuning. For example, if we would apply a too “relaxed” pre-processing, our model could be affected by biases or could not work well because of some noise. On the contrary, in we apply a too “strict” pre-processing we could cut off some important patterns of the text, so to not let the classifier predict well the outcome.

For all these reasons, it is important to understand with what data our model works well. In our experiments, we try both strict and simple pre-processing, in order to have a more complete idea about how text have to be handle for achieve better results.

As it is mentioned in the “Analysis of previous works” section, in the experiments of [17], they do not apply a strict pre-processing and so they let their model be influenced by biases.

Moreover, the pre-processing parameters change by article, which is an hyper-parametrisation that overfits the model.

The pre-processing parameters used are different for every article. This strong hyper-parametrisation approach leads to a strong overfitting of their models, especially since the number of cases for each article are small. We aim to apply the same pre-processing for all the samples of each article, in order to build a general model rather than a specific model for every single article. The goal of the project is to find a generic approach to apply also to different juridical datasets.

For this purpose, we use some of the most common python libraries, such as NLTK and regex.

Our **strict pre-processing** pipeline is the following:

1. Excluding some part of the text, such as the part “THE LAW” which contains the decisions of the Court. We also eliminate the first part of the each case law which consists of a list of judges and some other information such as the date and an ID number of the case;
2. as suggested by [16], we delete all the sentences that match with “the court”, “echtr” and “european court of human rights”, this is done for deleting all possible leaking of informations regarding the outcome in the other part of the case law;

3. deleting all the punctuations and multiple spaces;
4. applying lower case;
5. with the purpose of making the model not depending on the dates, we delete all them.;
6. we replace the strings such as “article 3” in “article3” and “articles 2 14 5” in “articles2-14-5”, so the model would learn this representation of the each article mentioned, e.g. “article3”, as a single word. The idea is that the model will be better able to find common patterns in the texts;
7. deleting all the stop-words, which are not just the one provided by NLTK. You can find them in the repository of this project at this link[6];
8. deleting all the numbers that remains at this point;
9. deleting all the letters which have not a latin format;
10. tokenising the text;
11. applying stemming.

In the end, we will have a list of string well structured, which represents a case law. This format is suitable for the class of the gensim library models.doc2vec.TaggedDocument, which will tag the documents with the respective labels.

As a possible upgrade, with respect to the point 5, we could normalise the date instead of deleting them.

On the other hand, the **simple pre-processing** pipeline is:

1. Excluding some part of the text, such as the part “THE LAW” which contains the decisions of the Court. We also eliminate the first part of the each case law which consists of a list of judges and some other information such as the date and an ID number of the case;
2. as suggested by [16], we deleting all the sentences that match with “the court”, “echtr” and “european court of human rights”, this is done for deleting all possible biases in the other part of the case law;
3. applying lower case;
4. deleting all the punctuations and multiple spaces;
5. deleting all the letters which have not a latin format;

[6] github.com/daminienrico/NLP_judicial_text_classification/blob/master/stop.txt

6. tokenising the text and deleting words not contained in the stop-words list provided by NLTK.

This pre-processing is the basically the same adopted in [17], however we delete the biases and clean the corpus from spurious data.

Classification

SVM

For the Support Vector Classification task in some experiments of this project we used sklearn.svm.LinearSVC implementation. As it is reported in the documentation[7]. It is implemented in terms of liblinear[8], so to have more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples. The underlying C implementation uses a random number generator to select features when fitting the model. It is thus not uncommon to have slightly different results for the same input data.

Normalisation

When it comes the time to create the vector through the Doc2vec model, it is also necessary to scale each sample individually to unit norm, before feeding these vectors to the Support Vector Machine. This is important because SVM works way better if the samples are scaled. For applying the normalisation we used sklearn.preprocessing.normalize with the following parameters: norm="l2" and axis=0; the first one indicates the norm to use to normalise each non-zero feature, while the latter represents the axis used to normalise the data along. In this case, we decided to normalise each features since we could have a better representation of the data across all the documents.

K-NN

For implementing the K-NN algorithm based on cosine similarity we exploit the functions of the Doc2vec provided by the Gensim library. More precisely, we used the object gensim.models.keyedvectors.Doc2VecKeyedVectors which contains the paragraph vectors. As it is written in the documentation9, the only difference between this model and Word2Vec is that besides the word vectors

[7] http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

[8] https://www.csie.ntu.edu.tw/~cjlin/liblinear/

[9] https://radimrehurek.com/gensim/models/doc2vec.html

we also include paragraph embeddings to capture the paragraph. In this way we can capture the difference between the same word used in a different context.
For retrieving the information about which document is the closets in terms of cosine similarity, we use the most_similar function which provides the top-N most similar document vectors from the training set. This method computes cosine similarity between a simple mean of the projection weight vectors of the given documents. Documents may be specified as vectors, integer indexes of trained document vectors, or if the documents were originally presented with string tags, by the corresponding tags.
It is important to underline the fact that we have as many top-N documents similarity as the number of tags. For instance, if we would label all the documents (with respect to the function TaggedDocument) with 0 or 1 for indicating the belonging class, we would end up having at most 2 most similar results, furthermore, with a relatively low similarity. The right way is to make every single documents unique and at same time given the supervised information about the document label. For example, by passing a tuple which consists of document's index and belonging class.

Grid search

In the model we have defined, we have many different possible variations; deciding to apply a configuration instead of another could lead to very different results. However, in terms of time, it is impossible to find the best configuration through enumeration. Thus, we are going to identify all the possible choosing points and select the ones that we think they are most likely leading to the best results.
We set up a pipeline which allows us to try a set of the possible combinations given a set of parameters and, in the end, make us able to choose the one with highest accuracy. In this way we are sure that the parameters are correctly set.
For most parameters there is a default configuration, which is likely the best one. Anyway, the text corpus could have a considerable impact on the model and, thus, the way these parameters are set affects how the model work.

We identify the Doc2Vec parameters as the most interesting for us, as they could have a huge impact since they create the embedding vectors. As it is reported in the documentation[10] of the Gensim library (in square brackets it is reported the possible values we have chosen) :

- **vector_size [100,500]**– Dimensionality of the feature vectors;
- **window [3,10]** – The maximum distance between the current and predicted word within a sentence;
- **min_count[1,100]** – Ignores all words with total frequency lower than this;
- **epochs [20,50]** – Number of iterations (epochs) over the corpus.
- **hs [1,0]**– If 1, hierarchical softmax will be used for model training. If set to 0, and *negative* is non-zero, negative sampling will be used.
- **negative [5,20]**– If > 0, negative sampling will be used, the int for negative specifies how many "noise words" should be drawn (usually between 5-20). If set to 0, no negative sampling is used;
- **ns_exponent [0,0.75,1]**– The exponent used to shape the negative sampling distribution. A value of 1.0 samples exactly in proportion to the frequencies, 0.0 samples all words equally, while a negative value samples low-frequency words more than high-frequency words. The popular default value of 0.75 was chosen by the original Word2Vec paper;
- **dm_mean [1,0]** – If 0 , use the sum of the context word vectors. If 1, use the mean. Only applies when *dm* is used in non-concatenative mode;
- **dbow_words [1,0]** – If set to 1 trains word-vectors (in skip-gram fashion) simultaneous with DBOW doc-vector training; If 0, only trains doc-vectors (faster).

