Information Systems Frontiers manuscript No.
(will be inserted by the editor)

GarNLP: A Natural Language Processing Pipeline for Garnishment Documents

Ilaria Bordino · Andrea Ferretti · Francesco Gullo · Stefano Pascolutti

the date of receipt and acceptance should be inserted later

Abstract Basic elements of the law, such as statuses and regulations, are embodied in natural language, and strictly depend on linguistic expressions. Hence, analyzing legal contents is a challenging task, and the legal domain is increasingly looking for automatic-processing support.

This paper focuses on a specific context in the legal domain, which has so far remained unexplored: automatic processing of *garnishment* documents. A garnishment is a legal procedure by which a creditor can collect what a debtor owes by requiring to confiscate a debtor's property (e.g., a checking account) that is hold by a third party, dubbed garnishee.

Our proposal, motivated by a real-world use case, is a versatile natural-language-processing pipeline to support a garnishee in the processing of a large-scale flow of garnishment documents. In particular, we mainly focus on two tasks: (i) categorize received garnishment notices onto a predefined taxonomy of categories; (ii) perform an information-extraction phase, which consists in automatically identifying from the text various information, such as identity of involved actors, amounts, and dates. The main contribution of this work is to describe challenges, design, implementation, and performance of the core modules and methods behind our solution. Our proposal is a noteworthy example of how data-science techniques can be successfully applied to a novel yet challenging real-world context.

Keywords applied data science · natural language processing · legal documents · garnishment · categorization · information extraction · supervised learning · word embeddings · named entity recognition

I. Bordino, A. Ferretti, F. Gullo
UniCredit, R&D Department, Italy
E-mail: {ilaria.bordino,andrea.ferretti2,francesco.gullo}@unicredit.eu

S. Pascolutti
Google, Switzerland
E-mail: pazqo@google.com
(Work completed while the author was employed at UniCredit)

1 Introduction

Natural language processing (NLP) is a field of artificial intelligence that helps machines "read" and understand text in natural language, such as business documents, news, emails, social-network posts. NLP technologies bring attractive benefits to a plethora of industry verticals, including finance, automotive, healthcare, and law. Entailed advantages encompass savings in terms of time and working resources, and effectiveness in many real-world tasks.

NLP and the legal domain. The law has language at its heart, thus it is not surprising that the legal domain is among those more interested in NLP. The application of artificial intelligence (AI) in the legal domain dates back to the 1960s, when the earliest systems for searching online legal content appeared, while legal expert systems were a hot topic in the 1970s and 1980s [3]. However, the last years have witnessed an upsurge of interest in the area.

One of the main obstacles to progress in the field of AI and law is the *natural language barrier* [28]. Legal knowledge is heavily intertwined with natural language and common sense, and law documents are typically characterized by a peculiar language, and *convoluted and unnatural* syntax [29]. This makes accessing the content embedded in legal texts a particularly challenging task, for which general-purpose NLP engines may not be the appropriate choice.

Garnishment. In simple words, a *garnishment* is a legal procedure by which a *creditor* can collect what a *debtor* owes by reaching a debtor's property, when the property is in the hands of a *third party* other than the debtor. In this scenario the creditor can initiate a garnishment judiciary action against the debtor as a defendant, and the third party holding the property as a *garnishee*. The most common type of garnishment involves the confiscation of a due amount from a checking account of the debtor. Therefore, the third party that acts as a garnishee is typically a bank or another credit institution holding a checking account of the debtor. During an active garnishment process, the garnishee is obliged to a truthful collaboration with the judicial authorities. Upon receiving due notification, the garnishee is required to block the garnishment amount in the checking account of the debtor. In fact, should the debtor withdraw such an amount from her account, the garnishee would be responsible for this act.

GarNLP: NLP for garnishees. A garnishee typically has to deal with a considerable number of documents related to garnishment procedures.[1] Processing and understanding these documents require to perform classification tasks, or extract relevant pieces of information from the text. Accomplishing these tasks on a large-scale flow of documents is complicated and time-consuming, and may require a lot of (human) effort without a proper technological support.

In this paper we focus on designing *NLP solutions for garnishment*, a specific use case in the legal domain that, to the best of our knowledge, has so far been overlooked in the NLP literature. Although the tasks for supporting the activities of a garnishee are, from a high-level perspective, rather

[1] As an example, the legal office of our partner garnishee receives ∼1k-2k documents/day.

standard, bringing them to practice and making them work for a specific context such as garnishment requires nontrivial effort. In fact, the garnishment setting comes with several challenges, such as the vast heterogeneity of garnishment documents, which typically originate from different sources and, as such, do not follow standard templates; their lack of structure; the use of a domain-specific language, which may be very different from the common-usage one; the peculiarities of the information to be classified/extracted. All of this prevents the adoption of general-purpose NLP solutions or the adaptation of domain-specific NLP pipelines designed for different application domains. Instead, there is an eager need for solutions that are ad-hoc suited for the garnishment context. To satisfy this demand, we introduce GarNLP, a *NLP framework to support a garnishee in the processing of garnishment documents.* The proposed framework can analyze and annotate various kinds of document exchanged among the actors of a garnishment process, by addressing two tasks: (*i*) *categorizing* received documents onto a predefined taxonomy of categories; (*ii*) performing an *information extraction* phase, which consists in automatically identifying, from the text, information such as identity and attributes of the main actors involved in the garnishment procedure (i.e., creditor, debtor, lawyer), the garnishment amount, the hearing date, and so on.

Benefits for the garnishee. Our work is motivated by a concrete request of UniCredit, which is a big pan-European commercial bank and obviously acts as a garnishee in a large number of garnishment procedures.[2] Prior to building GarNLP, the office assigned to the processing of garnishment documents was following a workflow that was mostly paper-based and handled by human operators. Although there was a semi-automatic handling of some functionalities, the workflow was affected by critical inefficiencies, and the lack of a proper IT support resulted in a significant amount of manual interventions. Providing automatic support for some of the required tasks, our pipeline improves the effectiveness and efficiency of the overall end-to-end process, enabling a plethora of potential benefits, including: (*i*) faster and better management of garnishment notices, reducing uncertainty and variability in the management of complex and non-standard cases; (*ii*) better use of human resources, reducing data-entry activities (and, consequently, the number of assigned employees) with the usage of an accurate extraction pipeline; (*iii*) reduced risks and losses from human errors.

Applicability. We remark that, although our solution has been conceived and deployed for a bank, it can be used (with minimal adaptation effort) by any other type of garnishee, or even actors other than garnishee involved in garnishment processes. Another remark on the potential (re-)usability of GarNLP concerns the language. Although we have worked with documents in Italian, we claim that our methodology is in great part language-indepedent, and can easily be adapted to other languages (more details on this aspect are reported in Section 4).

[2] The GarNLP framework is currently being productionized by the partner bank.

Contributions and roadmap. The main contributions of this paper are:

- We address a specific use case of the legal domain that has so far remained unexplored: automatic processing of garnishment documents (Section 2).
- We devise GarNLP, a NLP pipeline for garnishment documents, performing document categorization and information-extraction tasks. The main contribution here consists in designing an end-to-end NLP solution that is well-suited for the garnishment setting, thus achieving all its domain-specific challenges. Advancing the literature on the general tasks underlying our solution is instead beyond the scope of this work (Sections 3–4).
- We assess the performance of our GarNLP framework on a real-world use case and dataset provided by UniCredit, a big pan-European commercial bank. Results attest the high quality of our methods (Section 5).

Section 6 overviews related work, while Section 7 concludes the paper.

We believe this work is a relevant example of applied data science, i.e., how data-science techniques – from the areas of NLP, machine learning, and information extraction – can be successfully customized, combined, and finally deployed in a novel yet challenging real-world application context.

2 Application Scenario

A *garnishment* is a legal procedure by which a *creditor* can collect what a *debtor* owes by reaching a debtor's property, when the property is in the hands of a third-party, i.e., a *garnishee*. The specific procedure underlying a garnishment order depends on state law, and thus may differ from country to country. However, the main principles of the process are roughly the same everywhere.

