Reconhecimento de Conflitos em Contratos – Contribuição de Pesquisa

Sylvio A. de O. Corrêa Filho[1]

[1]Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Escola Politécnica – Curso de Ciência da Computação
Av. Ipiranga, 6681 – 90619-900 – Porto Alegre – RS – Brasil

sylvio.correa@acad.pucrs.br

Resumo. *Relatório de contribuição feita sobre a pesquisa descrita no artigo Norm Conflict Identification in Contracts. O artigo descreve pesquisa voltada para a automação do reconhecimento de conflitos entre normas de um mesmo contrato. Os autores da pesquisa criaram um programa capaz de fazer este reconhecimento. A contribuição apresenta os resultados obtidos com o uso de uma forma diferente de cálculo da similaridade semântica entre sentenças normativas, assim como uma nova forma de reconhecimento das partes envolvidas em um contrato. Testes comparativos entre diferentes métodos e a abordagem original são apresentados.*

1. Introdução

Este relatório descreve contribuição sobre a pesquisa relatada no artigo Norm Conflict Identification in Contracts [Aires et al. 2017]. O artigo trata de pesquisa sobre reconhecimento automático de conflitos entre normas de um mesmo contrato. Usando a linguagem de programação Python, os autores criaram um programa capaz de realizar esta tarefa e então produziram experimentos para demonstrar sua efetividade.

O trabalho dos autores é dividido em duas partes: obter dados a partir de contratos escritos em linguagem natural e computar estes dados a fim de identificar conflitos. Para a primeira parte, é necessário identificar as partes contratantes, as normas acordadas e a quem cada uma delas se refere, assim como a conversão das cláusulas normativas em proposições da lógica deôntica. Para a segunda parte, é necessário criar uma forma de computar os dados obtidos de modo a identificar neles a existência ou não de normas conflitantes.

Elementos fundamentais da abordagem dos autores serão trazidos novamente e então contrapostos a uma abordagem diferente apresentada pelo autor deste artigo. O propósito deste trabalho é contribuir de alguma forma para a pesquisa original. Com este objetivo em mente, duas novas linhas de pesquisa são exploradas neste artigo: Primeiro um novo método de cálculo da similaridade semântica é testado usando *sentence embeddings* com a biblioteca sent2vec. Depois uma nova maneira de encontrar automaticamente as partes de um contrato é testada usando a biblioteca StanfordCoreNLP.

Testes serão realizados usando as novas abordagens e então os resultados serão comparados aos originais.

2. Elementos Essenciais de um Contrato

Um contrato é a formalização de um acordo entre duas ou mais partes. Este acordo estabelece uma série de normas que devem ser respeitadas pelas partes acordantes. A formalização torna legalmente exequíveis as obrigações acordadas. Quase sempre, estes contratos são firmados por escrito em linguagem natural, o que muito dificulta o processo de automação que é o objetivo do trabalho dos autores.

É comum que a estrutura dos contratos contenha um cabeçalho e um corpo. O cabeçalho contém a informação das partes: nome, pseudônimo ou abreviação, cadastros de identificação, endereço etc. O corpo do contrato contém as normas acordadas.

Normalmente cada uma das normas de um contrato faz menção a uma das partes juntamente com algum tipo de obrigação, proibição ou permissão que lhe diga respeito.

A abordagem utilizada para obter dados computáveis a partir do texto dos contratos é baseada em processamento de linguagem natural. Os elementos críticos a serem obtidos são de dois tipos: as partes e as normas.

3. Identificação das Partes do Contrato

Para fins de realização do trabalho original, os autores se limitaram a utilizar contratos entre apenas duas partes. O método de identificação parte da definição de um padrão de cabeçalho a ser processado. A primeiro método têm por base a procura em uma lista de nomes de empresas constante em um repositório da Wikipédia [Wikipedia contributors 2018]. Não havendo sucesso, faz-se uso de um anotador do Natural Language Toolkit (NLTK) [Bird 2009].

Extraídos os nomes das partes, resta ainda localizar possíveis apelidos ou pseudônimos. Uma expressão regular é utilizada sobre o texto para obter os pseudônimos.

Uma das contribuições pretendidas neste artigo é a modificação do algoritmo de detecção de partes. Para isto serão usados anotadores da biblioteca de processamento de linguagem natural StanfordCoreNLP.

