Data Augmentation Techniques for Legal Text Analytics

by

Maria Papadaki

A thesis submitted to Athens University of Economics and Business in fulfillment of the requirements for the degree of Master in Data Science.

Department of Computer Science

Athens University of Economics and Business

October 2017

Acknowledgements

I would like to thank my supervisor Prof. Ion Androutsopoulos for providing me the chance to get involved in a very interesting NLP task. I would also like to thank Ilias Chalkidis and Achilleas Michos for their valuable guidance throughout the development of the thesis.

Abstract

Data augmentation is the practice of applying transformations to a given training set in order to synthetically expand it. This master thesis involves the implementation and evaluation of the domain-agnostic data augmentation techniques that operate in the input feature space. More concretely, in the context of contract element extraction (Chalkidis et al., 2017a, 2017b), this thesis explores the effectiveness of three domain-agnostic data augmentation methods, namely: adding Gaussian noise, applying interpolation or extrapolation at the word representation level (word embeddings). The performance of the aforementioned augmentation techniques were tested with all LSTM-based contract element extraction methods of Chalkidis et al. (2017b), on a subset of six out of eleven contract elements types considered by Chalkidis et al., focusing mostly on element types with fewer data samples in the (non-augmented) training set of Chalkidis et al. The experimental results of this thesis indicate that interpolation performs better compared to adding Gaussian Noise or applying extrapolation. None of data augmentation techniques considered in this thesis, however, managed to consistently improve the performance of the LSTM-based extractors, and in some cases the performance of the LSTM-based extractors deteriorated, compared to using no data augmentation.

Table of contents

List of Tables and Figures

Tables

Figures

Chapter 1

Introduction

A contract is a legal text which concludes a voluntary, deliberate, and legally binding agreement between two or more competent parties. Several tasks, which are an integral part of the operation of law firms, companies, government agencies etc., involve monitoring contracts. For example, law firms need to inform their clients when contracts are affected by law replacements or amendments, because their contracts may need to be modified accordingly. Termination dates, governing laws, or agreed payments also constitute some of the key elements of contracts, which can summarize them. The aforementioned tasks can be automated by extracting key contract elements. The current methods of contract element extraction, however, require human involvement. Researchers currently attempt to reduce the human involvement in order to render the process of extracting key information contracts less time-consuming and costly.

Chalkidis et al., (2017a, 2017b) studied how contract element extraction can be automated. They devised several contract element extraction methods, which were tested on a dataset of approximately 3500 English contracts they constructed. The dataset is publicly available and it is manually annotated with 11 types of contract elements. It is also encoded for privacy issues. The encoding involved the replacement of each vocabulary word with a unique integer identifier (Androutsopoulos et al., 2000). The word “legislation”, for instance, may have been replaced by the integer ‘1517’. For every token of the dataset, Chalkidis et al. also provide hand-crafted features, such as the part-of-speech (POS) tag, whether or not a token started with a capital letter before the encoding etc. Apart from the aforementioned features, word embeddings per vocabulary word are also provided. Word embeddings are dense real-valued vectors, constructed so that vocabulary words with morpho-syntactic and/or semantic similarity are in close proximity in vector space. Pre-trained word embeddings can be obtained via unsupervised techniques. In the dataset of Chalkidis et al., the word embeddings were pre-trained by applying word2vec (Mikolov et al., 2013b) to an unlabeled dataset of approximately 750,000 contracts. Additionally, POS-tag embeddings were obtained by applying word2vec to a subset of the same unlabeled dataset, with the only difference that the words were replaced by their POS-tags. Chalkidis et al. experimented with manually written rules, trainable linear classifiers and LSTM-based methods, with the latter leading to the best results.

This thesis does not attempt to develop new contract element extraction methods. It explores if and how domain-agnostic data augmentation techniques, namely: adding Gaussian noise, applying interpolation or extrapolation on the word representation level (word embeddings), affect the performance of the LSTM-based methods of Chalkidis et al. Due to time constraints of the thesis, the performance of the aforementioned data augmentation techniques were tested only with the LSTM-based methods of Chalkidis et al.,

as opposed to also applying them to their other methods, focusing on a subset of six out of eleven contract elements types considered by Chalkidis et al., mainly those with the fewer data samples in the non-augmented training set of Chalkidis et al.

The experimental results of this thesis showed that applying interpolation was the method which performed better compared to adding Gaussian Noise or applying extrapolation. None of data augmentation techniques considered in this thesis, however, managed to consistently improve the performance of the LSTM-based extractors, and in some cases the performance of the LSTM-based extractors deteriorated, compared to using no data augmentation.

1.1 Contract Structure and Elements

Chalkidis et al. (2017a) realized that they could benefit from the structure of the contracts and the high frequency of the key contract elements in specific zones of the contracts. According to their observations, a contract starts mostly with a preamble, which contains the contract title (Fig.1 point 1), the start or effective date (Fig.1, point 2) and contracting parties (Fig.1, point 3). The following figure (Fig.1) contains a typical example of a contract preamble.

(1)SUPPLY SERVICES

This agreement is made (2) the 25th day of September 2017 between:

(3) 1. Book Owl LTD., whose office is at 27 Mount Pleasant Ln, Llanrumney, Cardiff CF3 ("Publisher");

(3)2. ABC Papers Inc., whose office is at 83 Abberley Rd, Liverpool L25 9QY ("Supplier");

Figure 1: Typical example of annotated contract preamble.

It is common for the same information to be also provided in a cover page, which precedes the preamble along with a table of contents (also before preamble). The preamble is usually followed by the recitals, which provide background information. The remainder of the contract is organized in clauses, often called 'chapters', 'articles' etc., whose headings indicate their topics. The following figure (Fig.2) contains typical examples of contract clauses.

(9)ARTICLE II – TERMINATION

The supply period will be (4) for four years from the effective date. The agreement is considered to be terminated in (5) September 25, 2021.

(9)ARTICLE III – PAYMENTS & FEES

During the supply period monthly payments should occur. The estimated fees are (6) £50,000.

(9)ARTICLE IV – GOVERNING LAW

This agreement shall be governed and construed in accordance the (7) Laws of England & Wales. Each party hereby irrevocably submits to the exclusive jurisdiction of the (8) courts sitting in Northern London.

Figure 2: Typical example of annotated contract clauses.

Let us now provide some more information about the contract element types that are considered in the work of Chalkidis et al. and, hence, also in this thesis.

Contract Title (Fig.1, point 1): The title indicates the type or even the version of a contract. Extracting the contract title is useful, because it is a piece of information that facilitates the classification of contracts by type and/or clustering multiple versions (e.g., amendments) of the same contract.

Contracting Parties (Fig.1, point 3): Extracting contracting parties facilitates the retrieval of contracts involving particular parties (e.g., by building an index of contracting parties), or even the construction of graphs showing the interdependencies between contractors.

Start, Effective, Termination Dates, Contract Period, Value: According to Chalkidis et al., the start (Fig. 1, point 2), effective and termination dates (Fig. 2, point 5) "specify when the contract was signed, becomes effective and terminates, respectively". The contract period (Fig. 2, point 4) is "the number of working or calendar days the contract will be effective for", while the contract value (Fig. 2, point 6) is "the price of the agreed transaction". These elements are particularly useful. For example, law firms need to know when contracts are about to expire or become effective, in order to notify their clients.

Governing Law, Jurisdiction, Legislation Refs: The governing law (Fig. 2, point 7), jurisdiction (Fig. 2, point 8) and legislation references specify "the country or state whose laws apply, the courts responsible to resolve disputes and the laws the contract depends on, respectively". These elements are also particularly important, because contracts should be revised when the laws that affect them are amended or replaced.

Clause Headings (Fig. 2, point 9)**:** As already mentioned, clause headings indicate the topic of the clauses. Hence, clause headings are useful as they facilitate the identification of clauses where other contract elements are expected to be found. In case a contract does not include a table of contents, clause headings can also be used to construct it.

Extraction Zones (at testing)	Example Clause Heading Words	Contract Elements Typically Included
Cover page and preamble	–	Contract Title, Contracting Parties, Start Date, Effective Date
Term clause	'Term', 'Period', 'Term of Agreement'	Termination Date, Contract Period
Termination clause	'Termination', 'Termination of Agreement'	Termination Date
Governing Law clause	'Governing Law', 'Applicable Law'	Governing Law, Jurisdiction
Jurisdiction clause	'Jurisdiction', 'Jury Trial', 'Venue'	Jurisdiction
Miscellaneous clause	'Miscellaneous', 'Entire Agreement'	Governing Law, Jurisdiction
Contract Value clause	'Lump Sum', 'Salary'	Contract Value
In the text after the recitals, zones starting up to 20 tokens before and ending up to 20 tokens after each line break, not crossing other line breaks		Clause Headings
In the entire contract, zones starting up to 20 tokens before and ending up to 20 tokens after each occurrence of words like 'Act', 'Treaty' etc.		Legislation References

republished from Chalkidis et al. (2017)

Table 1: Extraction zones per contract element type.

