Processamento de linguagem natural (PLN) na indústria AEC: uma abordagem para tradução de regulamentos edilícios brasileiros para o domínio BIM

Natural language processing in the AEC industry: an approach on the translation of Brazilian building regulations for the BIM domain.

Thiago Nieves
Universidade de São Paulo | São Paulo| Brasil | thiago.nieves@usp.br
Eduardo Alves de Mendonça
Universidade de São Paulo | São Paulo| Brasil | mendonca.eduardo@usp.br
Sérgio Leal Ferreira
Universidade de São Paulo | São Paulo| Brasil | sergio.leal@usp.br

Resumo

A Modelagem da Informação da Construção (BIM) permite ampla geração, armazenagem e manipulação de informação sobre as edificações, abrindo caminhos para maior integração entre áreas do conhecimento. A evolução do entendimento sobre PLN contribui para relacionar as regulamentações edilícias e aplicativos BIM, fundamentando sistemas computacionais para a verificação automática de conformidade com regras de edificação. Abordando um dos três elementos essenciais de sistemas de verificação automática de regras, a geração de regras computáveis, através da PLN, este trabalho espera contribuir em direção à efetividade desses sistemas no contexto brasileiro.

Palavras-chave: Modelagem da Informação da Construção. BIM. Verificação Automatizada de Regras. Processamento de Linguagem Natural. PLN.

Abstract

Building Information Modeling (BIM) allows wide generation, storage, and manipulation of information regarding buildings, opening paths to greater integration between knowledge areas. The evolution on the understanding of Natural Language Processing (NLP) contributes to link BIM applications and construction regulations, supporting computer systems for automated Code-checking. Addressing one of the three essential elements in Code-checking systems, the

Como citar:

NIEVES, T.; MENDONÇA, E. A. de .; FERREIRA, S. L. . Processamento de Linguagem Natural na indústria AEC: uma abordagem para tradução de regulamentos edilícios brasileiros para o domínio BIM. *In*: SIMPÓSIO BRASILEIRO DE TECNOLOGIA DA INFORMAÇÃO E COMUNICAÇÃO NA CONSTRUÇÃO, 3., 2021, Uberlândia. **Anais [...]**. Porto Alegre: ANTAC, 2021. p. 1-14. Disponível em: https://eventos.antac.org.br/index.php/sbtic/article/view/613. Acesso em: 3 ago. 2021.

generation of computable rules, through NLP, this work expects to contribute towards the effectiveness of these systems in the Brazilian context.

Keywords: Building Information Modeling. BIM. Automated Code-checking. Natural Language Processing. NLP.

INTRODUÇÃO

A consecução de um edifício contemporâneo envolve a complexidade física do objeto em si e das estruturas legais, financeiras e organizacionais demandadas para sua implantação. As regulamentações, normas e legislações edilícias, naturais dos diversos mercados imobiliários ao redor do mundo, colocam um importante desafio para os projetistas e proprietários desses edifícios, assim como a crescente complexidade e sofisticação da infraestrutura e instalações embarcadas nesses objetos [1].

No Brasil, a responsabilidade de garantir a observância dessas regras recai sobre os projetistas das diversas disciplinas técnicas da indústria AEC, enquanto já existem implementações tecnológicas capazes de auxiliar esses profissionais nas suas atuações de avaliação de projetos, objetivando garantir a conformidade com o regramento edilício. Essas implementações são denominadas de sistemas de *Code-checking* ou *Rule-checking*. Em português poderia ser utilizado o termo Verificação Automática de Regras Edilícias – VARE, sigla utilizada neste trabalho.

No cenário da compulsoriedade da observância de regras, questiona-se a possibilidade de tratá-las de forma digital, automatizando ainda mais os sistemas VARE para auxiliar projetistas no cumprimento de suas responsabilidades e a resposta a esse questionamento pode estar vinculada a métodos automatizados de tradução das regras em Linguagem Natural (LN) para regras computáveis, o que é denominado de PLN.

Na revisão da literatura não se identificaram trabalhos nacionais relacionando PLN e regras edilícias da indústria AEC entre 2009 e 2019, nos repositórios das dez principais instituições universitárias de pesquisa do Brasil[1]. As buscas por esses trabalhos foram realizadas através da plataforma digital da Biblioteca Digital Brasileira de Teses e Dissertações (BDTD)[2]. Nessas buscas encontrou-se que os maiores participantes nacionais na discussão sobre PLN estão no campo da ciência da computação e da linguística, produtores de quase 85% das publicações na área [2].

O objetivo deste artigo é demonstrar como se pode explorar esses campos do conhecimento para aplicar um tratamento inicial a um texto regulatório da língua portuguesa, utilizando técnicas de PLN, para dar início à automação da tradução dessas regras.

---

[1] Ranking Universitário Folha 2019 (RUF), disponível em: https://ruf.folha.uol.com.br/2019/ranking-de-universidades/principal/

[2] Biblioteca Digital Brasileira de Teses e Dissertações, disponível em: https://bdtd.ibict.br/vufind/

SISTEMAS VARE

Sistemas VARE são constituídos de três elementos principais: a informação da construção disponibilizada de maneira computável, ou seja, um modelo BIM; a capacidade de capturar os códigos de edificação, transformando-os em regras de formato computável; e uma aplicação que avalie o modelo BIM perante essas regras [3].

Entende-se por LN aquela utilizada pelos seres humanos em suas comunicações ordinárias, tais como a língua Inglesa, o Português ou o Hindi [4], sendo esse o formato tradicional de armazenamento e registro das regras edilícias. No entanto, as regras computáveis, os equivalentes digitais de suas contrapartes tradicionais, devem ser traduzidas desses corpos de textos em LN para que possam ser utilizadas em sistemas VARE.