3.1. Biblioteca StanfordCoreNLP

A StanfordCoreNLP (https://stanfordnlp.github.io/CoreNLP) é uma biblioteca java que tem como propósito reunir os anotadores de texto mais usados para processamento de linguagem natural [Manning et al. 2014]. Esta biblioteca foi projetada para ser acessível a uma ampla gama de profissionais, e ser um framework leve e de fácil instalação.

Os múltiplos anotadores estão organizados em um pipeline de processamento de texto. Este pipeline pode ser visto na figura 1.

Interessa, para este trabalho em particular, o anotador Named Entity Recognition (NER). O NER é capaz de, usando modelos pré-treinados, identificar nomes de pessoas, organizações e locais. Este anotador será usado em substituição ao algoritmo do trabalho original que faz uso de um anotador de classes gramaticais do NLTK.

Este anotador irá rodar sobre os mesmos recortes de texto utilizados no algoritmo original. A seleção de cabeçalho e sua divisão em dois, portanto, permanece inalterada. Outra parte do programa anterior que permanece é a que busca, a a partir das palavras do texto o nome de alguma das organizações presentes em uma lista da wikipedia, posto que

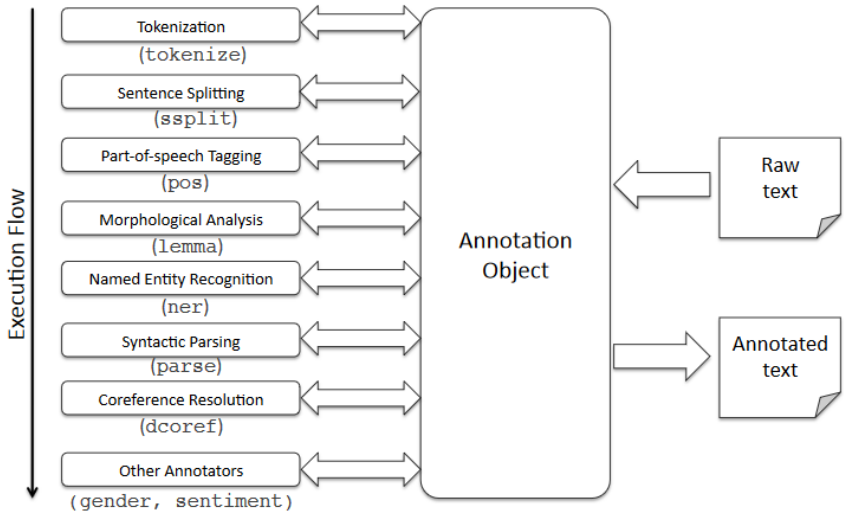

Figura 1. Pipeline do StanfordCoreNLP

esta lista seria a fonte preferencial de resultados ainda que limitada. Do output do anotador, são selecionadas todas as palavras que receberam a tag ORGANIZATION. Estas palavras são concatenadas e serão consideradas o nome da parte contratual.

O algoritmo de identificação de alcunhas e apelidos a partir de expressões regulares se mantém inalterado.

Os modelos usados pelo anotador são os modelos padrão chamados pelo StanfordCoreNLP (é possível usar outros modelos).

4. Identificando Conflitos Normativos

Para extração das normas de um contrato, os autores procuram identificar sentenças contendo verbos modais aos quais são atribuídos significados da lógica deôntica. A tabela 1 apresenta os verbos modais e sua respectiva semântica deôntica.

Verbo modal	Significado deôntico
CAN, MAY	Permissão
MUST, OUGHT, SHALL, WILL	Obrigação
CANNOT, MAY NOT, MUST NOT, SHALL NOT, WILL NOT	Proibição

Tabela 1. Verbos modais e seus respectivos significados deônticos.

Extraídos os dados de um contrato, inicia-se a segunda parte do trabalho, que realiza a proposta dos autores: o processamento destes dados para identificação de conflitos existentes. As normas de um contrato são comparadas em pares e passam por um teste que busca três condições que definem a existência de um potencial conflito. São estas precondições:

- Ambas as normas são aplicadas à mesma parte do contrato
- As normas possuem significados deônticos conflitantes
- As normas referem-se ao mesmo ato a ser observado pela parte.

A definição de significados deônticos conflitantes é aquela descrita por Sadat-Akhavi [Sadat-Akhavi 2003] como a primeira causa de conflitos normativos:

- Obrigação × Permissão
- Obrigação × Proibição
- Permissão × Proibição

A grande dificuldade desta abordagem é identificar a terceira precondição: ambas as normas dizem respeito ao mesmo ato.