Each type of contract element is almost always found in particular types of clauses or other zones of the contract. These zones are called **extraction zones**. For example, start dates can always be found in the preamble or cover page. Attempting to find the start date in other zones could unnecessarily increased the false positives during testing. It would also substantially increase the negative instances during training, resulting in a sizeable imbalance between negative and positive instances. Table 1 summarizes the extraction zones where contract elements of different types can be expected to be found, according to Chalkidis et al. (2017a).

The process of contract element extraction starts with the following steps:

1. Identifying the main parts of each contract (e.g., cover, table of contents, preamble) by using regular expressions.
2. Identifying the clause headings by employing the extractors for this type of contract elements (the F_1-score of the best clause heading extraction method is 0.97).
3. Splitting the text after the recitals into headings.
4. Identifying the topic of each clause using manually crafted lists of indicative words[1].
5. Identifying the extraction zones of each contract element type.

The methods that extract the other types of contract elements (other than clause headings), which will be discussed in following chapters, are then applied only to the corresponding extraction zones of Table 1.

For each test or train contract and each contract element type, the tokens of the corresponding extraction zones that are parts of contract elements of that type (as indicated by the gold annotations) are treated as positive (test or train) instances, whereas the other tokens of the extraction zones are treated as negative (test or train) instances. The provided labeled datasets[2] consisted of 993 contracts (893 training, 100 test) annotated with gold clause headings, and 2,461 contracts (2,111 training, 350 test) with gold annotations for the other 10 types of contract elements. Table 2 provides more information about the number of instances and tokens per element type in the training and test part of the dataset. Consult Chalkidis et al. (2017a) for further details about the dataset and the way it was constructed.

Contract Element Type	Instances	Tokens
Contract Title	4,363	17,282
Contracting Parties	8,235	34,560
Start Date	2,519	10,990
Effective Date	734	3,218
Termination Date	534	2,140
Contract Period	421	1,628
Contract Value	1062	2,714
Governing Law	2,956	15,497
Jurisdiction	1,841	13,095
Legislation Refs	5,997	32,472
Clause Headings	38,269	~183K

republished from Chalkidis et al. (2017)

Table 2: Number of instances and tokens per contract element type.

[1]This lists of indicative words used to detect the topics of the clauses were constructed by inspecting only the training part of the labeled dataset.

[2]The datasets are available at http://nlp.cs. aueb.gr/publications.html.

1.2 The purpose of this work

The purpose of this thesis is to implement and evaluate domain-agnostic data augmentation techniques that operate on the pre-trained word embeddings of the extraction zones (per contract element type), in order to synthetically augment the training set of six of the aforementioned contract element types: contract period, contract value, termination date, jurisdiction, contracting parties and effective date. We evaluate the data augmentation techniques indirectly, by evaluating the performance of the LSTM-based methods of Chalkidis et al. (2017b) with and without applying data augmentation to the training sets.

1.3 Relation to Named-Entity Recognition

Named-entity recognition (NER) is a subtask of information extraction that seeks to identify named entities in texts and classify them into pre-defined categories, such as the names of persons, organizations etc. Existing generic Named-Entity Recognizers are not directly applicable to contract element extraction, at least not without retraining them on contracts and possibly modifying their features and classes. For example, they can typically locate, for instance, a date but they cannot identify its type (start, effective or termination date). Consult Chalkidis et al. (2017a) for further details about the dataset and the way it was constructed.

1.4 Notation

The following table includes the abbreviations used throughout the remainder of this thesis:

Abbreviation	Meaning
NLP	Natural Language Processing
NER	Name Entity Recognition
RNN	Recurrent Neural Network
BIRNN	Bidirectional Recurrent Neural Network
LSTM	Long Short-Term Memory (Model)
BILSTM	Bidirectional Long Short-Term Memory (Model)
CNN	Convolutional Neural Network
LWLM	Latent Words Language Model
POS	Part of Speech

1.5 Outline

The rest of the thesis is organized as follows:

Chapter 2-Contract Element Extraction Methods: Description of the contract extraction methods of Chalkidis et al. (2017a, 2017b) that are used in this thesis.

Chapter 3-Data augmentation: Description of the data augmentation task at hand and the techniques we implemented to synthetically expand the training dataset.

Chapter 4-Experiments: A description of how the experiments of this thesis were conducted and a presentation of the experimental results.

Chapter 5-Conclusions and future work: Summary of the findings of the thesis and ideas for future work.

Chapter 2

Contract Element Extraction Methods

This section presents the contract element extraction methods of Chalkidis et al. (2017a, 2017b), especially their LSTM-based methods, which were employed in order to evaluate the data augmentation techniques. Chalkidis et al. treat contract element extraction as a sequence labeling task, where each element (in our case, word) of a given sequence (in our case, extraction zone) is classified by considering either a window of members around the element being classified or all the elements of the sequence. These contract element extraction methods of Chalkidis et al. involve linear classifiers that consider fixed-length sliding windows of the extraction zones (Chalkidis et al., 2017a), or bidirectional recursive neural networks that consider entire extraction zones (Chalkidis et al., 2017b) instead of sliding windows.

2.1 Linear Classifiers Operating on Sliding Windows

The first contract element extraction methods of Chalkidis et al. (2017a) involved Logistic Regression (McCullagh & Nelder, 1989; Yu et al., 2011) and SVM (Vapnik, 1995; Cristianini & Shawe-Taylor, 2000) classifiers[3], one classifier per contract element type (11 classifiers in total). Each classifier scans each token of the corresponding extraction zones and it either classifies it as positive, if it decides that is part of a contract element of the corresponding contract element type, or as negative (all classifiers are binary). When classifying a token, each classifier considers not only the particular token itself, but also a sliding window of 5-6 tokens around it (11-13 tokens in total). The size of the window depends on the contract element type. The classifiers do not actually take words as input, but feature vectors. Chalkidis et al. (2017a) experimented with three alternatives to map the words to feature vectors:

1. In the first case, each token of the sliding window is represented by a 200-dimensional word-embedding (a dense real-valued vector). The word-embeddings were obtained by pre-training them on an unlabeled dataset of contracts using word2vec[4]. The embeddings of all the tokens in the sliding window were concatenated in one feature vector, given as input to the classifier. The methods which used Logistic Regression and SVM classifiers with word embeddings are called **SW-LR-EMB** and **SW-SVM-EMB**, respectively.
2. In the second case, each token of the sliding window is represented by 17-21 hand-crafted binary features. The first 14 features are all the same for all the contract element types. They are:
    - 4 binary features indicating if the token being classified consists of all upper, all lower, mixed case letters, or contains numbers.

[3] The SCIKIT-LEARN implementations of LR and SVM (http://scikit-learn.org/) classifiers were employed.
[4] The Gensim's (v. 0.12.4) implementation of word2vec (http://radimrehurek.com/gensim/), with 10 minimum occurrences per word and default values for other parameters was employed

- 7 binary features indicating the length of the token. Each feature corresponded to a specific length range, in characters: the first feature to 1-2 characters, the second feature to 3-4 characters, and so on up to the last feature, which corresponded to more than 12 characters.
- 3 binary features indicating if the token is numeric, a special character, or stop-word.

The features that differ per contract element type indicate if the token is frequently a part or near parts of a particular element type. Furthermore, given that the POS tag of each token of the labeled dataset is provided[5] (45 distinct POS tags in total), 45 additional binary features per token can be used, along with previous hand-crafted ones, to indicate the POS tag of the token. The feature vectors of all the tokens in the sliding window are again concatenated in one feature vector, given as input to the classifier. The methods that employ Logistic Regression and SVM classifiers with the hand-crafted and POS-tag features are called **SW-LR-HCF** and **SW-SVM-HCF**, respectively.

3. In the third case, each token of a sliding window is represented a 200-dimensional word embedding, a 25-dimensional POS tag embedding and 17-21 hand-crafted binary features. The word embeddings and the hand-crafted features are the same as in the previous two cases above. The POS-tag embeddings were pre-trained by applying word2vec to 50,000 contracts of the same unlabeled dataset that was used to pre-train the word embeddings; however, the only difference was that the words had been replaced by their POS tags. The methods that employed Logistic Regression and SVM classifiers with word embeddings, POS tag embeddings and hand-crafted features are called **SW-LR-ALL** and **SW-SVM-ALL**, respectively.

	F1-score					
Element Type	SW-LR-EMB	SW-SVM-EMB	SW-LR-HCF	SW-SVM-HCF	SW-LR-ALL	SW-SVM-ALL
Title	0.88	0.87	0.87	0.89	0.91	0.91
Parties	0.85	0.87	0.86	0.88	0.89	0.89
Start	0.85	0.85	0.86	0.83	0.87	0.86
Effective	0.62	0.68	0.60	0.80	0.67	0.72
Termination	0.76	0.69	0.64	0.66	0.76	0.69
Period	0.65	0.67	0.57	0.58	0.67	0.66
Value	0.62	0.64	0.52	0.52	0.62	0.64
Gov. Law	0.91	0.93	0.91	0.92	0.94	0.94
Jurisdiction	0.82	0.82	0.72	0.75	0.81	0.82
Legisl. Refs.	0.80	0.83	0.82	0.85	0.83	0.86
Headings	0.72	0.78	0.78	0.78	0.80	0.80

Table 3: F_1-scores of linear classifiers operating on sliding windows

Table 3 shows the F1-scores of the methods described above. It can easily be observed that, in the vast majority of the element types, the best scores resulted from the combination of linear classifiers with hand-crafted features, word and POS tag embeddings.

