UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS I
CENTRO DE CIÊNCIA E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ADRIELLY DE MEDEIROS SANTOS

USO DO TEXTATTACK PARA VALIDAÇÃO DE MODELOS DE CLASSIFICAÇÃO NO CONTEXTO DE NOTAS FISCAIS ELETRÔNICAS

CAMPINA GRANDE
2021

ADRIELLY DE MEDEIROS SANTOS

USO DO TEXTATTACK PARA VALIDAÇÃO DE MODELOS DE CLASSIFICAÇÃO NO CONTEXTO DE NOTAS FISCAIS ELETRÔNICAS

Trabalho de Conclusão de Curso (Artigo) apresentado ao Departamento do Curso de Ciências da Computação da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de bacharel em Computação.

Área de concentração: Inteligência Artificial.

Orientador: Profa. Dra. Kézia de Vasconcelos Oliveira Dantas.

CAMPINA GRANDE
2021

S237u Santos, Adrielly de Medeiros.

Uso do *TextAttack* para validação de modelos de classificação no contexto de notas fiscais eletrônicas [manuscrito] / Adrielly de Medeiros Santos. - 2021.

27 p. : il. colorido.

Digitado.

Trabalho de Conclusão de Curso (Graduação em Computação) - Universidade Estadual da Paraíba, Centro de Ciências e Tecnologia , 2021.

"Orientação : Profa. Dra. Kézia de Vasconcelos Oliveira Dantas , Coordenação do Curso de Computação - CCT."

1. Processamento de linguagem natural. 2. Inteligência artificial. 3. Nota fiscal eletrônica. I. Título

21. ed. CDD 006.3

Elaborada por Lucas H. A. da Silva - CRB - 15/898 BC/UEPB

ADRIELLY DE MEDEIROS SANTOS

Uso do TextAttack para Validação de Modelos de Classificação noContexto de Notas Fiscais Eletrônicas

Trabalho de Conclusão de Curso de Graduação em Ciência da Computação da Universidade Estadual da Paraíba, como requisito à obtenção do título de Bacharel em Ciência daComputação.

Aprovada em 05 de Outubro de 2021.

Kézia de Vasconcelos Oliveira Dantas

Profa. Dra. Kézia de Vasconcelos Oliveira Dantas (DC - UEPB)
Orientador(a)

Sabrina de F. Souto.

Profa. Dra. Sabrina de Figueiredo Souto (DC - UEPB)
Examinador(a)

Paulo Barbosa

Prof. Dr. Paulo Eduardo e Silva Barbosa (DC - UEPB)
Examinador(a)

Dedico este trabalho aos meus pais e meu irmão, que sempre me incentivaram a continuar adquirindo cada vez mais conhecimento.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CNN	Convolutional Neural Network
EAN	European Article Number
LSTM	Long Short-Term Memory
NF-e	Nota Fiscal Eletrônica
NCM	Nomenclatura Comum do Mercosul
NUTES	Núcleo de Tecnologias Estratégicas e Saúde
PLN	Processamento de Linguagem Natural
RNN	Recurrent Neural Network
SEFAZ	Secretaria de Estado da Fazenda
UEPB	Universidade Estadual da Paraíba
WSNCS	WordSwapNeighboringCharacterSwap
WSO	WordSwapOrtography
WSQ	WordSwapQWERTY
WSRCD	WordSwapRandomCharacterDeletion

SUMÁRIO

USO DO TEXTATTACK PARA VALIDAÇÃO DE MODELOS DE CLASSIFICAÇÃO NO CONTEXTO DE NOTAS FISCAIS ELETRÔNICAS

USE OF TEXTATTACK FOR VALIDATION OF CLASSIFICATION MODELS IN THE CONTEXT OF ELECTRONIC INVOICES

Adrielly Santos*

RESUMO

Com o surgimento das NF-e o número de fraudes fiscais diminuiu pois elas se tornaram obrigatórias em todo o território brasileiro, mas a consequência disso é que todo mês são geradas um número exorbitante de NF-e e se torna inviável fazer a fiscalização manual de todas elas, isso porque as descrições dos produtos são em texto livre não padronizado. Por tanto, existe uma variação muito grande de descrições para o mesmo produto somadas aos erros ortográficos e as abreviações. Nesse contexto, um classificador de produtos foi desenvolvido pelo NUTES/UEPB em parceria com a Secretaria de Estado da Fazenda da Paraíba (SEFAZ-PB), o objetivo é usar inteligência artificial e técnicas de PLN para a partir da descrição do produto encontrada na NF-e identificar a que categoria ele pertence e relacioná-lo ao seu NCM/EAN. Este artigo vai mostrar a aplicação do TextAttack nesse classificador de produtos com o objetivo de testar o modelo e entender como ele está se comportando em relação as entradas que recebe.