5. Similaridade Semântica entre Sentenças Normativas

Na pesquisa original, os autores desenvolveram um algoritmo que utilizava a função de similaridade Wu-Palmer disponível na biblioteca do Natural Language Toolkit. Neste artigo, a troca da função de similaridade é uma das contribuições propostas. A nova função de similaridade fará uso da técnica de *sentence embedding* a partir da biblioteca sent2vec.

5.1. Biblioteca sent2vec

Sent2vec (https://github.com/epfml/sent2vec) é uma biblioteca para python que tem como objetivo proporcionar ao usuário uma ferramenta para a transformação de sentenças em vetores de números, processo conhecido como *sentence embedding* (embutimento/incorporação de palavras) [Pagliardini et al. 2017]. A partir desta transformação é possível executar computações sobre conjuntos de sentenças de modo a descobrir informações sobre seus significados semânticos. No case específico deste trabalho, interessa utilizar os vetores obtidos de forma a estabelecer a similaridade semântica entre duas sentenças.

O sent2vec utiliza técnicas de machine learning para obter os embeddings de sentenças. Trata-se de um método de aprendizado não supervisionado que gera, a partir de um corpus contendo sentenças, um modelo treinado que pode então ser usado para obter embeddings de uma dada sentença qualquer. O sent2vec usa como base a biblioteca FastText do Facebook e é uma extensão do modelo word2vec (ambos usados para embedding de palavras).

Existem duas correntes de pesquisa contrapostas no processamento de linguagem natural. Modelos usando redes neurais profundas são bastante expressivos mas, devido à complexidade, o treinamento com o uso de grandes datasets é lento demais. Em contrapartida, técnicas de aprendizado "superficial" que usam fatorização de matrizes ou modelos bilineares são muito mais rápidas e, consequentemente, podem utilizar datasets muito maiores. O sent2vec segue esta segunda linha de pesquisa.

Para uso do embedding de sentenças do sent2vec é necessário treinar um modelo usando algum dataset suficientemente grande ou então obter um modelo pré-treinado. Felizmente, o repositório do sent2vec oferece algumas opções de modelos de diferentes tamanhos e treinados sobre diferentes datasets. Para os testes apresentados neste artigo, o modelo utilizado é o toronto_unigrams.bin. O uso destes modelos gera uma demanda muito intensa de memória RAM. Por este motivo, só foram realizados testes com este que é o menor modelo disponível no repositório (logo, o menos expressivo) com apenas 2 GB.

Com este modelo, é possível transformar sentenças em vetores de *floats* de 700 dimensões.

5.2. Novo Cálculo de Similaridade Semântica

Quaisquer duas sentenças passadas pela função de embedding retornam vetores de mesmo tamanho. Uma maneira de calcular a similaridade semântica entre estas sentenças é calcular a distância euclidiana entre os vetores resultantes. A função de cálculo é simples e já possui diversas versões em diferentes bibliotecas de python que já são necessárias para a execução de outras etapas do programa apresentado neste trabalho. A implementação usada aqui é a do scipy (scipy.spatial.distance.euclidean()).

Outra maneira de cálculo de distância entre vetores menos conhecida é a distância de cosseno. A função usada nesta pesquisa é a implementada também na biblioteca scipy (scipy.spatial.distance.cosine()). A fórmula correspondente é:

$$\textit{distância\_de\_cosseno}(u, v) = 1 - \frac{u \cdot v}{\|u\|\|v\|}$$

6. Experimentos e Comparações

Uma série de experimentos foram realizados a fim de verificar o impacto das alterações efetuadas sobre o programa original. O novo método de cálculo da similaridade semântica foi usado sobre as mesmas frases de teste da pesquisa original. Os resultados podem ser vistos na tabela 2. A coluna wup traz os resultados usando o programa original que passava as palavras pela função de similaridade semântica de Wu Palmer. As outras duas colunas fazem o sentence embedding das frases usando o sent2vec, uma delas compara os vetores resultantes usando distância euclidiana, outra usando distância de cosseno.