Uma das formas de efetuar essa tradução é reescrever as regras através de linguagens de programação para que se integrem a um determinado sistema informatizado. As regras ficam, então, codificadas dentro daquele sistema e são personalizadas para ele. Essa abordagem pode chamar-se *hard-coded.* As regras estarão rigidamente atreladas àquele sistema. Contudo, a automação completa de sistemas VARE não pode ser obtida através do uso dessa abordagem [5].

Atualmente, sistemas VARE, ou estão vinculados a plataformas comerciais, ou a um domínio específico. Somando-se às aplicações *hard-coded*, a maioria desses sistemas são inflexíveis, resistentes a mudanças e de manutenção dispendiosa [6]. Por outro lado, os códigos e regulamentações edilícias sofrem revisões e atualizações constantes.

Uma proposta recente de implementação de sistema VARE totalmente automatizado [5] integra três tipos de algoritmos em uma única plataforma computacional: algoritmos de PLN semânticos para extração automática de informação de textos regulatórios e transformação em cláusulas lógicas; algoritmos semânticos em linguagem EXPRESS[3] para extração de informações dos projetos em BIM; e algoritmos de racionalização lógica baseados em semântica para análise de conformidade entre as informações extraídas dos textos regulatórios e dos modelos BIM.

O interesse deste trabalho recai sobre os algoritmos do primeiro tipo.

PLN

O PLN faz parte do campo da Inteligência Artificial (IA) objetivando tornar mais fácil a interação entre pessoas e computadores [8] e origina sistemas computacionais capazes de mimetizar as aptidões da linguagem humana valendo-se também de

[3] EXPRESS é uma linguagem de especificação de dados definida na ISO 10303-1. Consiste de elementos de linguagem que permitem definição de dados não ambíguos e especificação de restrições sobre os dados definidos [7].

conceitos e técnicas dos campos da filosofia, linguística, psicologia e disciplinas correlatas [8].

Em outras palavras, o PLN pode ser entendido como qualquer manipulação computadorizada da LN, desde uma simples contagem de palavras em um texto até a compreensão útil de enunciados completos [4].

MÉTODO

Este estudo seguiu as seguintes etapas de trabalho:

- Arbitragem, escolha e delimitação de um excerto de norma regulatória brasileira para ser objeto do PLN;
- Delimitação do escopo do PLN;
- Determinação do ferramental tecnológico para o PLN;
- Captura do excerto de norma: fonte e tratamento;
- Implementação e aplicação do PLN;
- Análise e apresentação dos resultados.

OBJETO DO TRABALHO E LIMITAÇÕES

O estudo delimitou-se às Normas Técnicas Brasileiras voltadas para a indústria AEC, no entanto, o corpo de regulamentações edilícias vai além destas, englobando outras normas e leis das esferas estadual, municipal e federal, além de regulamentações de concessionárias de serviços públicos e outras.

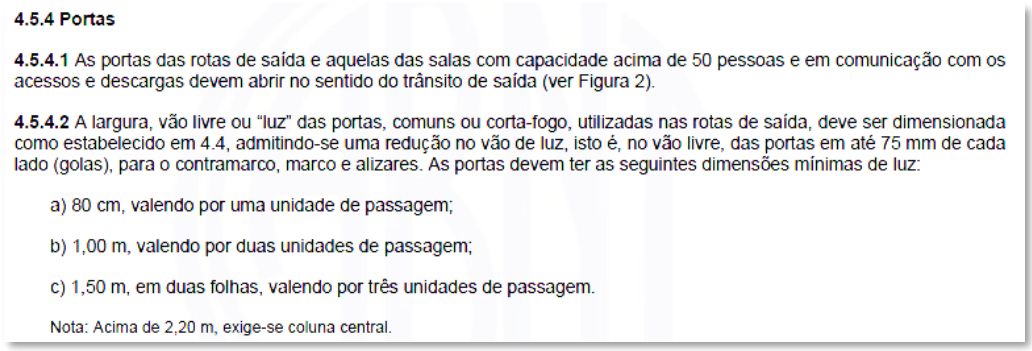
ARBITRAGEM, ESCOLHA E DELIMITAÇÃO DO EXCERTO DE NORMA PARA PLN

Para delimitação do excerto de regulamentação a ser processado, arbitrou-se retirá-lo da ABNT NBR 9077:2001, norma que trata de saídas de emergência em edifícios. Fundamentou-se em adotar a cidade de São Paulo como cenário, pois neste assunto e local específicos há sobreposição de regulamentação entre essa norma e outras três regulamentações [9-11].

A sobreposição de regulamentações aumenta a carga de trabalho que recai sobre os profissionais projetistas da construção, e é mais um motivo para procurar o auxílio computacional.

Do corpo da ABNT NBR 9077:2001, limitou-se aos itens 4.5.4 ao 4.5.4.2, Figura 1, que tratam de portas.

Figura 1: Excerto da NBR 9077



4.5.4 Portas

4.5.4.1 As portas das rotas de saída e aquelas das salas com capacidade acima de 50 pessoas e em comunicação com os acessos e descargas devem abrir no sentido do trânsito de saída (ver Figura 2).

4.5.4.2 A largura, vão livre ou "luz" das portas, comuns ou corta-fogo, utilizadas nas rotas de saída, deve ser dimensionada como estabelecido em 4.4, admitindo-se uma redução no vão de luz, isto é, no vão livre, das portas em até 75 mm de cada lado (golas), para o contramarco, marco e alizares. As portas devem ter as seguintes dimensões mínimas de luz:

a) 80 cm, valendo por uma unidade de passagem;

b) 1,00 m, valendo por duas unidades de passagem;

c) 1,50 m, em duas folhas, valendo por três unidades de passagem.