We decided to use all the possible values for binary parameters, and values close to the suggested/default for the others. We needed to selected subset of them to not have an excessively big grid. For instance, the possible values for vector size are $[1,\infty)$, and, clearly, we can not try all of them. Thus, we follow a simple principle. Let say that best solution found is a local maximum, we could choose values near the local maximum for finding the best solution or, instead,

[10] https://radimrehurek.com/gensim/models/doc2vec.html

we could try a random value for exploring the solution space which could lead to the global maximum we are looking for. However, it is though to guess which could be the value that heads to the best solution. However, before trying and deciding the values, we run some experiments, which are not reported, in order to let us have a rough idea.
With the parameter values we have chosen, we have totally 256 possible combinations, which means in case of model 3, where we have to train the Doc2vec every time, 4 days of computation. For this purpose, a server had to be set up.

Another point of choice would be the SVM parameters, but however just one is interesting to tune; the penalty parameter C of the error term. The best results are obtained leaving the default value.
The result we obtained, depending also on the preprocessing used, are reported in the conclusion of each model.

Bayesian comparison

The Bayesian comparison is a technique that, recently, has started being used instead of the more common null hypothesis significance testing (NHST). The aim is to ensure statistical validity of obtained results, however, the latter method has been shown as unsuitable [19].
The Bayesian comparison model is able to answer the question of whether or not a model is better than another. Given a dataset, it offers a formal way of assessing models and finding out which is the best one in terms of quality and predictivity. Basically, whenever a more complex model is used, the Bayesian model is able to tell if the data requires that complexity or not.
For what concern us, we need to compare the accuracies of two competing classifier through cross-validation, which would be the current state of the art model and our best model, considering articles separately on the same dataset. The best suitable approach is the **Bayesian correlated t-test** which is proposed in [20], [21] and [24]. As it is reported in [20], this test is adopted for the analysis of cross-validation results on a single dataset and it explains the

correlation due to the overlapping training sets. Basically, we will be able to assess the probability that two classifiers are equivalent.

As it has been well analysed in [24], the Bayesian analysis calculates three posterior probabilities, which are the follows (considering a classifier A against a classifier B) :

- $P_{A>>B}$: the posterior probability of the mean difference of accuracies being practically in favour of Algorithm A
- $P_{A==B}$: the posterior probability of the two classifier being practically equivalent
- $P_{A<<B}$: the posterior probability of the mean difference of accuracies in favour of Algorithm B

with $P_{A>>B}$ + $P_{A==B}$ + $P_{A<<B}$ = 1. Since it is possible to retrieve meaningful information even when they are under the 95% threshold, we can get, overall, a more informative outcome than the NHST.

In case we want to compare two models across multi-dataset, which means among all the articles, we need to apply another bayesian comparison test, which is called the **Bayesian sign and signed-tank**. This test is based on the Dirichlet process and it has been proposed by [22]. The Dirichlet process is a stochastic process and it can be seen as a distribution over probability distributions. This Bayesian analysis computes a posterior probability density function $p(z)$ and we can think about that as a hierarchical model where $p(z)$ depends on the weights w that are Dirichlet distributed.

$$p(z) = w_0\delta_{z_0}(z) + \sum_{j=1}^{n} w_j\delta_{z_j}(z) \qquad (w_0, w_1, \ldots, w_n) \sim Dir(s,1,...,1)$$

where: z denotes the scalar variable of interest, $z = \{z_1, \ldots, z_q\}$ is a vector of observations of z, s is the prior strength ($s > 0$), δ_{z_0} is a Dirac's delta centered on the pseudo-observation z_0. For any further details, see [20], chapter 4 section 2.1 .

The code we used for this task can be found at this link[11].

[11]https://github.com/BayesianTestsML/tutorial/tree/master/Python

Visualizing documents via t-SNE and PCA

In this section we explore how documents are represented by the Doc2Vec model. Since the embedded vectors are high-dimensional, we need to employ a technique which allows to reduce the dimensionality of the vectors in order to visualize them in a 2 or 3 dimensional space. We adopt two well known methods called t-SNE (t-Distributed Stochastic Neighbor Embedding) which is implemented via Barnes-Hut approximations; and PCA (Principal Component Analysis). The main difference between t-SNE and PCA is that t-SNE tries to deconvolute relationships between neighbors in high-dimensional data. t-SNE attempts to understand the underlying structure by prioritizing neighboring points. On the other hand, there are some reasons to prefer PCA instead of t-SNE. On top of them, PCA is deterministic, interpretable (PCA is just a diagonal rotation of our initial covariance matrix and the eigenvectors represent a new axial system in the space spanned by our original data; we can directly explain what a particular PCA does) and, given k as the dimension we want to reduce the vectors, it offers always the k best linear combination in terms of variance explained. While t-SNE solves a problem known as the crowding problem, which is somewhat similar points in higher dimension collapsing on top of each other in lower dimensions. Now as you increase the dimensions used the crowding problem gets less severe ie. the problem we are trying to solve through the use of t-SNE gets attenuated.

In the end, we try both methods in order to see which one works better. For this purpose, firstly, we use the implementation of sklearn, manifold.TSNE and secondly, we employ the module sklearn.decomposition.PCA. We plot the results by using the bokeh library.

We can decide to investigate the documents in many ways. For example by taking into account just the case law belonging to the dataset of a certain article or the documents belonging to a class considering more datasets. We show in figure below just as an example: the mapping in a two-dimensional space of the most frequent words contained in the documents belonging to the class Article 3. A strict preprocessing has been applied.

