ReqSCity: uma ferramenta de análise de requisitos de aplicações para cidades inteligentes

Bruno Carvalho da Silva[1], Rodolfo Alves[1], Vinícius Santos[1], Rodrigo Siqueira[1], Jone Correia[1], Luis Rivero[1], Luciano Coutinho[1], Ariel Soares Teles[2] e Davi Viana[1]

[1]Universidade Federal do Maranhão (UFMA)
São Luís – MA – Brasil

[2]Instituto Federal do Maranhão (IFMA)
Araioses – MA – Brasil

{bruno.carvalho1,santos.vinicius}@discente.ufma.br,davi.viana@ufma.br

Abstract. *Errors on requirements elicitation, such as ambiguity and incompleteness, in the project documents can harm the rest of the development process, so an analysis of these documents is important for the correct progress through the steps of an application's design. Manually detecting these requirements is costly and prone to failure, even more, when the system is highly complex. Therefore, the automatic detection of these errors presents itself as a promising approach. This work aims to automatically help the requirements analysis process, focusing on detecting ambiguous and incomplete requirements through the use of Natural Language Processing (NLP). We applied Tokenization and post-tagging techniques from the Python NLTK library. Additionally, the generated tool can assess whether the requirement is in the context of smart cities and suggest the incompleteness of intelligent city requirements, using the M3-Ontology ontology as a basis.*

Resumo. *Erros na fase de elicitação de requisitos, como ambiguidade e incompletude nos documentos gerados podem prejudicar o restante do processo de desenvolvimento, por isso, uma análise desses documentos é importante para o avanço adequado do processo de desenvolvimento. A análise manual destes requisitos é custosa, e tendenciosa a falhas, ainda mais quando o sistema possui uma grande complexidade. A detecção automática desses erros se apresenta como uma abordagem promissora. O objetivo deste trabalho é apresentar a ferramenta ReqSCity que auxilia o processo de análise de requisitos de maneira automática, focando na análise de requisitos ambíguos e incompletos através do uso de Processamento de Linguagem Natural (PLN). Utilizaram-se técnicas de tokenização e pos-tagging a partir da biblioteca NLTK do Python. Adicionalmente, a ferramenta gerada é capaz de avaliar se o requisito está no contexto de cidades inteligentes e fornece sugestões para melhorar a completude de requisitos, usando como base a ontologia M3-Ontology.*

1. Introdução

De acordo com relatório da ONU [Nations 2019], em 2050 a população mundial deverá chegar a 9,7 bilhões. Uma vez que os centros urbanos se tornam cada vez mais cheios, crescem junto os problemas, tais quais, aumento de trânsito, crescimento descontrolado

e meio ambiente cada vez mais afetado, com a poluição de ar, por exemplo. Todas essas problemáticas podem ser atenuadas com o uso de soluções para cidades inteligentes (do inglês, *smart cities*).

Para tornar essas soluções possíveis, o uso de sensores e atuadores são indispensáveis. Além disso, soluções para o contexto de cidades inteligentes fazem uso massivo de tecnologias de informação e comunicação (TIC). O conceito de Internet das Coisas (do inglês, *Internet of Things* - IoT) está relacionado com o contexto de cidades inteligentes [Atzori et al. 2010], pois permite compor sistemas a partir de objetos endereçáveis de maneira única (coisas) equipado com a identificação, sensação ou atuação de comportamentos e capacidades de processamento que podem se comunicar e cooperar para alcançar um objetivo.

Soluções para o contexto de cidades inteligentes precisam ser identificadas e especificadas corretamente para garantir a melhoria na qualidade de vida para as pessoas. Para isso, os requisitos dessas soluções precisam ser especificados adequadamente. A Engenharia de Requisitos tem por objetivo analisar, documentar e verificar os requisitos de software. Por requisitos entende-se como uma descrição daquilo que o sistema deve realizar e as restrições que podem existir para a sua atuação [Sommerville 2018].