Nota: Acima de 2,20 m, exige-se coluna central.

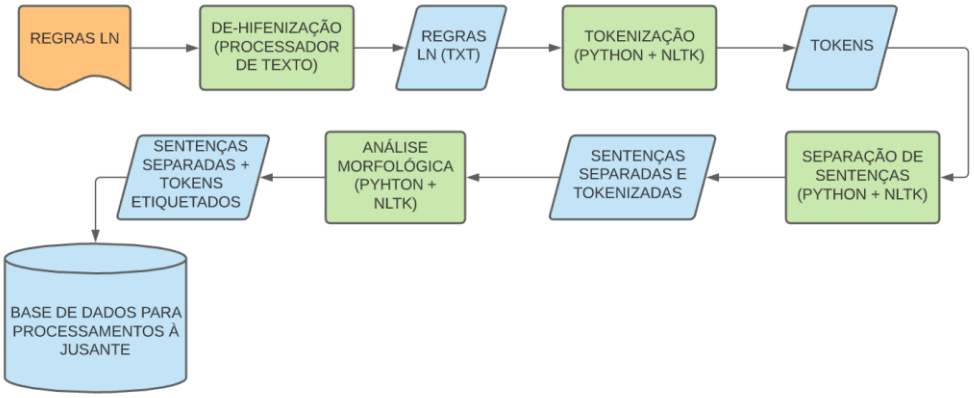
Fonte: ABNT NBR 9077:2001.

DELIMITAÇÃO DO ESCOPO DO PLN

O escopo de PLN em um sistema VARE pode ser delimitado a duas etapas principais: o pré-processamento do texto original em LN (Figura 2) e a geração de características sintáticas e semânticas que descrevem esse texto [5].

A etapa de pré-processamento pode ser subdividida em quatro técnicas de PLN: de-hifenização, *tokenização*, separação de sentenças e análise morfológica [5]. Este estudo apresenta a aplicação dessas quatro técnicas.

Figura 2: Fluxograma da etapa de pré-processamento



Fonte: os autores.

FERRAMENTAL TECNOLÓGICO PARA O PLN

A plataforma computacional de PLN escolhida para este estudo foi o *Python*[4].

O *Python* possui licença *open source*[5], portanto de uso e distribuição livres. Possui também uma extensa disponibilidade de bibliotecas de funções e métodos livres.

4 *Python*: https://www.python.org/about/

5 *Open source*: https://opensource.org/about

Uma dessas bibliotecas é a *Natural Language Toolkit – NLTK*[6], plataforma específica para trabalhar com dados em LN. Esta biblioteca foi a principal ferramenta utilizada neste trabalho.

CAPTURA DO EXCERTO DE NORMA: FONTE E TRATAMENTO

Muitos textos estão disponíveis em formatos binários como PDF e MSWord que só podem ser abertos com a utilização de software especializado. É possível escrever uma rotina (*script*) em Python para executar esta captura e transformação do formato PDF para TXT, no entanto, optou-se aqui, por uma captura mais simples, utilizando um aplicativo de edição de documentos em formato PDF e copiando o excerto para um editor de textos para salvá-lo no formato TXT. Esse formato é uma das fontes de dados para o PLN utilizando *Python* e a biblioteca NLTK.

IMPLEMENTAÇÃO DO PLN

A linguagem de programação utilizada foi a Python 3.8 distribuída na plataforma *Anaconda*[7] e os *scripts* foram desenvolvidos na aplicação *web Jupyter Notebook*[8], distribuída na mesma plataforma.

DE-HIFENIZAÇÃO

É necessário evitar potenciais erros futuros de processamento através da remoção dos hifens que indicam a continuação de palavras entre duas linhas de texto e este processo se denomina de-hifenização [5]. Pode ser feito, a princípio, utilizando os processadores de texto disponíveis comercialmente ou de uso livre, como o *Word* da *Microsoft* ou o *Writer*, do pacote *LibreOffice* da organização *The Document Foundation*. No entanto também é possível escrever uma rotina, em Python, para aplicar essa técnica de PLN.

