Neural Legal Outcome Prediction with Partial Least Squares Compression

Charles Condevaux

stats

MDPI

Article

Neural Legal Outcome Prediction with Partial Least Squares Compression

Charles Condevaux

CHROME, University of Nîmes, 30021 Nîmes, France; charles.condevaux@unimes.fr

Received: 31 July 2020; Accepted: 9 September 2020; Published: 18 September 2020

Abstract: Predicting the outcome of a case from a set of factual data is a common goal in legal knowledge discovery. In practice, solving this task is most of the time difficult due to the scarcity of labeled datasets. Additionally, processing long documents often leads to sparse data, which adds another layer of complexity. This paper presents a study focused on the french decisions of the European Court of Human Rights (ECtHR) for which we build various classification tasks. These tasks consist first of all in the prediction of the potential violation of an article of the convention, using extracted facts. A multiclass problem is also created, with the objective of determining whether an article is relevant to plead given some circumstances. We solve these tasks by comparing simple linear models to an attention-based neural network. We also take advantage of a modified partial least squares algorithm that we integrate in the aforementioned models, capable of effectively dealing with classification problems and scale with sparse inputs coming from natural language tasks.

Keywords: NLP; text classification; legal knowledge; transformer; PLS regression

1. Introduction

With the emergence of deep learning algorithms (neural networks), significant works have been done related to the legal domain and predictive justice. While classification tasks are important topics [1], some applications have been developed around, such as information extraction [2], legal norms classification [3] or topic classification [4]. Predicting the outcome of a case from factual data is a major topic in this literature, as it can offer great benefits to practitioners and can also be useful for citizens. If a lawyer or a judge can estimate the likelihood of an outcome, it can translate into greater efficiency, better access to justice, and greater fairness assuming this tool plays a supporting role.

The task of predicting the outcome has received much attention recently. In specific situations, high accuracies have been achieved [5] mainly using deep neural networks [6,7] which are able to significantly outperform standard and widely used algorithms like logistic regression, tree based models or SVM [8]. Some attention has recently been paid to the European Court of Human Rights (ECtHR) for a number of reasons. One of them is the accessibility of court decisions, another one has to do with the ability to create an automatic annotation process of these judgments. Solving tasks related to the legal domain requires experts to manually put labels on documents, it is often expensive and time consuming, leading to scarce data. ECtHR decisions possess a specific structure which can be exploited using simple regex, making them a good candidat for automatic legal analysis and forecasting. As the literature focuses on english decisions, some performances comparisons have already been made around standard classification algorithms [9,10] and deep learning approaches using state-of-art models [11,12] which require high computation cost compared to linear classifiers.

In the paper, we compare several algorithms on french ECtHR decisions on different binary tasks where the target is the outcome (violation of a given article). Another task is built as a multiclass problem and aims to find which article has potentially been violated. We do not voluntarily perform meaningful processing outside of tokenization and extraction of facts and circumstances in order to

deal with sparse inputs. Sparse inputs are addressed using a modified and efficient PLS algorithm to improve overall performances, stabilize results and ease convergence. Finally we take advantage of a pretrained word embedding and the transformer architecture [13] to build a neural network approach.

2. Data

Our main task is to predict the violation of article of the convention given a set of facts as inputs. We use the published ECtHR judgments (see (https://hudoc.echr.coe.int/)) in french as our main data source. The choice of the language has two reasons: one can pretrain a large model on french legal documents and state of the art english embeddings are not able to process long sequences without huge memory consumption.

2.1. Structure

In theory, judgments are structured with titles and paragraphs for ease of readability since these decisions typically contain hundreds of sentences. A document can be divided into four parts. The first is called the procedure and provides general information about the procedure followed before the Court. This section is the smallest most of the time and does only lists past results from a local court. Note that in the ECtHR, the parties are individuals against a state.

The second section is the facts section which is the main input of our model. It provides some background about the case itself and everything unrelated to legal arguments (i.e., articles from the European Convention on Human Rights ECHR). This part is generally divided into two subsections: the circumstances of the case and the relevant laws. The first relates the factual background which is a crucial element for a legal document. This section is formulated by the Court itself but we consider that it provides a reasonable representation of facts. This section is also the most heterogeneous because its size and vocabulary may change a lot even for two similar cases. The part on relevant laws adds information on domestic laws and legal elements, with the exception of articles of the ECHR.

The third section is the law section and does rely on legal arguments to consider the merit of the case. To pronounce an outcome, the Court must justify its decision using rules and principles selected taking into account an alleged violation of an ECHR article and arguments provided by the parties.

Finally the last section is the results of the case. It enumerates all potential violations of ECHR articles and whether they actually took place. The overall structure is presented Figure 1.

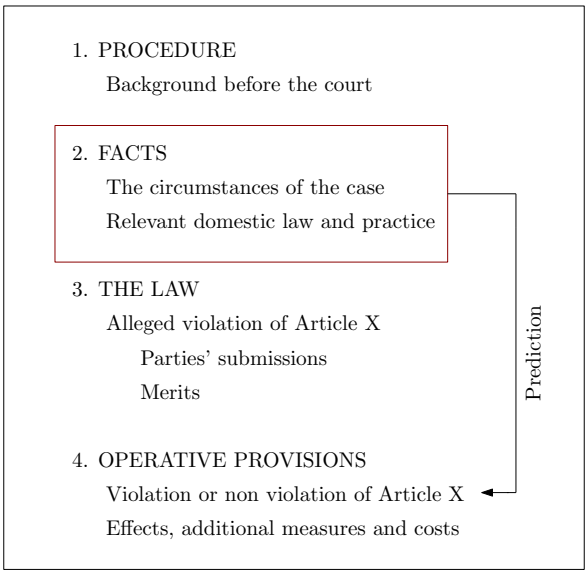

Figure 1. Structure of a court decision.

The majority of documents follow this structure, in both french and english versions. Note that the available decisions are not always translated, resulting in differences between the datasets coming from the two languages. Related papers [9,10] which focus on english already have different inputs because the extraction process is tricky and slightly differs between the two studies, new decisions are also periodically added to the database which may explain differences in size.

2.2. Extraction

Extraction is simple in theory because we can easily extract headings and paragraphs. In practice we observe a certain variability between documents which makes automatic extraction more challenging. Some titles may change or some section may move or disappear for no specific reason. For example, we sometimes find the procedure inside the fact section, or the fact section does exist but we cannot extract its subsections which should feed our models. The second challenge is removing ambiguous outcomes. As our main task is to find out whether a given article of the convention has been violated or not, we need to filter the decisions which have different outcomes for the same article. Everything is done using plain regex to first detect headings and paragraphs, check their validity, extract all the outcomes and compare them and their associated articles. The overall extraction procedure follows seven steps:

- We scrape all french decisions available.
- We check using multiple regex if the four sections titles (and some variations) can be found inside the decision to make sure everything is well located and available.
- We check if both facts subsections are well defined using regex also, namely circumstances and relevant domestic laws.
- We process the outcome section (operative provision) of the case by extracting sentences containing words “violation” and “article”.
- We process all these sentences by listing all named articles that belong to the European convention using a regex.
- For each retrieved article, we infer the outcome (violation or non-violation). This is simple given that these sentences always have the same structure (e.g., “Holds that there has been a violation of Article 6 of the Convention.” in english).
- We remove all decisions where a given article has been violated and non violated at the same time. This happens if there are multiple claims based on the same article in a given decision.