Conseguir identificar uma não conformidade logo na etapa de elicitação de requisitos é menos custoso que identificar na etapa de testes, além de ser mais fácil de corrigir erros [Umber and Bajwa 2011]. Erros como requisitos ambíguos, incompletos, ou fora do contexto tendem a prejudicar o desenvolvimento, gerando falhas no produto final e não correspondendo ao produto solicitado pelo cliente.

Em projetos de software, os documentos de requisitos são escritos em linguagem natural. Por isso, tecnologias de processamento de linguagem natural (PLN) vem sendo cada vez mais utilizadas para análise de requisitos [Huertas and Juárez-Ramírez 2012].

A aplicação de PLN na área de engenharia de requisitos não é um tema novo [Zhao et al. 2021]. Todavia, a aplicação de PLN para requisitos no contexto de aplicações para cidades inteligentes é um problema em aberto. Desta forma, são necessárias pesquisas e ferramentas que visam apoiar o desenvolvimento de aplicações para contextos recentes, como cidades inteligentes e IoT. Ressalta-se que ainda não há um conjunto de conceitos bem definidos na engenharia de requisitos para aplicações de contextos contemporâneos [da Silva et al. 2021]. A aplicação de PLN para domínios específicos é um problema em aberto, uma vez que cada domínio possui seu jargão específico, suas regras de negócio e práticas.

Neste sentido, este artigo apresenta a ReqSCity, uma ferramenta para análise de requisitos de aplicações para o contexto de cidades inteligentes. A ferramenta foi desenvolvida especificamente para analisar os requisitos considerando três aspectos: ambiguidade, incompletude e contextualização com cidades inteligentes.

A linguagem natural que o presente trabalho se propõe a analisar é a língua inglesa por uma série de motivos dentre os quais cabe citar que inglês atualmente é a linguagem utilizada mais frequentemente na área da computação, o que permitiria uma internacionalização da solução.

O restante do artigo está organizado da seguinte forma: a Seção 2 apresenta uma

descrição de PLN e de ferramentas de análise de requisitos. A Seção 3 descreve as tecnologias utilizadas para a construção da ferramenta. A Seção 4 mostra a arquitetura geral da aplicação e do funcionamento das classes da ferramenta. A Seção 5 apresenta a interface da ferramenta e um exemplo de uso, como prova de conceito. Por fim, a Seção 6 descreve as considerações finais e trabalhos futuros relacionados a esta ferramenta.

2. PLN e Ferramentas Relacionadas

O objetivo do PLN é fornecer aos computadores a capacidade de entender textos. Entender um texto significa reconhecer o contexto, fazer análise sintática, semântica, léxica e morfológica [Nazir et al. 2017].

Como as palavras são as peças vitais dos textos em linguagem natural, a análise morfológica se concentra na análise a nível de palavra, podendo vê-la como uma sequencia de caracteres ou como um objeto mais abstrato. A partir da análise morfológica pode-se depreender a análise sintática que irá se ater a estrutura da frase com o intuito de observar o adequadamento à gramática formal [DAMERAU 2010].

Há um conjunto de técnicas de PLN que podem ser aplicadas em ferramentas de apoio, independentemente do seu uso, são elas [Dale 2010]: Segmentação de Sentenças; *Tokenização*; *Part of Speech Tagging (Pos - Tagger)*; e Reconhecedor de Entidades (*Chunking*).

2.1. Ferramentas Relacionadas

A literatura apresenta um conjunto de ferramentas que visam apoiar a análise de requisitos. Todavia, ao comparar essas ferramentas com a ferramenta proposta neste trabalho, verificou-se que não há, até o presente momento, uma ferramenta que apoie a análise de requisitos para aplicações no contexto de cidades inteligentes. Desta forma, apresentam-se algumas soluções que serviram de inspiração para a ReqSCity.