TOKENIZAÇÃO

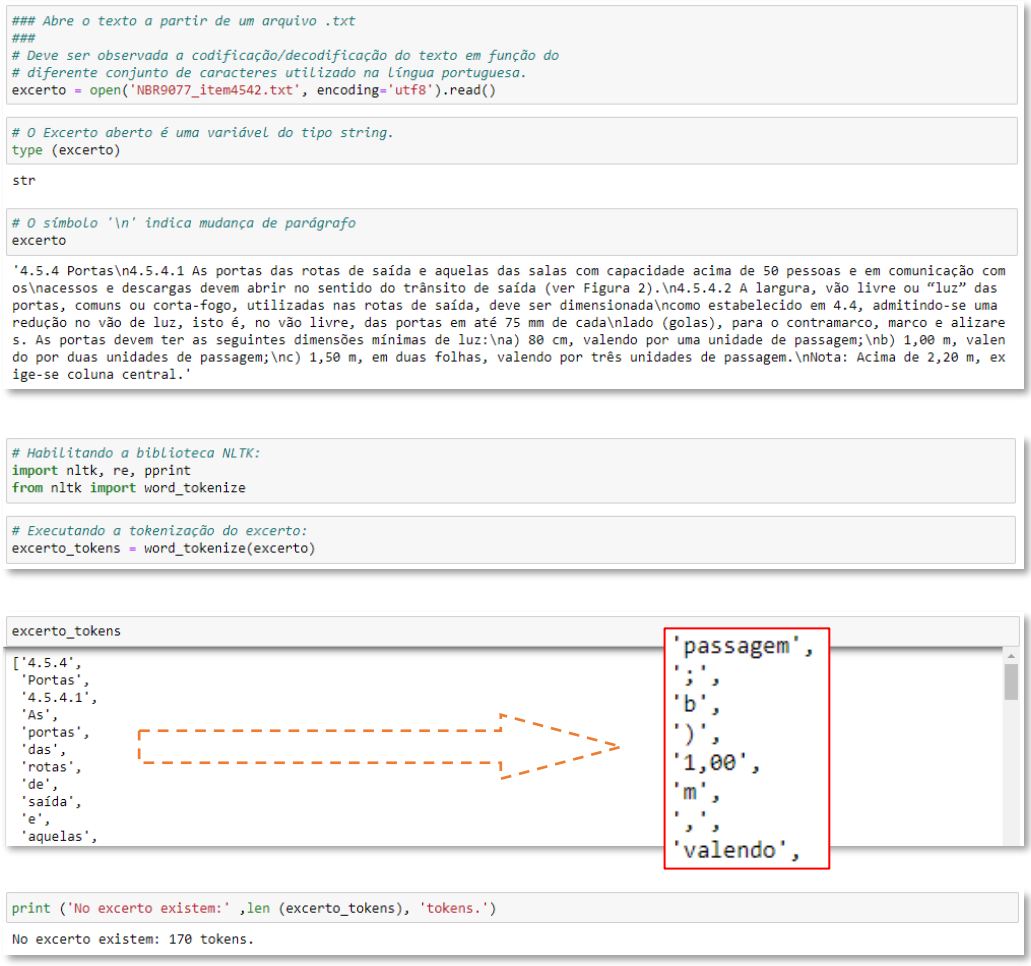
Em PLN, *token* é o nome de uma sequência de caracteres que pode ser uma palavra, como 'escada', ou um símbolo de pontuação como ';' [4]. Sendo assim, *tokenização* é separação e indexação dessas sequências de caracteres, conforme demonstrado na Figura 3.

[6] Natural Language Toolkit: http://www.nltk.org/

[7] https://www.anaconda.com/

[8] https://jupyter.org/

Figura 3: *Tokenização* no *Jupyter Notebook*



Fonte: os autores.

No desenvolvimento de um *tokenizador é importante* levar em consideração o auxílio de textos que tenham sido *tokenizados* manualmente e que, portanto, podem constituir o padrão de qualidade a ser seguido [4].

Com a base comparativa selecionada, um *script* que execute automaticamente o controle de qualidade deverá ser implementado. Esse *script* fará a varredura dos resultados e identificará as diferenças em relação à base padrão de qualidade.

Ao fim da etapa de tokenização, o resultado é o excerto distribuído dentro de uma lista indexada, tornando cada *token* do texto acessível [4].

SEPARAÇÃO DE SENTENÇAS

O reconhecimento dos limites de uma sentença auxilia na distinção das disposições de um corpo de regulamentações e é obtido com a técnica de separação de sentenças [5]. Tem como objetivo retornar automaticamente unidades mínimas compreensíveis de texto, ou seja, enunciados de sentido completo.

Pode ser compreendida como uma tarefa de classificação de pontuação, ou seja, quando se encontra um símbolo que potencialmente termina uma sentença, decide-se ou não pelo seu fim [4].

Figura 4: Separação de sentenças no *Jupyter Notebook*

```
# Separação de sentenças - aplicação da função da biblioteca NLTK
sentencas = nltk.sent_tokenize(excerto)
```

```
cont = 1
for item in sentencas:
    print ('Sentença',cont,':')
    print (item)
    print ()
    cont = cont+1
```

```
Sentença 1 :
4.5.4 Portas
4.5.4.1 As portas das rotas de saída e aquelas das salas com capacidade acima de 50 pessoas e em comunicação com os
acessos e descargas devem abrir no sentido do trânsito de saída (ver Figura 2).

Sentença 2 :
4.5.4.2 A largura, vão livre ou "luz" das portas, comuns ou corta-fogo, utilizadas nas rotas de saída, deve ser dimensionada
como estabelecido em 4.4, admitindo-se uma redução no vão de luz, isto é, no vão livre, das portas em até 75 mm de cada
lado (golas), para o contramarco, marco e alizares.

Sentença 3 :
As portas devem ter as seguintes dimensões mínimas de luz:
a) 80 cm, valendo por uma unidade de passagem;
b) 1,00 m, valendo por duas unidades de passagem;
c) 1,50 m, em duas folhas, valendo por três unidades de passagem.

Sentença 4 :
Nota: Acima de 2,20 m, exige-se coluna central.
```

Fonte: os autores.

Assim como na *tokenização*, o processo de separação de sentenças (Figura 4) deverá estar associado a um banco de texto previamente tratado, de modo que, dele, sejam extraídas características (*features)* que permitam o subprocesso de aprendizado de máquina (*Machine Learning*) resultando no *script* segmentador.

ANÁLISE MORFOLÓGICA

A análise morfológica ajuda no reconhecimento de conceitos ontológicos. Ela executa o reconhecimento das formas de uma dada palavra e as mapeia em sua forma lexical [5]. Ela se dá através da tarefa de etiquetagem morfossintática, onde as palavras de um texto são selecionadas e etiquetadas conforme suas distribuições morfológicas e sintáticas [12]. Esse processo denomina-se *Part-Of-Speech tagging (POS-tagging)* [4].

O *POS-tagging* é um procedimento complexo. Envolve a adoção de um etiquetador automático que pode ser construído a partir de um corpus de texto que tenha sido manualmente etiquetado. O corpus deve ser extenso, pois será utilizado como fonte em um subprocesso de aprendizado de máquina. Através de *scripts* apropriados é possível criar uma rotina computacional para etiquetar um outro texto que se deseje processar.