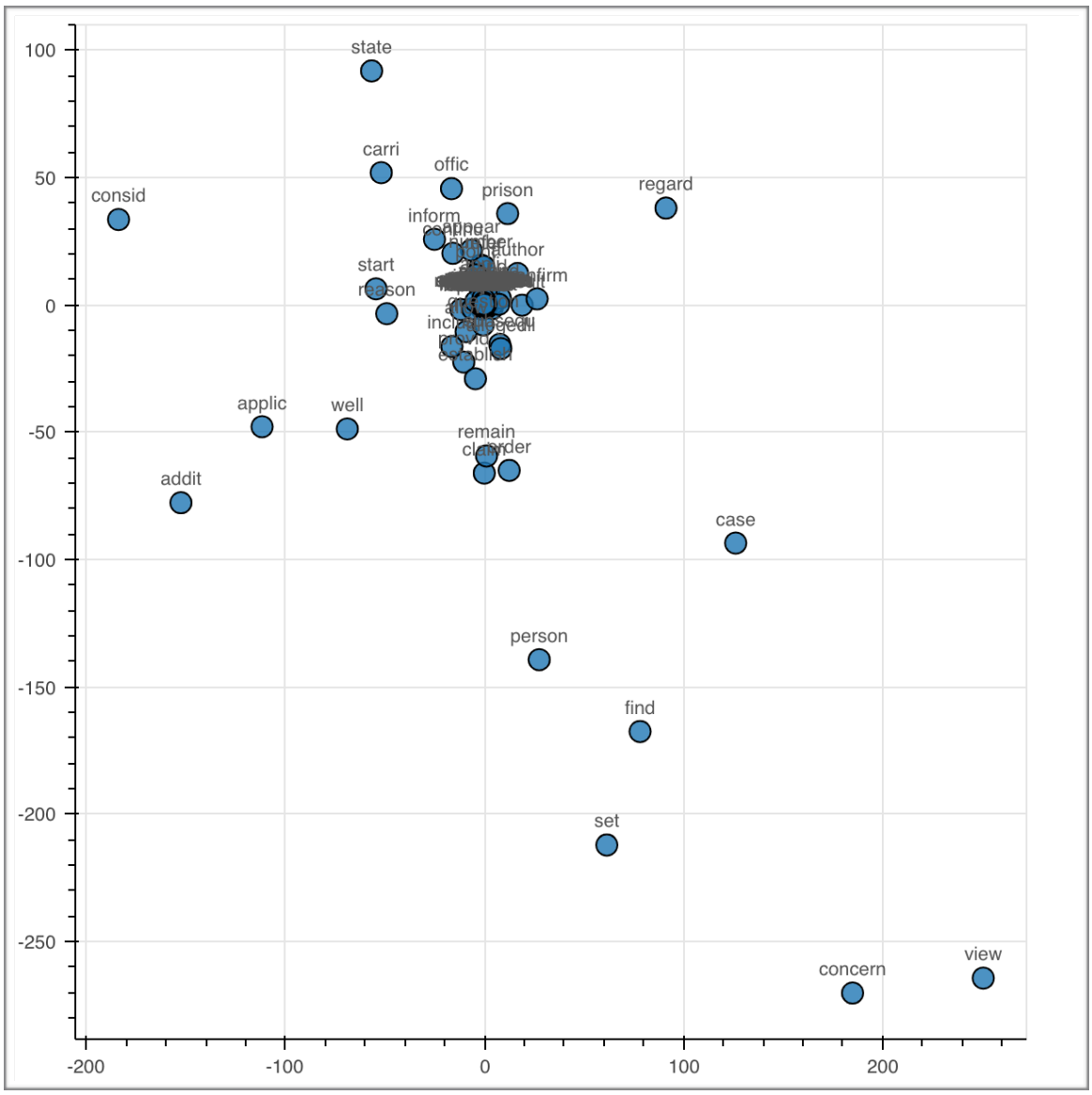


Figure 7: most frequent words of the documents belonging to the Article 3, using t-SNE.

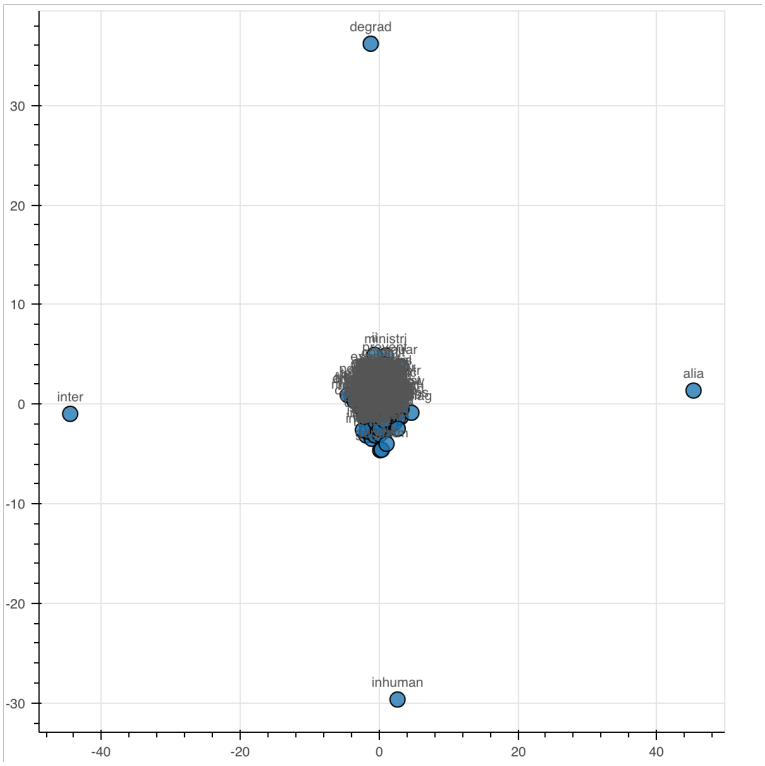
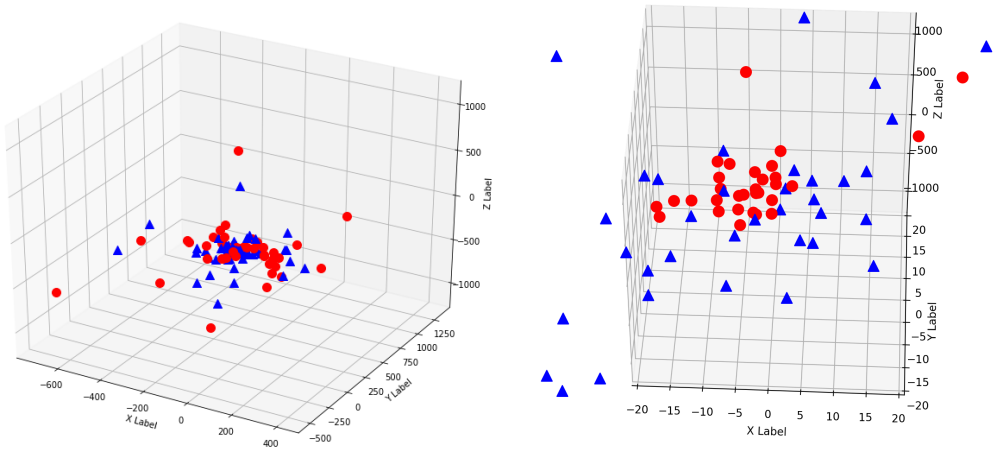


Figure 8: most frequent words of the documents belonging to the Article 3, using PCA.

As we can see, the t-SNE works a little bit better since the vectors are much more outdistanced, but not completely solving the crowding problem we mentioned above.

In addition, we show a different example which consists in training the Doc2vec on just one dataset (Article 3) and in separating the samples belonging to violations from the non-violations. We then plot the all dataset, showing how the two different classes are mapped in a three-dimensional space.