After extraction, we only keep decisions related to six articles which are also the most frequent ones to ensure a sufficient number of observations: article 3 (torture or inhuman and degrading treatment), article 5 (right to liberty and security), article 6 (right to a fair trial), article 8 (respect for private and family life), article 10 (freedom of expression) and article 13 (access to justice).

2.3. Datasets

For each selected article, we build a dataset consisting of valid decisions and their associated binary label (violation or non-violation of the article). The extraction process yields Table 1.

Table 1. Datasets sizes and labels distributions.

	Art. 3	**Art. 5**	**Art. 6**	**Art. 8**	**Art. 10**	**Art. 13**	**Stack**
Initial size	540	561	3704	571	476	541	6393
% Violation	0.832	0.892	0.937	0.776	0.853	0.887	-
Balanced size	182	122	468	256	140	122	2327
% Violation	0.500	0.500	0.500	0.500	0.500	0.500	-

As shown, the outcome is highly unbalanced and favors most of the time the claimant against the respondent state. It is however influenced by our extraction process as we have removed ambiguous

cases where we can find violation and non violation of a given article at the same time. For each selected set of judgments, we extract the circumstances of the case and the entire fact section as we only want to predict a fact-based outcome. The stack dataset is used to test a multiclass problem by trying to find, from a set of facts, whether a given item is relevant to the case.

To more accurately measure model performances and follow a similar approach as described in the related papers [9,10], we balance the datasets to ensure the same number of violations and non-violations. Note that circumstances and facts can last over a hundred sentences in few situations. They are also closely related: the facts are actually circumstances plus domestic legal arguments which are very heterogeneous as the applicants come from different countries.

We also ensure that the stack dataset is a single label problem by removing cases where two or more articles are relevant, however is it not balanced. Each article representing 13%, 10%, 35%, 22%, 11% and 9% of the stack respectively.

In order to solve these tasks, we compare various models and a transformer based neural network architecture. We also test a modified and optimized partial least squares algorithm (PLS) suitable for classification tasks and sparse input as we rely on it to skip any preprocessing outside of tokenization.

3. Models

3.1. Embeddings and Attention

3.1.1. Word Embeddings

Word embeddings are a fundamental building block for solving natural language tasks as they provide a way to represent textual data in a low dimensional space. Given a sentence made up of words, subwords or characters, a word embedding maps a discrete representation into a vector of real values. For a sequence of tokens $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t] \in \mathbb{R}^{t \times d}$, where t is the length of the sequence, d the vocabulary size and x_i a one-hot vector, a linear pretrained word embedding provides a weight matrix $\mathbf{W} \in \mathbb{R}^{d \times d'}$ with $d' \ll d$ such that it can project the sequence into a lower dimensional space. This embedding matrix can either be estimated in an unsupervised fashion using compression models like PCA or SVD, or through a self supervised approach with a neural network. In Word2Vec [14], the best-known method, the matrix is computed by solving a large classification task: finding the correct word given its context. Although this model is simple, scalable and efficient to estimate with backpropagation, it lacks certain properties. Firstly, it cannot handle out of vocabulary tokens, the tokens also have a fixed representation, thus it is difficult to disambiguate a word which may have a different meaning depending the sentence. Secondly, the model is fully linear which limits its capabilities, it also uses a small context window which cannot handle long term dependencies. Word embeddings have been improved several times by replacing words with sub-words (FastText [15]) and combining them to handle unseen tokens. Taking advantage of the contextual and the sequential aspect of sentences has also been improved by adding non linear connections and recurrent layers [16], namely LSTM [17], which are able to process and store information in memory cells. Finally, significant progress has been made with the emergence of transformer [13] based word embeddings like BERT [18], which relies on a masked objective and a stack of self attention modules: each token in a given sentence is weighted by all other elements, allowing fast convergence and strong performances across a wide range of tasks.

3.1.2. Attention Mechanism

While recurrent neural networks like LSTM are able to model long term dependencies, they require a sequential processing of the input, which is expensive and difficult to optimize for long sequences. The attention mechanism partially solves this problem by computing all dependencies in parallel using a score (context) matrix. This context matrix stores information between all pairs of tokens taken from

two inputs. These scores are squashed using a softmax function to get a weighting matrix representing a probability distribution (Figure 2).

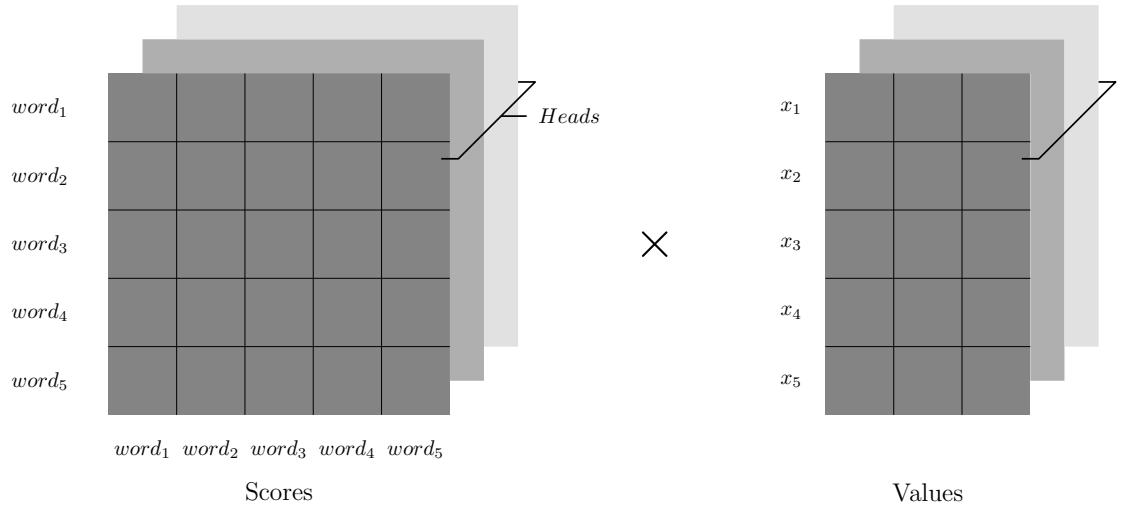

Figure 2. Self attention mechanism on word level.