[Hussain et al. 2007] traz uma solução baseada em um algoritmo de *machine learning*. Primeiramente, um manual de classificação é feito com base em quatro especialistas que descreveram os requisitos entre ambíguos e não ambíguos que servem como base pro teste e treinamento. O algoritmo usado foi a árvore de decisão C4.5, a partir disso foi feito o classificador de sentença responsável por gerar dinamicamente uma árvore de decisão com base nos dados e realiza a classificação dos textos. Por fim tem-se um classificador de discurso com base no anterior, aqui o classificador é feito com base na contagem do número de sentenças ambíguas do classificador de sentença e também usa como treinamento o corpora inicial.

Oliveira [Vinay S 2009], apresenta uma solução para análise de requisitos denominada RANT. Este trabalho concentra a sua construção em PLN, utilizando a biblioteca SpaCy[1] do Python. Os pontos analisados são: *Completeness* (os requisitos têm de estar na voz ativa), *Clearness* (Um requisito por sentença), *Consistency* (Cada requisito tem que conter um sujeito e um predicado), *Feasibility* (A palavra ”shall”deve ser usada em requisitos no imperativo), *Verifiability* (Não se pode ter requisitos duplicados) e *Traceability* (sem requisitos duplicados). Não se tem qualquer verificação do contexto de Cidades Inteligentes nesta aplicação. O RANT é constituído por um arquivo XML, que contém

[1] https://spacy.io/

as informações necessárias para o funcionamento da aplicação; O JavaScript, que contém as funções que irão ser chamadas na aplicação e, além disso, é responsável pelo back-end e pela conexão com o Python que irá ser responsável pelo PLN. Por fim, um arquivo CSS responsável pela estilização da ferramenta. Por sua vez, [Lu et al. 2008] traz o editor MOR, uma ferramenta para auxiliar os processos dos documentos de requisitos de software. A ferramenta auxilia nos aspectos de consistência, manutenibilidade, rastreabilidade e integridade. Foi feito um estudo de caso para averiguar a eficácia do MOR.

3. ReqSCity: arquitetura e funcionalidades

Para o desenvolvimento da ReqSCity a linguagem utilizada é o Python[2]. A linguagem possui uma gama de bibliotecas que complementam as suas aplicações como, por exemplo, as utilizadas neste trabalho: NLTK e Flask.

A biblioteca de PLN utilizada nesta ferramenta foi a *Natural Language Toolkit* ou NLTK que proporciona uma forma fácil de usar vários corpus e recursos léxicos, além de conter um conjunto de ferramentas de processamento de texto. Dentre os Recursos léxicos disponíveis no NLTK, cabe citar o Wordnet[3] um grande banco de dados lexical de inglês, onde as classes gramaticais são separadas em grupos de sinônimos cognitivos, os chamados synsets. Mesmo se assemelhando a um tesauro, o Wordnet não trata as palavras apenas pela sua estrutura, mais sim, pelo seu sentido; e também rotula relações semânticas entre as palavras, como hiperonímia e meronímia. [Fellbaum 2005].

Para analisar os requisitos em relação ao contexto de cidades inteligentes, utilizou-se uma ontologia de domínio. A ontologia é um modelo de dados utilizado que representa um conjunto de conceitos dentro de um domínio [Santosa et al. 2021]. Nesta ferramenta, foi utilizada a M3-Ontology[4] no qual o seu domínio é o de cidades inteligentes.

Para tornar a solução como uma aplicação web, o framework utilizado foi o Flask[5], escolhido por ser: simples, rápido e voltado para projetos pequenos. É baseada em dois kits de ferramentas, o Werkzeg WSGI e a Jinja2. A ReqSCity está disponivel no repositório GitHub[6].

3.1. Arquitetura da ferramenta

A Figura 1 apresenta os principais componentes da ferramenta ReqSCity. No início do processo, os **requisitos** são recebidos em formato de documento de texto .txt. Esses requisitos são tratados, divididos e colocados individualmente como elementos em uma lista, antes de serem encaminhados para a **Classe Texto**.