Os *scripts* para construção do etiquetador morfológico apresentados neste trabalho referenciaram-se no trabalho de George-Bogdan Ivanov [13] e o procedimento envolve a utilização de um corpus licenciado pela Universidade de São Paulo, denominado Mac-Morpho [14] [15] [16]. Uma versão do Mac-Morpho está disponível dentro da biblioteca NLTK utilizada neste estudo.

O procedimento inicia-se com o carregamento das ferramentas na aplicação *Jupyter Notebook*, determinando o acesso ao corpus e às frases etiquetadas, como apresentado na Figura 5.

Figura 5: Inicialização do etiquetador morfológico no *Jupyter Notebook*

```
# Este script tem como objetivo treinar um etiquetador morfológico para classificação automática de palavras para posterior
# análise morfológica de textos de códigos edilícios

import nltk, re, pprint
from nltk import word_tokenize, pos_tag

# word_tokenize e pos_tag são funções embarcadas na bilbioteca NLTK
# word_tokenize cria um lista de palavras, ou tokens, a partir de um string
# pos_tag faz a etiquetagem morfológica de uma lista de palavras na língua inglesa.
```

```
# Aqui está sendo carregada a base de dados para treinamento do etiquetador
# Trata-se de um corpus: Mac-Morpho is a corpus of Brazilian Portuguese texts annotated with part-of-speech tags.
# http://nilc.icmc.usp.br/macmorpho/

# Essa base de dados também está disponível dentro da biblioteca NLTK e é a base que está sendo utilizada aqui.

frases_etiquetadas = nltk.corpus.mac_morpho.tagged_sents()
```

```
frases_etiquetadas
```

```
[[('Jersei', 'N'), ('atinge', 'V'), ('média', 'N'), ('de', 'PREP'), ('Cr$', 'CUR'), ('1,4', 'NUM'), ('milhão', 'N'), ('em', 'PR
EP|+'), ('a', 'ART'), ('venda', 'N'), ('de', 'PREP|+'), ('a', 'ART'), ('Pinhal', 'NPROP'), ('em', 'PREP'), ('São', 'NPROP'),
('Paulo', 'NPROP')], [('Programe', 'V'), ('sua', 'PROADJ'), ('viagem', 'N'), ('a', 'PREP|+'), ('a', 'ART'), ('Exposição', 'NPRO
P'), ('Nacional', 'NPROP'), ('do', 'NPROP'), ('Zebu', 'NPROP'), (',', ','), ('que', 'PRO-KS-REL'), ('começa', 'V'), ('dia',
'N'), ('25', 'N|AP')], ...]
```

Fonte: os autores.

No fim da Figura 5, observa-se que cada palavra vem associada a uma sigla em letra maiúscula, por exemplo (‘atinge’, V). Este é um *token* etiquetado morfologicamente, ‘V’ significa verbo. O corpus Mac-Morpho possui mais de 1,1 milhão de palavras etiquetadas, sendo, portanto, um corpus extenso (Figura 6). Ao fim destas primeiras ações obtém-se uma variável composta de frases cujos *tokens* estão todos etiquetados. No exemplo, a variável é denominada *frases_etiquetadas*.

Figura 6: Extensão do corpus Mac-Morpho (51.397 frases e 1.170.095 palavras)

```
print (frases_etiquetadas[0])
print ("Frases etiquetadas: ", len(frases_etiquetadas))
print ("Palavras etiquetadas:", len(nltk.corpus.mac_morpho.tagged_words()))
```

```
[('Jersei', 'N'), ('atinge', 'V'), ('média', 'N'), ('de', 'PREP'), ('Cr$', 'CUR'), ('1,4', 'NUM'), ('milhão', 'N'), ('em', 'PRE
P|+'), ('a', 'ART'), ('venda', 'N'), ('de', 'PREP|+'), ('a', 'ART'), ('Pinhal', 'NPROP'), ('em', 'PREP'), ('São', 'NPROP'), ('P
aulo', 'NPROP')]
Frases etiquetadas:  51397
Palavras etiquetadas: 1170095
```

Fonte: os autores.

Na sequência, o *script* responsável por atribuir características a cada *token* que será processado deve ser criado (Figura 7). O *script* vincula dezessete características a cada *token* (linhas iniciadas em vermelho e precedidas dos números 00 até 16). Entre essas características (*features*) estão: o *token* em si; se o *token* é o primeiro ou o último dentro da frase; se o primeiro caractere do *token* é uma letra maiúscula; prefixos e sufixos do *token*; entre outros. Estas são as características a serem relacionadas com o corpus modelo no subprocesso de aprendizado de máquina que resultará no *script* etiquetador automático.

Figura 7: Função de atribuição de características (*features*)

```
def features(frase, ind):
    """ frase: [p1, p2, ...], índice: o índice da palavra """
    return {
        '00_palavra': frase[ind],
        '01_primeira': ind == 0,
        '02_ultima': ind == len(frase) - 1,
        '03_prm_maiuscula': frase[ind][0].upper() == frase[ind][0],
        '04_int_maiuscula': frase[ind].upper() == frase[ind],
        '05_int_minuscula': frase[ind].lower() == frase[ind],
        '06_prefixo-1': frase[ind][0],
        '07_prefixo-2': frase[ind][:2],
        '08_prefixo-3': frase[ind][:3],
        '09_sufixo-1': frase[ind][-1],
        '10_sufixo-3': frase[ind][-3:],#gerundio
        '11_sufixo-5': frase[ind][-5:],#adverbio de modo "...mente"
        '12_plvr_ante': '' if ind == 0 else frase[ind - 1],
        '13_plvr_post': '' if ind == len(frase) - 1 else frase[ind + 1],
        '14_possui_hifen': '-' in frase[ind],
        '15_numerica': frase[ind].isdigit(),
        '16_possui_maiusculas_int': frase[ind][1:].lower() != frase[ind][1:]
    }
```

Fonte: os autores.