Attention can be computed between different sources, generally called query, key and value, respectively $\mathbf{Q} \in \mathbb{R}^{t \times d}$, $\mathbf{K} \in \mathbb{R}^{t \times d}$ and $\mathbf{V} \in \mathbb{R}^{t \times d}$ where t is the sequence length of the source, d the embedding dimension and *sigma* a softmax function:

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \sigma\Big(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\Big)\mathbf{V} \tag{1}$$

Using attention once for each pair of sentences is not enough to fully capture dependencies. In practice the computation is done several times in parallel by dividing the input into smaller pieces called heads. The embedding dimension can be decomposed into h heads of size d_h such that $d = h \times d_h$. In the self attention setup, Q, K and V usually have the same shape and are distinct linear projection from a same input $X \in \mathbb{R}^{t \times d}$. Given $W_q, W_k, W_v \in \mathbb{R}^{d \times d_m}$ learned matrices with d_m the hidden dimension of the projection subspace, self attention is computed head wise as:

$$SelfAttention(\mathbf{X}) = \sigma\Big(\frac{\mathbf{X}\mathbf{W}_q(\mathbf{X}\mathbf{W}_k)^T}{\sqrt{d}}\Big)\mathbf{X}\mathbf{W}_v \tag{2}$$

In practice, these products are also done batch wise on a set of examples at the same time. In order to stack multiples sentences of different length in a single tensor, they are artificially padded with zeros.

As computing the score matrix is memory intensive as its size depends on t^2, we use a slightly modified version which compute sequentially two score matrices in order to reduce the overall computational cost [19]:

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \sigma_{row}(\mathbf{Q})\Big(\sigma_{col}(\mathbf{K})^T\mathbf{V}\Big) \tag{3}$$

Finally we build a transformer architecture by stacking the attention mechanism to various feed forward layers with a skip connection [20] and a normalization:

$$\tilde{\mathbf{X}} = Norm(\mathbf{X} + Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V})) \tag{4}$$

The transformer layer is represented in Figure 3:

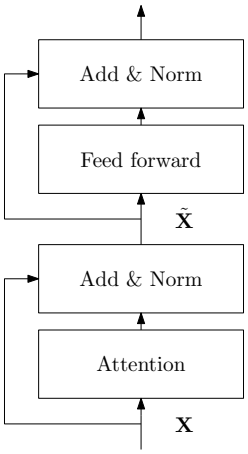

Figure 3. A transformer layer.

3.2. The Reduction Algorithm

As we deal with sparse inputs, we want to rely on a dimensional reduction tool to densify our matrix. Note that in most natural language tasks, the number of features can be extremely high due to the size of the vocabulary. One way to solve this problem in a supervised manner is to rely on the PLS algorithm and its NIPALS variant [21] which is known to provide independent, very informative components and is fairly efficient to compute. However the algorithm has few caveats: it is built for regression and sometimes underperforms for classification problems, it has to loop over the target columns which is slow for multitargets tasks, it does rely on an iterative process which can cause some variations during the training phase. Many alternatives has been proposed such as the logit PLS [22] or the PLS-DA [23] but they can be expensive to compute for large inputs, especially the first one. We propose an alternative algorithm which relies on a point biserial covariance.

3.2.1. Point Biserial Covariance

Let $\mathbf{X} \in \mathbb{R}^{n \times m}$, a real matrix of regressors $\mathbf{x}_j$ $(j = 1, \ldots, m)$ where $x_{(j,i)}$ denotes the ith observation $(i = 1, \ldots, n)$ of the jth variable and $\mathbf{y} \in \{0, 1\}^n$ a target binary vector. All regressors are assumed to be centered and scaled to unit variance. The point-biserial $r_{pb} \in [-1, 1]$ does compute a correlation between a continuous and a dummy variable and is defined as :

$$r_{pb}(\mathbf{x}_j, \mathbf{y}) = \frac{(\bar{\mathbf{x}}_j^{(1)} - \bar{\mathbf{x}}_j^{(0)})}{s(\mathbf{x}_j)} \sqrt{\frac{n^{(1)} n^{(0)}}{n^2}} \tag{5}$$

where $n^{(1)}$ and $n^{(0)}$ denote each class sample size, s the standard deviation and $\bar{\mathbf{x}}_j^{(0,1)}$ is the jth variable conditional mean which can be efficiently computed as :

$$\begin{aligned} \bar{\mathbf{x}}_j^{(1)} &= \frac{1}{n^{(1)}} \mathbf{x}_j^\top \mathbf{y} \\ \bar{\mathbf{x}}_j^{(0)} &= \frac{1}{n^{(0)}} \mathbf{x}_j^\top (\mathbf{1}_n - \mathbf{y}) \end{aligned} \tag{6}$$

Point-biserial covariance is defined as :

$$\begin{aligned} cov_{pb}(\mathbf{x}_j, \mathbf{y}) &= (\bar{\mathbf{x}}_j^{(1)} - \bar{\mathbf{x}}_j^{(0)}) \frac{n^{(1)} n^{(0)}}{n^2} \\ &= \frac{1}{n^2} \mathbf{x}_j^\top \left(n^{(0)} \mathbf{y} - n^{(1)} (\mathbf{1}_n - \mathbf{y}) \right) \end{aligned} \tag{7}$$

More generally, the point biserial covariance matrix is similar to a vanille covariance matrix and can be efficiently computed as:

$$cov_{pb}(\mathbf{X}, \mathbf{y}) = \mathbf{X}^\top \tilde{\mathbf{y}} \tag{8}$$

Assuming $\tilde{\mathbf{y}}$ comes from:

$$\tilde{\mathbf{y}} = \frac{1}{n^2}\Big(n^{(0)}\mathbf{y} - n^{(1)}(\mathbf{1}_n - \mathbf{y})\Big) \tag{9}$$

This formula can be extended to a larger binary matrix $\mathbf{Y} \in \{0,1\}^{n\times k}$ which can be a target matrix for a multiclass or a multilabel classification task. In these conditions, we define $\mathbf{n}^{(0)}, \mathbf{n}^{(1)} \in \mathbb{N}^{1\times k}$ for which $\mathbf{n}_i^{(1)}$ $(i = 1, \ldots, k)$ denotes the number of individuals with value 1 in column i of matrix $\mathbf{Y}$. Thus, with $\odot$ denoting a row wise product the computation of $\tilde{\mathbf{Y}}$ is done as follow:

$$\tilde{\mathbf{Y}} = \frac{1}{n^2}\Big(\mathbf{n}^{(0)} \odot \mathbf{Y} - \mathbf{n}^{(1)} \odot (\mathbf{1}_{n\times k} - \mathbf{Y})\Big) \tag{10}$$