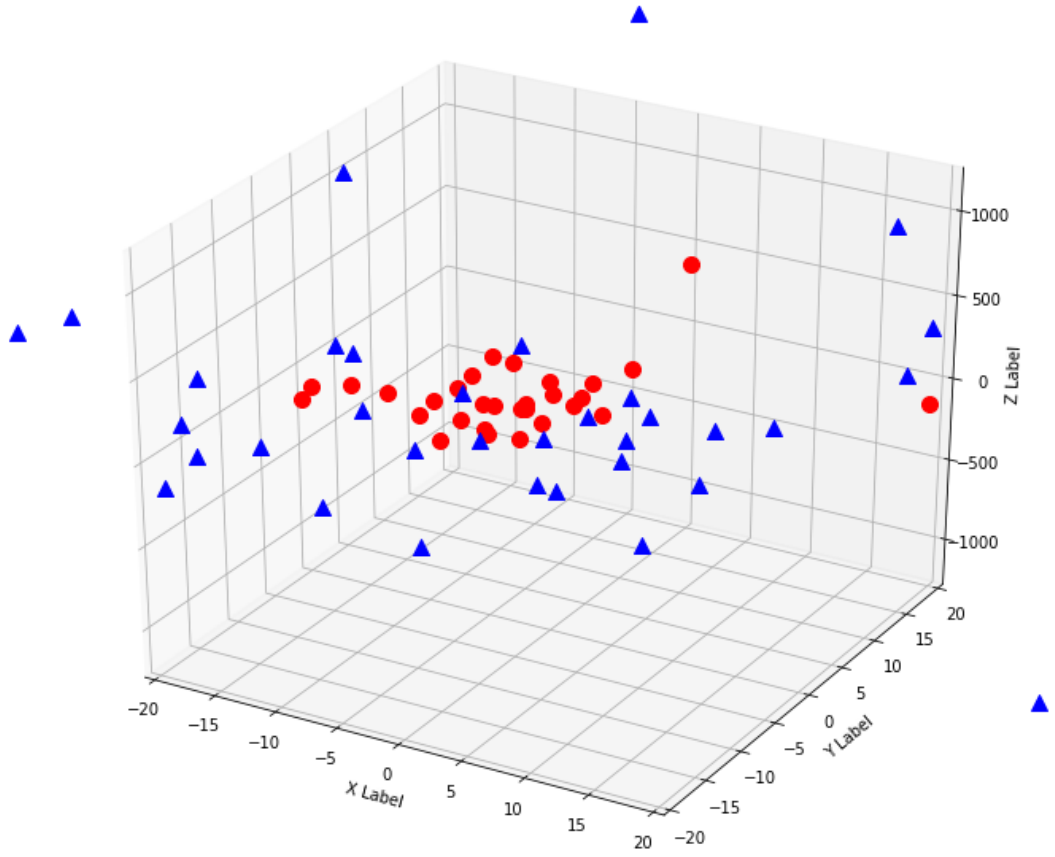


Figure 9: representation of two different classes belonging to the dataset of the article 3, after having applied t-SNE.

If we take a closer look to the figures above, we can observe that the two classes are mapped according to some criteria. One class is mapped in the center forming a cluster, while the other is also forming a cluster in the same position but with the samples more distanced.

We conclude that the two classes are not clearly linearly separable, so we have just found out that the kernel of the SVM could play a really important role in the model.

Experimental results

Model 1

Environment description

In this first experiment we are going to adopt an approach similar to what [17] have realised in their study. Given an article, for instance the number 3, we build a model that received in input the preprocessed dataset with respect to article we are taking into account, and, as output, a 10x10 matrix. This matrix represents the one hundred accuracies, that have been computed through the process. We obtain these results because we shuffle the dataset 10 times and we apply 10-fold cross validation. This is a standard approach in classifier comparison and it is necessary to apply the Bayesian techniques presented before. For the reproducibility of the experiments is essential setting a seed that is used to shuffle the dataset. This has been done because we want to be sure that a particular configuration of the dataset do not help the model in predicting the outcome, or, vice versa, we do not want that a particular configuration penalize the results. The 10-shuffles mitigates this problem.
We then compute the mean of all this matrix for being sure we get a trustable result.
For this experiment we first apply a strict preprocessing and then we explore the case which consists to apply a simpler preprocessing.
The dataset for this case consists of the merging between the train and the test20, taking into account each article separately. The code we used for merging the dataset is included in the module_preprocessing.py with the other preprocessing functions.
The Doc2Vec model is trained on the 9-folds remaining after the split of the cross validation. After having transformed the raw text corpus in vectors through the Doc2Vec, we are able to train the SVM (which is consequently trained on the 9-folds). We finally test the SVM on the rest 1-fold of the data.
The ten accuracy values (one for each fold) are saved and the routine starts from the beginning with a new shuffle. The seeds we used can be found in the code, just after the import.

As we said above, this procedure is repeated for ten times, see the figure below which illustrates the all process.

Doc2vec

Parameters	Configuration
vector_size	100
min_count	1
epochs	20

SVM

Parameters	Configuration
C	1
Kernel	linear

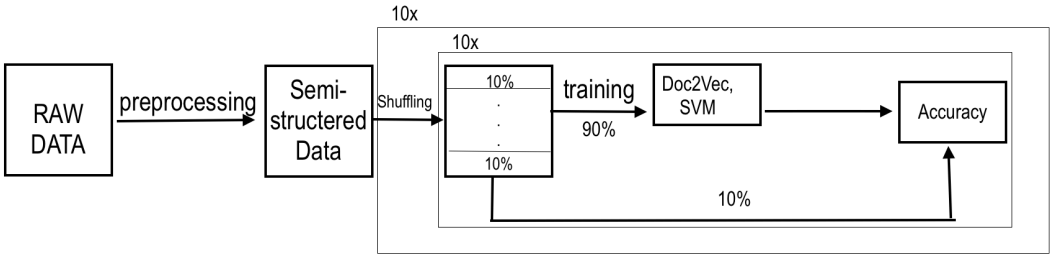


Figure 10 : description of model 1 routine

Algorithm 1 Model 1

```
data = applyPreprocessing(rawText_articleN)
shuffles ← 10
folds ← 10
while i < shuffles do
    shuffling(data)
    while j < folds do
        test,train = split(data)
        doc2vec=trainingD2V(train)
        svm=trainingSVM(train,doc2vec)
        accuracy[i,j]=test(svm,test)
    end while
end while
```

Results

The intermediate results and the final accuracy are shown in the table below. As we can see, the model we build does not perform significantly better than the the result achieve in [17]. However, the output we get is clearly more

authentic since we shuffle the dataset and compute the mean across all the intermediate accuracies. Moreover, the vector length is undoubtedly reduced, compared to [17], which means that we can work with less features for each sample.

In this first model we did not perform a grid search, thus we expect better results if we apply an hyper-parametrization.

Furthermore, it is important to underline the fact that, the more the data the better the NN works, thus in the next experiment we try to give as many text samples as possible. In addition, by applying a simpler pre-processing the text that represents each document is significantly more.

In conclusion, we compare the two pre-processing (see the tables above) and we can see that the simpler one works better, according with what we have just pointed out.

Furthermore, we apply the simple pre-processing without discarding the biases, we basically skip the point two and one in the preprocessing pipeline. This let us understand how the biases influence the classifier. As we can see, we can increase the accuracy up to 3%.

	min	max	**mean**
accuracy	0,57	0,84	**0,73**

	k-fold									
mean accuracy	0,73	0,74	0,73	0,73	0,70	0,75	0,73	0,74	0,72	0,76

Table 4 and 6: results after having applied a strict pre-processing to article 3.

	min	max	**mean**
accuracy	0,62	0,86	**0,75**

	k-fold									
mean Accuracy	0,75	0,75	0,75	0,76	0,74	0,75	0,75	0,76	0,74	0,75

Table 6 and 7: results after having applied a simple pre-processing to article 3

	min	max	**mean**
accuracy	0,66	0,87	**0,78**

	k-fold									
mean Accuracy	0,79	0,77	0,77	0,77	0,76	0,78	0,76	0,78	0,79	0,78

Table 8 and 9: results after having applied a simple pre-processing to article 3, leaving the biases.

```
CPU times: user 3h 54min 41s, sys: 2min 40s, total: 3h 57min 21s
Wall time: 3h 10min 57s
```

Figure 11: the total running time of article 3, strict pre-processing. This experiment was run on the Server.

```
CPU times: user 10h 4min 35s, sys: 5min 8s, total: 10h 9min 43s
Wall time: 6h 10min 18s
```

Figure 12: the total running time of article 3, simple pre-processing with biases. This experiment was run on the Server.

Model 2

Environment description

The model 2 differs from the first model we presented in the dataset with which we trained the Doc2vec. In this case we intend to feed the Neural Network with as many documents as possible. In order to do that, we merge in one dataset all the sample contained in the violations, train and test20 folder. Moreover, we apply the as we called strict pre-processing so to also reduce the dimension of each document into its essential.

Once the Doc2vec has been trained, we create a vectors dataset from the merging of the train and test20, considering each article separately. We then feed the SVM with this dataset and apply 10-cross validation. Because we shuffle the dataset ten times, we have as output a 10x10 matrix .