The two main types of garnishment are *wage* or *attachment*. The former deducts money from an employee's monetary compensation: in this case, the garnishee is usually the debtor's employer. Attachment garnishment, instead, requires the garnishee to deliver the defendant's money in the hands of the garnishee at the time of process to the court. Attachment garnishees are typically *banks*, or other credit institutions. In this work we focus on attachment garnishment, paying attention to the actions required by a garnishee.

A garnishment procedure is initiated by the creditor, who must be in possession of an *enforceable order*, i.e., a document attesting the credit against the debtor. The garnishment of the debtor's property starts when the debtor receives the notification of the enforceable order and the *confiscation order*, an order of payment that the debtor must fulfil within a specified time interval. The *foreclosure order* is then notified to the debtor and the garnishee.

During a garnishment procedure, a garnishee is obliged to a truthful collaboration with the judicial authorities. Upon receiving the foreclosure notification, the garnishee must block the garnishment amount in the debtor's property. The garnishment procedure is ruled by a judge in a court hearing, which takes place on a date indicated in the foreclosure notice.

2.1 Target context: garnishment from a garnishee perspective

We aim at supporting the tasks a garnishee should perform upon receiving a garnishment document. This procedure consists of two phases: (*i*) *Document processing*, i.e., categorizing the document and extracting relevant information from it; (*ii*) *Implementation*, i.e., taking the actions requested in the text, e.g., providing information to a legal entity, or seizure/release of a bank account.

The focus of this work is right on the document-processing tasks. We devise a framework that supports the daily activities of employees who are devoted to those tasks. We term our framework GarNLP, as an obvious allusion to application of NLP techniques to the garnishment domain.

Specifically, the document-processing phase typically consists of two main subtasks: (*i*) *categorization*, and (*ii*) *information extraction*.

Categorization. This task deals with assigning the document at hand the correct garnishment-specific category, which is a critical initial step of the overall garnishment process for a garnishee, as different categories may imply totally different ways of dealing with the document.

Given a garnishment document D, the goal of categorization is to assign D a category from a pre-defined set $\mathcal{C}$ of garnishment-specific categories. Categories $\mathcal{C}$ mainly depend on the garnishment legislation in place. For the use case we consider here, i.e., garnishment in Italy as of 2018-2019, they are:

- *practice*, whose main purpose is for a legal entity to ask a garnishee information about the debtor of the underlying garnishment process, such as debtor's available money (in the form of account balance/payment/wage).
- *assignment*, i.e., a court order that requires a garnishee to seize a certain amount of money for a debtor.
- *renunciation*, which denies a previous assignment order, because, e.g., the creditor has changed her mind about the garnishment proceeding or the garnishment amount has been finally assigned to a different garnishee.

Set $\mathcal{C}$ may also contain further garnishee-specific categories, like, e.g., subcategories defined based on the creditor/lawyer of the underlying proceeding.

Information extraction. The second task is to extract relevant information from a garnishment document. The main types of information we consider are:

- *Actors* of the garnishment procedure. Here the goal is to identify *creditor*, *debtor*, and *lawyer* of the procedure, extracting first/last names (for natural persons) or business names (for legal persons). A desideratum may be to extract further information, e.g., date/place of birth, address, and so on.
- *Amounts*, e.g., the debt owed to the creditor, or the amount to be seized.[3]
- *Dates*, such as *notification date* (when the garnishment document has been notified to the garnishee) or *hearing date* (when the hearing will take place).
- *Codes*, i.e., specific codes that are typically assigned to certain parts of the garnishment document (e.g., *injunction number*), and whose tracking may be useful for, e.g., reconstructing the history of a garnishment process.

[3] Owed amount and seized amount may differ as a court order may require to seize an amount that is (slightly) more than the owed one (for tax or interest reasons).

3 The GarNLP framework

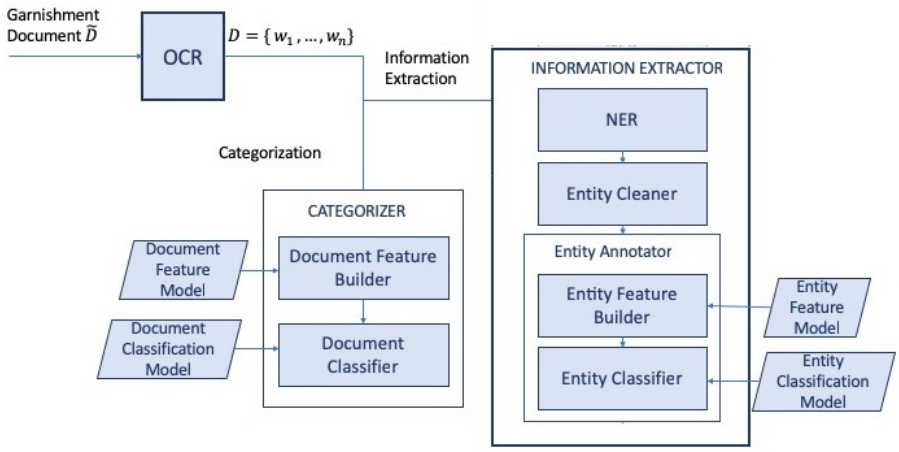

Fig. 1: Overview of the proposed GarNLP framework.

In this section we describe the solution we propose for handling the application scenario presented in Section 2. The main components of the proposed framework are depicted in Figure 1. A preliminary common step of the framework consists in performing *optical character recognition* (OCR), as it is not uncommon that a garnishee is provided with non-digital garnishment documents, and, as such, they have to be preliminarily scanned and, indeed, OCR-ized to get to a textual, machine-readable format. After this preliminary step, a raw garnishment document $\tilde{D}$ may be represented as a sequence $D = [w_1, \ldots, w_n]$ of n tokens (i.e., terms or punctuation symbols), where $n = |D|$, and passed to the two core components that are in charge of categorization and information extraction, i.e., the *categorizer* and the *information extractor*, respectively.

3.1 Categorizer

Given a garnishment document $D = [w_1, \ldots, w_n]$ and a set $\mathcal{C} = \{C_1, \ldots, C_m\}$ of categories, the goal of the categorizer module is to assign D a category from $\mathcal{C}$. In this work we assume the availability of a ground truth of documents whose category has been manually assigned, and cast the problem at hand as a (multi-class) supervised-learning task. As a result, our categorization happens in two phases: *training*, where the ground truth is exploited (offline) to build a proper model, and *inference*, where the trained model is exploited (online) to ultimately perform category assignment. Orthogonally to this distinction between training and inference, the categorizer is logically split into two subcomponents, i.e., *document-feature builder* and *document classifier*.

Document-feature builder. The very first step of our categorizer is to represent a garnishment documents by a suitable format that can be handled by (standard) machine learning. To this end, we adopt a frequentist approach and represent a document D by a k-dimensional integer vector (k defined below), which encodes the frequency of the terms that help the most discern between any two categories in $\mathcal{C}$. The main intuition here is that, for every pair $C_i, C_j \in \mathcal{C}$ of categories, the terms whose frequency within (documents of) category C_i is considerably higher than the frequency in category C_j (or vice versa) are the most discriminant ones to distinguish between C_i and C_j.