Palavras-chave: TextAttack. PLN. Classificador de produtos.

ABSTRACT

With the emergence of NF-e, the number of tax frauds decreased as they became mandatory throughout the Brazilian territory, but the consequence of this is that every month an exorbitant number of NF-e are generated and it becomes impossible to carry out manual inspection of all of them, this is because the product descriptions are in non-standard free text, so there is a very large variation of descriptions for the same product added to spelling errors and abbreviations. In this context, a product classifier was developed by NUTES/UEPB in partnership with the State Secretariat of Finance of Paraíba (SEFAZ-PB), the objective is to use artificial intelligence and PLN techniques to identify from the description of the product found in the NF-e which category it belongs to and relate it to it's own NCM/EAN. This article will show the application of TextAttack in this product classifier in order to test the model and understand how it is behaving in relation to the inputs it receives.

Keywords: TextAttack. NLP. Product classifier.

* Aluna de Computação, UEPB, Campus I. E-mail: adrielly.santos@aluno.uepb.edu.br.

1 INTRODUÇÃO

As NF-e são de grande utilidade e são responsáveis, entre outras coisas, pela diminuição das fraudes em documentos fiscais. Essas fraudes ocorrem quando o contribuinte omite, altera ou manipula de alguma forma os dados desses documentos com o objetivo de sonegar impostos ou legitimar despesas que não ocorreram, isso é crime e está previsto na Lei 8.137/90. No entanto, ao se tornar obrigatória em todas as transações de venda, um volume exorbitante de NF-e são geradas de maneira que a fiscalização manual dessas notas se torna inviável.

Foi nesse contexto que a parceria entre a Secretaria de Estado da Fazenda da Paraíba (SEFAZ-PB) e o Núcleo de Tecnologias Estratégicas e Saúde (NUTES/UEPB) foi firmada, tendo como objetivo o desenvolvimento de uma ferramenta que, a partir da descrição presente nas NF-e, pudesse apontar a que categoria o produto em questão pertence. Dessa maneira, é possível também a identificação de fraudes causadas pela divergência entre a descrição do produto declarado pelo contribuinte e sua respectiva alíquota e a categoria presente na NF-e. A grande dificuldade que existe para a fiscalização atualmente é que não há uma padronização para as descrições presentes nas NF-e que são exclusivamente em formato textual, por tanto o mesmo produto pode ser declarado de *n* maneiras diferentes, sem contar todos os erros ortográficos e abreviações incomuns que são encontrados com frequência.

Com o objetivo de auxiliar na resolução deste problema um classificador de produtos foi desenvolvido utilizando inteligência artificial e técnicas de processamento de linguagem natural (PLN) [8]. A solução é um modelo de rede neural recorrente, o Long Short-Term Memory [7], em conjunto com a técnica de *word embeddings* [4, 11]. Dentre as técnicas de PLN o *word embeddings* se destaca pois representa as palavras como vetores matemáticos que possuem um grande poder de generalização. Atualmente o classificador está em fase de aperfeiçoamento, mas já inclui as categorias de bebidas quentes, cerveja, aguardente, refrigerante, água e energéticos.

A acurácia atual do classificador é de 97% (para mais detalhes ver Seção 4.1), porém além da acurácia de um modelo, a confiança também é um aspecto bem importante para que se possa medir a assertividade. Para tal, se faz necessário realizar testes no modelo com diferentes entradas para que possamos entender o seu comportamento, identificando erros e acertos e, além disso, buscar explicar o porquê dos erros. Quando se pensa no classificador de produtos e a sua importância não é diferente, é preciso realizar testes para que a partir deles possamos compreender as mudanças a serem realizadas para melhorar cada vez mais o modelo. Esta pesquisa restringiu-se à categoria de bebidas alcoólicas e, nesse contexto, alguns exemplos que podemos destacar seriam para as descrições "BEBIDA MISTA ISIS GUARANA ACAI ML" e "BECO LACTA CHOCOLATE ITALAC ML", embora a confiança para a categoria de **outros** seja de 99% e para **bebidas quentes** de 91%, o classificador ainda errou na categorização dessas descrições classificando-as como bebidas quentes quando deveriam estar na classe de outros.