Doc2vec

Parameters	Configuration
vector_size	100
min_count	1
epochs	20

SVM

Parameters	Configuration
C	1
Kernel	linear

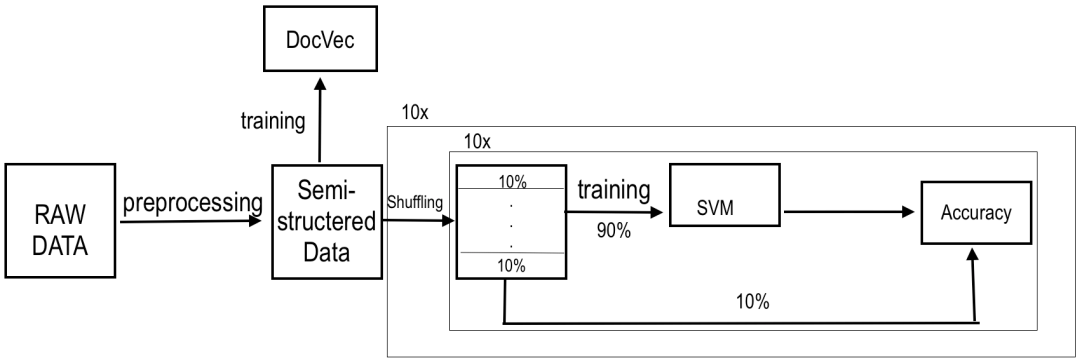


Figure 13: description of model 2 routine

Algorithm 2 Model 2

$data_{all} = applyPreprocessing(rawText_{allArticles})$
$data_N = applyPreprocessing(rawText_{articleN})$
$shuffles \leftarrow 10$
$folds \leftarrow 10$
$doc2vec = trainingD2V(data_{all})$
while $i < shuffles$ **do**
    $shuffling(data_N)$
    while $j < folds$ **do**
        $test, train = split(data_N)$
        $svm = trainingSVM(train, doc2vec)$
        $accuracy[i, j] = test(svm, test)$
    end while
end while

Results

As in the previous experiment, in the table below are shown the intermediate accuracies and the mean over the k-folds.

In this case, we aimed to feed the NN with as many samples as possible, because it is likely to work better. In fact, the results really improved, the mean

accuracy increased to 80%, which is 7 percent more than the previous model. We can see that maximum peak is up to 90%, while the minimum is 70%.
This output is reached without applying any hyper-parametrisation, which we know could lead to increase the mean accuracy percentage.
However, we believe that this results could be affected by data leakage.
The classifier never observes unseen examples, but for the NN we used all the samples in the dataset. More in detail, when we train the Doc2vec model, it is required tagging the documents with labels. We decided to tag each sample with its tag, 0 or 1 respectively if they are violated or not.
In this sense, for getting really trustable results, we should not permit to our model to see samples that we are going to test. This is what we are going to explore in the next experiments.

SVM

Parameters	Configuration
C	1
Kernel	linear

	min	max	**mean**
accuracy	0,70	0,91	**0,81**

	k-fold									
mean Accuracy	0,80	0,80	0,80	0,81	0,81	0,82	0,81	0,80	0,80	0,80

Table 10: results of article 3.

```
CPU times: user 1h 33min 18s, sys: 33.9 s, total: 1h 33min 52s
Wall time: 1h 42min 3s
```

Figure 14: the total running time of article 3. This experiment was run on the MacBook Pro.

Model 3

Environment description

In this experiment our goal is to avoid the data leakage of the previous experiment. In order to do that, we need to exclude the samples we are going to use for testing the accuracy of the classifier.

The pre-processing we apply in this experiment is the simpler one, while the dataset we take into account for training the Doc2vec consists of all the violations samples, train samples and all the test20 samples except the ones for testing the classifier, which is a Support Vector Machine. We test the samples directly without applying any cross validation. Consequently, the Doc2vec model would be trained as many times as the number of articles in the dataset. Thus, we expect a considerable decrease in the performance in terms of computational time.

In addition, we perform a grid search over all the articles, so as to determinate which is the best parameters for embedding the text corpus. In this case we execute a the same routine mentioned above but with just 1 shuffle for each parameter configuration. The reason is simple: it would take too long for getting the results of all the combination of parameters. Therefore, the best result would be the set of parameters that would head to the highest mean accuracy over all the articles. Furthermore, for the same time-consuming reason, we decided to train the Doc2vec model just one time for all the articles for each parameter configuration we are going to test. Hence, all the test20 samples have to be excluded from all the model routine.

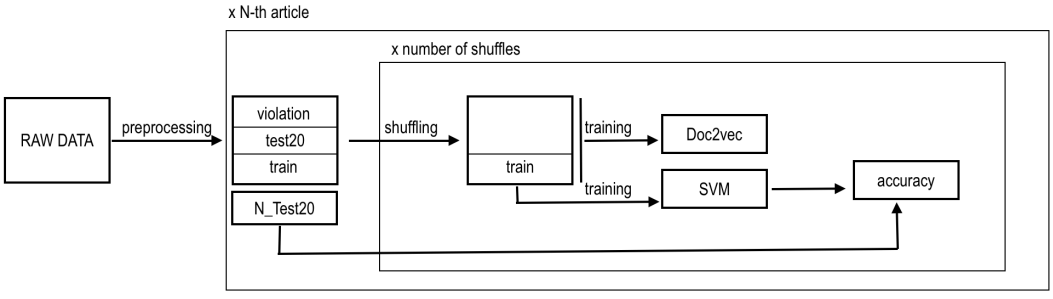


Figure 15: description of model 3 routine

Algorithm 3 Model 3

$data_{violations} = applyPreprocessing(rawText_{violations})$
$data_{train} = applyPreprocessing(rawText_{train})$
$data_{test20} = applyPreprocessing(rawText_{test20})$
$shuffles \leftarrow [1, ..., 10]$
N ::= denotes the N-th article
for *articles* **do**
    $data_{Ntrain} = applyPreprocessing(rawText_{trainArticleN})$
    $data_{Ntest20} = applyPreprocessing(rawText_{test20ArticleN})$
    $data_{D2V} = data_{violations} \cup data_{train} \cup data_{test20 \cap Ntest20}$
    while $i < shuffles$ **do**
        $shuffling(data_{Ntrain})$
        $shuffling(data_{Ntest20})$
        $doc2vec = trainingD2V(data_{D2V})$
        $svm = trainingSVM(data_{Ntrain}, doc2vec)$
        $accuracy[i] = test(svm, data_{Ntest20})$
    end while
end for

Algorithm 4 Model 3, Grid search

$data_{violations} = applyPreprocessing(rawText_{violations})$
$data_{train} = applyPreprocessing(rawText_{train})$
$shuffles \leftarrow 1$
N ::= denotes the N-th article
$data$ =data$_{violations} \cup data_{train}$
for *parameters* **do**
    $doc2vec = trainingD2V(data, parameter)$
    for *articles* **do**
        $data_{Ntrain} = applyPreprocessing(rawText_{trainArticleN})$
        $data_{Ntest20} = applyPreprocessing(rawText_{test20ArticleN})$
        while $i < shuffles$ **do**
            $shuffling(data_{Ntrain})$
            $shuffling(data_{Ntest20})$
            $svm = trainingSVM(data_{Ntrain}, doc2vec)$
            $accuracy_{parameter}[i] = test(svm, data_{Ntest20})$
        end while
    end for
end for

Results

The following table shows the results that we get after run the grid search program. In total, it took 19 days, 5 hours and 12 minutes (as reported in the figure below). The program had been obviously run on the server.