3.2.2. Binary Algorithm

The algorithm consists of computing h orthogonal latent variables $\mathbf{t}_1, \ldots, \mathbf{t}_h \in \mathbb{R}^{n\times 1}$ purged from multicollinearity and able to explain $\mathbf{y}$'s binary class behavior. The first step consists of finding a weight vector $\mathbf{w}_1 \in \mathbb{R}^{m\times 1}$ in order to compute the first component $\mathbf{t_1}$:

$$\mathbf{t}_1 = \mathbf{X}\mathbf{w}_1$$

The weight vector is found by maximizing the biserial covariance between $\mathbf{t}_1$ and $\mathbf{y}$ in a similar manner as in the vanilla PLS algorithm:

$$\begin{array}{ll} \max\limits_{\mathbf{w}_1} & cov_{pb}(\mathbf{X}\mathbf{w}_1, \mathbf{y}) \\ \text{s.t.} & \mathbf{w}_1^\top \mathbf{w}_1 = 1 \end{array} \tag{11}$$

The solution to this program is straightforward, assuming $\mathbf{C} = \mathbf{X}^\top \tilde{\mathbf{y}}$:

$$\mathbf{w}_1^* = \frac{\mathbf{C}}{\|\mathbf{C}\|_2} \tag{12}$$

To ensure the orthogonality between components, each variable $\mathbf{x}_j$ from $\mathbf{X}$ is deflated by running a simple linear regression and extracting its residuals for the next step. For $j = 1$ to m:

$$\mathbf{x}_j = \mathbf{t}_1 \beta_j + \tilde{\mathbf{x}}_j \tag{13}$$

In practice, the deflation is done in parallel by solving all linear regressions at the same time using the OLS estimator $\boldsymbol{\beta} \in \mathbb{R}^{1\times m}$ with a regularization parameter $\gamma \geq 0$:

$$\begin{aligned} \boldsymbol{\beta} &= (\mathbf{t}_1^\top \mathbf{t}_1 + \gamma)^{-1} \mathbf{t}_1^\top \mathbf{X} \\ &= \frac{\mathbf{t}_1^\top \mathbf{X}}{\|\mathbf{t}_1\|_2^2 + \gamma} \end{aligned} \tag{14}$$

More generally, the input matrix $\tilde{\mathbf{X}}_d$ at step $d \in [1, \ldots, h]$, with $\tilde{\mathbf{X}}_0 = \mathbf{X}$, is equal to:

$$\begin{aligned} \tilde{\mathbf{X}}_d &= \tilde{\mathbf{X}}_{d-1} - \tilde{\mathbf{t}}_d \boldsymbol{\beta} \\ &= \tilde{\mathbf{X}}_{d-1}(\mathbf{I}_m - \mathbf{w}_d \boldsymbol{\beta}_d) \end{aligned} \tag{15}$$

The maximization and the computation of a new component is done again using the deflated input matrix. The same process is followed until we reach the require number of latent variables. Note that for step $d \geq 2$, the projection weight vector $\mathbf{w_d}$ does project the deflated matrix and not the

original input. By unrolling $\tilde{\mathbf{X}}_d$, we can extract the projection matrix $\mathbf{W}^* = [\mathbf{w}_1^*, \ldots, \mathbf{w}_h^*] \in \mathbb{R}^{m \times h}$ such that the component matrix $\mathbf{T} = [\mathbf{t}_1, \ldots, \mathbf{t}_h] \in \mathbb{R}^{n \times h}$:

$$\mathbf{T} = \mathbf{X}\mathbf{W}^* \tag{16}$$

where $\mathbf{w}_1^* = \mathbf{w}_1$ and for $d \geq 2$:

$$\mathbf{w}_d^* = \prod_{j=1}^{d-1} (\mathbf{I}_m - \mathbf{w}_j \boldsymbol{\beta}_j) \mathbf{w}_d \tag{17}$$

In most situations, these weights are computed using a matrix $\mathbf{P} \in \mathbb{R}^{m \times m}$ such that:

$$\mathbf{w}_d^* = \mathbf{P}_d \mathbf{w}_d \tag{18}$$

The overall algorithm is shown in the next table.

The f function generally comes from a logistic regression which is parameterized by $\boldsymbol{\theta}$. As this algorithm is also linear, we can make a class prediction by computing:

$$\hat{\mathbf{y}} = Round(\sigma(\mathbf{X}\mathbf{W}^*\boldsymbol{\theta} + \theta_{bias})) \tag{19}$$

Where σ is a sigmoid function and θ_{bias} is the intercept of the model. Note that in the PLS logistic model, each component requires to solve m logistic regressions. As this algorithm relies on backpropagation, it is computationally expensive even for moderately sized matrices.

3.2.3. General Algorithm

While Algorithm 1 works well for a one dimensional target $\mathbf{y}$, it has some drawbacks for a multiclass problem. For a k classes target matrix $\mathbf{Y} \in \{0,1\}^{n \times k}$, it does compute k components at the same time which are not independent in this situation. One natural way to compute a single component is to solve the following problem instead:

$$\begin{array}{ll} \max\limits_{\mathbf{w}_1} & cov_{pb}^2(\mathbf{X}\mathbf{w}_1, \mathbf{Y}) \\ \text{s.t.} & \mathbf{w}_1^\top \mathbf{w}_1 = 1 \end{array} \tag{20}$$

which can be rewritten as:

$$\begin{array}{ll} \max\limits_{\mathbf{w}_1} & \mathbf{w}_1^\top \mathbf{X}^\top \tilde{\mathbf{Y}} \tilde{\mathbf{Y}}^\top \mathbf{X} \mathbf{w}_1 \\ \text{s.t.} & \mathbf{w}_1^\top \mathbf{w}_1 = 1 \end{array} \tag{21}$$

Setting $\mathbf{C} = \tilde{\mathbf{Y}}^\top \tilde{\mathbf{X}}$, the Lagrangian is written as:

$$\mathcal{L} = \mathbf{w}_1^\top \mathbf{C}^\top \mathbf{C} \mathbf{w}_1 - \lambda(\mathbf{w}_1^\top \mathbf{w}_1 - 1) \tag{22}$$