Na **Classe Texto**, os requisitos são tokenizados utilizando o tokenizador padrão da biblioteca NLTK. O resultado desta atividade é armazenado na instância criada. Além disso, o resultado é enviado para a função padrão do etiquetador (*Part of Speech Tag*) do NLTK, a fim de se ter os requisitos com as etiquetas morfológicas. Para além disso, essa classe é a responsável por receber a saída das outras classes, concentrando assim todo o produto da aplicação.

[2]https://www.python.org/
[3]https://wordnet.princeton.edu/
[4]https://databus.dbpedia.org/ontologies/sensormeasurement.appspot.com/m3
[5]https://flask.palletsprojects.com/en/2.1.x/
[6]https://github.com/carvalhosilva42/ReqSCity

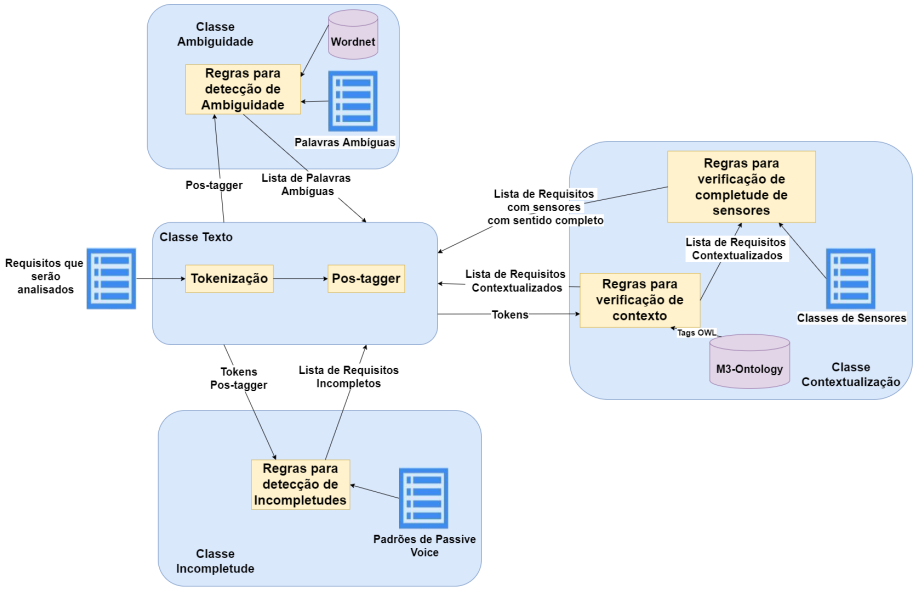

Figura 1. Arquitetura da ReqSCity. Fonte: Autores

3.2. Classe Ambiguidade

Essa classe é definida em duas partes: (1) trata as ambiguidades léxicas, aquelas que centram sua análise nas palavras; e (2) as ambiguidades sintáticas que concentram análise na estrutura dos requisitos.

Nas Ambiguidades léxicas, há os seguintes gatilhos (do inglês, *triggers*):

- `Palavras Ambíguas`: para o funcionamento deste gatilho, tem-se o recebimento de uma lista de palavras ambíguas, palavras essas que podem levar os requisitos à incompreensão ou podem ser vagas, como *big*, *sometime* e *low* [Sandhu and Sikka 2015].
- `Algoritmo Flexible Ambiguity`: O próximo gatilho tem como base [Bäumer and Geierhos 2018]. Uma vez recebido os tokens e as palavras etiquetadas, o processo se inicia com a retirada de *stopwords*, ou seja, palavras que são somente estruturais da língua inglesa e para isso usa-se uma lista de *stopwords* advindas do NLTK. Após essa atividade, sobram as palavras realmente relevantes para o sentido do requisito e, então, se utiliza o Wordnet. Esse uso ocorre da seguinte maneira, se consulta o synset de cada palavra relevante, caso a palavra tenha mais de três elementos em seu synset, ela é potencialmente ambígua visto que ela possui uma alta quantidade de sentidos. Uma vez que a quantidade de tokens potencialmente ambíguos exceder 75% em relação aos tokens totais, esse requisito é considerado ambíguo.