Parameters	Configurations		
vector_size	100	100	100

Parameters	Configurations		
epochs	20	20	20
min_count	1	1	1
window	3	3	3
hs	1	0	0
negative	20	5	5
ns_exponent	1	1	0.75
dm_mean	1	1	0
dbow_words	1	0	1
Total Accuracy across all articles	**0.7558**	**0.7484**	**0.7482**

Satisfied by the result we obtained, we run the algorithm of the model 3 with best set of parameters found. With this model, without any biases and data leakage, we achieve 0,76% of total mean accuracy across all the articles.

The main purpose of this experiment is finding a set of parameters that allows us to achieve good performances across all the articles. We are aware that testing on the test20 can not be the final result, however it is a good assessment of our model and it helps us to understand what parameters influence most the final accuracy. The reason why we decided to test just on the test20 is simply because a grid search takes long time, which we do not have.

In addition, we also run the grid search program with the leaving all the biases so to have a clear idea of what could be our upper bound in terms of the maximum accuracy we can achieve. The table below shows the results, after 18 days, 10 hours and 55 minutes.

Parameters	Configurations		
vector_size	500	500	100
epochs	20	20	20
min_count	1	1	100

Parameters	Configurations		
window	3	3	10
hs	0	0	1
negative	20	5	20
ns_exponent	0.75	1	0.75
dm_mean	1	0	0
dbow_words	1	0	0
Total Accuracy across all articles	**0.7880**	**0.7936**	**0.7932**

Model 4

Environment description

The model we are going to introduce is similar to the previous one except for the classifier. In this case, we want to explore a difference strategy by using the well known K-NN methods. We are going to use a metric known as cosine similarity for measuring the nearest neighbours.

The pre-processing we intended to use is the strict one, however, we also are going to do some other experiments with the simplest one.

For the same reasons observed in the previous model, we must exclude some the samples we are going to test, thus, the Doc2vec is trained with all violations and train samples. In addition, we use all the articles of test20 expect the ones that refer to the article we are going to test. So we directly test without applying any cross validation.

A crucial aspect, when we want to apply the K-NN technique, is how each sample is tagged. This is done through the TaggedDocument function. In this case, we pass a tuple which consists of ID and belonging class. Given a new instance to classify, we aim to find the most similar sample among all the documents passed during the training phase. Once found the most similar one, we classify the new instance with same label. If we just tagged each document with the belonging class, we would have in the end the similarity between the instance and all the documents.

Algorithm 6 Model 4, Grid search

$data_{violations} = applyPreprocessing(rawText_{violations})$
$data_{train} = applyPreprocessing(rawText_{train})$
$shuffles \leftarrow 1$
N ::= denotes the N-th article
$data$ =data$_{violations} \cup data_{train}$
for *parameters* **do**
    $doc2vec = trainingD2V(data, parameter)$
    for *articles* **do**
        $data_{Ntrain} = applyPreprocessing(rawText_{trainArticleN})$
        $data_{Ntest20} = applyPreprocessing(rawText_{test20ArticleN})$
        while $i < shuffles$ **do**
            $shuffling(data_{Ntrain})$
            $shuffling(data_{Ntest20})$
            $predictions = get_most_similar(doc2vec, data_{Ntest20})$
            $accuracy[i] = test(predictions, data_{Ntest20})$
        end while
    end for
end for

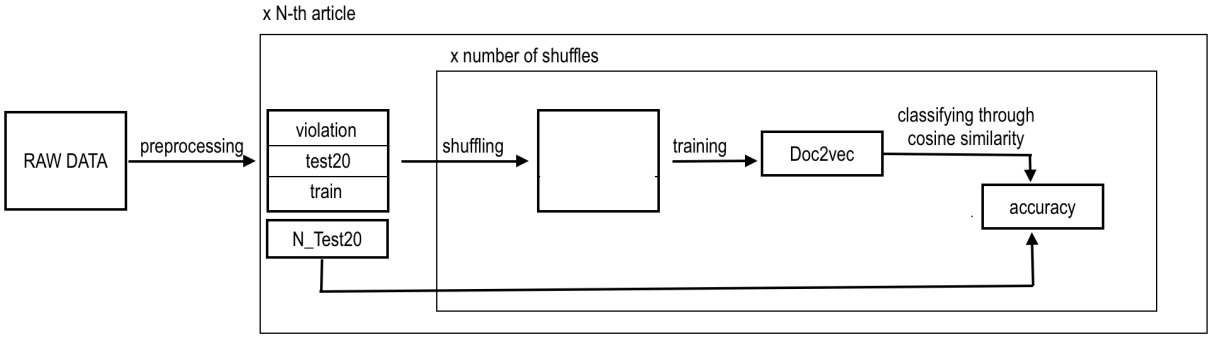


Figure 16: description of model 4 routine

Results

In conclusion, using techniques such as K-NN based on cosine similarity does not seem leading to better results. In the datasets, the case law sometimes are really similar to each other even if they belong to different classes. Thus, we can not rely on this approach. We can see from the Jupyter notebook[12] how the model works well by identify the most similar document with 90% of similarity. The results in table below are taken by using a simple preprocessing.

Parameters	Configurations		
vector_size	500	100	100

[12]https://github.com/daminienrico/NLP_judicial_text_classification/blob/master/MODELLO4.ipynb

Parameters	Configurations		
window	20	20	20
min_count	1	1	1
epochs	3	3	10
hs	0	1	1
negative	5	20	20
ns_exponent	1	1	1
dm_mean	0	1	1
dbow_words	0	1	1
Total Accuracy	**0.7030**	**0.7031**	**0.7028**

Model 5

Environment description

The model we are going to present has the following routine: for each article, we perform a 10-cross validation on the merging of the test20 and train dataset, thus, the SVM has been trained on 9-folds and tested on the remain fold. The Doc2Vec has been trained, not only on the 9-folds, but also on the violations of all the articles and on the merging of the test20 and train of the other articles. In this way, we are sure that there is no data leakage. Moreover, this is the maximum we can do in terms of feeding the Doc2Vec with as many samples as we possible.

In this model we use a simple preprocessing and a Linear support vector machine. The parameters used for the Doc2Vec and for SVM are reported in the figure below. This set of parameters are based on knowledge matured from the previous experiments, in which we performed a grid search.

Furthermore, we expected that the computational time increases with respect to the previous models, because we need to trained the NN at every single step of the cross-validation.