[5] The POS tag of each token was obtained via NLTK's (v. 3.2.1) default POS tagger (http://nltk.org/).

2.2 Deep Learning Methods

In addition to simple linear classifiers, Chalkidis et al. (2017) experimented with recurrent neural networks (RNNs) were applied. RNNs iteratively perform the same computation for every element of a sequence (the words of an extraction zone in our case), with the output at each time step depending on the previous computations. Another way to think about RNNs is that they have a "memory" which captures information about what has been calculated so far. An RNN unrolled over time is depicted in its simplest form below (Fig.3):

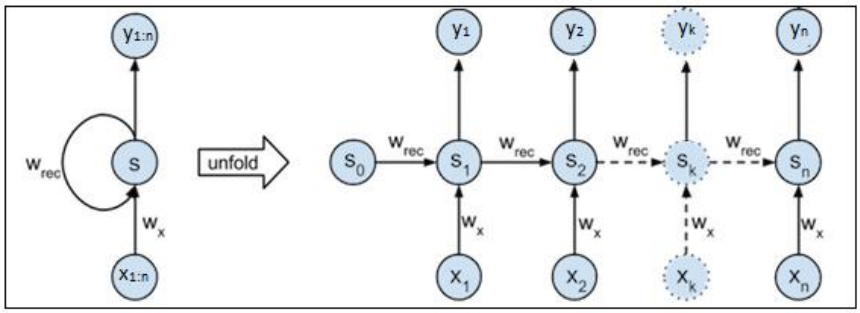


Figure 3: A typical RNN, unrolled.

By unrolled we simply mean that the network is written out for the complete sequence, as if it were non-recurrent. For example, if the sequence is a sentence of five words, the network would be unrolled into a 5-layer neural network, one layer for each word. Notice, however, that the weights of the network would be shared (they would be the same) across all five layers.

RNNs take as input an ordered sequence (of arbitrary length) of n d_{in}-dimensional vectors $x_{1:n} = (x_1, ..., x_n)$,e.g. word embeddings, and produce an ordered sequence of n d_{out}-dimensional vectors $y_{1:n} = (y_1, ..., y_n)$, where each y_i "summarizes" the sub-sequence $x_{1:i} = (x_1, ..., x_i)$.The RNN is defined recursively, by means of a function f (usually a nonlinearity such as sigmoid, tanh or ReLU) taking as input a hidden state vector s_{i-1} and an input vector x_i at time step i and returning a state vector s_i. The state vector s_i is then mapped to an output vector y_i using a simple deterministic function O_R. The initial state vector, s_0, is also an input to the RNN, and it is often assumed to be a zero vector. Given an input sequence $x_{1:n}$, the formulas that a standard RNN uses are the following:

$$s_i = f(w_{rec} s_{i-1} + w_x x_i + b_s),$$
$$y_i = O_R(s_i) = w_y s_i + b_y,$$
$$\forall i \in [1, n],$$
$$x_i \in \mathbb{R}^{d_{in}},$$
$$y_i \in \mathbb{R}^{d_{out}}.$$

A bidirectional-RNN (BIRNN) is an extended version of an RNN, which allows, the prediction for the i_{th} element of a given sequence to be based not only on the previous elements, but also on the following ones. The BIRNN maintains two separate states, s_i^b and s_i^f, for each input position i. The forward state s_i^f, at time step i, is based on $\mathrm{x}_1,...,x_i$, while the backward state s_i^b is based on $\mathrm{x}_n,...,x_i$. The forward and backward states are generated by two different RNNs, which are fed the input sequence $\mathrm{x}_{1:n}$ and its reverse version, respectively. Given an input sequence $\mathrm{x}_{1:n}$, the formulas that a BIRNN uses are the following:

$$s_i^f = f(\mathrm{w}_{rec}^f s_{i-1}^f + w_x^f x_i + \mathrm{b}_s^f),$$
$$s_i^b = f(\mathrm{w}_{rec}^b s_{i-1}^b + w_x^b x_i + \mathrm{b}_s^b),$$
$$y_i = [y_i^f; y_i^b] = [w_y^f s_y^f + b_y^f; w_y^b s_y^b + b_y^b].$$

The BIRNN is very effective for sequence tagging tasks and was introduced to the NLP community by Irsoy and Cardie (2014). RNNs or BIRNNs can also be stacked in layers. Each layer, i.e. RNN (or BIRNN), takes as input the output of the RNN (or BIRNN) below it, except for the first RNN (or BIRNN), which takes as input the sequence of vectors $\mathrm{x}_{i:n}$.

RNNs and BIRNNS that use the simple formulas above face the 'vanishing gradient' problem. The quickly diminishing error signals in the back-propagation process do not allow them to capture long-range dependencies. The problem was explored in depth by Hochreiter (1991) and Bengio et al. (1994). Long Short Term Memory networks (LSTMs) are a special kind of RNN, capable of learning long-term dependencies to a larger extent. They were introduced by Hochreiter & Schmidhuber (1997). In LSTMs, the state vector s_j, at time-step j, is composed of two components, the memory and the hidden state component h_j. The memory component c_j preserves the long-term memory, with the help of gates and prevent error gradients from vanishing, across time. At each time-step, a gate is used to decide how much of the new input should be written to the memory component, and how much of its current content should be forgotten. There are three gates, i, f, and o, controlling for input ,forget ,and output. The forget and input gates control how much of the previous memory and the proposed update should be maintained, respectively, whereas the output gate controls the value of h_j (i.e., the output corresponding to j_{th} sequence position). The formulas that a LSTM recursively computes the following formulas (Goldberg, 2017):

$$s_j = [c_j; h_j],$$
$$c_j = f \odot c_{j-1} + i \odot z,$$
$$\mathrm{h}_j = o \odot \tanh(c_j),$$
$$i = \sigma(w^{xi} x_j + w^{hi} h_{j-1} + b_i),$$
$$f = \sigma(w^{xf} x_j + w^{hf} h_{j-1} + b_f),$$
$$o = \sigma(w^{xo} x_j + w^{ho} h_{j-1} + b_o),$$

$$\mathrm{z} = \tanh\left(w^{xz} x_j + w^{hz} h_{j-1} + b_z\right),$$

$$y_j = O_L\left(s_j\right) = h_j,$$

$$c_j, \mathrm{h}_j, i, f, o, \mathrm{z} \in \mathbb{R}^{d_h},$$

$$s_j \in \mathbb{R}^{2d_h},$$

$$x_j \in \mathbb{R}^{d_x},$$

$$\mathrm{w}^{hz}, \mathrm{w}^{hi}, \mathrm{w}^{hf}, \mathrm{w}^{ho} \in \mathbb{R}^{d_h \times d_h},$$

$$\mathrm{w}^{xz}, \mathrm{w}^{xi}, \mathrm{w}^{xf}, \mathrm{w}^{xo} \in \mathbb{R}^{d_h \times d_x},$$

$$\mathrm{b}_z, \mathrm{b}_i, \mathrm{b}_f, \mathrm{b}_o \in \mathbb{R}^{d_h},$$

where σ is the logistic sigmoid function and $\odot$ denotes the element-wise multiplication. LSTMs are responsible for many state-of-the-art sequence modeling results.

In their more recent work, Chalkidis et al. (2017b) employed a bi-directional LSTM (BILSTM). A BILSTM consists of two LSTMs, a forward and a backward one, which are fed the input sequence $\mathrm{x}_{1:n}$ and its reverse version, respectively. The advantage of BILSTM compared to LSTM is that it preserves information from both past and future as it maintains, similarly to BIRNNs, two separate states, s_j^f and s_j^b, for each input position j. Each LSTM uses the formulas mentioned above. In the end, the outputs, y_j^f and y_j^b, obtained from the two LSTMs are concatenated into one vector $y_j = [y_j^f; y_j^b]$. On top of the output vector of the BILSTM, Chalkidis et al. (2017) connected a single neuron. Each vector y_j passed through a sigmoid activation function in this layer:

$$p_j = \sigma\left(w_y y_j + b_y\right).$$

If $p_j > 0.5$,then the j_{th} token of a given sequence is classified as positive ,i.e., its tag is 1, because it has more than 50% probability to be part of the element type which we wish to extract. The method described above is called **BILSTM-LR**.

It should be noted that for each contract element type and for each LSTM-based method described in this section, a separate extractor is constructed (11 extractors per method in total). Moreover, each extractor takes as input a sequence of concatenated word, POS tag[6], and token shape embeddings; the latter are 5-dimensional embeddings pre-trained on approximately 2,000 contracts from the unlabeled dataset, containing information regarding the form of the token, for example, if uppercase letters or numbers are included in a token. Each concatenated word (token) embedding represents a word of an extraction zone[7] of the element type considered by the particular extractor. The BILSTM converts each (concatena-

[6]In these methods the word and POS tag embeddings are the same pre-trained ones on the unlabeled dataset of contacts by the word2vec algorithm, which were used in the methods involving linear classifiers . The words, for which no word embedding was available, were mapped into random embeddings.