Para as ambiguidades sintáticas, tem-se dois gatilhos:

- `Coordination Ambiguity`: este algoritmo se baseia em [Sandhu and Sikka 2015]. Uma *coordination ambiguity* ocorre quando um modificador, adjetivo, por exemplo, se refere a um substantivo que é sucedido por uma conjunção e não fica claro se a referência está no substantivo mais próximo ou no mais longe. Este algoritmo, primeiramente, recebe os requisitos

Tense	Passive Voice Formula	Passive Voice detection formula POS equivalent
Simple Present	am, is, are + simple participle	VBZ+VBN VBP+VBN
Present continuous	am being, is being, are + past participle	VBZ+VBG+VBN VBP+VBG+VBN
Simple past	was, were + past participle	VBD+VBN VBZ+VBN
Past continous	was being, were being + past participle	VBD+VBG+VBN
Present perfect	has been, have been + past participle	VBZ+VBN+VBN VBP+VBN+VBN
Past perfect	had been + past participle	VBD+VBN+VBN
Future	will be + past participle	MD+VB+VBN
Future perfect	will have been + past participle	MD+VB+VBN+VBN

Tabela 1. Quadro Passive Voice. Fonte: Autores

etiquetados, depois realiza a união de todas as tags de um requisito em uma única *string* e, por fim, pesquisa pelos seguintes padrões:

$$JJNNSCC\, e\, CCNNSCC$$

sendo: JJ = adjetivos, NNS = substantivo no plural e CC = conjunção. Se pelo menos um dos padrões forem encontrados na *string*, esse requisito é considerado como ambíguo;

- `Passive Voice`: uma vez que os requisitos de software focam sua estrutura em linguagem direta, requisitos em voz passiva podem levar a um não entendimento sobre qual o papel do sistema e o que ele deve executar. O algoritmo recebe os requisitos etiquetados, depois realiza a junção de todas as tags do em uma string e busca pelos padrões da terceira coluna da tabela 1. Se pelo menos um padrão for encontrado, o requisito é considerado ambíguo.

3.3. Classe Incompletude

Essa classe é a responsável por verificar se os requisitos apresentam um sentido completo. Uma vez que um requisito não tem um sentido possível por causa de uma estrutura sintática defeituosa, esse requisito será completamente inadequado ao desenvolvimento de aplicações. Há quatro gatilhos possíveis para analisar a incompletude de requisitos:

- `Sentence Fragment`: é o gatilho responsável por verificar se uma sentença termina em verbo [Traffis 2021]. Uma vez que, para um sentido completo, um verbo necessita de um objeto que o complemente, um requisito terminando em verbo pode causar a impressão para quem lê de que algo está faltando.
- `Missing Subject`: é responsável por verificar se o requisito possui um sujeito, algo ou alguém que realiza uma ação.
- `Missing Verb Mistake`: é responsável por verificar se existe verbo no requisito. Uma vez que um requisito indica algo que o software deva fazer ou deva ter, a ausência de verbo indicaria um requisito muito vago, sem uma ação.

- `Dummy Subject`: este gatilho foca em duas palavras utilizadas no sujeito da frase, no caso *"it"* e *"there"*. Ao utilizar essas palavras no sujeito, o sentido se torna vago, passando a ideia de que há algo para se fazer, mas sem saber quem será o responsável pela execução.