Parameters	Configurations
vector_size	100
epochs	20
min_count	1
window	3
hs	1
negative	20
ns_exponent	1
dm_mean	1
dbow_words	1

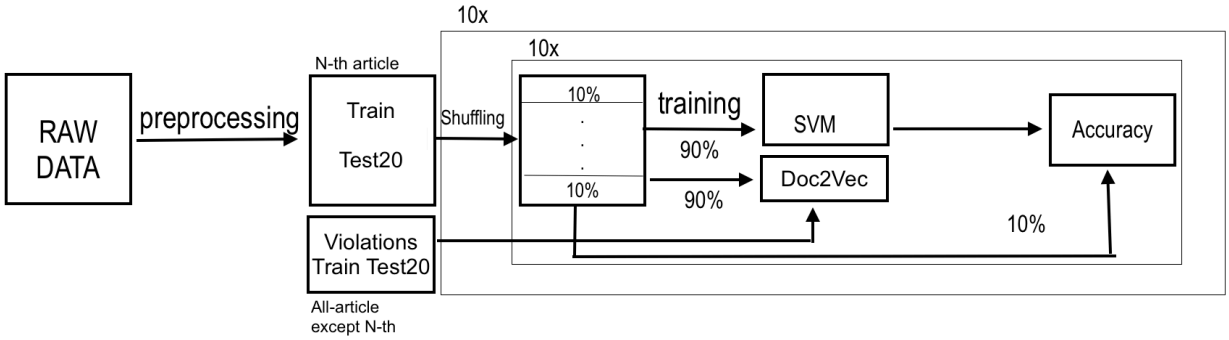


Figure 17: description of model 5 routine

Results

As expected, the computational time is very high: for each computation of the model Doc2vec it took on average 20 minutes, which means that the program run for $20[min] \times 100[iter] \times 9[dataset] = 18000[min] = 12{,}5[day]$. Since in some dataset the ninth part consists of few examples, the accuracy of the part could be very high or very low, however this is a normal results. We can finally presents a model which is reliable and robust. The results are shown below:

	Articles									
	2	**3**	**5**	**6**	**8**	**10**	**11**	**13**	**14**	**mean**
Accuracy	0,705	0,760	0,710	0,661	0,680	0,610	0,75	0,784	0,754	**0,713**

The total accuracy is decreased by 3% respect to the previous models, but this is not surprising since the latter was tested just with one shuffle and on the test20 samples, therefore this results are way more reliable.

Classifiers comparison

As we expected, since the data are intrinsically not well separated it was not possible to increase the accuracy significantly. We could achieve better results by having a specific preprocessing and specific parameters for each article, however we must take into account the generality of the models. Tuning the method parameters on each single article leads to a strong overfitting. Especially since the dataset size is particularly small. We aim to build a model which is more robust, reliable and realistic rather than a model that is not general, even if it works well on a single dataset. In this way we can draw conclusions that have higher probability of being right in other juridical datasets.

We believe that the last model presented is the most reliable in terms of correctness and best practices.
The results we achieved are the follow:

	Articles									
Accuracy	**2**	**3**	**5**	**6**	**8**	**10**	**11**	**13**	**14**	**Mean**
Model5	0,705	0,760	0,710	0,661	0,680	0,610	0,751	0,784	0,754	**0,713**
Medvedeva	0,714	0,78	0,695	0,773	0,649	0,651	0,728	0,797	0,770	**0,728**

We can finally say that the accuracy is not as high as the target we aimed to reach, however the model has many positive aspects to take into account in the final assessment.

Bayesian analysis

In order to compare our model with the state of the art, we used the Bayesian analysis.
Firstly, we compare the two models on the same datasets with the correlated t-test. For example for article 8 we have the follow results:

$$P(Medvedeva > Model5) = 0.0041,\ \ P(rope) = 0.0196,\ \ P(Model5 > Medvedeva) = 0.9762$$

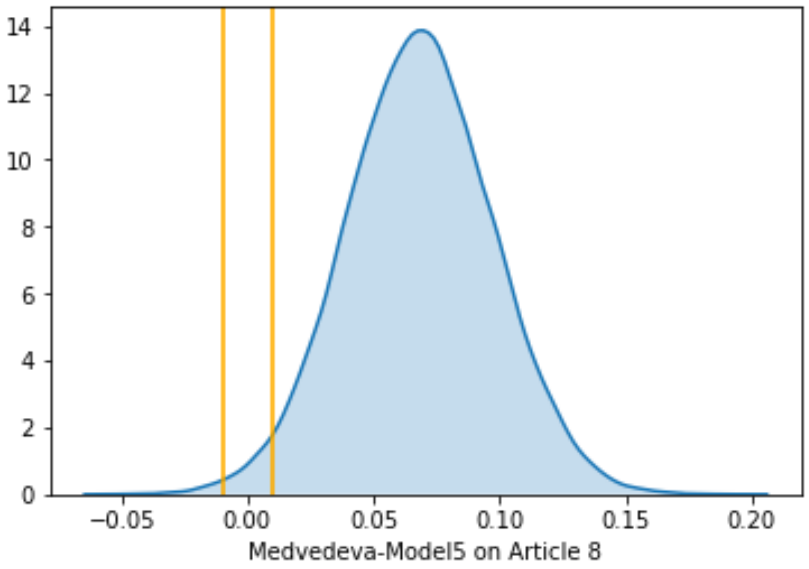


Figure 17: the posterior distribution Medvedeva-Model5 models on the dataset of the article 8.

In this case the our model performs better than the state of the art, however, if we try the test on another case, such as the article 3, we have opposite results:

$$P(Medvedeva > Model5) = 0.5314,\ \ P(rope) = 0.2965,\ \ P(Model5 > Medvedeva) = 0.1719$$

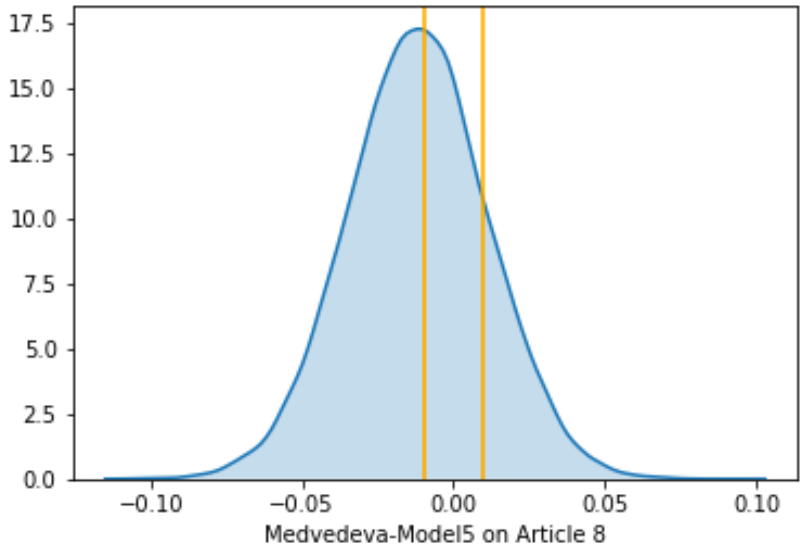


Figure 18: the posterior distribution Medvedeva-Model5 models on the dataset of the article 3.

In order to have a better comparison we use another Bayesian comparison test (described in a previous chapter) so to be able to compare the same two models on more than one datasets. The results are the follow:

$$P(Medvedeva > Model5) = 0.768,\ \ P(rope) = 0.001,\ \ P(Model5 > Medvedeva) = 0.231$$