Das ferramentas que existem para realizar esses testes em redes neurais destacamos duas que embora recentes tem apresentado ótimos resultados, o CheckList[1] e o TextAttack[2]. Essas duas ferramentas são bem similares quanto ao

[1] Ferramenta disponível em: <https://github.com/marcotcr/checklist>
[2] Ferramenta disponível em: <https://github.com/QData/TextAttack>

seu objetivo, ambas lidam especificamente com ataques/testes em modelos que trabalham com PLN. O CheckList já foi inclusive testado no modelo de análise de sentimento da Microsoft pela própria equipe interna responsável e eles afirmaram que o uso dessa ferramenta trouxe muitos benefícios, por exemplo possibilitando a aplicação de testes que antes não haviam sequer sido considerados [10]. No entanto, para o classificador optou-se pelo uso do TextAttack porque ele é bem mais abrangente em relação aos tipos de ataques possíveis e a liberdade em realizá-los. O CheckList testa capacidades diferentes, onde é possível fazer várias combinações com os tipos de testes disponíveis, mas com apenas três tipos de testes o TextAttack supera esse aspecto com as várias possíveis transformações que ele possui. Além disso, o TextAttack ainda nos possibilita a escolha entre ataques prontos da literatura ou a criação de ataques próprios com poucas linhas de código, característica que não encontramos no CheckList.

Diante do exposto, o objetivo deste trabalho é apresentar de maneira prática como o uso do TextAttack pode contribuir na obtenção de melhores resultados quando o objetivo é a criação e treinamento de modelos neurais que lidam com dados textuais. Para alcançar tal objetivo o estudo de caso será realizado sobre o modelo do classificador de produtos, serão discutidos dois cenários distintos onde o primeiro diz respeito aos resultados do classificador sem aplicação do TextAttack e outro onde técnicas do TextAttack são utilizadas. Por fim será realizada a análise dos resultados obtidos.

O restante deste artigo é organizado da seguinte maneira: na Seção 2 encontra-se toda a fundamentação teórica do trabalho contendo a apresentação das técnicas utilizadas para solução; na Seção 3 é apresentada a metodologia utilizada; na Seção 4 temos a apresentação do cenário em que essa pesquisa foi realizada; na Seção 5 são apresentados os resultados deste trabalho; e, por fim, na Seção 6 são apresentados as conclusões e os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção serão descritas todas as tecnologias envolvidas durante o desenvolvimento deste estudo de caso. Primeiramente será apresentado as técnicas de PLN utilizadas e o porquê de elas terem sido escolhidas, depois veremos um pouco sobre o que é uma rede neural LSTM, como ela funciona e quando seu uso é indicado, por último o TextAttack será apresentado de forma mais detalhada explicando como esse framework funciona e quais suas aplicações.

2.1 Processamento de linguagem natural

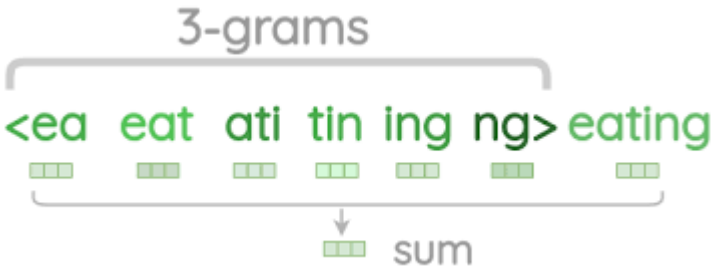
Como os dados trabalhados estão em formato textual se fez necessário trabalhar com técnicas de Processamento de Linguagem Natural (PLN) [8]. Existem diferentes técnicas de PLN que podem ser utilizadas, mas no contexto deste trabalho era preciso uma ferramenta que considerasse não apenas a sintaxe, mas também a semântica das palavras envolvidas, nesse sentido uma técnica que tem se destacado é o *word embeddings* [4].