3.4. Classe Contextualizada

O objetivo desta classe é avaliar requisitos de software estão adequados ao contexto de cidades inteligentes. Temos dois gatilhos para essa classe:

- `Contextualiza`: é a responsável por avaliar se os requisitos estão de fato no contexto de cidades inteligentes. Para isso, a ontologia M3-Ontology é utilizada. Por meio de um tratamento utilizando python, armazenou-se em uma lista as classes da ontologia. O algoritmo recebe, então, os tokens dos requisitos e verifica se existe no requisito alguma das palavras contidas no arquivo de texto, caso exista, esse requisito é classificado como contextualizado ao cenário de cidades inteligentes;
- `Completude`: dentre o contexto de cidades inteligentes, este trabalho também avalia se os requisitos descrevem adequadamente os sensores. Por adequadamente, pode-se entender se foi descrito o tipo de sensor e o que esse sensor faz. Para tal, a onotologia M3-Ontology também foi utilizada. Neste trabalho, criou-se um arquivo texto contendo as classes de sensores. O algoritmo avalia somente os requisitos que estão contextualizados, portanto, ele recebe a saída do gatilho anterior.

4. Interface da ReqSCity e Prova de Conceito

A Ferramenta de Análise de Requisitos de Aplicações para Cidades Inteligentes, foi construída como uma ferramenta web simples e de fácil uso. O software será disponibilizado gratuitamente. A seguir, apresenta-se a interface da ferramenta e uma prova de conceito, como exemplo de uso da ReqSCity.

4.1. Interface

A interface principal da aplicação é única e comporta todas as funcionalidades da ferramenta. Inicialmente, é apresentada a funcionalidade de submissão (*upload*) de arquivos. Em seguida, o usuário pode selecionar o tipo de análise que deseja ter como resultado. Por fim, as visualizações dos resultados são exibidas na mesma tela. A aplicação também possui uma interface secundária, onde apresenta as instruções de como os gatilhos funcionam para o conhecimento dos usuários.

A Figura 2 apresenta a tela inicial, que apresenta a funcionalidade de submissão do arquivo texto contendo os requisitos em formato .txt, sendo que cada requisito está em uma linha do arquivo. Após a submissão do arquivo, é necessário escolher qual a análise deve ser feita. As opções são: (1) ambiguidade; (2) incompletude; e (3) contextualização. Uma vez acionado o botão "enviar", o retorno do processamento se dará nesta mesma página em um formato de tabela, abaixo dos campos de entrada.

A Figura 3 demonstra a parte da aplicação destinada a dar instruções aos usuários. Nesta tela, há os botões que apresentam estes diagramas, uma vez clicados, aparecerão logo abaixo os diagramas de: entrada, ambiguidade, incompletude e contextualização. A ferramenta está disponível um repositório do GitHub.

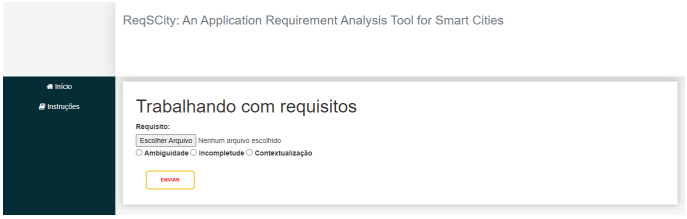

Figura 2. Tela Inicial da aplicação. Fonte: Autores

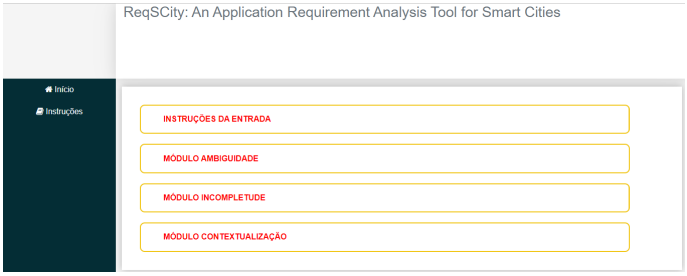

Figura 3. Tela de Instruções da aplicação. Fonte: Autores

4.2. Prova de conceito

Para analisar a adequabilidade da ferramenta proposta, ao seu propósito e exemplificar seu uso, foi realizada uma prova de conceito. Na prova de conceito, foram utilizados os requisitos definidos em projetos da disciplina de Tópicos Avançados em Cidades Inteligentes do programa de pós-graduação em engenharia elétrica (PPGEE) e em ciência da computação (PPGCC) da UFMA. Para a condução da prova de conceito, foram utilizados 55 requisitos [7] de oito projetos da disciplina.