Formally, let $\mathcal{G}$ be a set of ground-truth documents, where, for every $\hat{D} \in \mathcal{G}$, $c(\hat{D}) \in \mathcal{C}$ denotes the category manually-assigned to $\hat{D}$, and, for every category $C \in \mathcal{C}$, let $\mathcal{G}_C = \{\hat{D} \in \mathcal{G} \mid c(\hat{D}) = C\}$ be the subset of $\mathcal{G}$ whose category corresponds to C. For a term w, let $tf(w, \hat{D})$ be the term frequency of w within document $\hat{D}$ (i.e., the number of occurrences of w within D), and $tf(w, C) = \sum_{\hat{D} \in \mathcal{G}_C} tf(w, \hat{D})$ the term frequency of w within $\mathcal{G}_C$. The relative frequency of a term w with respect to categories $C_i, C_j \in \mathcal{C}$ is defined as:

$$\overline{tf}(w, C_i, C_j) = \max\left\{\frac{1 + tf(w, C_i)}{1 + tf(w, C_j)}, \frac{1 + tf(w, C_j)}{1 + tf(w, C_i)}\right\}.$$

Given an integer $h > 0$, for every pair $C_i, C_j \in \mathcal{C}$ of categories, we compute the top-h frequent terms $T_{ij}(h)$ by sorting the various terms w in non-increasing order of their $\overline{tf}(w, C_i, C_j)$ relative frequency, and taking the first h terms of such a resulting ordered list (ties broken randomly). Ultimately, for an input garnishment document D, we compute the overall set of terms of interest as:

$$T(h) = \bigcup_{C_i, C_j \in \mathcal{C}} T_{ij}(h), \tag{1}$$

while the k-dimensional integer vector $\mathbf{v}(D)$, $k = |T(h)| \leq h\ \frac{m(m-1)}{2}$, used for representing document D is defined as:

$$\mathbf{v}(D) = [tf(w, D)]_{w \in T(h)}\,. \tag{2}$$

To summarize, the training phase of the document-feature builder takes h as input and computes the term set $T(h)$. This way, the *document-feature model* in Figure 1 simply corresponds to $T(h)$. To handle class imbalance, the training is performed a ground truth with the same number of documents for every class. Given a document D, the inference phase instead consists in computing vector $\mathbf{v}(D)$ (Equation (2)) based on the term set (model) $T(h)$.

Document classifier. During training, the document classifier exploits the vectorial representation $\mathbf{v}(\hat{D})$ of every ground-truth document $\hat{D} \in \mathcal{G}$, along with the corresponding $c(\hat{D})$ label, to learn a *document-classifier model* (as termed in Figure 1). Here training is carried out on a ground truth where the number of documents of every class is kept proportional to the original frequency of that class in the whole dataset. This was purposely done to help the model better learn the representativity of each class.

Once it has been built, the document-classifier model is exploited in the inference phase to ultimately assign a category to the target input garnishment document D (represented by its vector $\mathbf{v}(D)$).

Both training and inference are carried out with standard supervised-learning techniques. Specifically, each document belongs to exactly one category: therefore, the underlying learning task is a classic multi-class classification one. More details in this regard are reported in Sections 4–5.

3.2 Information extractor

The information-extractor component of GarNLP identifies relevant information from a garnishment document D, by employing a *filter-and-verify*, two-step approach. The first step performs *named entity recognition* (NER) [32], whose main goal is to extract all *named entities* from the document. The second step classifies every named entity as an entity of interest or not, while also identifying its type (e.g., creditor/debtor/lawyer name, hearing date, seized amount, etc.). Between such two main steps is *entity cleaning*, which aims at filtering out named entities that are easily recognizable as non-interesting.

NER. The NER module is devoted to a coarse-grained identification of the parts of the input document that are potentially of interest. Specifically, it performs the well-established NLP task of *recognizing named entities*, and classifying them into pre-defined, general categories. The categories considered by GarNLP are persons, organizations, places, dates, amounts, and (alpha)numeric codes, as these are the main types of information to be extracted in the garnishment scenario. Further types can however be included with minimal effort. Formally, given an input garnishment document $D = \{w_1, \ldots, w_n\}$, the NER's output is a set $\mathcal{E} = \{\langle e_1, \ell_{\text{NER}}(e_1), so_1, eo_1\rangle, \ldots, \langle e_{|\mathcal{E}|}, \ell_{\text{NER}}(e_{|\mathcal{E}|}), so_{|\mathcal{E}|}, eo_{|\mathcal{E}|}\rangle$, where, for all $i \in [1..|\mathcal{E}|]$, e_i is an entity of D, i.e., a set of consecutive terms drawn from D, $\ell_{\text{NER}}(e_i)$ denotes the NER category of e_i, while so_i and eo_i are the start and end offset of the mention of e_i in the text (needed to retrieve the context of e_i in the entity-feature-builder module, see next). We accomplish the NER task by resorting to well-established techniques. However, as the language used in garnishment documents is rather ad-hoc and different of the one used in generic texts, we also perform an ad-hoc retraining of state-of-the-art models on (a random sample of) our real dataset used in the experiments. As expected, this retraining was really beneficial, as the resulting model is recognized as our best one (see Sections 4–5 for more details on this).

Entity cleaner. The cleaner module extracts a subset $\mathcal{E}^* \subseteq \mathcal{E}$ of entities, by discarding entities from $\mathcal{E}$ that are easily recognizable as non-relevant information. We accomplish this by exploiting simple rules of two kinds: (*i*) *ground-truth-driven*, i.e., derived from simple analyses/statistics on the available ground-truth documents, and (*ii*) *language-specific*, i.e., syntactic rules of the specific language of the input garnishment documents at hand. An example of ground-truth-driven rule is *stop-entity removal*: We run NER on

the ground-truth documents, and, for every entity e extracted from document $\hat{D}$, we compute its *stop-frequency* as the number of times e does not match any ground-truth information for $\hat{D}$. Our entity cleaner ultimately removes the top-f entities exhibiting the highest stop-frequency (with $f = 100$ in our implementation). A second ground-truth-driven rule removes entities whose length (in terms of number of characters) is not within an $[lb, ub]$ range, where lb and ub are computed according to the distribution of the length of the ground-truth entities. Specifically, in our implementation lb and ub correspond to the 5th and 95th percentiles of such a distribution

Language-specific rules include filtering the candidate actors of the garnishment process by taking a list of proper names and a list of company legal endings, and discarding entities that do not match any proper name or do not end with any valid legal endings. A second rule of this kind is discarding entities containing literal patterns that cannot comply with the language at hand (e.g., in our use case where documents are written in Italian, entities having no vowels). This rule basically allows for detecting obvious OCR mistakes.

Entity annotator. This is the core subcomponent of our GarNLP's information extractor. It basically takes the filtered set $\mathcal{E}^* = \{\langle e_1, \ell_{\text{NER}}(e_1)\rangle, \ldots, \langle e_{|\mathcal{E}^*|}, \ell_{\text{NER}}(e_{|\mathcal{E}^*|})\rangle$ of entities (and corresponding NER categories) and assigns a proper garnishment label to every entity e_i, $i = [1..|\mathcal{E}^*|]$, that classifies it as (a specific type of) relevant garnishment information or not. Particularly, recall that the main types of garnishment information to be extracted are $\mathcal{T} = \{actors, amounts, dates, codes\}$ (Section 2). For every type $T \in \mathcal{T}$, let $\mathcal{L}(T)$ be the set of garnishment labels for T. For instance, $\mathcal{L}(actors) = \{creditor, debtor, lawyer\}$, $\mathcal{L}(dates) = \{hearing\ date\}$, and so on (the actual $\mathcal{L}(T)$ labels considered in our use case are discussed in Section 5). Also, for a type $T \in \mathcal{T}$, let $\mathcal{E}^*(T) \subseteq \mathcal{E}^*$ be the subset of entities $e_i \in \mathcal{E}^*$ that are relevant for T based on NER category $\ell_{\text{NER}}(e_i)$. Specifically, $\mathcal{E}^*(actors)$ consists of all entities $e_i \in \mathcal{E}^*$ whose NER category $\ell_{\text{NER}}(e_i) \in \{person, organization\}$, as actors may only be persons or organizations. Likewise, $\mathcal{E}^*(dates) = \{e_i \in \mathcal{E}^* \mid \ell_{\text{NER}}(e_i) = date\}$, $\mathcal{E}^*(amounts) = \{e_i \in \mathcal{E}^* \mid \ell_{\text{NER}}(e_i) = money\}$, $\mathcal{E}^*(codes) = \{e_i \in \mathcal{E}^* \mid \ell_{\text{NER}}(e_i) \notin \{person, organization\}\}$.