Durante este teste foi possível observar que o sent2vec necessita uma frase limpa e sem pontuação para retornar um resultado confiável. A frase "*An English Dictionary.*" retornava um vetor nulo (contendo apenas zeros) caso fosse deixada com seu ponto final. Curiosamente, isso não ocorreu com qualquer outra frase deste teste. Isso é extremamente relevante para o cálculo de distância de cosseno, pois pode acarretar em divisão por zero. A função de distância de cosseno do scipy não interrompe a computação, permitindo que outros resultados importantes sejam obtidos, mas imprime um aviso no terminal. Em razão disso, a função original que limpava as frases foi alterada de modo a também remover pontos finais, de exclamação e de interrogação. Isso não resolveu completamente o problema, mas melhorou os resultados, inclusive do cálculo de vetores não nulos.

O experimento seguinte foi já sobre o corpus de contratos contendo normas conflitantes. Trata-se do mesmo experimento principal realizado no trabalho original a fim de detectar normas conflitantes. É bom lembrar, o restante do algoritmo de detecção continua o mesmo, isto é, para ser candidato para a computação de similaridade, um par de normas deve fazer referência à mesma parte contratual e representar um dos conflitos deônticos vistos. A tabela 3 mostra os resultados com o cômputo de similaridade semântica por distância euclidiana de embedded sentences. Um par de sentenças normativas é considerado semanticamente igual se sua distância euclidiana é inferior ao *threshold*.

Usando o mesmo sent2vec para produzir os vetores a partir de sentenças, desta vez foi calculada a similaridade usando distância de cosseno. Os resultados para diferentes limiares estão na tabela 4. Um par de sentenças normativas é considerado semanticamente igual se sua distância de cosseno é inferior ao *threshold*.

Frases	wup	sent2vec(euclidiana)	sent2vec(cosseno)
(“I have a pen.”, “Where do you live?”)	0.14	9.03	0.99
(“Red alcoholic drink.”, “An English dictionary.”)	0.37	18.27	1.00
(“I have a pen.”, “Where is ink.”)	0.28	11.27	0.82
(“I have a hammer.”, “Take some apples.”)	0.28	12.66	0.90
(“Canis Familiaris are animals.”, “Dogs are common pets.”)	0.45	10.94	0.59
(“Red alcoholic drink.”, “Fresh apple juice.”)	0.43	18.16	0.83
(“I have a hammer.”, “Take some nails.”)	0.32	12.46	0.88
(“I like that bachelor.”, “I like that unmarried man.”)	0.83	7.32	0.55
(“Red alcoholic drink.”, “A bottle of wine.”)	0.44	13.77	0.71
(“Red alcoholic drink.”, “Fresh orange juice.”)	0.46	16.60	0.77
(“It is a dog.”, “It is a log.”)	0.87	7.91	0.69
(“A glass of cider”, “A full cup of apple juice.”)	0.46	10.44	0.74
(“It is a dog.”, “That must be your dog.”)	0.37	4.82	0.30
(“Dogs are animals.”, “They are common pets.”)	0.64	12.90	0.66
(“It is a dog.”, “It is a pig.”)	0.87	7.69	0.60
(“John is very nice.”, “Is John very nice?”)	0.68	4.73	0.17

Tabela 2. Comparativo dos resultados obtidos por diferentes métodos de cálculo de similaridade entre os pares de frases.

Threshold	1.50	1.80	2.15	2.50	3.00
True positives	79	79	84	84	84
True negatives	13636	13634	13633	13623	13512
False positives	19	21	22	32	143
False negatives	42	42	37	37	37
Accuracy	1.00	1.00	1.00	0.99	0.99
Precision	0.81	0.79	0.79	0.72	0.37
Recall	0.65	0.65	0.69	0.69	0.69
F-score	0.72	0.71	0.74	0.71	0.48
True positive rate	0.65	0.65	0.69	0.69	0.69
Specificity	1.00	1.00	1.00	1.00	0.99
False positive rate	0.00	0.00	0.00	0.00	0.01
Total de conflitos	111				

Tabela 3. Resultados para teste realizado usando o embedding de sentenças do sent2vec e calculando a similaridade entre sentenças por distância euclidiana.

O resultado é parecido com o visto na distância euclidiana. É possível notar uma propensão do método em captar mais positivos verdadeiros. Em contrapartida, mais classificações resultam também em falsos positivos.