Os requisitos foram avaliados, manualmente, por dois especialistas da área de engenharia de software. Ao receber os requisitos, os especialistas foram instruídos a classificá-los como ambíguos, incompletos ou no contexto de cidades inteligentes. Esse resultado foi registrado [8] para comparação com o resultado da ferramenta.

Em relação a ReqSCity, foi possível detectar 11 requisitos considerados ambíguos e 20 requisitos considerados incompletos. Além disso, a ferramenta detectou que somente 11 dos requisitos apresentados, se lidos de maneira isolada, poderiam ser considerados no contexto de cidades inteligentes.

Ao comparar o resultado da ferramenta com o resultado dos especialistas, verificou-se que o especialista A detectou seis requisitos considerados ambíguos, 19 requisitos incompletos e 18 caracterizados no contexto de cidades inteligentes. Já o especialista B detectou 15 requisitos considerados ambíguos, 23 incompletos e 23 no contexto de cidades inteligentes.

Temos na tabela 2 quatro métricas utilizadas amplamente no contexto de classificação binária, sendo elas: Acurácia, Precisão, Revocação ou *Recall* e F1. Cada métrica foi feita a partir da comparação da predição, resultados da ReqSCity em comparação com os especialistas

[7]Lista de requisitos utilizada na prova de conceito: https://drive.google.com/file/d/1YXoBPnDBFOmo-YN1V7MNlCTua2SDfKWB/view?usp=sharing

[8]https://drive.google.com/file/d/1W7sYEhn9gBDYQztbzR10ZW$_w t2NAhrEr/view?usp = sharing$

[redacted]	Especialista A			Especialista B		
[redacted]	Incompletude	Ambiguidade	Contexto Smart City	Incompletude	Ambiguidade	Contexto Smart City
Acurácia	58%	84%	69%	73%	82%	64%
Precisão	40%	36%	55%	70%	73%	64%
Recall	42%	67%	33%	61%	53%	30%
F1	41%	47%	41%	65%	62%	41%

Tabela 2. Resultados da avaliação da prova de conceito. Fonte: Autores

Cabe ressaltar que não foi gerado uma lista única dos especialistas, pois eles possuem experiências diferentes nas atividades de análise de requisitos e nas atividades de desenvolvimento de aplicações para o contexto de IoT e Cidades Inteligentes. Os dados expostos ainda não servem para validar a ferramenta, mas servem para analisar seu funcionamento. Os dados obtidos nessa prova de conceito serão fundamentais para a melhoria e a evolução da ferramenta no futuro.

5. Conclusão

Este artigo apresentou a ReqSCity, uma ferramenta que visa automatizar o processo de análise de requisitos, por meio de técnicas de PLN. Essa análise considera três aspectos: ambiguidade, incompletude e contextualização dos requisitos para Cidades Inteligentes. A ferramenta foi analisada em uma prova de conceito e obteve alguns resultados promissores, além de resultados que permitem fazer ajustes nos gatilhos utilizados.

Como trabalhos futuros, pretende-se evoluir a ferramenta com base nos resultados já obtidos e executar novos estudos visando adequação para uso comercial. Além disso, pretende-se evoluir a forma de submissão de arquivos para outros formatos para além do .txt, podendo contar ainda com uma normalização interna, de modo a deixar o usuário com menos atribuições. Por fim, existe a necessidade de realizar uma avaliação dos resultados do algoritmo, de modo a atestar sua acurácia e garantir resultados satisfatórios.