The goal of the information extractor is, for every information type $T \in \mathcal{T}$ and entity $e_i \in \mathcal{E}^*(T)$, to yield a garnishment label $\ell_i \in \mathcal{L}(T) \cup \{\perp\}$, where $\ell_i = \perp$ denotes the fact that e_i is not relevant information. To accomplish this, we exploit a ground-truth set of documents, and adopt a supervised-learning approach that is similar to the one used for the categorization task (Section 3.1). Our approach basically consists in first building suitable (numerical) representations for entities in $\mathcal{E}^*(T)$ (*entity-feature builder* module), and then classifying them based on such a representation (*entity classifier* module). We discuss the details of such two steps in the following.

Entity-feature builder. We resort to the well-established *word embedding* technique, whose goal is to represent textual units (words, sentences, paragraphs, etc.) by means of numerical vectors such that (semantically) similar units are assigned vectors that are close in the corresponding vector space [4].

Our intuition is that entities of the same type share semantically similar contexts in the various garnishment documents where they appear, i.e., entities of the same type are likely mentioned in sentences/paragraphs that are semantically similar to each other, at least for what concerns a number of key, discriminating terms. For instance, all entities recognized as *debtor* will likely be mentioned in proximity of terms like "owe to", or "seizure". Motivated by this, in our entity-feature builder we adopt the *paragraph vector* approach [23], which is an extension of *word2vec* [31] suitable for learning representations for words and their contexts simultaneously, The basic word2vec learns a model that is able to assign vector representations to single words. Paragraph vector, instead, learns vector representations for both single words and paragraphs (or whatever proper textual unit, such as phrases, sentences, entire documents), where that word appears in, producing a more general model. This way, paragraph vector learns a more general model, which allows to assign vectors to any (previously unseen) combination of (seen) words.

More precisely, the *training phase* of our entity-feature builder takes every document $\hat{D}$ within the given ground truth $\mathcal{G}$, and, for every ground-truth entity $\hat{e}$ of $\hat{D}$, builds a context $\omega(\hat{e})$ by considering every mention of $\hat{e}$ within $\hat{D}$ and concatenating all windows of words that are in the neighborhood of such mentions. In particular, for every mention $m_{\hat{e}}$, the corresponding window $W(m_{\hat{e}})$ is given by the W_r terms preceding $m_{\hat{e}}$ and the W_r terms following $m_{\hat{e}}$ ($W_r = 10$ in our implementation). The context $\omega(\hat{e})$ is thus computed by concatenating the $W(m_{\hat{e}})$ windows, for all mentions $m_{\hat{e}}$. The set $\{\omega_{\hat{e}}\}_{\hat{e}\in\hat{D},\hat{D}\in\mathcal{G}}$ of all these contexts form the base paragraphs to be used for training paragraph vector, and ultimately yielding what in GarNLP is referred to as the *entity-feature model* (Figure 1). At *inference* time, given a document D and an entity e extracted from D, the context $\omega(e)$ of e is first built (again, by concatenating the windows $W(m_e)$ of all mentions of e within D), and the context $\omega(e)$ is provided to the trained entity-feature model, which ultimately outputs the desired vector representation for e. We denote such a vector by $\mathbf{p}(e)$.

Entity classifier. This module takes the numerical representations $\mathbf{p}(e)$ of every entity $e \in \mathcal{E}^*(T)$, and employs standard supervised-learning, for both training and inference (more details in Sections 4–5). Specifically, we formulate the entity-classification problem as a multi-class classification one. As only a tiny fraction (2%) of our dataset contained entities with multiple (i.e., creditor and lawyer) roles, the adoption of a more sophisticated formulation such as multi-label classification was not considered as necessary. In the training phase the numerical representations $\{\mathbf{p}(e)\}_{e\in\mathcal{E}^*(T)}$, along with their ground-truth labels, are exploited to build a *document-classification model*. Such a model is then exploited at inference time to assign a label to unseen entities.

4 Implementation and Technology

In this section we discuss implementation details and technology behind the proposed GarNLP. We implemented the OCR component and the categorizer

in Python, while the information-extractor component is written in Scala. In the following we report such details for each GarNLP's component one by one.

OCR. Every page of a (non-textual) garnishment document is preliminarily converted to a .png image by making use of ImageMagick (www.imagemagick.org) and GhostScript (www.ghostscript.com). After that, every page is OCRized by employing Tesseract (github.com/tesseract-ocr/tesseract) as a library wrapped in ctypes (docs.python.org/3/library/ctypes.html).

We adopt a number of tricks to improve the OCR performance. Specifically, we first apply a *blur* filter to enhance the connectedness of pixels within the same character, then a *maxfilter* to drop the noisy background pixels, and finally an *unsharp mask* to make the overall image tidier. We also attempted a language-model-based approach to fix spelling mistakes. Specifically, we trained a character-level language model on a mixture of a general Italian corpus and a corpus extracted from a subset of our documents where the OCR quality was high (as measured by the percentage of tokens that are part of a dictionary). The resulting language model was applied on the extracted text, and characters were corrected whenever the prediction of the language model would differ from the actual character and the confidence of the prediction exceeded a threshold. This helped fix many common spelling mistakes. During the development of GarNLP, however, the OCR software was upgraded from Tesseract v3 to Tesseract v4. Version 4 of Tesseract includes an OCR which is based on an LSTM neural network applied to the output of a convolutional network. This helps the model learn at once both the graphical features of the text and the statistical properties of the language of the document. In some sense, the LSTM OCR already features a language model as part of its structure. After switching to Tesseract v4, applying an additional language model on top of the Tesseract output was not giving significant improvements anymore, so it was removed from the final version of GarNLP.

Document-feature builder. The feature set associated with a document is the set $T(h)$ of terms of interest defined in Equation (1). Apart from term frequency and unigrams, we also experimented with *tf-idf* and n-grams (up to $n = 5$), without noticing any significant difference.

Document classifier. For document categorization we use a classic *Logistic Regression* classifier (although we have tried several others; see Section 5). To this purpose, the module involves scikit-learn (scikit-learn.org/stable) and pandas (pandas.pydata.org) libraries. The scikit-learn library is exploited in both training and inference, while pandas is utilized in training only, just to pass the datasets around. To train the *Logistic Regression* classifier we use a multinomial loss (as the problem at hand is multi-class), with newton-cg as a solver (i.e., an implementation of the *Newton-Conjugate-Gradient* method).

NER. We use the well-established StanfordNER library (nlp.stanford.edu/software/CRF-NER.html), which is part of the StanfordCoreNLP suite. It implements a *conditional random field* sequence model, together with additional well-engineered features, and provides models to recognize seven NER cat-

egories (i.e., location, person, organization, money, percent, date, time), for four different languages (i.e., English, Chinese, German, and Spanish). The library gives also the chance of training a model on another language. For our use case, we use the Italian model offered within Tint (`http://tint.fbk.eu`), a (`Java`-based) pipeline for NLP in Italian that is built on top of Stanford-CoreNLP. We also perform an ad-hoc retraining on a random sample of our dataset of garnishment documents, and the resulting model is recognized as our best one (see Section 5, Table 5).

Entity annotator. As for the paragraph-vector technique, we use the *Doc2Vec* implementation within the Deeplearning4J library (`deeplearning4j.org`), for both training and inference. The parameters used for training (after an extensive tuning) are: 0.025 as learning rate, 10^{-4} as minimum learning rate, 500 as batch size, 100 as number of epochs, 100 as dimensionality of the output $\mathbf{p}(\cdot)$ vectors, 10 as context-window radius W_r. As far as the entity classifier, we perform both training and inference with the well-known Weka suite (`www.cs.waikato.ac.nz/ml/weka`). Although we experimented with a variety of well-established classifiers (including *Naïve Bayes*, *Random Forest*, *LinearSVM*, *J48*), we ultimately stick to *Logistic Regression*, as it yielded the most accurate results in our assessment. Specifically, we use the Weka wrapper of the *LibLINEAR* library [18], available at `wiki.pentaho.com/display/DATAMINING/LibLINEAR`, and set *L2-regularized logistic regression* as a solver.