[7]In these methods the word sequences (in the form of vector sequences), with which the extractors are fed, are equivalent to entire export zones and not sliding windows of 11-13 tokens as in the case of the linear classifiers.

ted) token embedding into a context-aware embedding, which eventually passes through a Logistic Regression (LR) layer to estimate the probability that the particular token is positive (i.e., part of the corresponding contract element type) to be calculated. Each (concatenated) input token embedding is a 230-dimensional vector, whereas the hidden states of each LSTM (both forward and backward), are 300-dimensional vectors. Dropout (Srivastava et al., 2014) is also applied after the token embeddings layer, before the BILSTM chain, and again before the Logistic Regression layer, in order to avoid overfitting. More specifically, dropout zeroes each dimension of an input sequence of the corresponding sequence of vectors with a probability tuned via k-fold cross-validation. Furthermore, Glorot initialization (Glorot & Bengio, 2010), binary cross-entropy loss, and the Adam optimizer (Ki- ngma & Ba, 2015) are used to train each extractor.

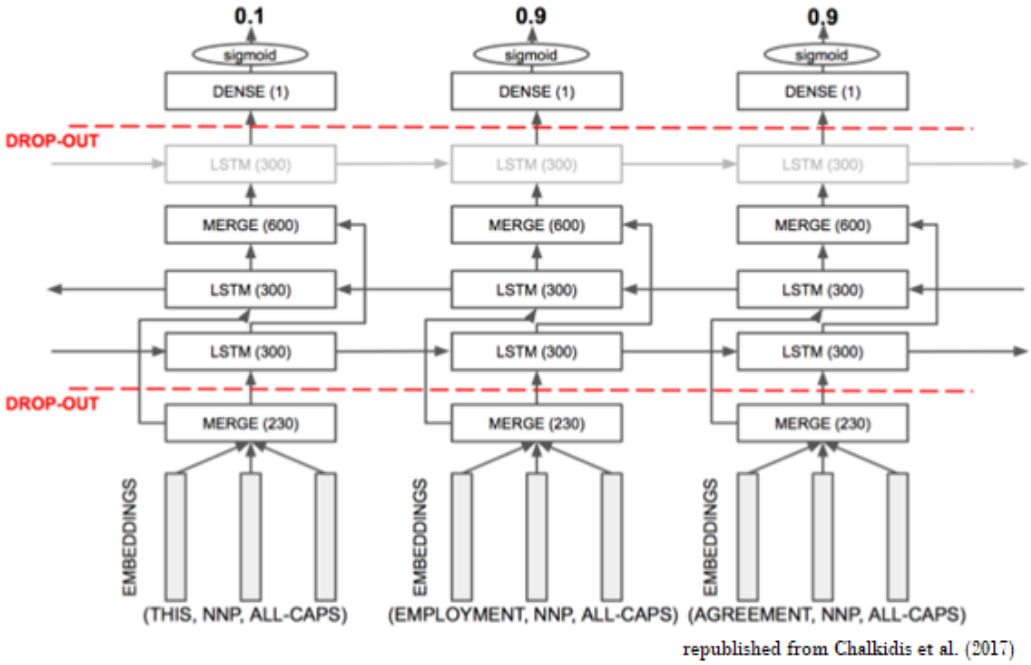


Figure 4: BILSTM-(LSTM)-LR architecture for sequence labeling.

The second stacked LSTM-based method of Chalkidis et al. (2017b) was called **BILSTM-LSTM-LR**. The only difference with the previous method is that it has an additional LSTM between the BILSTM and the Logistic Regression layer, which is fed with the 600-dimensional context-aware token embeddings derived from the BILSTM. The figure above (Fig. 4) depicts the BILSTM-LSTM-LR (and BILSTM without the upper LSTM boxes) architecture for sequence labeling.

The last LSTM-based method of Chalkidis et al. (2017b) is called **BILSTM-CRF.** Linear-chain Conditional Random Fields (CRFs) layer (Lafferty et al., 2001) is used in top of the BILSTM. Roughly speaking, the CRF layer predicts a label for each element of the input sequence by considering the predictions for its neighbors. The linear-chain CRF is quite popular in natural language processing and it is mostly used in sequence labeling tasks. In our case the linear-chain CRF layer predicts the label (class) of each one of the tokens in an extraction zone, instead of using a Logistic Regression layer, by also taking into account the predictions for the neighboring tokens. The CRF layer's objective is to assign a probability to each possible tag

sequence $t = t_1, ..., \mathrm{t}_n \in \{0,1\}^n$ over a word sequence, $w_1, ..., w_n$. The optimum label assignment is the one that maximizes the following joint conditional probability (Chalkidis et al., 2017b):

$$P(\mathrm{t}_1, ..., \mathrm{t}_n \mid w_1, ..., w_n; \mathrm{V}, U, \alpha, \mathrm{b}) = \frac{\exp\left(a^T \tau_1 + b^T \tau_n + \sum_{j=1}^{n} \left(V\tau_j\right)^T y_j + \sum_{j=2}^{n} \tau_{j-1}^T U \tau_j\right)}{\sum_{t'_1,...,t'_n} \exp\left(a^T \tau'_1 + b^T \tau'_n + \sum_{j=1}^{n} \left(V\tau'_j\right)^T y_j + \sum_{j=2}^{n} \tau'^T_{j-1} U \tau'_j\right)},$$

where $y_j \in \mathbb{R}^{600}$ is the embedding produced by the BILSTM chain (after dropout) that corresponds to token w_j, $V \in \mathbb{R}^{600\times 2}, U \in \mathbb{R}^{2\times 2}, a \in \mathbb{R}^2, b \in \mathbb{R}^2$ are the parameters to be learned , and τ_k is a 2-dimensional one-hot vector indicating the value of t_k (2 possible values).

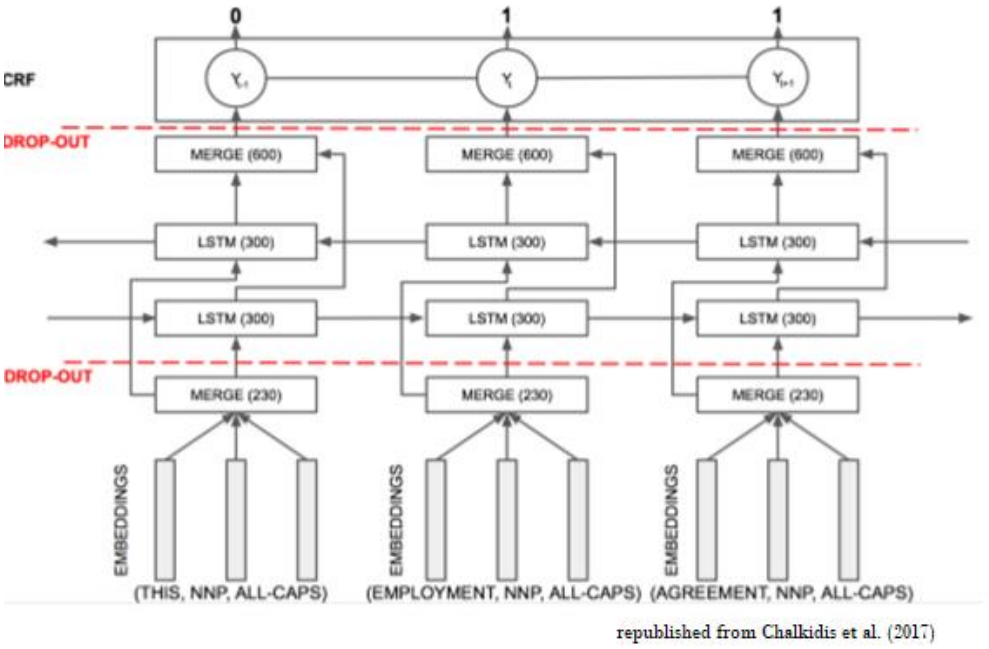


Figure 5: BILSTM-CRF architecture for sequence labeling.

Given the i.i.d. assumption and assuming that T training sequences are available, the joint conditional log-likelihood of the correct label assignments of all the training sequences is the following:

$$l\left(\mathrm{V}, U, \alpha, \mathrm{b}\right) = \sum_{i=1}^{T} \log\left(P(\mathrm{t}_{i1}, ..., \mathrm{t}_{in} \mid w_1, ..., w_n; \mathrm{V}, U, \alpha, \mathrm{b})\right),$$

where t_{ij} the correct tag of the j_{th} word of the i_{th} sequence. The goal is to learn the parameters V, U, a, b which maximize the joint conditional log-likelihood $l\left(\mathrm{V}, U, \alpha, \mathrm{b}\right)$. Training, in this case, requires the combination of dynamic programming or beam search decoding with backpropagation; see Yao et al. (2014) and Andor et al. (2016). The figure above (Fig.5) shows the **BILSTM-CRF** architecture.

The results of the LSTM-based methods of this section (with and without augmentation) will be presented in following sections.

2.2 Related work

LSTM-based methods have been successful in various sequence labeling tasks. For example, Ling et al. (2015) employed stacked LSTM-based models for POS-tagging. The first layer of their model was a BILSTM, which operated on the characters of the tokens and constructed word embeddings aiming to capture morphological features of the tokens. The resulting morphology-aware vectors were then combined with pre-trained word embeddings, produced by employing word2vec, and were passed on to the second LSTM or BILSTM layer (each with a softmax), to perform language modeling and POS tagging respectively.