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Codigo de Financiamento 001 e PROCAD-Amazônia. Este trabalho também contou com o apoio do INCT de Internet do Futuro para Cidades Inteligentes (CNPq 465446/2014-0, CAPES 88887.136422/2017-00, FAPESP 14/50937-1 e 15/24485-9), CNPq (311608/2017-5, 420907/2016-5 e 312324/2015-4). O último autor agradece a FAPEMA (UNIVERSAL-00745/19) e Bolsa de Produtividade FAPEMA (BEPP-01608/21). Por fim, o trabalho teve o apoio de bolsas de Iniciação Científica e Iniciação Tecnológica da FAPEMA, UFMA e CNPq.

Referências

Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer networks*, 54(15):2787–2805.

Bäumer, F. and Geierhos, M. (2018). Flexible ambiguity resolution and incompleteness detection in requirements descriptions via an indicator-based configuration of text analysis pipelines.

da Silva, D. V., de Souza, B. P., Gonçalves, T. G., and Travassos, G. H. (2021). A requirements engineering technology for the iot software systems. *Journal of Software Engineering Research and Development*.

Dale, R. (2010). *Handbook of natural language processing*. Boca Raton: Chapman & Hall, second edition.

DAMERAU, N. I. F. J. (2010). *HANDBOOK OF NATURAL LANGUAGE PROCESSING SECOND EDITION*. CRC Press Taylor&Francis Group.

Fellbaum, C. (2005). Wordnet and wordnets. *Keith et al. (eds.), Encyclopedia of Language and Linguistics*, Second Edition:665–670.

Huertas, C. and Juárez-Ramírez, R. (2012). Nlare, a natural language processing tool for automatic requirements evaluation. In *Proceedings of the CUBE International Information Technology Conference*, pages 371–378.

Hussain, I., Ormandjieva, O., and Kosseim, L. (2007). Automatic quality assessment of srs text by means of a decision-tree-based text classifier. In *Seventh International Conference on Quality Software (QSIC 2007)*, pages 209–218.

Lu, C.-W., Chang, C.-H., Chu, W. C., Cheng, Y.-W., and Chang, H.-C. (2008). A requirement tool to support model-based requirement engineering. In *2008 32nd Annual IEEE International Computer Software and Applications Conference*, pages 712–717. IEEE.

Nations, U. (2019). População mundial deve chegar a 9,7 bilhões de pessoas em 2050, diz relatório da onu.

Nazir, F., Butt, W. H., Anwar, M. W., and Khan Khattak, M. A. (2017). The applications of natural language processing (nlp) for software requirement engineering-a systematic literature review. In *International conference on information science and applications*, pages 485–493. Springer.

Sandhu, G. and Sikka, S. (2015). State-of-art practices to detect inconsistencies and ambiguities from software requirements. In *International Conference on Computing, Communication Automation*, pages 812–817.

Santosa, N. C., Miyazaki, J., and Han, H. (2021). Automating computer science ontology extension with classification techniques. *IEEE Access*, 9:161815–161833.

Sommerville, I. (2018). *Engenharia de Software*. Pearson Education.

Traffis, C. (2021). What is a sentence fragment?

Umber, A. and Bajwa, I. S. (2011). Minimizing ambiguity in natural language software requirements specification. In *2011 Sixth International Conference on Digital Information Management*, pages 102–107. IEEE.

Vinay S, Shridhar Aithal, P. D. (2009). An nlp based requirements analysis tool. *IEEE International Advance Computing Conference*, pages 2355–2360.

Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K. J., Ajagbe, M. A., Chioasca, E.-V., and Batista-Navarro, R. T. (2021). Natural language processing for requirements engineering: A systematic mapping study. *ACM Computing Surveys (CSUR)*, 54(3):1–41.