Multilinguality. Our GarNLP framework is in great part language-independent, given that most of its components and algorithms make little or no assumption on the language of the input text. As for OCR, the final version of GarNLP employs Tesseract 4, without further use of additional language models to make corrections. The document-categorization module is completely language independent: it applies a logistic-regression classifier on top of frequent-word features. For what concerns the information-extraction module, its main steps also naturally work on multiple languages. In fact, training the NER model and the paragraph-vector model obviously requires language-specific data (documents in the desired language), because not all features might be equally useful for every language. However, the NER library that we employed, i.e., StanfordNER, is designed to work with multiple languages, and provides pretrained models for several major languages, as well as support to retrain the model on the desired input data. As far as paragraph vector, it also requires language-specific retraining, given that it essentially exploits word collocation, and different languages might have different syntactic structures. However, like NER, training a paragraph-vector model on a desired language (or on multiple languages) is rather easy: this is one of the reasons why word2vec-like models have been employed in machine translation between language pairs [30].

4.1 Industrialization

The GarNLP framework has been successfully deployed in production and is currently being used to enhance the work of the UniCredit garnishment office.

The industrialized version of GarNLP is actually provided with a graphical user interface, which the garnishment operators employ to monitor the behavior and performance of the framework. Such a user interface displays, for any processed document, the assigned category label and the various information items extracted from the text; in addition, the operator is allowed to visualize the original document, so as to be able to compare the information-extraction output with the original text. The user interface includes some functionalities that may facilitate this comparison, such as semi-dynamical highlight of the extracted information, possibility to scroll the document up and down to better check all the extracted information fields, possibility (depending on document type) to copy/paste data from the original document, so as to correct the output of the information extractor.

5 Experiments

Table 1: Dataset characteristics.

month	*#documents*	*category*	*%documents*	*actors*	*size*
M1	19 652	*collection agency* (NEQ)	49%	*creditors*	98 586
M2	21 827	*private* (NPR)	18%	*debtors*	79 299
M3	21 458	*renunciation* (RNC)	16%	*lawyers*	62 628
M4	17 586	*other* (OTH)	9%		
M5	21 039	*assignment* (ASS)	6%		
		authority (treasurer) (NET)	1%		
		authority (no treasurer) (NEN)	1%		

We evaluated the performance of GarNLP on a real-world use case, i.e., supporting the legal office of UniCredit, a large pan-European commercial bank, in handling a garnishment process. We used a (proprietary) *real* dataset, i.e., a random sample of 101 562 garnishment documents received by the bank during five months in 2018. The documents are in Italian, enriched with ground-truth labels for the two tasks, and span 7 categories. The *ASS* and *RNC* categories include practices with a known final outcome (i.e., *assignment* and *renunciation*, respectively). The $N*$ categories cover practices for which no outcome has been ruled yet: *NPR* includes *private* subjects, *NEQ* the practices involving a big Italian *collection agency*, *NET* and *NEN* public authorities for which the partner bank respectively is and is not treasurer. The *OTH* category collects all remaining cases, e.g., optional information requests or reminders. The main characteristics of the dataset are summarized in Table 1, where we report: number of documents per month, percentage of documents per category, number of actors (for every actor type) within the whole set of documents.

Next we report separate experiments for the two tasks of GarNLP.

Document categorization. We evaluated the document categorizer proposed for GarNLP, which we dub GarNLP-Cat, by comparing it with 3 baselines that use simpler strategies (such as handcrafted key terms) to build the feature vocabulary and/or assign a category:

Table 2: Document categorization (Test: M5): GarNLP-Cat vs. baselines.

training	*method*	*accuracy*
–	HandFirst-Cat	0.727
M1–M4	HandSup-Cat	0.92
–	HandFreq-Cat	0.899
M1–M4	GarNLP-Cat	**0.986**

Table 3: Document categorization (Train: M1/M4; Test M5): varying classifiers in GarNLP-Cat.

classifier	*accuracy*
Decision Trees	0.947
Passive Aggressive	0.959
Extra Trees	0.963
Perceptron	0.953
SGD	0.963
Logistic Regression	**0.986**

Baseline #1: Handcrafted Terms + First Found (HandFirst-Cat). A completely unsupervised method, which handpicks a few representative terms for each category, and determines the category of a document as the one associated (by majority) with the first ten representative terms in the text. The rationale is that the start of a document contains enough information for categorization.

Baseline #2: Handcrafted Terms + Most Frequent Found (HandFreq-Cat). Another unsupervised method, similar to the previous one. The only difference here is that the category of a document is inferred by computing the frequencies of all representative terms in the handcrafted vocabulary, and picking the category associated with the most frequent term.

Baseline #3: Handcrafted Terms + Supervised Classification (HandSup-Cat). A supervised method, which still employs the category representation based on hand-picked representative terms, but exploits ground-truth annotations to train a supervised-classification model on the term-frequency vectors of the documents in the training set. As a classification algorithm, we employ the one that was finally chosen for our main method, i.e., *Logistic Regression*.

Results. We tested GarNLP-Cat and the baselines on the same dataset, i.e., the fifth month (M5) in our document collection (21 039 documents). The supervised methods were trained on the first four months of the collection (M1–M4), using 10-fold cross-validation for parameter tuning. Table 2 compares the four methods in terms of *accuracy*, i.e., the percentage of correctly-classified documents in the test set. GarNLP-Cat considerably outperforms all the baselines, especially the unsupervised ones. We achieved this result employing *Logistic Regression*. Thus our feature building strategy is much more effective than the handcrafted vocabulary. We also experimented with other off-the-shelf classification algorithms, which are reported for completeness in Table 3.

Information extraction: actor identification. We start our discussion of the information-extraction results by focusing on the identification of the main actors involved in a garnishment process: *creditor*, *debtor*, and *lawyer*.

Table 4: Actor identification: classification results of proposed GarNLP-IE and Freq-IE baseline (Train: M1–M4; Test: M5).

method	*accuracy*
Freq-IE	0.809
GarNLP-IE	**0.926**

Table 5: Actor identification: Precision (P), Recall (R), and F1 of GarNLP-IE with varying the NER model (Train: M1/M4, Test: M5).

actor	Category: ALL						Category: NEQ					
	NER: Tint			NER: GarNLP			NER: Tint			NER: GarNLP		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
creditor	0.830	0.301	0.442	0.763	0.322	0.453	0.990	0.696	0.817	0.878	0.706	0.782
debtor	0.748	0.320	0.448	0.781	0.426	0.551	0.874	0.645	0.742	0.859	0.718	0.782
lawyer	0.783	0.662	0.717	0.758	0.672	0.715	0.985	0.671	0.798	0.981	0.676	0.800

Baseline. We compared the information extractor of GarNLP described in Section 3.2 – here termed GarNLP-IE – with a baseline that is still supervised, but employs a simpler *frequentist* feature model. Specifically, it extracts the *textual context* of a candidate actor in a document, i.e., a person or organization mentioned in the text and identified via NER, by concatenating all the text windows around its mentions in the text. We tested windows of variable radius W_r around each mention, and a different number W_n of windows, considering (i) the case where left and right context of a mention are kept separate, and (ii) the case where left and right context are merged. We denote these two cases by $W_n = 2$ (left and right contexts separated) and $W_n = 1$ (left and right contexts merged). We extract from each context a *bag of features*, considering lemmatized non-stopword terms, bigrams, or both, and we retain as feature vocabulary the union of the top-K features of each type that occur most frequently in each class in the training dataset. We represent the context of a candidate actor as the frequency vector of the features included in the vocabulary. We dub this baseline Freq-IE.