Huang et al. (2015) used LSTM-based models combined with CRF layers to perform POS tagging, chunking and named-entity recognition (NER). Ma and Hovy (2016) used a similar model consisting of a BILSTM and a CRF layer for POS tagging or NER purposes, with the only difference being that they fed their model with embeddings produced by a Convolutional Neural Network (CNN), which operated on characters.

Regarding the extraction of information from legal documents, mostly linear methods and hand-crafted extraction rules have been applied so far, with the exception of Chalkidis et al. (2017b), who also considered LSTM-based methods as already discussed. Curtotti and Mccreath (2010) classified lines of Australian contracts into 32 classes, four of which correspond to classes extracted by Chalkidis et al. (2017a,2017b); the four classes are: contract titles, clause headings, contracting parties and start date. Curtotti and Mccreath employed several algorithms, such as SVM and decision trees, along with hand-crafted rules. Indukuri and Krishna (2010) also used SVMs to identify the clauses of each contract and determine clauses that describe payment terms. Gao et al. (2012) used only manually crafted rules to locate exception clauses in contracts.

Chapter 3

Data Augmentation

Labeled training datasets are crucial for any supervised machine learning algorithm, especially, for deep neural architectures which are susceptible to overfitting. Deep learning has resurged and flourished mostly in domains where large training datasets are readily available. The lack of large labeled datasets in some domains, including the legal domain which this work focuses on, prevents state-of-art deep learning methods from performing as well as in others domains where much larger labeled datasets are available. The process of acquiring more labeled data requires investment of human effort; hence, the need for automated, less time-consuming and cheaper methods emerges.

Data augmentation is the practice of applying transformations to existing training datasets in order to synthetically expand them. In this chapter, we discuss about data augmentation techniques, including where and how they have been applied so far and, more importantly, we describe the methods that were used in this thesis to augment the contract element extraction dataset of Chalkidis et al. (2017a, 2017b). The data augmentation techniques of this thesis involved perturbing, interpolating or extrapolating existing examples (training instances). For simplicity, we chose to operate in the input feature space, meaning the word embeddings level of the LSTM-based methods of Chalkidis et al., instead of working in the vector space of higher layers, such as the space of the hidden states produced by the LSTMs or BILSTMs of Chalkidis et al. We note that Bengio et al. (2013) and Ozair & Bengio (2014) claimed that augmenting the training data by operating in the representation spaces of higher layers (e.g., perturbing, interpolating, or extrapolating the points of the manifold that a deep neural network maps the input feature space to) makes it less likely to augment the dataset with non-realistic samples. We note, however, that the word embeddings level, where our data augmentation techniques operate, is already a learned level and, thus, may have similar properties, for data augmentation purposes, with higher layer representations. For example, the word embeddings are pre-trained on legal texts, hence they already capture morpho-syntactic and semantic properties of words occurring in legal texts; consequently, perturbing, interpolating, or extrapolating the sequence of word embeddings of a training extraction zone may lead to sequences of word embeddings that, although synthetic, may still be realistic, in the sense that they may correspond to word sequences that are likely to be found in extraction zones of the same contract element type.

We were mostly interested in augmenting the training sets of contract element types that had few training instances. For completeness, however, we also applied the data augmentation techniques to the training sets of some contract element types with more training instances. In both cases, we wanted to examine if the performance of the corresponding contract element extractors could be improved and to what extent.

3.1 Related Work on data augmentation

This section provides a review of related work that has applied data augmentation techniques to existent training data to improve the performance of the models induced from the training data. For many machine learning tasks, data augmentation has been employed as a tool against overfitting while training supervised learning models. The more (and more diverse) examples the models see during training, the better they generalize, and consequently the better they predict when presented with new instances.

Salamon and Bello (2016) applied various audio data augmentations (deformations), while training a deep CNN for environmental sound classification. The deformations involved time stretching (i.e., slowing down or speeding up the audio sample), pitch shifting, and inserting background noise (i.e., mixing the existing sound samples with another recording containing background sounds from different types of acoustic scenes) or dynamic range compression. Each deformation was applied directly to the audio signal prior to converting it to the input representation used to train the neural network. Combined with data augmentation, their proposed model produced state-of-the-art results for environmental sound classification.

Data augmentation is also very popular for visual recognition tasks, due to the fact that the generation of new data can be easily achieved by simple image manipulations, such as shifting, scaling, rotation, mirroring, adjusting contrast or grayscale, or randomly cropping. LeCun et al. (1998), while training LeNet5, or Krizhevsky et al. (2012), while training AlexNet[8], applied a series of such transformations to the input images in order to improve the performance of these models.

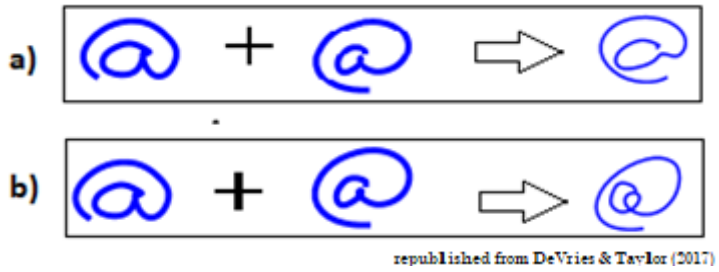
As far as NLP tasks are concerned, one of the most commonly used method to augment a text dataset is to replace words with their closest synonyms. Mueller and Thyagarajan (2015) augmented their training dataset by employing thesaurus-based augmentation while training a siamese adaptation of the Long Short-Term Memory (LSTM) network to assess semantic similarity between sentences. Mueller and Thyagarajan generated additional training examples by replacing random words of the training dataset with one of their synonyms found in WordNet (Fellbaum, 1998); the best results were obtained by training their model on the augmented dataset. Kolomiyets et al. (2011), in an attempt to improve the performance of the Logistic Regression classifier they used to identify time expressions, augmented their training dataset by substituting words found in the training dataset with likely synonyms. The synonyms were selected both via WordNet, and via predictions from the Latent Words Language Model (LWLM) (Deschacht and Moens, 2009).

The methods mentioned above are domain-specific ones (e.g., for image classification, or sentence similarity). Domain-agnostic methods have also been employed. Lu et al. (2006), in order to enhance the classification performance of an SVM model they employed for biomedical text analysis, applied a semi-supervised learning algorithm described by Zhu et al. (2005). This algorithm is based on the theory of Gaussian random fields, which allows the labels of the training cases to be propagated to the unlabeled data probabilistically. Schlüter& Grill (2015), apart from shifting the pitch of the audio signal, time stretching, vary-

[8]Both models were used in image recognition tasks.

ing the loudness of the audio signal or applying random frequency filters, also applied domain-agnostic methods such as adding Gaussian noise to the input, or interpolating between samples in input space. The domain-agnostic methods, unfortunately, did not work as well the domain-specific ones.

DeVries and Taylor (2017) applied domain-agnostic techniques to various datasets, such as images of hand-written characters or audio clips. Based on the hypothesis of Bengio et al. (2013) and Ozair & Bengio (2014) as already discussed, DeVries and Taylor used a sequence autoencoder to produce higher level representations (feature vectors), and then applied simple transformations to the latter to generate new data. The autoencoder consists of two parts, an encoder, which receives each input and, via nonlinear transformations, converts it into a new representation, and a decoder which, subsequently, takes the new representation and tries to reconstruct the original input in the same way. As a sequence autoencoder they used a stacked LSTM with two layers for both the encoder and decoder. In order to augment the dataset, they fed the sequence autoencoder with each available sample, extracted the encoder's hidden state vector at the final time step (i.e., the projection of the raw input to the feature space) and applied a set of transformations on the new data points such as adding noise, interpolating or extrapolating between them. The following figure (Fig. 6) shows an example of what it is produced if we interpolate (a) or extrapolate (b) between the feature vectors of two hand-written characters.



republished from DeVries & Taylor (2017)

Figure 6: Interpolation (a) and extrapolation (b) between two handwritten characters. The original characters are shown in bold.

The experimental results of DeVries and Taylor showed that the most effective method was extrapolating between neighbor feature vectors representing existing samples, whereas the rest of the methods (i.e., interpolating between them or adding random noise) proved less effective not only comparing with extrapolation, but also with the case of not augmenting the available dataset at all.

Wang and Yang (2015) improved significantly the performance of SAGE (Eisenstein et al., 2011), a state-of-the-art mixed-effect topic model that they used to conduct behavioral analysis using social media text (tweets), by adopting a novel approach of creating additional training instances. Wang and Yang generated new training instances by searching the nearest neighbor word, in terms of cosine similarity between word-embeddings, for each token in a tweet and then replacing each word by its nearest neighbor.