O próximo passo é a elaboração da função com objetivo de identificar as etiquetas contidas no corpus referência para fornecê-las ao etiquetador automático, conforme a Figura 8.

Figura 8: Função de coleta de etiquetas morfológicas

```
# função para separar as etiquetas do corpus referência e fornecê-las ao etiquetador
def rem_etq(frase_etiquetada):
    return [p for p, etq in frase_etiquetada]
```

Fonte: os autores.

Com isto, chega-se ao início do subprocesso de aprendizagem de máquina, quando se faz a separação dos dados do corpus para criação de dois conjuntos de dados: os dados de treinamento e os dados de teste. Estes dados são as frases etiquetadas do corpus referência, sendo que 75% das frases serão destinadas ao treino e 25% para a aferição do treinamento, conforme a Figura 9.

Figura 9: Script para separação dos conjuntos de dados de treino e teste

```
# Separar conjunto de dados para treino e teste
corte = int(.75 * len(frases_etiquetadas))
frases_treino = frases_etiquetadas[:corte]
frases_teste = frases_etiquetadas[corte:]

print (len(frases_treino))   # 2935 - caso fosse com o corpus Treebank (língua inglesa)
print (len(frases_teste))   # 979 - caso fosse com o corpus Treebank (língua inglesa)

def conv_em_cj_dados(frases_etiquetadas):
    X, y = [], []

    for etq in frases_etiquetadas:
        for ind in range(len(etq)):
            try:
                X.append(features(rem_etq(etq), ind))
                y.append(etq[ind][1])
            except:
                print('ERO+ROOOOO -----')
                print(rem_etq(etq))
    return X, y

X, y = conv_em_cj_dados(frases_treino)
```

```
38547
12850
ERO+ROOOOO -----
['Com', '"', 'B', 'A', 'B', 'Y', '']
ERO+ROOOOO -----
['A', 'entrada', 'é', 'por', 'a', 'Olavo', 'Fontoura', '', ',', 's/nº']
```

Fonte: os autores.

A seguir, executa-se o aprendizado de máquina e o teste do aprendizado, o que resulta na precisão (*accuracy*). A ferramenta utilizada neste caso foi a biblioteca *Scikit-learn* [17] e uma de suas ferramentas de árvores de decisão (*Decision Trees*).

As árvores de decisão (ADs) são métodos não paramétricos de aprendizado supervisionado utilizados para classificação e regressão [18].

O processamento efetuado neste estudo resultou em uma precisão de 79,5% (Figura 10). Entende-se que há grande influência do *script* da Figura 7 nesta precisão, assim como o fato dos procedimentos levarem em consideração, exclusivamente, as características sintáticas do excerto.

É esperado que a captura de características semânticas de um texto produza melhores resultados quando da utilização de uma ontologia de domínio, se comparada com a captura puramente sintática [19].

Zhang e El-Gohary (2011, apud [19]) revelaram melhora no desempenho da extração de informação, comparando processos exclusivamente sintáticos e processos semânticos baseados em ontologia. Segundo os autores, a precisão aumentou de 75% para 100%.

Figura 10: Aprendizado de máquina e precisão atingida

```
## Decision tree classifier
## O cálculo da acurácia demora um pouco, por volta de 5 min.
## São pouco mais de 38 mil frases de treino e pouco mais de 12 mil frases de teste

from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_extraction import DictVectorizer
from sklearn.pipeline import Pipeline

clf = Pipeline([
    ('vectorizer', DictVectorizer(sparse=False)),
    ('classifier', DecisionTreeClassifier(criterion='entropy'))
])

clf.fit(X[:10000], y[:10000])   # Use only the first 10K samples if you're running it multiple times. It takes a fair bit :)

print ('Treino terminado')

X_test, y_test = conv_em_cj_dados(frases_teste)

print ("Accuracy:", clf.score(X_test, y_test))
```

```
Treino terminado
Accuracy: 0.7947405209201942
```

Fonte: os autores.

A associação dos *scripts* descritos anteriormente resulta na função *etiquetar* representada na Figura 11. Esta função é o etiquetador automático e ao utilizá-la sobre as sentenças contidas no excerto escolhido, obtém-se o resultado da Figura 12, onde cada *token* de cada sentença, ou frase, em processamento, recebe sua etiqueta de classe gramatical.

Figura 11: Função *etiquetar*

```
def etiquetar(frase):
    etiquetas = clf.predict([features(frase, ind) for ind in range(len(frase))])
    return list(zip(frase, etiquetas))
```

Fonte: os autores.

As etiquetas são fornecidas pelo corpus referência e são associadas ao excerto através do etiquetador treinado no aprendizado de máquina. A descrição de cada etiqueta pode ser encontrada no manual do Mac-Morpho [20].