Results of GarNLP-IE vs. Freq-IE baseline. We evaluated the proposed GarNLP-IE and Freq-IE in the classification task performed within the GarNLP entity classifier: we trained on (M1–M4) with 10-fold cross-validation, and tested the two competing methods on M5. Table 4 reports accuracy for the best configuration of each competitor, i.e., $\{$*lemmas*, $W_r = 10$, $W_n = 2$, $K = 25$, *Logistic Regression*$\}$ for Freq-IE, and $\{W_r = 10, N = 100$, *batch size*=500, *epochs*=100, *Logistic Regression*$\}$ for GarNLP-IE (parameter tuning is discussed in Tables 6–7). These results attest the superiority of GarNLP-IE, which outperformed Freq-IE by more than 12%.

Results with varying the NER model. We further tested GarNLP-IE (still training on M1–M4 and testing on M5), varying the NER model to assess its impact on performance. In addition to using the Italian one provided by Tint [34] for StanfordNER [19], we trained our ad-hoc model on a random sample of the training dataset, with the aim of better capturing the peculiar language used in garnishment documents. Table 5 reports precision, recall, and F1 for every actor label, on the whole test set (ALL) and on the subset

Table 6: Actor Identification: parameter tuning of the proposed GarNLP-IE method.

W_r	N	*batch size*	*epochs*	*J48*	*Naïve Bayes*	*Random Forest*	*Logistic Regr.*
5	50	500	20	0.545	0.569	0.681	0.849
		500	50	0.504	0.388	0.741	0.856
	100	500	20	0.693	0.686	0.772	0.81
10	50	500	20	0.512	0.39	0.749	0.82
		500	50	0.574	0.48	0.733	0.849
	100	500	50	0.646	0.668	0.778	0.91
		500	100	0.735	0.714	0.886	**0.926**
		1000	100	0.604	0.52	0.73	0.883

of test documents in the most frequent category (NEQ, which approximately spans half of the data).

On the whole test set (ALL), the inital model (Tint) achieves acceptable precision (average 0.784), while recall is lower (average 0.427), especially for the creditor and debtor classes. In our use case the obtained precision met the requirements of the partner institution, whereas we were explicitly requested to improve recall. From a recall-oriented perspective, we consider the results obtained with the GarNLP NER model better than those achieved with Tint. In fact, the average recall on ALL increases by 11% – from 0.427 to 0.473. This comes at the cost of a loss of 2.5% in average precision (from 0.787 to 0.767), which we consider a reasonable price to pay. The improvement in performance is especially noticeable for the debtor class, where recall increases by 33%, and F1 increases by 23%. On the NEQ class, which accounts for almost half of our dataset, our GarNLP NER model also achieves a small improvement over Tint in terms of average recall (4.3%), at the cost of a bit more significant loss (5%) in average precision (with F1 staying basically the same).

Parameter tuning of proposed GarNLP-IE. We tuned the parameters of paragraph vectors (radius W_r of the context window around each mention, dimensionality N of the paragraph vectors), and two hyperparameters of the SGD algorithm used to train the paragraph-vector model (batch size and number of epochs), by experimenting with the same test settings as before, i.e., we trained on M1–M4 and measured classification accuracy with 10-fold cross-validation. We also varied the classifier used in the entity-classifier module. Table 6 reports the results of parameter tuning for our GarNLP-IE, showing that, using $N = 100$ and a *Logistic Regression* classifier, it achieves the best results, while the other parameters do not seem to influence performance much.

Parameter tuning of Freq-IE baseline. For completeness, we also report in Table 7 the parameter tuning for the Freq-IE baseline. The involved parameters were: type of feature, radius W_r of the text window extracted around each mention, whether the left and right window around a mention are merged ($W_n = 1$) or not ($W_n = 2$), and number K of top-frequent features in the vocabulary for each class and for each feature type. The simplest, lemma-based vocabulary achieves the best accuracy (0.8 for $W_r = 10$ and $W_n = 2$, with *Logistic Regression*). Using bigrams alone yields worse performance, and

Table 7: Actor Identification: parameter tuning of the Freq-IE baseline.

W_r	W_n	K	Features: Lemmas				Features: Lemmas & Bigrams			
			J48	*Naïve Bayes*	*Random Forest*	*Logistic Regr.*	*J48*	*Naïve Bayes*	*Random Forest*	*Logistic Regr.*
5	1	25	0.749	0.716	0.771	0.768	0.728	0.707	0.757	0.762
		50	0.744	0.709	0.768	0.777	0.737	0.682	0.756	0.781
	2	25	0.761	0.712	0.770	0.791	0.759	0.701	0.768	0.789
		50	0.766	0.694	0.773	0.794	0.757	0.684	0.768	**0.809**
10	1	25	0.730	0.705	0.755	0.767	0.729	0.700	0.750	0.756
		50	0.736	0.701	0.758	0.775	0.733	0.664	0.749	0.772
	2	25	0.754	0.701	0.773	**0.809**	0.747	0.681	0.760	0.793
		50	0.755	0.671	0.757	**0.800**	0.742	0.682	0.759	0.790

taking both does not improve with respect to the lemma-only case (results are omitted for the sake of brevity).

Information extraction: identifying codes, dates, amounts. We conclude our report on information extraction by discussing the performance of GarNLP on the remaining attributes other than the actor labels. These include some codes (*authority*, *court*, *injunction number*, and *RGE*), the *amount*, and the *hearing date*. Codes were extracted with simple regular expressions. For the amount and hearing date, we used the same method designed for actor identification: we trained a supervised classifier on the paragraph vectors of mentions (the *money* and *date* tags extracted by NER). Table 8 reports the accuracy on M5 (with supervised extractors trained on M1–M4). GarNLP performs well in the extraction of codes, with the exception of injunction number, which we will investigate in future work. We perform well on amount and hearing date, although the latter needs to be improved: by error analysis we found that accuracy on the hearing date is affected by not-so-good NER performance in the extraction of date tags.

Table 8: Performance of GarNLP on identification of codes, dates, and amounts.

entity	*accuracy* (%)
authority	99.32
injunction #	40.09
court	95.18
amount	91.06
RGE	78.32
hearing date	82.63

Scalability. We tested the scalability of the proposed GarNLP framework by running the complete pipeline (including OCR, document categorization, and information extraction) on document sets of different sizes. The results of this experiment are reported in Table 9. The table shows that the total running time is linear in the number of documents, with an average time per document being roughly constant and equal to slightly more than 10 seconds. This attests the high scalability of our proposal. In this regard, note also that the reported times refer to a single-machine single-core execution. However, the computation can be easily parallelized, as executions of the pipeline on different documents are independent on each other.

Table 9: Scalability: times of running the overall GarNLP pipeline (including OCR, document categorization, and information-extraction) on different document-set sizes.

# documents	*total time (s)*	*avg time per document (s)*
100	1 206	12.06
1 000	11 326	11.33
10 000	117 056	11.71

Qualitative evaluation. A qualitative evaluation of GarNLP was performed during the industrialization phase. The final deployment was preceded by a *progressive roll-out* of two weeks, during which the framework was employed in a *controlled go-live* mode by the operators of the garnishment office. In this phase the operators manually monitored a sample of the incoming document feed, using the graphical user interface to correct the output of the information extractor, when needed. First, the operators provided a quantitative report of the effectiveness of the tool, measured in terms of *slot error rate* (SER) [25], which is a measure of the cost for the users of the tool, to make different types of errors, i.e., insertions, deletions, or substitutions. The choice of the SER metric was motivated by the request of the operators to have a single measure of performance that deals with the three possible types of error simultaneously, as opposed to employing the more widely-adopted precision and recall, which respectively track substitution and insertion errors, and substitution and deletion errors. In their pre-deployment experiments the operators achieved a SER of 31.8%, which roughly means that slightly less than 70% of the information extracted by the framework was correct and did not require manual intervention. Please note that this result is not directly relatable to the performance reported in our manuscript, because the pre-roll-out experiments were conducted on new documents.