Yielding the following eigen problem:

$$\mathbf{C}^\top \mathbf{C} \mathbf{w}_1 = \lambda \mathbf{w}_1 \tag{23}$$

In practice, this problem is solved by using an iterative algorithm called NIPALS which does approximate the solution using multiples projection matrices. While being very fast on small problems, it does not scale well for large inputs. Additionally, it requires to loop over $\mathbf{Y}$ columns which is inefficient outside binary problems. Solving the eigen problem is computationally expensive if m is large or if $m > k$ which is the case in the majority of classification tasks. We solve the transposed problem instead to improve efficiency:

$$\mathbf{C}\mathbf{C}^\top \mathbf{w}_e = \lambda \mathbf{w}_e \tag{24}$$

with $\mathbf{w}_e^*$ denoting the eigenvector associated to the highest eigen value λ^*, we immediately get the unormalized weight:

$$\mathbf{w}_1 = \mathbf{C}^\top \mathbf{w}_e^* \tag{25}$$

Algorithm 1 Naive PLS binary algorithm

Require: $\mathbf{X} \in \mathbb{R}^{n \times m}$, $\mathbf{x}_j$ centered and scaled $\forall j \in (1, \ldots, m)$
Require: $\mathbf{y} \in \{0, 1\}^n$ a binary vector
Require: $\lambda \geq 0$ a regularization parameter
Require: h the number of components
$\tilde{\mathbf{y}} \leftarrow \frac{1}{n^2}\left(n^{(0)}\mathbf{y} - n^{(1)}(\mathbf{1}_n - \mathbf{y})\right)$
$\tilde{\mathbf{X}} \leftarrow \mathbf{X}$
$\mathbf{P} \leftarrow \mathbf{I}_m$
for $d = 1$ to h **do**
    Compute weights: $\mathbf{w}_d \leftarrow \tilde{\mathbf{X}}^\top \tilde{\mathbf{y}} / \|\tilde{\mathbf{X}}^\top \tilde{\mathbf{y}}\|_2$
    Compute component: $\mathbf{t}_d \leftarrow \tilde{\mathbf{X}}\mathbf{w}_d$
    Compute the deflation factor: $\boldsymbol{\beta}_d \leftarrow (\mathbf{t}_d^\top \mathbf{t}_d + \lambda)^{-1} \mathbf{t}_d^\top \tilde{\mathbf{X}}$
    Store projection weights: $\mathbf{w}_d^* \leftarrow \mathbf{P}\mathbf{w}_d$
    if $d < h$ **then**
        Deflate: $\tilde{\mathbf{X}} \leftarrow \tilde{\mathbf{X}} - \mathbf{t}_d \boldsymbol{\beta}_d$
        Update $\mathbf{P}$ matrix: $\mathbf{P} \leftarrow \mathbf{P}(\mathbf{I}_m - \mathbf{w}_d \beta_d)$
    end if
end for
Set $\mathbf{W}^* = [\mathbf{w_1}^*, \ldots, \mathbf{w_h}^*]$
Estimate the model: $\mathbf{y} = f(\mathbf{X}\mathbf{W}^*; \boldsymbol{\theta}) + \epsilon$

Note that if $k = 1$, $\mathbf{w}_e^*$ is a scalar, thus we get the same formula as in the binary algorithm. To further improve efficiency, we do not use a projection matrix $\mathbf{P} \in \mathbb{R}^{m \times m}$ to compute the optimal weights $\mathbf{W}^*$ because the product $\mathbf{P_d}(\mathbf{I}_m - \mathbf{w}_d \boldsymbol{\beta}_d)$ $(d = 1, \ldots, h)$ is expensive if m is large which is often the case with natural language tasks. One way to improve performances is to compute the whole $\mathbf{W}^*$ matrix at the end of the algorithm. In practice, the component matrix $\mathbf{T} = \mathbf{X}\mathbf{W}^*$ can be written as [24]:

$$\mathbf{T} = \mathbf{X}\mathbf{W}(\mathbf{B}\mathbf{W})^{-1} \tag{26}$$

with $\mathbf{B} = [\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_h]$ and $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_h]$. Thus we can compute the final projection weight matrix $\mathbf{W}^*$:

$$\mathbf{W}^* = \mathbf{W}(\mathbf{B}\mathbf{W})^{-1} \tag{27}$$

The overall algorithm is written in the following table:

3.3. General Algorithm for Multiclass Case

3.3.1. Scalability

Most PLS models do not scale well to medium sized or large inputs due to their iterative nature. In practice, Algorithm 2 does scale better overall if we increase both n or m (Figure 4).

We also plot a lower precision variant which had no effect on performances presented in the next section. NIPALS is actually efficient if the target is one dimensional. However is fails to scale for larger targets as shown in Figure 5.

NIPALS can be volatile as it requires some convergence before computing the next component. Note also that the fast PLS variant complexity highly depends on k, more precisely on $min(k, m)$, due to the eigenvalue decomposition. To limit this dependency, we may rely on batching on n and m or k which

will increase the number of required components but will also considerably reduce the complexity. In practice, the number of classes k is often very limited except outside word embeddings estimations.

Algorithm 2 Fast PLS algorithm

Require: $\mathbf{X} \in \mathbb{R}^{n \times m}$, $\mathbf{x}_j$ centered and scaled $\forall j \in (1, \ldots, m)$
Require: $\mathbf{Y} \in \{0,1\}^{n \times k}$ a binary matrix
Require: $\mathbf{n}^{(0)}, \mathbf{n}^{(1)} \in \mathbb{N}^{1 \times k}$ class count vectors
Require: $\gamma \geq 0$ a regularization parameter
Require: h the number of components
$\tilde{\mathbf{Y}} \leftarrow \frac{1}{n^2}\left(\mathbf{n}^{(0)} \odot \mathbf{Y} - \mathbf{n}^{(1)} \odot (\mathbf{1}_{n \times k} - \mathbf{Y})\right)$
$\tilde{\mathbf{X}} \leftarrow \mathbf{X}$
$\mathbf{C} \leftarrow \tilde{\mathbf{Y}}^\top \tilde{\mathbf{X}}$
for $d = 1$ to h **do**
  if $k < m$ **then**
    Solve the eigen problem $(\mathbf{C}\mathbf{C}^\top - \lambda \mathbf{I}_k)\mathbf{w} = 0$
    Get the highest eigen value λ^* and its associated eigen vector $\mathbf{w}_e$
    Store normalized weights: $\mathbf{w}_d \leftarrow \mathbf{C}^\top \mathbf{w}_e / \|\mathbf{C}^\top \mathbf{w}_e\|_2$
  else
    Solve the eigen problem $(\mathbf{C}^\top \mathbf{C} - \lambda \mathbf{I}_m)\mathbf{w} = 0$
    Get the highest eigen value λ^* and its associated eigen vector $\mathbf{w}_d$
    Store normalized weights: $\mathbf{w}_d \leftarrow \mathbf{w}_d / \|\mathbf{w}_d\|_2$
  end if
  Compute the component: $\mathbf{t}_d \leftarrow \tilde{\mathbf{X}}\mathbf{w}_d$
  Store the deflation estimator: $\boldsymbol{\beta}_d \leftarrow (\mathbf{t}_d^\top \mathbf{t}_d + \gamma)^{-1} \mathbf{t}_d^\top \tilde{\mathbf{X}}$
  if $d < h$ **then**
    Deflate: $\tilde{\mathbf{X}} \leftarrow \tilde{\mathbf{X}} - \mathbf{t}_d \boldsymbol{\beta}_d$
    Update $\mathbf{C}$ matrix: $\mathbf{C} \leftarrow \mathbf{C}(\mathbf{I}_m - \mathbf{w}_d \boldsymbol{\beta}_d)$
  end if
end for
Set $\mathbf{W} = [\mathbf{w_1}, \ldots, \mathbf{w_h}]$
Set $\mathbf{B} = [\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_h]$
Compute the projection matrix $\mathbf{W}^* = \mathbf{W}(\mathbf{B}\mathbf{W})^{-1}$
Estimate the model on components: $\mathbf{Y} = f(\mathbf{X}\mathbf{W}^*; \boldsymbol{\theta}) + \epsilon$