3.2 Data augmentation methods used in this thesis

For some of the contract element types of the dataset of Chalkidis et al. (2017a, 2017b), we hypothesized that the number of instances (Table 2) was insufficient to train the LSTM-based extractors. More specifically, we believed that the lack of data afflicted the performance and, in some cases, even the robustness of at least some of the LSTM-based extraction methods (Section 2.1.2) and this is why this data augmentation work started. We note, however, that there does not seem to be a consistent relation between the number of samples of each contract element type and the performance of its extractors. Contract period was the element type with the lowest number of available samples, and also with the worst results. Termination date was the second lowest element type in number of samples, and its extractors performed better than those of contract period. The performance of the extractors for effective date was among the best (its highest F_1-score was 0.97), even though effective date was the third lowest in number of samples. For Contract Value, more than twice the number of instances was provided, compared with Contract Period, and its extractors performed better than those corresponding to Contract Period. Regarding Jurisdiction, its number of instances was significantly higher, and the performance of its extractors was better compared to the aforementioned element types. Contracting Parties is the second highest in data availability element type and its performance was also among the best.

We chose to apply domain-agnostic data augmentation techniques to the word embeddings of the training extraction zones of the aforementioned contract element types, in order to see if and how the additional, synthetically created data would affect the performance of the extractors of those contract element types, whose results and sizes of available data differ from each other. Each dataset is split in 3 parts for training, validation and testing. We operated on the training part of each dataset. For words with no available word2vec embeddings, Chalkidis et al. use a randomly generated embedding.

We increased the number of training instance sizes in two ways (before and while training), depending on the data augmentation technique. In the case of adding Gaussian noise, the number of training samples is increased during training. More specifically, another layer is added to the BLSTM-based extractors after the token embedding layer and before dropout being applied, which receives the original training sequences of token embeddings as input and adds Gaussian noise to the word embedding of each token of each training sequence with probability equal to 50% . In this way, each original training sequence may be modified completely, partially, or not at all in each epoch. Also, the noise added (if it is added) to the word embedding representing a particular token of a training sequence differs from epoch to epoch. Hence, the BILSTM-based extractors are prevented from translating the added noise uniformly, as it would probably happen if the noise was added once before training. In the case of applying interpolation or extrapolation, we increased the number of training extraction zones (each viewed as a sequence of embeddings) of each dataset before training by 50%, i.e., we randomly selected half of the existing training extraction zones, and created a new synthetic training extraction zone from each selected one. In order to create a new training extraction zone, we iterated over the tokens of the original extraction zone, excluding line breaks, punctuation, brackets, tokens containing digits, other special

characters etc., as well as tokens we did not have pre-trained word embeddings for, applying one of the transformations we describe below to the word embedding of each token. The resulting new synthetic word embedding was then concatenated with the POS tag and token shape embeddings of the original token. No transformation was applied to the POS tag and token shape embeddings, because we wished the properties of these embeddings to remain the same as in the original tokens, to avoid generating synthetic embeddings that would be too far from the original ones. It would also be impossible to produce new POS tag and token shape embeddings, given that the new synthetic word embeddings do not necessarily correspond to actual words.

3.2.1 Gaussian Noise

The simplest way to transform the original word embeddings of an extraction zone is adding Gaussian noise to them. In our experiments, when an extraction zone passes through the additional layer we described above, for each word embedding of the particular extraction zone, a new noise vector is generated in each epoch with probability equal to 50% by drawing from a Gaussian distribution with mean equal to zero and standard deviation equal to 0.5 and then the noise vector is added to the original word embedding. More specifically, the element w_i' of the i_{th} dimension of the new word embedding was computed as follows:

$$w_i' = w_i + x_i,$$
$$x_i \sim N(0, 0.5),$$

where w_i denotes the element of the i_{th} dimension of an original word embedding and x_i denotes the element of the i_{th} dimension of the noise vector drawn from a Gaussian distribution with zero mean and standard deviation = 0.5.

3.2.2 Interpolation and Extrapolation

We also employed two alternative approaches for data augmentation, according to which, roughly speaking, we can create a new word embedding by finding the K nearest neighbors (words with the closest word embeddings) of the original embedding, computing the centroid of the K nearest word embeddings (the average of the neighboring embeddings) and finally interpolating or extrapolating between the original word embedding and the centroid. We use cosine similarity when computing embedding distances. Recall, also, that each token of an input training sequence (training extraction zone) is manually labeled as positive or negative, with respect to a contract element type, depending on whether or not it is part of a contract element of the particular type. The same word may be labeled as positive in one training sequence and negative in another. Additionally, if a word is encountered more than once in the same sequence, it is possible for the occurrences of the word to have different labels.

When searching for the K nearest words of a particular word that we wish to replace in a training extraction zone, we actually search the embeddings of words that occur with the same label in every training extraction zone of the same contract element type. For instance, let us assume that "This agreement shall be governed and construed in accordance to the Laws of England & Wales" is a training extraction zone of the Governing Law contract element type, and that the token "laws" is labeled as positive. Searching for the two nearest neighbors in the entire vocabulary (meaning words we have pre-trained embeddings for) without taking into consideration the (positive) label of "laws" in the particular extraction zone, would return "jurisdictions" as the second closest neighbor. However, "jurisdictions" is never a positive token in the training extraction zones of Governing Law, i.e., it has never been encountered as part of a Governing Law element. Hence, including "jurisdictions" in the centroid would be a poor choice; for example, when using interpolation, it would contribute towards replacing "laws" by words that are close to "jurisdictions", and the new word would still be labeled as positive. This is like feeding the extractors with an incorrect synthetic example. Moreover, in an attempt to create syntactically realistic sequences, we created 18 broader groups out of the 45 existing POS tags. For example, one group included all the types of nouns, such as singular (proper) nouns, plural (proper) nouns, another group included all the types of verbs, such as verbs in past or present tense, etc., and we did not allow words belonging to a POS tag group different than the focus token's group to be included in its K nearest neighbors; by focus token we mean the token to be replaced.

More concretely, the steps we followed in order to create new training sequences (training extraction zones) by applying interpolation or extrapolation are the following:

1. For each token *t* (excluding line-breaks or special characters, such as comma, period, bracket, tokens containing digits etc.) in the training dataset of a particular contract element type *e*, we found the 5 nearest neighbors (in terms of cosine similarity of word embeddings) of *t*, searching for neighbors in all the other tokens (with the same exclusions) of the dataset of *e*, but considering only tokens in the same POS tag group and binary class (positive or negative) as *t*.
2. In order to create a new training sequence (training extraction zone) based on an existing one, we iterated over the tokens of the existing sequence twice. In the first iteration, for each token *t* of the sequence (again excluding line-breaks or special characters, such as comma, period, bracket, tokens containing digits etc.), we randomly chose 2 out of its 5 nearest neighbors (found as in step 1 above) and created the centroid (average) of the 2 selected neighbors. We randomly chose 2 of the 5 available nearest neighbors, because we wanted their centroid to differ to a larger extent from sequence to sequence, to avoid generating very similar synthetic examples. In the second iteration, for each token *t* of the original sequence, we used the word embedding of *t* and the centroid of the neighbors of *t* (calculated during the first iteration) to generate a new embedding by interpolating or extrapolating between the embedding of *t* and the centroid. The formula we used when we applied interpolation was the following:

$$w'_j = (\mathrm{w}_k - w_j)\lambda + w_j,$$

where w_k , w_j and w'_j are the centroid , original word embedding and synthetic word embedding corresponding to the j_{th} member of the sequence, respectively, while λ is a parameter in the range [0,1] that controls the degree of interpolation.

Similarly, when we employed extrapolation, we used the following formula:

$$w'_j = (w_j - w_k)\lambda + w_j.$$

In case of extrapolation, λ is a value in the range $[0, \infty)$ which controls the degree of extrapolation. In our experiments, we tried various values for λ on the validation data of each element type, and the best results were obtained for $\lambda = 0.8$ or 0.5,in case of interpolation, and $\lambda = 0.2$ or 0.5,in case of extrapolation, depending on the dataset. In the following figure, we illustrate how interpolation (a) and extrapolation (b) work.

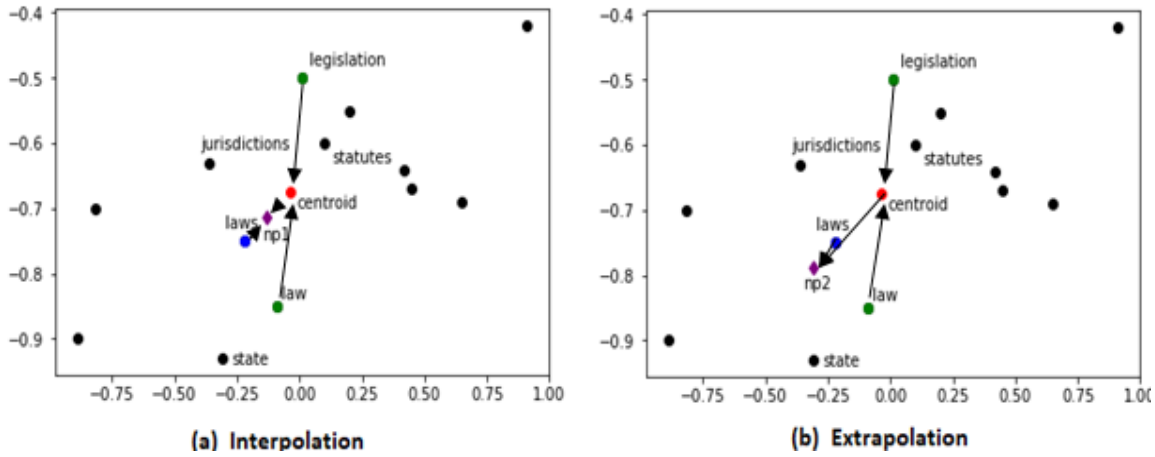


Figure 7: Interpolating (a) or extrapolating (b) between the centroid and the embedding of the focus word to create a new embedding.