O *word embeddings* tem por característica representar todas as palavras como um vetor matemático, isso é muito bom para o classificador utilizado no estudo de caso, pois permite o agrupamento de palavras similares, algo muito útil uma vez que precisamos agrupar vários tipos de produtos para realizar sua classificação de acordo com sua respectiva categoria.

Ainda no *word embeddings* existem várias técnicas diferentes que poderiam ser aplicadas, entre elas temos os modelos de redes neurais. Mas, após alguns testes chegou-se à conclusão de que a melhor opção seria utilizar o FastText [11].

A principal característica do FastText é que em uma frase ele não considera uma palavra como a menor unidade possível na representação vetorial, mas que cada palavra é formada por *n-grams* de caracteres, isso possibilita que palavras não presentes no dicionário possam ter sua representação vetorial. Essa característica é muito útil em nosso contexto visto que trabalhamos com descrições textuais de notas fiscais e essas descrições não possuem uma padronização, então abreviações de palavras e erros ortográficos são muito comuns, com o FastText mesmo esses casos extremos podem ser representados contribuindo de maneira positiva para a acurácia final do classificador.

Figura 1 – Representação do funcionamento dos *n-grams*



Fonte: Amit Chaudhary - A Visual Guide to FastText Word Embeddings (2020).[3]

Na Figura 1 vemos como os *n-grams* do FastText funcionam, a quantidade *n* de *grams* pode ser ajustada e no exemplo acima estamos trabalhando com 3-grams. Nesse caso cada palavra é representada como um conjunto composto por todas as possibilidades de subpalavras formada por 3 caracteres. Como comentado anteriormente, isso possibilita que a sequência “ati”, mesmo não sendo uma palavra, possa ter sua representação no vetor matemático.

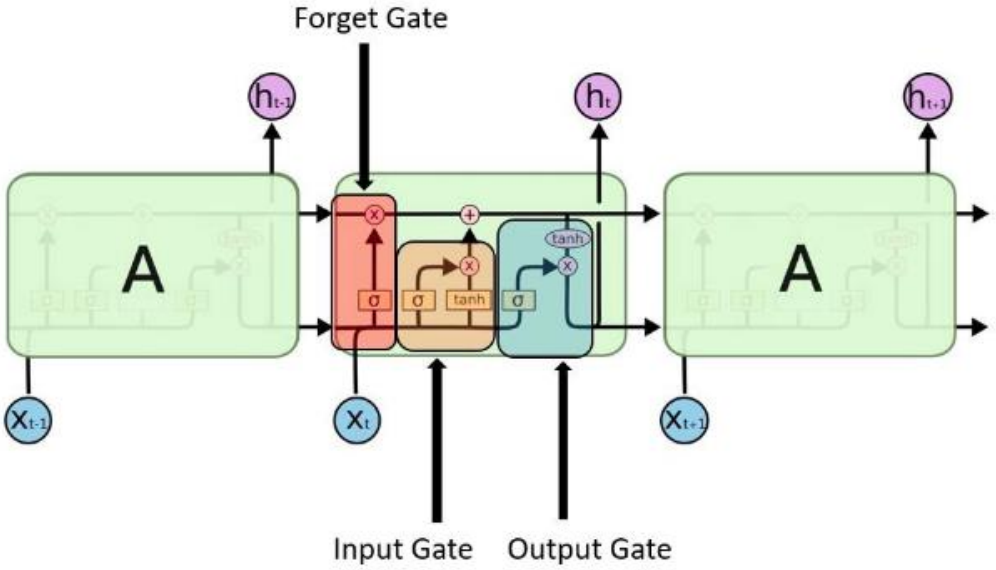
2.2 Rede neural LSTM

Uma rede neural LSTM é um tipo específico de rede neural recorrente ou RNN (Recurrent Neural Network). Essa rede foi desenvolvida com o objetivo de resolver o problema do *vanishing gradiente* [5]. De maneira simplificada, o *vanishing gradiente* é um problema bastante comum em redes neurais profundas onde ocorre a dissipação do gradiente e dificulta o aprendizado das redes [2].