Figura 12: Sentenças etiquetadas

```
frases_etiquetadas = []
cont = 0
for frase in sentencas:
    fe = etiquetar(word_tokenize(sentencas[cont]))
    frases_etiquetadas.append(fe)
    cont = cont+1
```

```
cont = 1
for frase in frases_etiquetadas:
    print ('Frase', cont,':')
    print (frase)
    print ()
    cont = cont + 1
```

```
Frase 1 :
[('4.5.4', 'NUM'), ('Portas', 'NPROP'), ('4.5.4.1', 'NUM'), ('As', 'NPROP'), ('portas', 'N'), ('das', 'NPROP'), ('rotas', 'N'),
('de', 'PREP'), ('saída', 'N'), ('e', 'KC'), ('aquelas', 'ADJ'), ('das', 'NPROP'), ('salas', 'N'), ('com', 'PREP'), ('capacidad
e', 'N'), ('acima', 'PREP|+'), ('de', 'PREP'), ('50', 'NUM'), ('pessoas', 'N'), ('e', 'KC'), ('em', 'PREP'), ('comunicação',
'N'), ('com', 'PREP'), ('os', 'ART'), ('acessos', 'N'), ('e', 'KC'), ('descargas', 'ADJ'), ('devem', 'V'), ('abrir', 'V'), ('n
o', 'PREP|+'), ('sentido', 'N'), ('do', 'NPROP'), ('trânsito', 'N'), ('de', 'PREP'), ('saída', 'N'), ('(', '('), ('ver', 'V'),
('Figura', 'NPROP'), ('2', 'N|AP'), (')', ')'), ('.', 'NUM')]

Frase 2 :
[('4.5.4.2', 'NUM'), ('A', 'NPROP'), ('largura', 'N'), (',', ','), ('vão', 'N'), ('livre', 'N'), ('ou', 'KC'), ('“', 'NUM'),
('luz', 'N'), ('”', 'NUM'), ('das', 'NPROP'), ('portas', 'N'), (',', ','), ('comuns', 'ADJ'), ('ou', 'KC'), ('corta-fogo',
'N'), (',', ','), ('utilizadas', 'PCP'), ('nas', 'N'), ('rotas', 'N'), ('de', 'PREP'), ('saída', 'N'), (',', ','), ('deve', 'VA
UX|+'), ('ser', 'VAUX'), ('dimensionada', 'PCP'), ('como', 'PREP'), ('estabelecido', 'PCP'), ('em', 'PREP'), ('4.4', 'N|AP'),
(',', ','), ('admitindo-se', 'V'), ('uma', 'ART'), ('redução', 'N'), ('no', 'PREP|+'), ('vão', 'VAUX'), ('de', 'PREP'), ('luz',
'N'), (',', ','), ('isto', 'PROSUB'), ('é', 'N'), (',', ','), ('no', 'KC'), ('vão', 'VAUX'), ('livre', 'N'), (',', ','), ('da
s', 'NPROP'), ('portas', 'N'), ('em', 'PREP'), ('até', 'PREP'), ('75', 'NUM'), ('mm', 'V'), ('de', 'PREP'), ('cada', 'PROADJ'),
('lado', 'PCP'), ('(', '('), ('golas', 'N'), (')', ')'), (',', ','), ('para', 'PREP'), ('o', 'ART'), ('contramarco', 'N'),
(',', ','), ('marco', 'N'), ('e', 'KC'), ('alizares', 'N'), ('.', 'NUM')]

Frase 3 :
[('As', 'ART'), ('portas', 'N'), ('devem', 'V'), ('ter', 'V'), ('as', 'ART'), ('seguintes', 'N'), ('dimensões', 'N'), ('mínima
s', 'N'), ('de', 'PREP'), ('luz', 'N'), (':', ':'), ('a', 'ART'), ('(', '('), ('80', 'NUM'), ('cm', 'V'), (',', ','), ('valend
o', 'V'), ('por', 'PREP'), ('uma', 'ART'), ('unidade', 'N'), ('de', 'PREP'), ('passagem', 'N'), (';', ';'), ('b', 'N'), (')',
')'), ('1,00', 'NUM'), ('m', 'V'), (',', ','), ('valendo', 'V'), ('por', 'PREP'), ('duas', 'NUM'), ('unidades', 'N'), ('de', 'P
REP'), ('passagem', 'N'), (';', ';'), ('c', 'N'), (')', ')'), ('1,50', 'NUM'), ('m', 'V'), (',', ','), ('em', 'PREP'), ('duas',
'NUM'), ('folhas', 'N'), (',', ','), ('valendo', 'V'), ('por', 'PREP'), ('três', 'NUM'), ('unidades', 'N'), ('de', 'PREP'), ('p
assagem', 'N'), ('.', 'NUM')]

Frase 4 :
[('Nota', 'N'), (':', ':'), ('Acima', 'NPROP'), ('de', 'PREP'), ('2,20', 'NUM'), ('m', 'V'), (',', ','), ('exige-se', 'V'), ('c
oluna', 'V'), ('central', 'ADJ'), ('.', 'NUM')]
```

Fonte: os autores.

ANÁLISES PRELIMINARES

Os resultados obtidos neste trabalho mostram que ainda há um extenso caminho a percorrer para compreensão de todos os fatores que influenciam na aplicação das quatro técnicas de PLN aqui relacionadas.

Em particular, é necessário estudar mais detidamente a atribuição das características (*features*) aos *tokens*. Especula-se, por exemplo, que a etiquetagem incorreta do *token* ‘Porta’, possa advir da letra maiúscula com a qual se inicia, definido na instrução de número 03 da função de atribuição.

Ademais, também se faz necessário maior observação sobre o corpus Mac-Morpho. Algumas inconsistências, como palavras ausentes e/ou repetidas, foram encontradas e prejudicam o processo de aprendizado de máquina [16].

Percebe-se também a necessidade de explorar de forma mais completa o processo de aprendizado de máquina, detendo-se sobre configurações possíveis e talvez outras ferramentas de classificação que não sejam do tipo árvores de decisão.

Por fim, verificou-se que a aplicação das quatro técnicas de PLN é possível, mas que os resultados precisam ser cuidadosamente avaliados e certificados para que possam alimentar processos à jusante, como a extração de regras.