Também foi realizado o mesmo experimento trocando o reconhecedor das partes contratuais pelo algoritmo que utiliza a biblioteca StanfordCoreNLP. Neste experimento, o cálculo de similaridade semântica foi o mesmo usado na tabela 3. Em razão disso, é possível observar que a perda de qualidade nos resultados obtidos se deve exclusivamente à troca do detector de partes contratuais. Quando notada esta perda, o algoritmo de reconhecimento foi executado sobre todos os contratos do corpus de teste para comparação com o algoritmo original. Foi possível notar que , apesar de várias das entidades reconhecidas resultarem em uma detecção mais limpa e mais fiel aos nomes originais das mesmas, algumas entidades não eram reconhecidas pelo anotador NER como organizações, resultando na ausência total de nomes para algumas partes. A solução tentada foi semelhante a

Threshold	0.05	0.07	0.10	0.15	0.20
True positives	77	81	82	84	85
True negatives	13635	13635	13632	13630	13628
False positives	20	20	23	25	27
False negatives	44	40	39	37	36
Accuracy	1.00	1.00	1.00	1.00	1.00
Precision	0.79	0.80	0.78	0.77	0.76
Recall	0.64	0.67	0.68	0.69	0.70
F-score	0.71	0.73	0.73	0.73	0.73
True positive rate	0.64	0.67	0.68	0.69	0.70
Specificity	1.00	1.00	1.00	1.00	1.00
False positive rate	0.00	0.00	0.00	0.00	0.00

Tabela 4. Resultados para teste usando o embedding de sentenças do sent2vec com cálculo de similaridade por distância de cosseno.

implementada pelos autores originais. Usando um anotador de classes gramaticais (part-of-speech – POS), mas, desta vez, também do StanfordCoreNLP, procurou-se reunir as palavras anotadas como nomes próprios e substantivos quando o anotador NER falhava. Esta é a versão cujo resultado pode ser visto na tabela 5.

Mesmo assim é possível observar uma queda na qualidade dos resultados quando comparamos com o experimento que utiliza o detector de partes contratuais do trabalho original. Um dado mais preocupante é revelado quando se observa a quantidade de negativos verdadeiros obtidos neste teste. É visível uma diminuição acentuada em relação aos outros experimentos realizados. Uma perda de mais de mil pontos deveria sugerir um erro de classificação de mais de mil pares de normas, mas não é isso o que os dados mostram, posto que os demais indicadores não absorveram um número equivalente de classificações. Isto se deve ao fato de que a falha da função de detecção de partes contratuais faz com que algumas sentenças contratuais sequer sejam reconhecidas como sentenças normativas, porque uma das características que constituem tais sentenças para o programa é a menção a uma das partes envolvidas.

O repositório desta contribuição possui o output de um teste de identificação das partes Identified_parties.txt.

Por fim, uma tabela comparativa foi criada (tabela 6). Nesta tabela, os resultados mais significativos de cada experimento foram reunidos para que seja possível observar as variações decorrentes das diferentes abordagens.

Os resultados apresentados com o uso do StanfordCoreNLP sobressaem negativamente como os menos satisfatórios. Este autor não soube utilizar devidamente a biblioteca para extrair bons resultados na pesquisa. Além disso, o processo de anotação é relativamente demorado e também ocupa memória RAM. Carregar os modelos necessários e anotar as sentenças leva algo em torno de 2 minutos. O mesmo programa (usando apenas o sent2vec) leva apenas de 7 a 20 segundos para executar o experimento. Como o sent2vec também faz uso intenso de memória RAM, as bibliotecas podem acabar ficando sem recursos para alocação de memória, o que aconteceu ao menos uma vez durante os experimentos.

Threshold	1.50	1.80	2.15	2.50	3.00
True positives	74	74	79	79	79
True negatives	12442	12440	12439	12429	12319
False positives	17	19	20	30	140
False negatives	47	47	42	42	42
Accuracy	0.99	0.99	1.00	0.99	0.99
Precision	0.81	0.80	0.80	0.72	0.36
Recall	0.61	0.61	0.65	0.65	0.65
F-score	0.70	0.69	0.72	0.69	0.46
True positive rate	0.61	0.61	0.65	0.65	0.65
Specificity	1.00	1.00	1.00	1.00	0.99
False positive rate	0.00	0.00	0.00	0.00	0.01

Tabela 5. Resultados para teste usando o embedding de sentenças do sent2vec, cálculo de similaridade por distância euclidiana e o reconhecedor de entidades do StanfordCoreNLP

Ainda sobre o tempo, talvez seja essa a maior vantagem sobre o programa do artigo original. Este levava algo em torno de 1 hora e 20 minutos para ser executado, devido à função de similaridade original (ver especificações do equipamento usado na seção 7). Como já dito, usando somente o sent2vec o programa demora apenas 20 segundos para terminar.