A arquitetura da LSTM ela permite que as informações dos neurônios sejam mantidas por mais tempo, ou seja, a rede pode passar mais tempo aprendendo sobre os dados que ela está trabalhando, por isso ela é o tipo de rede neural recorrente que mais se destaca atualmente, porque os seus resultados costumam ser superiores ao de uma RNN comum [1, 3, 7].

Figura 2 – Arquitetura de uma LSTM

[3] Disponível em: <https://amitness.com/2020/06/fasttext-embeddings/>



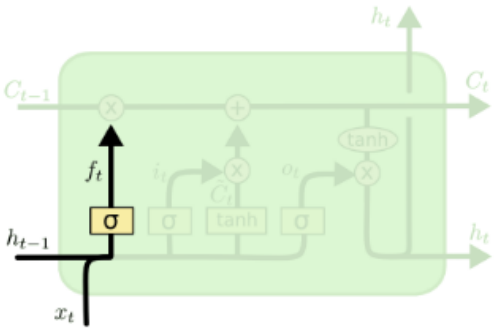
Fonte: Colah's blog - Understanding LSTM Networks (2015 , com adaptações).

A figura acima representa a arquitetura de uma LSTM, é possível notar que ela é formada por três portas que operam em conjunto, todos essas portas trabalham com o objetivo de manter as informações úteis à rede e descartar as informações que não são mais necessárias [3, 7]. O *forget gate* é o portão responsável por “esquecer” as informações que não são mais úteis a rede; o *input gate* garante que na entrada apenas as informações úteis iram permanecer; o *output* é a porta responsável por passar para a saída apenas os dados úteis.

Para facilitar o entendimento cada uma dessas portas destacados será melhor explicado a seguir.

O *forget gate* é responsável por passar para a célula de memória quais as informações que devem ser esquecidas, essa porta recebe duas entradas: x_t, a entrada atual, e h_{t-1}, a saída da célula anterior. Essas entradas são multiplicadas por uma matriz de pesos (W_f) e somadas ao bias (b_f). O resultado dessa operação é passado para uma função de ativação (σ) e o resultado é uma saída binária; se 0 a informação é esquecida, se 1 a informação é mantida. A Figura 3 abaixo destaca essa porta e apresenta a função f_t descrita.

Figura 3 – Arquitetura de uma LSTM: Forget Gate

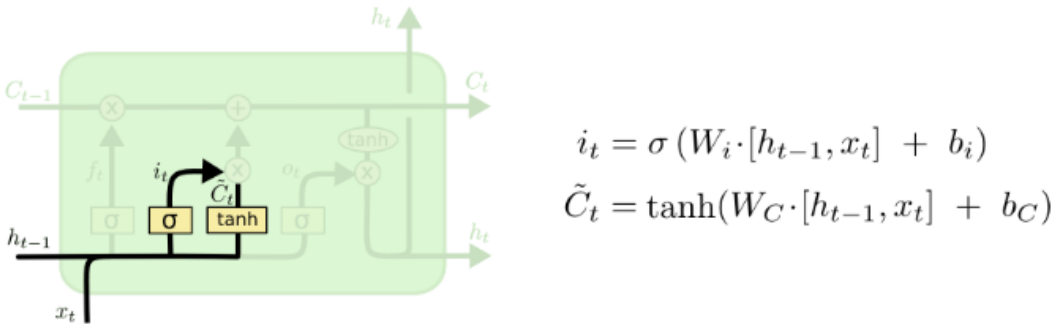


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Fonte: Colah's blog - Understanding LSTM Networks (2015).

O *input gate* vai determinar as informações que serão utilizadas no momento atual. Inicialmente faz-se uma filtragem das entradas (x_t e h_{t-1}) utilizando a função sigmóide (σ), mesma função utilizada pelo *forget gate*. Em seguida, se cria um vetor utilizando a função tanh, esse vetor é formado por todos os possíveis valores de x_t e h_{t-1}, os valores variam em um intervalo de [-1, 1]. Por fim, é feita uma multiplicação entre os valores filtrados e os valores do vetor para obter as informações úteis à rede. A Figura 4 mostra essa porta e apresenta respectivamente as funções i_t e C_t descritas.