6 Related Work

General-purpose NLP. Natural Language Processing (NLP) is a subfield of artificial intelligence whose goal is to automatically process pieces of text written in human (natural) language. NLP includes many different tasks, such as marking up syntactic and semantic elements (e.g., named entity recognition (and disambiguation), part-of-speech (POS) tagging, syntactic parsing, semantic role labeling), language modeling, sentiment analysis, and more. Every prominent NLP task – in its domain-agnostic formulation – has received considerable attention in the literature [22, 33].

From a system-oriented perspective, there have been attempts to incorporate methods for multiple NLP tasks into a single suite. These include the popular *Stanford Core* [26] and *Natural Language ToolKit (NLTK)* [7], respectively a Java- and a Python-based framework of tools for processing various natural languages, including libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries and easy-to-use interfaces to lexical resources such as WordNet. Similar in spirit to Stanford Core and NLTK are *Apache Lucene and Solr*

(`https://lucene.apache.org`) and *Apache OpenNLP* (`http://opennlp.apache.org`). Other NLP systems include *The Curator (Illinois)* [15], a management framework to easily incorporate any third-party components, and distribute components across multiple machines; *Hermes* [10], a tool for large-scale NLP, mainly focused on (Wikipedia-based) named entity disambiguation; *PASTA* [16] and *ToPIC* [35], two distributed self-tuning frameworks for general and topic-model-based text clustering, respectively. Orthogonally, there have also been efforts focused on the efficiency of information-extraction pipelines [43].

Although exhibiting generality and reusability, general-purpose NLP solutions overlook the peculiarities of a specific application domain, for which designing ad-hoc methodologies is the only viable option.

Domain-specific NLP. Numerous NLP pipelines specifically tailored to a single application domain have been proposed [39].

Wachsmuth *et al.* [42] devise a framework to efficiently find and analyze market forecast information, by means of retrieval, extraction and NLP techniques. Specifically, given a corpus of (Web) documents, Wachsmuth *et al.*'s framework extracts time and money information, and use support-vector classification to identify sentences that represent market statements. The statements' subjects are then found by relating recognized named entities to the time and money information, and the resulting information is eventually normalized and aggregated to effectively present it to the user.

ArguAna is a research project that was funded by the German Federal Ministry of Education and Research (BMBF), and originally ran from 2012 to 2014, with a follow-up during 2016–2019.[4] The project aimed at the development of text-analysis algorithms for fine-grained opinion mining from customer product reviews. In particular, a focus was on the analysis of the sequence of single arguments in a review in order to capture and interpret the review's overall argumentation. The main outcome of the project was a text-analysis pipeline to tackle the underlying text-classification and information-extraction tasks. The main steps of the pipeline are as follows. First, the body of each review text is segmented into its single subsentence-level discourse units. Every unit is classified as being either an objective fact, a positive, or a negative opinion. Discourse relations between the units are then extracted as well as products and aspects the units are about, together with their attributes. Finally, a sentiment score in the sense of the review's overall rating is predicted. The techniques employed in the various steps of the pipeline have been presented in several research works, including [40, 41, 44].

Another domain that has observed the proliferation of ad-hoc NLP frameworks is the healthcare/biomedical one [5], where the attention has been, among others, on tasks like text clustering [36] and data visualization [17].

Further attempts have focused on various other domains, such as, e.g., microblogging [9], investigative journalism [45], and cybersecurity [47].

Being specifically tailored to domains other than garnishment, all those pipelines focus on different inputs, and, especially, different customizations of

[4] `http://www.arguana.com`, `https://webis.de/research/arguana-for-the-web.html`

tasks and analyses than the ones required in the garnishment setting. This way, they are not (easily) adaptable to our context.

NLP in the legal domain. To the best of our knowledge, there is no work in the legal-NLP literature on designing domain-specific NLP pipelines, neither for garnishment, nor for any other legal subdomains. Related – but clearly different – problems include ontology learning, construction of knowledge resources, and semantic processing of legal texts. In the following we briefly overview the state of the art in such problems.

Learning ontologies from unstructured data for the specific context of the legal domain has been active for more than two decades. Already in 1994, Valente and Breuker [38] discussed role and benefits of ontologies in AI & Law. Since then, numerous frameworks and methods for learning ontologies from text have been proposed [1,12,14,24]. For a comprehensive review of the use of ontologies for legal-knowledge systems, see the survey of Casellas *et al.* [13], and the workshop of Francesconi *et al.* [21].

As far as knowledge resources for legal text processing, Francesconi *et al.* [20] propose a (semi-)automatic NLP-based approach for multilingual legal knowledge acquisition and modelling. Bosca and Dini [11] devise an ontology-induction method for individual laws, based on corpora comparison. Ajani *et al.* [2] propose the Legal Taxonomy Syllabus and a methodology to manage the conceptual representation of European laws. Bonin *et al.* [8] present a term-extraction approach to discriminate legal terms from regulated-domain terms.

A number of efforts have also been devoted to semantic processing of legal texts. Bartolini *et al.* [6] study the problem of automatically enriching legal texts with semantic annotations, by classifying law paragraphs based on regulatory content. Francesconi *et al.* [21] propose strategies for multilingual legal knowledge modelling. Mazzei *et al.* [27] present an approach for annotating modificatory provisions. Spinosa *et al.* [37] focus on automatic consolidation of Italian legislative texts. Wyner and Peters [46] devise a text-analysis tool for legal researchers, which integrates legal and linguistic resources.

Finally, in terms of SW libraries/tools, it is worth to mention *IusExplorer*,[5] a legal search engine gathering different law sources (case laws, legislation, jurisprudence, etc.) in the Italian language, and the *LexNLP* `Python` package.

7 Conclusions

We have presented GarNLP, a natural-language-processing pipeline for the garnishment domain, supporting the work of a garnishee in two main tasks, i.e., document categorization and information extraction. Extensive experiments on a real-world use case and dataset have attested the high quality of the methodologies at the core of the proposed pipeline.

In the future we plan to improve the performance of GarNLP on the considered tasks, to add methods for extracting further information, and to extend the framework by incorporating models for more languages.