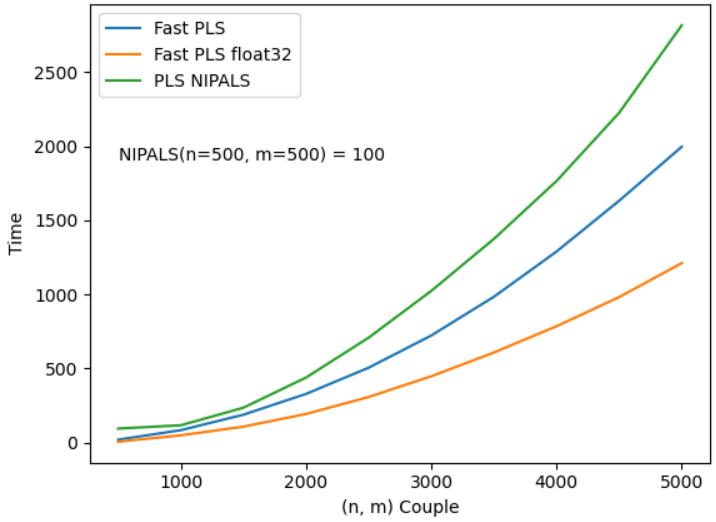

Figure 4. Effect of input size on training speed with $k = 1$.

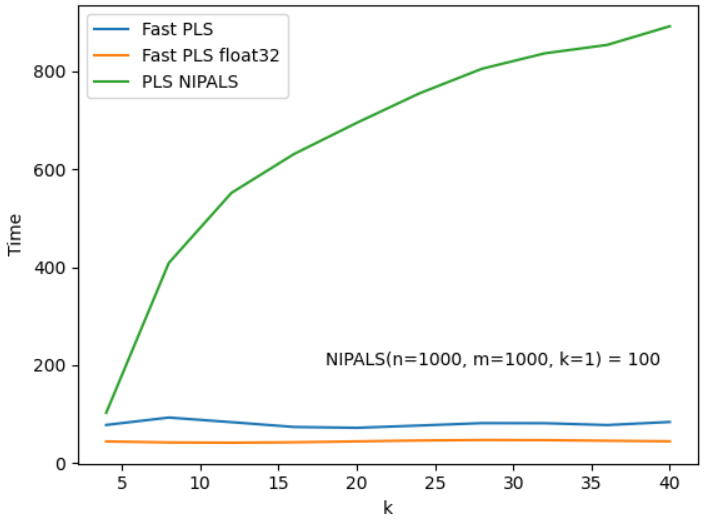

Figure 5. Effect of k on training speed with $(n = 1000, m = 1000)$.

4. Experiments and Results

We first compare different linear models using different inputs to show how sparsity can affect performances. We focus on two main tasks, the first relative to the violation of an article given a set of facts or circumstances which is a binary classification task. The second aims to find which of the convention articles is appropriate given factual informations, in this case it is a multiclass problem.

All these tasks are challenging, even more for the one with large base vocabulary size and small sample size. For example, the dataset related to article 5 only contains 122 documents for a vocabulary greater than 40,000 words for the circumstances and 50,000 for the facts which show how heterogeneous can the input be. Most of the time, there is no obvious relation between several documents from the same source. Few of the datasets are more manageable, the one related to article 13 has a smaller vocabulary size and rare words are less common which may improve overall performances.

For the first set of tasks, we use a simple preprocessing. We first tokenize at the word level by splitting the sequences on white spaces and punctuation then we build a binary count matrix and a TF-IDF representation. Since sentences are very long, we limit the vocabulary size to the 25,000 most frequent words. While using smaller sizes like 15 or 20 thousands words gives close performances, sparsity is less pronounced for simple classifiers. Choosing a size of 25,000 offered the best compromise between performance and sparsity to fully show how the PLS model handles this effect.

For the multiclass problem, we extend the limit to 50,000 words to further increase sparsity. Note that we are only relying on unigrams and we do not add any additional preprocessing step compared to the related literature [9,10], punctuation and stop words are not removed in this experiment. We also suppose that the modified PLS algorithm is able to discriminate and keep most informative components without the need of vocabulary selection, hand-crafted features or heuristics.

We compare three simple linear models: a logistic regression, a ridge regression where the regularization parameter as been optimized and a linear SVM. The choice of linear models is similar to related works which are relying on linear SVM. Since the sample size is very limited most of the time, these models are less prone to overfitting compared to tree-based approaches or kernel-based approaches which are expensive to compute for a large feature space.

We intentionally limit the number of PLS components to $h = 8$ in order to show the efficiency of the model given thousand of features. All experiments are compared using accuracy and a 10-cross validation approach.

For information, we copy the results from the related literature [9] obtained on english documents and similar tasks using a linear SVM (Table 2).

Table 2. English accuracy baselines on facts and circumstances from the literature.

Aletras (2016)	Art. 3	Art. 6	Art. 8
Circumstances	0.680	0.820	0.770
Facts	0.700	0.800	0.680

Beside a significant difference on article 8 which is difficult to explain for the circumstances, the results presented in the following section on french documents are similar to what has been achieved in the literature for linear models at least. Note that english is known to be grammatically simpler, which may improve overall performances.

4.1. Binary Input

Results for binary inputs are reported in Tables 3 and 4.

Table 3. Accuracy with binary input on circumstances.

Circumstances	Art. 3	Art. 5	Art. 6	Art. 8	Art. 10	Art. 13	Stack
Logit	0.693	0.656	0.776	0.676	0.721	0.820	0.446
Ridge	0.699	0.656	0.731	0.633	0.729	0.812	0.363
SVM	0.737	0.656	0.784	0.669	0.729	0.812	0.455
PLS + Logit	0.686	0.638	0.765	0.649	0.707	0.804	0.617
PLS + Ridge	0.681	0.647	0.765	0.634	0.700	0.804	0.552
PLS + SVM	0.692	0.671	0.763	0.634	0.707	0.795	0.629

Table 4. Accuracy with binary input on facts.

Facts	Art. 3	Art. 5	Art. 6	Art. 8	Art. 10	Art. 13	Stack
Logit	0.704	0.614	0.778	0.684	0.764	0.812	0.407
Ridge	0.704	0.647	0.740	0.700	0.764	0.820	0.364
SVM	0.698	0.655	0.787	0.739	0.764	0.820	0.467
PLS + Logit	0.687	0.655	0.769	0.692	0.743	0.837	0.620
PLS + Ridge	0.682	0.647	0.765	0.689	0.743	0.829	0.561
PLS + SVM	0.676	0.638	0.757	0.688	0.721	0.828	0.626

As expected at least for the binary cases, running the models without a PLS reduction bring slightly better results overall but with a higher computational cost. Computing components then estimating a model is significantly faster than running the model on raw inputs for the logistic regression and the SVM. Note that we only use 8 components and we are already achieving very close performances. For the multiclass problem, standard algorithms already fail to generalize to new inputs due to sparsity (50 k features). Regularization does not fix the problem as ridge regression does not bring strong performances.