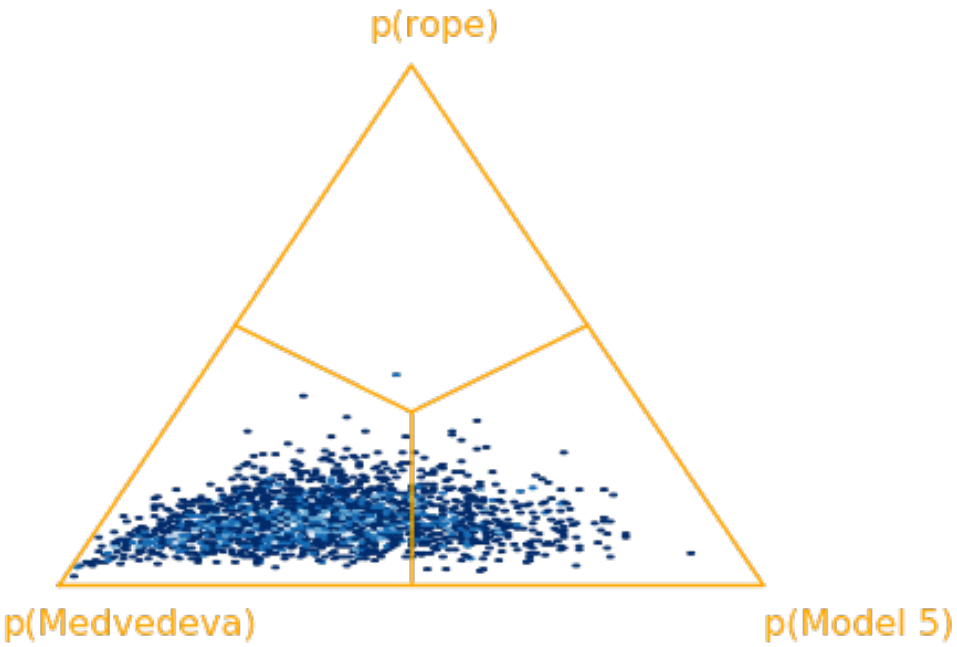


Figure 19: the posterior distribution Medvedeva-Model5 models on all the datasets.

It can be seen that the posterior mass is mainly in the region in favour of the State of the Art model, with just a 23% in favour of the model 5. From the posterior we have also an idea of the magnitude of the uncertainty and the "stability" of our inference.

From this comparison, a really important consideration can be done: we can see that the probability that the two model are in the practical equivalence is very low (0.1%). So the two models can not be considered practically equivalent, thus they are completely different. This means that there are some datasets in which the model 5 works better than the state of the art model.

In conclusion, we can say that, since $P(rope)$ is really low, the model 5 can not be ignored for ensemble methods or for classification tasks on new datasets.

Conclusion

Herein we presented a new NPL approach to juridical decision making which is based on two main parts: first, embedding the text documents by employing a more advanced technique that take advantage of Neural Networks and, second, using a classifier that works well in a high-dimensional space, such as a Support Vector Machine. We tried many different approaches to the problem, for example by considering the datasets separately or by changing the text preprocessing. We decided to explain here a subset of the models we tried, the ones that we believe are significant in terms of methodology and results. We gave particular attention to avoid data leakage and overfitting, issues that are present in the previous work.

The final model presented in this work is promising in terms of robustness, reliability and accuracy. Its performances are close to the state of the art in juridical decision.
Despite the fact that our model have a slightly lower average accuracy, its value can be recognise in the Bayesian comparison in the previous chapter. The probability of the two models being equal, $P(rope)$, is almost 0, while the probability of the new approach to beat of beating the state of the art are significant. This means that the two approaches exploit different structure of the datasets, making them behaving in significantly different ways. In many articles our model can extrapolate the significant informations in a more performant way (e.i. article 8). This diversity is particularly valuable in building ensemble classifiers [23]. Moreover, it means that during the deployment of similar applications on new previously unknown dataset our method should be evaluated as well, since there is the probability that outperform the previous approaches significantly.

We believe that our model has strong margin of improvement, its performances could be further enhanced with a larger dataset. In fact, published works using NN based techniques generally use corpuses of tens-of-thousands to millions of documents. We can have at most 12 thousands and they are not even class balanced. Due to the very time-consuming computations of the neural network

trainings, it was not possible to further extend the grid search and have a stronger hyper-parametrisation.

References

1. Dyevre, A., The promise and pitfalls of automated text-scaling techniques for the analysis of judicial opinions. 2015.
2. Garoupa, N., M. Gili, and F. Gómez-Pomar, Political Influence and Career Judges: An Empirical Analysis of Administrative Review by the Spanish Supreme Court. Journal of Empirical Legal Studies, 2012. **9**(4): p. 795-826.
3. Sulea, O.-M., et al., Exploring the Use of Text Classification in the Legal Domain. arXiv preprint arXiv:1710.09306, 2017.
4. Vols, M., P. Tassenaar, and J. Jacobs, Anti-social behaviour and European protection against eviction. International Journal of Law in the Built Environment, 2015. **7**(2): p. 148-161.
5. De Jaeger, T., Gerechtelijke achterstand: de piñata van de wetgever. NJW, 2017: p. 290-307.
6. Katz, D.M., M.J. Bommarito II, and J. Blackman, A general approach for predicting the behavior of the Supreme Court of the United States. PloS one, 2017. **12**(4): p. e0174698.
7. Lindholm, J. and M. Derlén, The court of justice and the Ankara agreement: Exploring the empirical approach. Europarättslig tidskrift, 2012(3): p. 462-481.
8. Derlén, M. and J. Lindholm, Goodbye van G end en L oos, Hello B osman? Using Network Analysis to Measure the Importance of Individual CJEU Judgments. European Law Journal, 2014. **20**(5): p. 667-687.
9. Holá, B., C. Bijleveld, and A. Smeulers, Consistency of international sentencing: ICTY and ICTR case study. European journal of criminology, 2012. **9**(5): p. 539-552.
10. Tarissan, F. and R. Nollez-Goldbach. Temporal properties of legal decision networks: a case study from the International Criminal Court. in 28th International Conference on Legal Knowledge and Information Systems (JURIX'2015). 2015.
11. Fürnkranz, J., et al., Multilabel classification via calibrated label ranking. Machine learning, 2008. **73**(2): p. 133-153.

12. Christensen, M.L., H.P. Olsen, and F. Tarissan. Identification of Case Content with Quantitative Network Analysis: An Example from the ECtHR. in JURIX. 2016.
13. Panagis, Y., M.L. Christensen, and U. Sadl. On Top of Topics: Leveraging Topic Modeling to Study the Dynamic Case-Law of International Courts. in JURIX. 2016.
14. Le, Q. and T. Mikolov. Distributed representations of sentences and documents. in International Conference on Machine Learning. 2014.
15. Wickramasinghe, N. *A convolutional net for medical records*. 2017. Engineering in Medicine and Biology Society.
16. Aletras, N., et al., Predicting judicial decisions of the European Court of Human Rights: A natural language processing perspective. PeerJ Computer Science, 2016. **2**: p. e93.
17. Medvedeva, M., M. Vols, and M. Wieling, Judicial Decisions of the European Court of Human Rights: Looking into the Crystal Ball. C onference on Empirical Legal Studies–Europe, 2018.
18. Dai, A.M., C. Olah, and Q.V. Le, Document embedding with paragraph vectors. arXiv preprint arXiv:1507.07998, 2015.
19. Demšar, J. On the appropriateness of statistical tests in machine learning. in Workshop on Evaluation Methods for Machine Learning in conjunction with ICML. 2008.
20. Benavoli, A., et al., Time for a change: a tutorial for comparing multiple classifiers through Bayesian analysis. The Journal of Machine Learning Research, 2017. **18**(1): p. 2653-2688.
21. Corani, G. and A. Benavoli, A Bayesian approach for comparing cross-validaed algorihms on muliple daa ses. Machine Learning, 2015. **100**(2-3): p. 285-304.
22. Benavoli, A., et al. A Bayesian Wilcoxon signed-rank test based on the Dirichlet process. in International conference on machine learning. 2014.
23. Kuncheva, L.I., Combining pattern classifiers: methods and algorithms. 2004: John Wiley & Sons.
24. Visentin, O'Sullivan, Predicting Judicial Decisions: A statistically Rigorous Approach and an Ensemble Classifier. 2018.