In Figure 7, assuming that the focus token (the token we wish to replace) is the word “laws” (blue point) and the words “legislation” and “law” (green points) are the two randomly chosen neighbors (among the 5 nearest ones) of “laws”, which are used to generate a centroid (red point). We can see the new data points (purple points) constructed by interpolating (point **np1** in Fig. 7a) or extrapolating (point **np2** in Fig. 7 b) between the centroid and the word embedding of the word “laws”.

Chapter 4

Experiments

We performed experiments for each combination of extractor, augmentation technique and augmentation size. We did not experiment with all the contract element types, due to time constraints. We used Glorot initialization (Glorot & Bengio, 2010), binary cross-entropy loss, and the Adam optimizer (Kingma & Ba, 2015) similarly to Chalkidis et al. (2017b), to train each extractor, with early stopping examining the validation loss. Furthermore, we used the same hyper-paremeters, such as dropout rate, learning rate, and batch size for each element as in the work of Chalkidis et al. (2017b). The additional parameters λ and K, which control the degree of interpolation or extrapolation and indicate the number of nearest neighbors based on which we generated each centroid (Section 3.2.1), respectively, were trained in the same way as the aforementioned hyper-parameters had been trained in the work of Chalkidis et al. (2017b), i.e., 3-fold cross validation on 80% of the training extraction zones (of the corresponding contract element type). The evaluation of each combination of extractor, augmentation technique and augmentation size is based on the decisions of the extractor per token. The performance measures are the following:

1. precision : $P = \frac{TP}{TP + FP}$
2. recall : $R = \frac{TP}{TP + FN}$
3. F_1-score: $F_1 = \frac{2 \cdot P \cdot R}{P + R}$,

where TP, FN and FP are the true positives (i.e., correctly classified as positive tokens), false negatives (i.e., incorrectly classified as negative tokens), and false positives (i.e., incorrectly classified as positive tokens), respectively. In order to show if and how each applied augmentation technique affected the performance of each extractor, we also provide the difference in performance in terms of F_1 score, more specifically we compared the F_1 scores of each extractor (for each element type) with the original and augmented training dataset, respectively. Along with the performance of each technique, we also wanted to see if the techniques improved the robustness of the extractors, so we calculated the standard deviation of the F_1–score for each combination of extractor, augmentation technique and augmentation size[9]. It can be observed that none of the augmentation techniques leads to consistently better results comparing to the results before data augmentation. Overall, however, interpolation appears to be the best among the three data augmentation techniques.

[9] We ran each type of experiment, i.e., for each extractor, augmentation technique and augmentation size 3 times. In Table 5, precision, recall and F_1-score are averaged over the corresponding results of the multiple runs.

BILSTM-LR																		
Element Type	Original			Noise					Interpolation					Extrapolation				
	P	R	F1	P	R	F1	+/-	sd	P	R	F1	+/-	sd	P	R	F1	+/-	sd
Termination	0.65	0.90	0.75	0.57	0.42	0.48	-27%	0.04	0.70	0.92	0.79	+4%	0.02	0.69	0.94	0.79	+4%	0.03
Period	0.44	0.82	0.57	0.43	0.81	0.56	-1%	0.04	0.51	0.80	0.61	+4%	0.05	0.35	0.67	0.46	-11%	0.13
Value	0.74	0.55	0.63	0.74	0.57	0.64	+1%	0.03	0.78	0.57	0.66	+3%	0.03	0.78	0.57	0.66	+3%	0.01
Jurisdiction	0.90	0.89	0.89	0.90	0.90	0.90	+1%	0.01	0.91	0.87	0.89	0%	0.01	0.90	0.87	0.89	0%	0.01
Contract Parties	0.97	0.92	0.95	0.96	0.93	0.95	0%	0.01	0.97	0.93	0.95	0%	0.00	0.97	0.92	0.95	0%	0.01
Effective Date	0.98	0.92	0.95	0.96	0.93	0.95	0%	0.01	0.98	0.90	0.94	-1%	0.02	0.97	0.93	0.95	0%	0.01
Macro-Average	0.78	0.83	0.79	0.76	0.76	0.75	-4%	-	0.81	0.83	0.81	+2%	-	0.78	0.82	0.78	-1%	-

BILSTM-LSTM-LR																		
Element Type	Original			Noise					Interpolation					Extrapolation				
	P	R	F1	P	R	F1	+/-	sd	P	R	F1	+/-	sd	P	R	F1	+/-	sd
Termination	0.70	0.92	0.79	0.48	0.53	0.50	-29%	0.01	0.71	0.93	0.81	+2%	0.01	0.67	0.92	0.77	-2%	0.04
Period	0.47	0.86	0.59	0.45	0.85	0.58	-1%	0.06	0.46	0.84	0.60	+1%	0.08	0.58	0.84	0.69	+10%	0.03
Value	0.74	0.63	0.68	0.71	0.64	0.67	-1%	0.02	0.76	0.62	0.68	0%	0.02	0.75	0.63	0.68	0%	0.02
Jurisdiction	0.90	0.88	0.89	0.90	0.88	0.89	0%	0.01	0.89	0.89	0.89	0%	0.01	0.90	0.90	0.90	+1%	0.01
Contract Parties	0.97	0.94	0.95	0.97	0.94	0.95	0%	0.00	0.97	0.94	0.95	0%	0.01	0.97	0.94	0.95	0%	0.01
Effective Date	0.97	0.96	0.97	0.97	0.98	0.97	0%	0.01	0.97	0.96	0.96	-1%	0.01	0.97	0.94	0.96	-1%	0.01
Macro-Average	0.79	0.87	0.81	0.75	0.80	0.76	-5%	-	0.79	0.86	0.81	0%	-	0.83	0.86	0.82	+1%	-

BILSTM-CRF																		
Element Type	Original			Noise					Interpolation					Extrapolation				
	P	R	F1	P	R	F1	+/-	sd	P	R	F1	+/-	sd	P	R	F1	+/-	sd
Termination	0.65	0.93	0.77	0.38	0.45	0.39	-38%	0.13	0.68	0.94	0.78	+1%	0.04	0.69	0.89	0.77	0%	0.04
Period	0.55	0.85	0.65	0.44	0.79	0.56	-9%	0.05	0.49	0.82	0.61	-4%	0.07	0.37	0.82	0.51	-14%	0.04
Value	0.72	0.60	0.66	0.68	0.56	0.62	-4%	0.01	0.74	0.61	0.67	+1%	0.02	0.74	0.55	0.63	-3%	0.02
Jurisdiction	0.90	0.88	0.88	0.90	0.88	0.89	+1%	0.00	0.90	0.88	0.89	+1%	0.01	0.90	0.88	0.89	+1%	0.01
Contract Parties	0.98	0.92	0.95	0.97	0.92	0.95	0%	0.01	0.98	0.93	0.95	0%	0.01	0.98	0.92	0.95	0%	0.01
Effective Date	0.95	0.89	0.92	0.94	0.93	0.94	+2%	0.01	0.98	0.91	0.94	+2%	0.01	0.95	0.86	0.90	-2%	0.02
Macro-Average	0.79	0.85	0.80	0.72	0.76	0.72	-8%	-	0.80	0.85	0.80	0%	-	0.77	0.82	0.77	-3%	-

Table 4: Precision (P), Recall (R) and F_1 score measured per contract token before (Original) and after (Noise, Interpolation, Extrapolation) applying augmentation techniques. The +/- column shows the increase or decrease in F_1 score, compared to the original results, while 'sd' indicates the standard deviation of the F_1 score over multiple repetitions.

Adding Gaussian noise is the technique with the worst performance. Combining a BILSTM-CRF extractor with any of the domain-agnostic augmentation techniques produced the worst results, as, in the majority of cases, the performance per element was adversely affected, and at the same time, no increase in F1 score exceeded 2 percentage points. Moreover, a tremendous decrease of 38 percentage points and a standard deviation of 13% were recorded in the F1 score of the BILSTM-CRF extractor for Termination Date when combined with adding Gaussian noise. Overall, interpolation was the technique that produced the most frequently positive results when using BILSTM-CRF, but never leading to an improvement larger than 2 percentage points in F1 score, and causing a maximum deterioration of 4 percentage points (for Contract Period).

In the case of BILST-LSTM-LR extractors, the only case of a tremendous decrease (of 29 percentage points) was recorded in the F1 score of the BILSTM-LSTM-LR extractor for Termination Date when combined with adding Gaussian noise. Except for the aforementioned case, no other decrease in F1 score exceeded 2 percentage points. Adding Gaussian noise affected adversely or not at all the performance of the BILSTM-LSTM-LR extractor, while applying interpolation or extrapolation produced both positive and negative results, across the element types, with the number of positive and negative results being almost equal. Noteworthy was the improvement in terms of performance and robustness of Contract Period's BILST-LSTM-LR extractor combined with extrapolation (F_1 score's increase reached 10 percentage points while its standard deviation was only 3%).