CONSIDERAÇÕES FINAIS

O PLN tem potencial para contribuir para a eficiência de sistemas VARE tornando sua manutenção menos dispendiosa e permitindo adaptação mais fácil à evolução das normas e regulamentações edilícias. Ao mesmo tempo, se encaixa bem na raiz de sistemas computacionais para auxiliar os profissionais projetistas a cumprirem quesitos de qualidade e adequação do projeto com maior segurança. Além disso, poderiam estar na base de sistemas inteligentes de revisão das regras em si, o que auxiliaria na melhoria da compreensão automatizada dessas regras, fechando um sistema circular evolutivo.

O PLN é uma realidade e muitas de suas técnicas são eficazes e acessíveis, mediante algum esforço. Estas técnicas estão disponíveis através de tecnologias *Open Source* como a linguagem *Python* e a Biblioteca *NLTK*.

Este trabalho prosseguirá objetivando uma melhor compreensão sobre a implementação completa das técnicas de PLN aqui apresentadas, para subsidiar processos posteriores como a extração de regras. Para tanto, identificou-se ser necessário ampliar a exploração do tema correlato, a aprendizagem de máquina, apoiando as técnicas de segmentação de sentenças e de análise morfológica, assim como a aplicação de ontologia de domínio para ampliar a eficácia dessas ferramentas.

REFERÊNCIAS

[1] EASTMAN, C. et al. **Manual de BIM:** Um guia de modelagem da informação da construção para arquitetos, engenheiros, gerentes, construtores e incorporadores. Porto Alegre. Bookman, 2014.

[2] LADEIRA, A. P. **Processamento de linguagem natural:** caracterização da produção científica dos pesquisadores brasileiros. Tese (Doutorado em Ciência da Informação). Belo Horizonte, 2010.

[3] KHEMLANI, L. **Automating Code Compliance in AEC.** AECbytes Feature: Automating Code Compliance in AEC. 2015. Disponível em: http://www.aecbytes.com/feature/2015/AutomatingCodeCompliance.html. Acesso em: 11 ago. 2019.

[4] BIRD, S.; KLEIN, E.; LOPER, E. **Natural language processing with Python**. Cambridge. O'Reilly, 2009.

[5] ZHANG, J.; EL-GOHARY, N. M. Integrating semantic NLP and logic reasoning into a unified system for fully-automated code checking. **Automation in Construction,** v. 73, p. 45–57, 2017.

[6] NAWARI, N. O. A Generalized Adaptive Framework (GAF) for Automating Code Compliance Checking. **Buildings,** v. 9, n. 4, p. 86, 2019.

[7] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO 10303-11:2004.** Disponível em: https://www.iso.org/standard/38047.html. Acesso em: 6 mar. 2021.

[8] CHERPAS, C. Natural language processing, pragmatics, and verbal behavior. **The Analysis of verbal behavior,** v. 10, p. 135–147, 1992.

[9] POLÍCIA MILITAR DO ESTADO DE SÃO PAULO. **Instrução Técnica nº 11/2018:** Saídas de emergência. São Paulo. Corpo de Bombeiros, 2018.

[10] SÃO PAULO (Cidade). **Decreto nº 57.776, de 7 de julho de 2017**, São Paulo, 2017.

[11] SÃO PAULO (Cidade). **Lei nº 16.642, de 9 de maio de 2017**, São Paulo, 2017.

[12] ALMEIDA, W. R. d. **PORTSERVICE-BR:** Uma plataforma para processamento de linguagem natural para língua portuguesa. Dissertação (Mestrado em Informática). Vitória, 2017.

[13] IVANOV, G.-B. **Complete guide for training your own POS tagger with NLTK & Scikit-Learn.** 2016. Disponível em: https://nlpforhackers.io/training-pos-tagger/. Acesso em: 16 jun. 2020.

[14] ALUÍSIO, S. et al. An Account of the Challenge of Tagging a Reference Corpus for Brazilian Portuguese. *In*: PARDO, T. A. S.; RINO, L. H. M.; NUNES, M. d. G. V. (Org.). **GistSumm:** A summarization tool based on a new extractive method, Heidelberger: Springer, 2003. p. 210-218 (Lecture Notes in Computer Science. p. 110–117.

[15] FONSECA, E. R.; ROSA, J. L. G.; ALUÍSIO, S. M. Evaluating word embeddings and a revised corpus for part-of-speech tagging in Portuguese. **Journal of the Brazilian Computer Society,** v. 21, n. 1, p. 110, 2015.

[16] FONSECA, E. R.; ROSA, J. L. G. Mac-Morpho revisited: Towards robust part-of-speech tagging. **Proceedings of the 9th Brazilian Symposium in Information and Human Language Technology.** p. 98–107. Porto Alegre, 2013.

[17] PEDREGOSA, F. et al. Scikit-learn: Machine Learning in Python. **Journal of Machine Learning Research,** v. 12, n. 85, p. 2825–2830, 2011.

[18] SCIKIT-LEARN PROJECT. **Scikit learn:** Decision Trees. c2020. Disponível em: https://scikit-learn.org/stable/modules/tree.html. Acesso em: 6 mar. 2021.

[19] ZHANG, J.; EL-GOHARY, N. M. Semantic NLP-Based Information Extraction from Construction Regulatory Documents for Automated Compliance Checking. **Journal of Computing in Civil Engineering,** v. 30, n. 2, p. 4015014, 2016.

[20] INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO ICMC - USP. **Manual Mac-Morpho v10.** 2003. Disponível em: http://nilc.icmc.usp.br/macmorpho/macmorpho-manual.pdf. Acesso em: 17 jun. 2020.