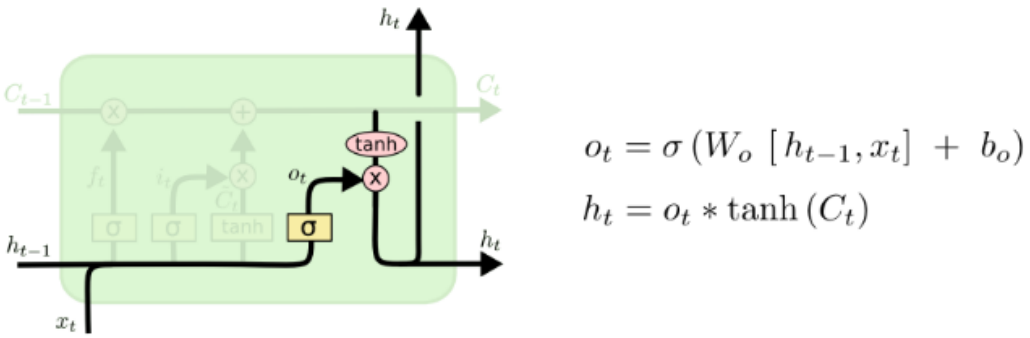
Figura 4 – Arquitetura de uma LSTM: Input Gate



Fonte: Colah's blog - Understanding LSTM Networks (2015).

O *output gate* é a porta responsável por selecionar as informações que serão dadas como saída atual e consequentemente entrada para o próximo estado. Nesse caso, primeiro se gera um vetor a partir da aplicação da função tanh à célula e em seguida as informações são filtradas utilizando a função sigmóide (σ) de modo a obter apenas os valores que devem ser lembrados. Logo após esse processo faz-se a multiplicação entre os valores filtrados e os valores do vetor para obter as informações úteis à rede, esse último processo é semelhante ao que é feito ao final do *input gate*. Os valores resultantes dessa multiplicação serão escolhidos como a saída final dessa célula e serão passados para a próxima. A Figura 5 abaixo destaca essa porta e apresenta as funções o_t e h_t descritas.

Figura 5 – Arquitetura de uma LSTM: *Output* Gate



Fonte: Colah's blog - Understanding LSTM Networks (2015).