Running models on full facts does significantly increase sentences length which provide additional information but also adds some noise due to domestic legal arguments. In most situations, performances increase slightly except for article 5 where standard models struggle to keep their previous accuracy.

For the multiclass problem, we do not observe any significant difference in accuracy using different inputs. As shown in Table 5, some articles are easier to predict such as article 10 which refers to the freedom of expression and article 3 relating to torture or inhuman treatment. The reason is linked to the structure of their respective vocabulary, with predominant themes such as violence or the media, making it easier to detect them. On the other hand, articles 5 related to the right to liberty are generally more vague with a much broader vocabulary. Finally, one can observe that as in the binary case, classes with a lot of rare words tend to be more difficult to recognize.

Table 5. Individual and global accuracies for the stack dataset (PLS + SVM models).

Stack	Art. 3	Art. 5	Art. 6	Art. 8	Art. 10	Art. 13	Global
Circumstances	0.625	0.572	0.645	0.615	0.664	0.630	0.629
Facts	0.615	0.567	0.646	0.609	0.668	0.626	0.626

4.2. TF-IDF Input

We now run the same experiment but replace the binary input with a TD-IDF representation. The input size is the same as before but does provide a different information as it relies on frequencies and inverse frequencies. The results for binary inputs are reported in Tables 6 and 7.

Table 6. Accuracy with tf-idf input on circumstances.

Circumstances	Art. 3	Art. 5	Art. 6	Art. 8	Art. 10	Art. 13	Stack
Logit	0.489	0.410	0.543	0.484	0.457	0.418	0.363
Ridge	0.516	0.418	0.699	0.540	0.471	0.510	0.363
SVM	0.467	0.410	0.453	0.492	0.443	0.410	0.363
PLS + Logit	0.708	0.623	0.780	0.641	0.721	0.821	0.612
PLS + Ridge	0.703	0.631	0.784	0.649	0.736	0.821	0.602
PLS + SVM	0.703	0.640	0.782	0.649	0.729	0.829	0.622

Table 7. Accuracy with tf-idf input on facts.

Facts	Art. 3	Art. 5	Art. 6	Art. 8	Art. 10	Art. 13	Stack
Logit	0.500	0.410	0.556	0.508	0.450	0.410	0.363
Ridge	0.532	0.410	0.708	0.582	0.471	0.534	0.363
SVM	0.473	0.410	0.458	0.472	0.436	0.410	0.363
PLS + Logit	0.698	0.622	0.784	0.680	0.736	0.821	0.618
PLS + Ridge	0.709	0.630	0.776	0.680	0.736	0.821	0.621
PLS + SVM	0.709	0.630	0.776	0.684	0.736	0.821	0.636

The additional variance due to the nature of the input significantly affects the overall performances without PLS reduction. Ridge regression has slightly higher accuracy due to the regularization parameter. We observe that the PLS algorithm is still consistent and even show better scores because the TF-IDF representation is generally more informative than a simple binary matrix. Accuracy on the multiclass problem is even worse and is very close to the frequency of the larger class of this dataset which is related to article 6.

We observe the same problem as before. PLS reduction does bring efficiency and consistency. Computing components and adding a linear classifier on top provides very similar performances regardless of the model, this is a consequence of a low number of components. Note that increasing h did not provide significant gains on average. The choice should be based on the dataset, but we choose to keep h the same to make fair comparisons.

On the multiclass task (Table 8), the same behavior as in Table 5 is generally observed with slightly better performances with facts as inputs, this was not the case before.

Table 8. Individual and global accuracies for the stack dataset (PLS + SVM models).

Stack	Art. 3	Art. 5	Art. 6	Art. 8	Art. 10	Art. 13	Global
Circumstances	0.608	0.559	0.642	0.611	0.660	0.621	0.622
Facts	0.635	0.589	0.648	0.619	0.672	0.640	0.636

4.3. Neural Approach

The neural network approach is different as it tales advantage of the temporal aspect thanks to the attention mechanism. The choice of attention is indeed natural considering the current state of the NLP literature which does rely entirely on transformer-based models. Training a RNN as an alternative is not appropriate here because convergence is extremely slow when sequences are very long. Another approach would have been to only use feedforward layers, but it requires using some averaging to remove the sequential aspect. Using a mean embedding over thousand of tokens provides a very noisy representation as all words are equally weighted in this case.

We estimate our models by mixing two different inputs, one comes from a pretrained FastText (128 dimensions) which has been trained on 10Gb of french legal documents (2 billions tokens). The second is either a binary input or a TF-IDF input. We use the attention mechanism on the embedding inputs while we simply rely on feedforward layers for the other one. We stack everything up before doing our prediction. Using this combination generally improves speed of convergence as one part of the network does focus on word discrimination and acts like a simple linear model while the second part is oriented toward disambiguation and the handling of rare words. We rely on cross entropy as the loss function, we stack 3 layers of transformers and use Adam [25] as the optimizer with a learning rate of 10^{-3} and a weigth decay of 0.01.

We also train similar models with a dimensional reduction provided by the PLS algorithm. All results are reported in Tables 9 and 10.

Table 9. Accuracy with attention mecanism on circumstances

Circumstances	Art. 3	Art. 5	Art. 6	Art. 8	Art. 10	Art. 13	Stack
Attention + binary	0.773	0.805	0.803	0.749	0.797	0.858	0.766
Attention + tf-idf	0.775	0.809	0.801	0.751	0.797	0.862	0.765
Attention + binary + PLS	0.761	0.782	0.791	0.734	0.783	0.846	0.747
Attention + tf-idf + PLS	0.768	0.794	0.790	0.748	0.787	0.848	0.740

Table 10. Accuracy with tf-idf input on facts.

Facts	Art. 3	Art. 5	Art. 6	Art. 8	Art. 10	Art. 13	Stack
Attention + binary	0.786	0.801	0.829	0.778	0.800	0.880	0.772
Attention + tf-idf	0.788	0.805	0.827	0.781	0.800	0.882	0.774
Attention + binary + PLS	0.781	0.792	0.815	0.768	0.791	0.862	0.756
Attention + tf-idf + PLS	0.782	0.793	0.811	0.772	0.793	0.869	0.756

The use of neural networks greatly improves performance, but the computational cost is also much higher. Applying a reduction does hurt performances with a loss of 1.5 points in accuracy in average but the model is also easier and much faster to train which can be a good trade off in some situations.