[5] `https://www.iusexplorer.it/`

References

1. Agnoloni, T., Bacci, L., Francesconi, E., Peters, W., Montemagni, S., Venturi, G.: A two-level knowledge approach to support multilingual legislative drafting. In: Proc. Conf. on Law, Ontologies and the Semantic Web (2009)
2. Ajani, G., Boella, G., Lesmo, L., Martin, M., Mazzei, A., Radicioni, D.P., Rossi, P.: Semantic processing of legal texts. chap. Multilevel Legal Ontologies. Springer (2010)
3. Allwood, W.: Expert systems in law. A jurisprudential inquiry. By Richard E. Susskind. The Cambridge Law Journal **47** (1988)
4. Almeida, F., Xexéo, G.: Word embeddings: A survey. CoRR **abs/1901.09069** (2019)
5. Ananiadou, S., Mcnaught, J.: Text Mining for Biology And Biomedicine. Artech House, Inc. (2005)
6. Bartolini, R., Lenci, A., Montemagni, S., Pirrelli, V., Soria, C.: Automatic classification and analysis of provisions in Italian legal texts: A case study. In: R. Meersman, Z. Tari, A. Corsaro (eds.) Proc. OTM Work. (2004)
7. Bird, S., Loper, E.: NLTK: The natural language toolkit. In: ACL Conf. (Poster and Demonstration) (2004)
8. Bonin, F., Dell'Orletta, F., Venturi, G., Montemagni, S.: Singling out legal knowledge from world knowledge: An NLP-based approach. In: LOAIT Work. (2010)
9. Bontcheva, K., Derczynski, L., Funk, A., Greenwood, M.A., Maynard, D., Aswani, N.: TwitIE: An open-source information extraction pipeline for microblog text. In: RANLP Conf., pp. 83–90 (2013)
10. Bordino, I., Ferretti, A., Firrincieli, M., Gullo, F., Paris, M., Pascolutti, S., Sabena, G.: Advancing NLP via a distributed-messaging approach. In: IEEE Big Data, pp. 1561–1568 (2016)
11. Bosca, A., Dini, L.: Semantic processing of legal texts. chap. Ontology Based Law Discovery. Springer (2010)
12. Breuker, J., Hoekstra, R.: Epistemology and ontology in core ontologies: FOLaw and LRI-Core, two core ontologies for law. Phycologia (2004)
13. Casellas, N.: Legal ontology engineering: methodologies, modelling trends, and the ontology of professional judicial knowledge. Springer (2011)
14. Cimiano, P., Völker, J.: Text2Onto – A framework for ontology learning and data-driven change discovery (2005)
15. Clarke, J., Srikumar, V., Sammons, M., Roth, D.: An NLP curator (or: How I learned to stop worrying and love NLP pipelines). In: LREC Conf., pp. 3276–3283 (2012)
16. Di Corso, E., Cerquitelli, T., Ventura, F.: Self-tuning techniques for large scale cluster analysis on textual data collections. In: SAC Conf., pp. 771–776 (2017)
17. Di Corso, E., Proto, S., Cerquitelli, T., Chiusano, S.: Towards automated visualisation of scientific literature. In: ADBIS Conf., pp. 28–36 (2019)
18. Fan, R., Chang, K., Hsieh, C., Wang, X., Lin, C.: LIBLINEAR: A library for large linear classification. Journal of Machine Learning Research **9** (2008)
19. Finkel, J.R., Grenager, T., Manning, C.: Incorporating non-local information into information extraction systems by gibbs sampling. In: Proc. ACL Conf. (2005)
20. Francesconi, E., Montemagni, S., Peters, W., Tiscornia, D.: Semantic processing of legal texts. chap. Integrating a Bottom-Up and Top-Down Methodology for Building Semantic Resources for the Multilingual Legal Domain. Springer (2010)
21. Francesconi, E., Montemagni, S., Peters, W., Tiscornia, D. (eds.): Semantic Processing of Legal Texts: Where the Language of Law Meets the Law of Language. Springer (2010)
22. Khurana, D., Koli, A., Khatter, K., Singh, S.: Natural language processing: State of the art, current trends and challenges. CoRR **abs/1708.05148** (2017)
23. Le, Q.V., Mikolov, T.: Distributed representations of sentences and documents. In: Proc. ICML Conf. (2014)
24. Lenci, A., Montemagni, S., Pirrelli, V., Venturi, G.: Ontology learning from italian legal texts. In: Proc. Conf. on Law, Ontologies and the Semantic Web (2009)
25. Makhoul, J., Kubala, F., Schwartz, R., Weischedel, R.: Performance measures for information extraction. In: In Proceedings of DARPA Broadcast News Workshop, pp. 249–252 (1999)

26. Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J.R., Bethard, S., McClosky, D.: The Stanford CoreNLP natural language processing toolkit. In: ACL Conf. (System Demonstrations, pp. 55–60 (2014)
27. Mazzei, A., Radicioni, D.P., Brighi, R.: NLP-based extraction of modificatory provisions semantics. In: Proc. ICAIL Conf. (2009)
28. McCarty, L.T.: Deep semantic interpretations of legal texts. In: ICAIL Conf. (2007)
29. McCarty, L.T.: Remarks on legal text processing – parsing, semantics and information extraction. (2009)
30. Mikolov, T., Le, Q.V., Sutskever, I.: Exploiting similarities among languages for machine translation (2013)
31. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: NIPS Conf. (2013)
32. Nadeau, D., Sekine, S.: A survey of named entity recognition and classification. Linguisticae Investigationes **30**(1), 3–26 (2007)
33. Otter, D.W., Medina, J.R., Kalita, J.K.: A survey of the usages of deep learning in natural language processing. CoRR **abs/1807.10854** (2018)
34. Palmero Aprosio, A., Moretti, G.: Italy goes to Stanford: A collection of CoreNLP modules for Italian. ArXiv (2016)
35. Proto, S., Di Corso, E., Ventura, F., Cerquitelli, T.: Useful ToPIC: Self-tuning strategies to enhance Latent Dirichlet Allocation. In: IEEE Big Data Conf., pp. 33–40 (2018)
36. Renganathan, V.: Text mining in biomedical domain with emphasis on document clustering. Healthcare Informatics Research **23**(3), 141–146 (2017)
37. Spinosa, P., Giardiello, G., Cherubini, M., Marchi, S., Venturi, G., Montemagni, S.: NLP-based metadata extraction for legal text consolidation. In: ICAIL Conf. (2009)
38. Valente, A., Breuker, J.: Ontologies, the missing link between legal theory and AI and Law. Mathematics of Computation (1994)
39. Wachsmuth, H.: Text Analysis Pipelines - Towards Ad-hoc Large-Scale Text Mining, *Lecture Notes in Computer Science*, vol. 9383. Springer (2015)
40. Wachsmuth, H., Kiesel, J., Stein, B.: Sentiment flow - A general model of web review argumentation. In: EMNLP Conf., pp. 601–611 (2015)
41. Wachsmuth, H., Potthast, M., Al-Khatib, K., Ajjour, Y., Puschmann, J., Qu, J., Dorsch, J., Morari, V., Bevendorff, J., Stein, B.: Building an argument search engine for the web. In: ArgMining@EMNLP Work., pp. 49–59 (2017)
42. Wachsmuth, H., Prettenhofer, P., Stein, B.: Efficient statement identification for automatic market forecasting. In: COLING Conf., pp. 1128–1136 (2010)
43. Wachsmuth, H., Stein, B., Engels, G.: Constructing efficient information extraction pipelines. In: CIKM Conf., pp. 2237–2240 (2011)
44. Wachsmuth, H., Trenkmann, M., Stein, B., Engels, G.: Modeling review argumentation for robust sentiment analysis. In: COLING Conf., pp. 553–564 (2014)
45. Wiedemann, G., Yimam, S.M., Biemann, C.: A multilingual information extraction pipeline for investigative journalism. In: EMNLP Conf., pp. 78–83 (2018)
46. Wyner, A., Peters, W.: Lexical semantics and expert legal knowledge towards the identification of legal case factors. In: Conf. on Legal Knowledge and Information Systems (2010)
47. Xie, T., Enck, W.: Text analytics for security: Tutorial. In: HotSos Conf., pp. 124–125 (2016)

Ilaria Bordino is a researcher at UniCredit, R&D Department. She received her PhD in Computer Engineering in 2010, from Sapienza University of Rome (Italy) and Pompeu Fabra University of Barcelona (Spain). Prior to joining UniCredit, Ilaria was a research scientist at Yahoo Labs, Barcelona, Spain. She has organized the MIDAS (MIning DAta for financial applicationS) Workshop @ECML-PKDD['16-'19].

Andrea Ferretti is a researcher at UniCredit, R&D Department. He received his PhD in Mathematics from Sapienza University (Rome, Italy) in 2009. Before joining UniCredit (in February 2013), he spent a period as a postdoc at

Max Planck Institute for Mathematics in Bonn, Germany (2009-2010), and at University of Lille, France (2010-2011). He also worked as a software engineer for I Mille and Moneyfarm. Andrea was co-Chair of the MIDAS (MIning DAta for financial applicationS) Workshop @ECML-PKDD['18-'19].

Francesco Gullo is a researcher at UniCredit, R&D department. He received his PhD from the University of Calabria, in 2010. Before joining UniCredit, he spent 1.5 years in the University of Calabria, and 4 years in Yahoo Labs, Spain. He served as a workshop co-chair of ICDM'16, and organized several workshops/symposia (MIDAS @ECML-PKDD['16-'19]; MultiClust @SDM'14, KDD'13; 3Clust @PAKDD'12).

Stefano Pascolutti is a software engineer at Google Zurich. Stefano received his PhD in Mathematics (Algebraic Geometry) in 2011, after completing a doctoral program in Sapienza University of Rome (Italy). He next spent a period as a math researcher at the Riemann Center in Hannover, and he was a researcher at UniCredit, R&D Department, from 2013 to 2019. Andrea was co-Chair of the MIDAS (MIning DAta for financial applicationS) Workshop @ECML-PKDD['18-'19].