Não apenas o tempo apresenta uma significativa melhora, os resultados também mostram desempenho superior, ainda que não na mesma proporção.

Método	Original	Sent2vec Euclidiana	Sent2vec Cosseno	Sent2vec Euclidiana + StanfordCoreNLP
Threshold	60%	2.15	0.20	2.15
True positives	81	84	85	79
True negatives	13630	13633	13628	12439
False positives	25	22	27	20
False negatives	40	37	36	42
Accuracy	1.00	1.00	1.00	1.00
Precision	0.76	0.79	0.76	0.80
Recall	0.67	0.69	0.70	0.65
F-score	0.71	0.74	0.73	0.72
True positive rate	0.67	0.69	0.70	0.65
Specificity	1.00	1.00	1.00	1.00
False positive rate	0.00	0.00	0.00	0.00

Tabela 6. Comparação entre resultados de diferentes métodos

7. Especificações da Máquina Usada

A reprodução do experimento original foi feita em um computador com 8 GB de memória RAM, processador Intel Core i7, 3.60 GHz, placa de video Nvidia GeForce GT 720,

sistema operacional Windows 64 bits. Esta mesma máquina foi usada para rodar uma máquina virtual onde foram executados os experimentos de contribuição. Esta máquina virtual foi executada com virtualbox, 5 GB de RAM, 2 *cores*, sistema operacional Linux Mint 19 Cinnamon 64 bits.

8. Conclusão

Este artigo buscou contribuir para a pesquisa original de [Aires et al. 2017] modificando a maneira de computar a similaridade semântica entre as sentenças normativas assim como a maneira pela qual o programa detecta as partes contratantes. A nova função de similaridade, fazendo uso de sentence embeddings da biblioteca sent2vec, apresenta um ganho significativo de tempo de execução em relação ao trabalho original passando de um total de 1 hora e 20 minutos para apenas 20 segundos. O novo programa exige o uso de modelos pré-treinados de machine learning que podem ser bastante pesados e tornar inviável o uso para certas máquinas. No entanto, o experimento foi realizado com o menor dos modelos encontrados, e em uma máquina virtual de apenas 5 GB de RAM. Ainda assim os resultados melhoraram em relação ao original.

No entanto, a nova função de detecção de partes contratuais usando a biblioteca StanfordCoreNLP não obteve o mesmo sucesso. Os resultados apresentados são inferiores aos do artigo original e fazem com que algumas sentenças contratuais sequer sejam reconhecidas como sentenças normativas, isto é, alguns pares de normas nem mesm chegam a passar pela função de classificação. Além disso, a função sobrecarrega ainda mais a memória, aumentando a especificação mínima para uso e estendendo o tempo de execução.

É possível pensar para o futuro na utilização de novos modelos mais robustos e pesados, disponíveis no repositório do sent2vec, para a função de sentence embedding em uma máquina com melhores especificações. O mesmo pode ser feito em relação ao StanfordCoreNLP embora outros modelos da biblioteca não estejam tão prontamente disponíveis.

Referências

Aires, J. P., Pinheiro, D., Lima, V. S. d., and Meneguzzi, F. (2017). Norm conflict identification in contracts. *Artificial Intelligence and Law*, 25(4):397–428.

Bird, Steven; Loper, E. K. E. (2009). *Natural Language Processing with Python*. O'Reilly Media Inc.

Gao, X., P. Singh, M., and Mehra, P. (2012). Mining business contract for service exceptions. *Services Computing, IEEE Transactions on*, 5:1 – 1.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

McNamara, P. (2018). Deontic logic. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, fall 2018 edition.

Pagliardini, M., Gupta, P., and Jaggi, M. (2017). Unsupervised learning of sentence embeddings using compositional n-gram features. *CoRR*, abs/1703.02507.

Princeton University. About wordnet. Disponível em: https://wordnet.princeton.edu/.

Sadat-Akhavi, S.-A. (2003). *Methods of Resolving Conflicts between Treaties*. Brill, Leiden, The Netherlands.

Vranes, E. (2006). The definition of 'norm conflict' in international law and and legal theory. *The European Journal of International Law*, 17:395 – 418.

Wikipedia contributors (2018). List of companies of the united states — Wikipedia, the free encyclopedia. [Online; accessed 23-September-2018].