We observe a similar behavior on facts. In either case, the binary or TF-IDF have little effect on performances due to the embedding which may already incorporate similar information.

In the multiclass task, one can observe huge performance gains over simple linear models with more than 10 points difference on average for each individual accuracy (Table 11). The most difficult classes to predict remain the same compared to those observed without the use of a neural network and the attention mechanism.

Table 11. Individual and global accuracies for the stack dataset (Attention + binary + PLS models).

Stack	Art. 3	Art. 5	Art. 6	Art. 8	Art. 10	Art. 13	Global
Circumstances	0.744	0.688	0.755	0.729	0.820	0.740	0.747
Facts	0.750	0.705	0.761	0.738	0.832	0.754	0.756

4.4. General Observations

Overall, article 13 is the easiest to predict while article 8 is the most difficult. For attention-based approaches, article 5 seems easier to predict compared to simple linear models as we can observe a significant performance gain from 10 to 15 points. If we pay attention to the overall vocabulary size, datasets with many rare words tend to be harder to predict for simpler models as they cannot properly handle words that appear only once. On the other hand, word embeddings are able to infer these words because they are trained on a large corpus before and may have seen them during pre-training. This advantage is also visible for the multiclass task where the overall vocabulary is very large (more than 100 K words) with a majority of rare words.

The natural ability of neural networks to disambiguate expressions also helps a lot when simplest models cannot properly handle sequences. However, while attention-based approaches are in theory able to provide some explainability, the exploitation of attention weights makes few sense for very long sequences (thousands of words). Most of them are actually very small due to the fact that they are positive and add up to one, they also provide very noisy outputs which cannot be properly exploited by practitioners.

5. Conclusions

This paper attempts to provide a dimension reduction algorithm for classification tasks when applying a model to raw inputs can lead to a bad solution or a high computational cost. We show that this PLS variant retains goods performances with varying sparse inputs size and can also be used to reduce the number of parameters of a large neural networks with limited effects on accuracy. Finally we observe that linear models while being simple and computationally cheap, can provide decent performances on complex tasks even when the sample size is small and the feature space large. However they are largely outclassed by sophisticated neural networks, capable of correctly processing sequences and handling rare words using a word embedding pretrained on a large corpus.

Funding: Granted by Région Occitanie.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Gonçalves, T.; Quaresma, P. Is Linguistic Information Relevant for the Classification of Legal Texts? In Proceedings of the 10th International Conference on Artificial Intelligence and Law, ICAIL'05, Bologna, Italy, 6–11 June 2005
2. Chalkidis, I.; Androutsopoulos, I.; Michos, A. Obligation and Prohibition Extraction Using Hierarchical RNNs. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*; Association for Computational Linguistics: Melbourne, Australia, 2018; pp. 254–259. [CrossRef]
3. Waltl, B.; Muhr, J.; Glaser, I.; Georg Bonczek, E.S.; Matthes, F. Classifying Legal Norms with Active Machine Learning. In Proceedings of the 30st International Conference on Legal Knowledge and Information Systems (JURIX), Luxembourg, 13–15 December 2017; pp. 11–20.
4. Chalkidis, I.; Fergadiotis, M.; Malakasiotis, P.; Aletras, N.; Androutsopoulos, I. Extreme Multi-Label Legal Text Classification: A case study in EU Legislation. *CoRR* **2019**, abs/1905.10892.
5. Zhong, H.; Guo, Z.; Tu, C.; Xiao, C.; Liu, Z.; Sun, M. Legal Judgment Prediction via Topological Learning. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*; Association for Computational Linguistics: Brussels, Belgium, 2018; pp. 3540–3549. [CrossRef]

6. Sulea, O.; Zampieri, M.; Vela, M.; van Genabith, J. Predicting the Law Area and Decisions of French Supreme Court Cases. *CoRR* **2017**, abs/1708.01681.
7. Hu, Z.; Li, X.; Tu, C.; Liu, Z.; Sun, M. Few-Shot Charge Prediction with Discriminative Legal Attributes. In *Proceedings of the 27th International Conference on Computational Linguistics*; Association for Computational Linguistics: Santa Fe, NM, USA, 2018; pp. 487–498.
8. Wei, F.; Qin, H.; Ye, S.; Zhao, H. Empirical Study of Deep Learning for Text Classification in Legal Document Review. *CoRR* **2019**, abs/1904.01723.
9. Aletras, N.; Tsarapatsanis, D.; Preoţiuc-Pietro, D.; Lampos, V. Predicting Judicial Decisions of the European Court of Human Rights: A Natural Language Processing Perspective. *PeerJ Comput. Sci.* **2016**, *2*, e93. [CrossRef]
10. Medvedeva, M.; Vols, M.; Wieling, M. Using machine learning to predict decisions of the European Court of Human Rights. *Artif. Intell. Law* **2020**, *28*, 237–266. [CrossRef]
11. O'Sullivan, C.; Beel, J. Predicting the Outcome of Judicial Decisions Made by the European Court of Human Rights *CoRR* **2019**, abs/1912.10819.
12. Chalkidis, I.; Androutsopoulos, I.; Aletras, N. Neural Legal Judgment Prediction in English. *CoRR* **2019**, abs/1906.02059.
13. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.; Polosukhin, I. Attention Is All You Need. *CoRR* **2017**, abs/1706.03762.
14. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.; Dean, J. Distributed Representations of Words and Phrases and their Compositionality. *CoRR* **2013**, abs/1310.4546.
15. Bojanowski, P.; Grave, E.; Joulin, A.; Mikolov, T. Enriching Word Vectors with Subword Information. *CoRR* **2016**, abs/1607.04606. [CrossRef]
16. Peters, M.E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; Zettlemoyer, L. Deep contextualized word representations. *CoRR* **2018**, abs/1802.05365.
17. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
18. Devlin, J.; Chang, M.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* **2018**, abs/1810.04805.
19. Shen, Z.; Zhang, M.; Yi, S.; Yan, J.; Zhao, H. Factorized Attention: Self-Attention with Linear Complexities. *CoRR* **2018**, abs/1812.01243.
20. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. *CoRR* **2015**, abs/1512.03385.
21. Wold, H. Estimation of principal components and related models by iterative least squares. *Multivar. Anal.* **1966**, 1, 391–420.
22. Tenenhaus, M. La Regression Logistique PLS; In Proceedings of the 32èmes journées de Statistique de la Société française de Statistique, Fes, Morocco, 17–21 May 1999.
23. Barker, M.; Rayens, W. Partial Least Squares For Discrimination. *Analyst* **2003**, *17*, 166–173. [CrossRef]
24. Manne, R. Analysis of two partial-least-squares algorithms for multivariate calibration. *Chemom. Intell. Lab. Syst.* **1987**, *2*, 187–197. [CrossRef]
25. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. In Proceedings of the 3rd International Conference for Learning Representations, ICLR, San Diego, CA, USA, 7–9 May 2015.