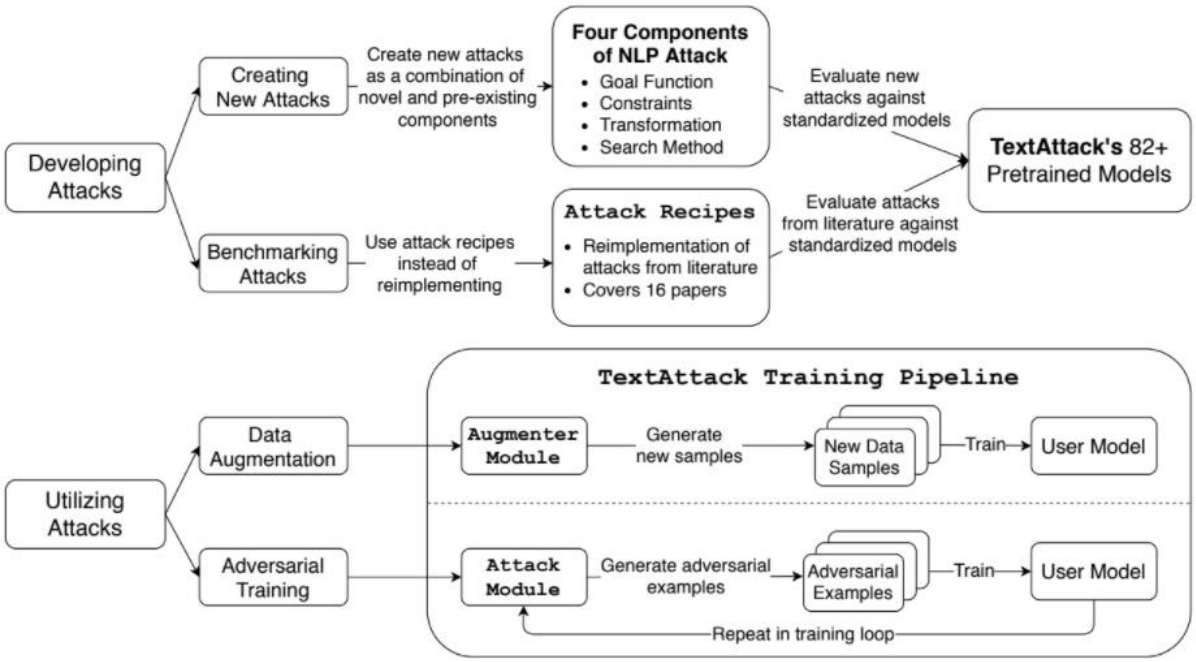
2.3 TextAttack

O TextAttack é um framework desenvolvido a partir do problema que existe até hoje que é a dificuldade de realizar ataques a modelos PLN e utilizá-los para melhorar a performance do modelo [9]. O TextAttack permite a realização de *adversarial attacks, data augmentation* e *adversarial training* em PLN, tudo isso é possível a partir de quatro componentes que o compõem: função objetivo, conjunto de restrições, transformação e método de busca. Veremos um pouco mais sobre esses componentes nas próximas subseções.

A Figura 6 apresenta as principais características do TextAttack. Quanto ao desenvolvimento dos ataques existem duas possibilidades, ou a criação de novos ataques ou o uso dos ataques prontos. A criação de novos ataques se dá utilizando os quatro componentes já citados, enquanto os ataques prontos são na verdade reimplementações de ataques da literatura. O TextAttack dispõe de 16 ataques adversariais da literatura completamente configurados que podem ser facilmente utilizados caso não se deseje criar o próprio ataque. Uma descrição desses ataques da literatura está disponível no anexo A. Todos esses ataques podem ser aplicados no seu próprio modelo de rede neural, mas o TextAttack disponibiliza também mais de 82 modelos pré-treinados onde é possível testar essa ferramenta.

Ainda na Figura 6 podemos ver que os ataques podem ser utilizados de duas maneiras no modelo, ele tanto pode servir como *data augmentation* quanto pode ser utilizado para a realização de ataques adversariais. O TextAttack dispõe de um módulo específico para tratar cada um desses casos, mas em ambos o fluxo é bem parecido. Se o objetivo for *data augmentation* após a geração desses novos dados faz-se um novo treinamento do modelo, já se for a realização de testes geram-se as perturbações que serão utilizadas no modelo.

Figura 6 – Principais características do TextAttack



Fonte: MORRIS et al. (2020, p. 120).[4]

[4] Referência [9]

2.3.1 Função objetivo

Uma **função objetivo** é usada para determinar se um ataque obteve sucesso ou não em relação à saída do modelo. O TextAttack traz algumas funções objetivo que podem ser aplicadas ao ataque, entre elas podemos citar a classificação não direcionada, a classificação direcionada, redução do input, entre outras. A classificação não direcionada ou **UntargetedClassification** é a que utilizamos neste estudo portanto está descrita em maiores detalhes na Seção 4.2; a classificação direcionada ou **TargetedClassification** é usada quando o objetivo é maximizar o valor de uma classe específica da rede; já a redução do input ou **InputReduction** tenta reduzir a quantidade de palavras da entrada ao máximo enquanto mantém a mesma classificação original.

2.3.2 Conjunto de restrições

O **conjunto de restrições** determina se uma perturbação *x'* é válida. Uma perturbação *x'* é o resultado de uma alteração realizada na entrada original *x*, como estamos tratando de testes em um conjunto de dados textual todas as perturbações dizem respeito a mudanças de palavras e/ou caracteres no conjunto de dados original.

Quando realizamos as perturbações sobre o conjunto de dados de entrada existem um número *n* de critérios que podemos levar em conta para validar ou não a perturbação formada, esses critérios são as restrições. Uma restrição bem comum que pode ser aplicada, se assim for necessário, é um verificador gramatical, para garantir que as perturbações não afetaram de maneira negativa a parte gramatical da sentença.

No estudo descrito neste trabalho foram utilizadas três restrições, **LevenshteinEditDistance, RepeatModification** e **StopwordModification**, todas elas são descritas na Seção 4.2.

As restrições presentes no TextAttack são classificadas em três categorias: semântica, gramatical e de sobreposição.

As **restrições semânticas** validam as perturbações com base no nível de similaridade semântica entre a entrada original *x* e a perturbação *x'*.