Data augmentation worked much better overall with the BILSTM-LR extractors. The increase in F1 score ranged from 2 to 5 percentage points. Each method produced both positive and negative results, across the element types, with the number of positive results exceeding slightly the number of negative ones (in total). Again, interpolation was overall the best among the three data augmentation techniques. Notably, adding Gaussian noise deteriorated the F1 score of Termination Date by 27 percentage points and extrapolation deteriorated the F1 score of Contract Period by 11 percentage points.

Studying the results of Table 4 per contract element type, we can observe that Contract Parties (the element type with the second highest number of instances) was not affected by any form of data augmentation. Jurisdiction (an element with sufficient data whose performance before data augmentation was very satisfying and stable) had only some minor positive changes with respect to F_1 score that did not exceed 1 percentage point. Contract Value (the second worst element in performance even before data augmentation, but only the fourth lowest in number of instances) was affected more intensely than the aforementioned elements, however not consistently. Termination date (an element with half the number of instances compared to Contract Value, but with significantly better performance before data augmentation) showed very pronounced fluctuations in the performance of its extractors with respect to F_1 score, especially in the case of adding Gaussian noise. Effective Date (an element with more than impressive performance before data augmentation, despite its small number of instances) was slightly, but inconsistently affected. Finally, Contract Period (the element with the worst and least robust performance before data augmentation, and with the lowest number of instances) performed either unexpectedly well or terribly badly after data augmentation.

To conclude, none of the three data augmentation techniques worked well for every type of extractor and element we experimented with, probably because contracts are complex and difficult to understand documents, and domain-agnostic augmentation techniques cannot preserve the semantic and syntactic properties of the original contracts when creating new synthetic training instances, or because applying the augmentation methods at this early stage, i.e., at the level of word embeddings, did not allow the techniques to produce consistent results.

Chapter 5

Conclusions and future work

In this thesis, we explored three domain-agnostic data augmentation techniques applied to legal documents, more specifically contracts. We experimented with (a) adding Gaussian noise to the word embeddings of training instances differently in each epoch, (b) interpolating or (c) extrapolating between the word embeddings of existing training instances and neighboring embeddings of words with similar properties (class, POS tag group) obtained from other existing training instances of the same contract element type. The evaluation of these three techniques was based on how they affected the performance of the BILSTM-based extractors proposed by Chalkidis et al. (2017b). We showed that interpolation was the technique that produced the best results, compared to extrapolation and adding Gaussian noise. However, none of the three techniques was consistently successful, regardless of the type of LSTM-based extractor it was combined with. Furthermore, the contract element types whose performance was very good before data augmentation were slightly or not affected by these techniques (e.g., Contract Parties, Effective Date, Jurisdiction). Regarding the other types of contract elements, the results after data augmentation were affected more intensely, however inconsistently.

Future work could explore domain-specific augmentation techniques. Alternatively, it could apply the techniques of this thesis to the vector spaces of higher layers, for example to the output of the lower BILSTM layer of the methods of Chalkidis et al. (2017b), as opposed to applying them to the input word embeddings.

References

Andor, D., Alberti, C., Weiss, D., Severyn, A., Presta, A., Ganchev, K., Petrov, S., & Collins, M. (2016). Globally normalized transition-based neural networks. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (long papers), pp. 2442–2452, Berlin, Germany.

Androutsopoulos, I., Koutsias, J., Chandrinos, K. V., & Spyropoulos, C. D. (2000). An experimental comparison of Naive Bayesian and keyword-based anti-spam filtering with encrypted personal e-mail messages. In Proceedings of the 23rd ACM SIGIR Conference, pp. 160–167, Athens, Greece.

Bengio,Y., Mesnil,G., Dauphin,Y., and Rifai,S. (2013). Better mixing via deep representations. In ICML (1), pp. 552–560.

Bengio,Y., Simard,P., & Frasconi,P. (1994). Learning Long-Term Dependencies with Gradient Descent is Difficult. IEEE Transactions on Neural Nets, Volume 5, Issue 2, pp. 157-166.

Chalkidis, I., Androutsopoulos, I., & Michos, A., 2017. Extracting contract elements. In Proceedings of the 16th International Conference on Artificial Intelligence and Law, pp. 88–97, London, UK

Chalkidis, I. & Androutsopoulos I., 2017. A Deep Learning Approach to Contract Element Extraction. In Proceedings of the JURIX 2017: 30th International Conference on Legal Knowledge and Information Systems, Luxemburg City, Luxembug.

Cristianini,N., Shawe-Taylor,J. (2000). An introduction to Support Vector Machines and other kernel based learning methods. Cambridge University Press.

Curtotti, M., & Mccreath, E. (2010). Corpus based classification of text in Australian contracts. In Proceedings of the Australasian Language Technology Association Workshop, pp. 18–26, Melbourne, Australia.

Deschacht, K., and Moens,M.F. (2009). Using the Latent Words Language Model for SemiSupervised Semantic Role Labeling. In Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing

DeVries,T., and Taylor,G.W. (2017). Dataset Augmentation in Feature Space. In Workshop track of the 5th International Conference on Learning Representations, Toulon, France.

Fellbaum C. (1998). WordNet: An Electronic Lexical Database. MIT Press.

Gao, X., Singh, M. P., & Mehra, P. (2012). Mining business contracts for service exceptions. IEEE Transactions on Services Computing, 5, 333–344.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the International Conference on Artificial Intelligence and Statistics, pp. 249–256, Sardinia, Italy.

Goldberg, Y. (2017). Neural Network Methods in Natural Language Processing. Morgan and Claypool Publishers.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780.

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen Netzen (diploma thesis). Technical University Munich, Institute of Computer Science.

Huang, Z., Xu, W., & Yu, K. (2015). Bidirectional LSTM-CRF models for sequence tagging. CoRR, abs/1508.01991.

Indukuri, K. V., & Krishna, P. R. (2010). Mining e-contract documents to classify clauses. In Proceedings of the 3rd Annual ACM Bangalore Conference, pp. 7:1–7:5, Bangalore, India.

Irsoy, O., & Cardie, C. (2014). Deep recursive neural networks for compositionality in language. In Proceedings of the 27th International Conference on Neural Information Processing Systems, pp. 2096–2104, Montreal, Canada.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In Proceedings of the International Conference on Learning Representations, San Diego, CA.

Kolomiyets,O., Bethard,S., and Moens,M.F. (2011). Model-Portability Experiments for Textual Temporal Analysis. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: shortpapers, pp. 271–276, Portland, Oregon.

Krizhevsky,A., Sutskever,I., and Hinton,G. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pp. 1097–1105.

Lafferty, J. D., McCallum, A., & Pereira, F. C. N. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In Proceedings of the 18th International Conference on Machine Learning, pp. 282–289, Williamstown, MA.

Ma, X., & Hovy, E. (2016). End-to-end sequence labeling via bi-directional LSTM-cNNsCRF. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pp. 1064–1074, Berlin, Germany

McCullagh, P., & Nelder, J. A. (1989). Generalized Linear Models, (Chapman & Hall/CRC Monographs on Statistics & Applied Probability) (2 edition). Chapman and Hall/CRC.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013a). Efficient Estimation of Word Representations in Vector Space. In Proceedings of the International Conference on Learning Representations (ICLR), Scottsdale, AZ.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013b). Distributed Representations of Words and Phrases and their Compositionality. In Proceedings of the 26th International Conference on Neural Information Processing Systems, Stateline, NV.

Mikolov, T., tau Yih, W., & Zweig, G. (2013c). Linguistic regularities in continuous space word representations. In Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 746–751, Atlanta, GA.

Mueller,J., and Thyagarajan, A. (2016). Siamese Recurrent Architectures for Learning Sentence Similarity In the Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16), Phoenix, Arizona, USA.

LeCun,Y., Bottou,L., Bengio,Y., and Haffner,P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

Lu,X., Zheng,B., Velivelli,A., Zhai, C. (2006). Enhancing Text Categorization with Semantic Enriched Representation and Training Data Augmentation. Journal of the American Medical Informatics Association Volume 13,pp. 526-534.

Ozair,S., and Bengio,Y. (2014). Deep directed generative autoencoders. arXiv preprint arXiv:1410.0630, 2014.

Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global vectors for word representation. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, pp. 1532–1543, Doha, Qatar.

Salamon,J., and Bello,J.P. (2016). Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification.

Schlüter,J., and Grill,T. (2015). Exploring data augmentation for improved singing voice detection with neural networks. In International Society for Music Information Retrieval Conference (ISMIR).

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15, 1929–1958.

Vapnik, V. N. (1995). The Nature of Statistical Learning Theory. Springer-Verlag.

Wang,W.Y., and Yang,D. (2015).That's So Annoying!!!: A Lexical and Frame-Semantic Embedding Based Data Augmentation Approach to Automatic Categorization of Annoying Behaviors using #petpeeveTweets. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Lisbon, Portugal.

Yao, K., Peng, B., Zweig, G., Yu, D., Li, X., & Gao, F. (2014). Recurrent conditional random field for language understanding. In Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, pp. 4077–4081, Florence, Italy.

Zhu,X. (2005). Semi-supervised learning literature survey. Computer science technical report 1530, University of Winsconsin.