São classificadas como **restrições gramaticais** aquelas que consideram válidas as perturbações que preservam a sintaxe das palavras evitando os erros gramaticais.

As **restrições de sobreposição** fazem a validação das perturbações baseadas em análises a nível de caracteres. A restrição **LevenshteinEditDistance** utilizada nesse estudo se encaixa nessa categoria.

2.3.3 Transformação

A **transformação** é o componente responsável por gerar um conjunto de possíveis perturbações sobre os dados de entrada. O TextAttack disponibiliza diversos tipos de transformações, algumas mais comuns são a troca de *n* palavras da sentença, a troca de posição das letras em uma determinada palavra ou a substituição de caracteres homógrafos.

Nesse estudo foram utilizadas as transformações **WordSwapNeighboringCharacterSwap, WordSwapRandomCharacterDeletion, WordSwapQWERTY** e **WordSwapOrtography** descritas na Seção 4.2. Essas

transformações em particular foram escolhidas dado o contexto específico do classificador, por ele trabalhar com descrições de produtos em formato textual onde não há uma padronização ou rigor quanto a ortografia das palavras, é importante saber como o classificador está se comportando em relação as entradas que possuem palavras com letras faltantes, ou onde alguns de seus caracteres tiveram suas posições invertidas por exemplo, visto que esse tipo de erro é comum quando se digita em texto livre.

2.3.4 Método de busca

O **método de busca** é o componente que seleciona as perturbações do conjunto de transformações que serão utilizadas no modelo. Como em todos os outros componentes existem vários métodos de busca diferentes para se escolher e a partir da união desses componentes é possível formar *n* tipos de ataques diferentes.

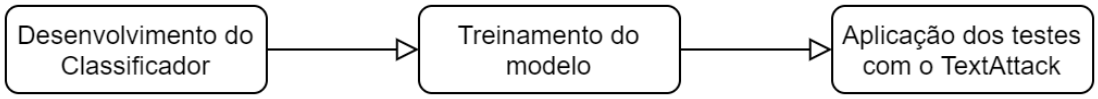
No nosso estudo o método de busca utilizado foi o **GreedyWordSwapWIR** (ver descrição na Seção 4.2) para que fossem aplicadas ao nosso modelo apenas as perturbações com maior significância gerando assim resultados mais satisfatórios.

3 METODOLOGIA

Foram muitos os processos utilizados até chegar aos resultados atuais do classificador, uma vez que este projeto está sendo desenvolvido já a alguns anos. Inicialmente foi necessário fazer um levantamento bibliográfico, focando principalmente em artigos, que pudessem apontar as diversas maneiras em que o problema poderia ser abordado para que a melhor solução fosse encontrada.

Quando o processo de pesquisa foi finalizado já iniciamos o desenvolvimento do classificador. Tivemos acesso a base de dados da SEFAZ-PB que continham as NF-e para que extraíssemos esses dados para usar no nosso modelo. Depois que essa parte foi concluída foi realizado o treinamento da rede e por último os testes com o TextAttack. A Figura 7 mostra os passos seguidos durante o desenvolvimento dessa pesquisa.

Figura 7 – Fluxograma com a metodologia utilizada



Fonte: Elaborada pelo autor, 2021.

Para desenvolver o classificador os dados coletados passaram por um pré-processamento removendo principalmente números e *stopwords* (palavras sem relevância) das descrições, só então eles foram usados para o treinamento da rede. Embora não seja o foco deste trabalho, e por isso não é abordado de maneira mais aprofundada, foram realizados testes para descobrir qual seria o modelo e a técnica de PLN que melhor se adequaria ao nosso projeto, ao analisar os resultados desses

testes chegou-se à conclusão de que a melhor opção seria utilizar o modelo do FastText com técnicas de *word embeddings*.

Assim surgiram as primeiras versões do classificador, mas ainda existiam alguns problemas como por exemplo a falta de representatividade de alguns dados, ainda mais por se tratar de dados textuais, alguns casos com erros ortográficos ou abreviações incomuns que possuíam poucos exemplos na nossa base de treino e acabavam por não apresentar os resultados desejados, nesse contexto foi aplicado o TextAttack, um framework desenvolvido justamente para realização de ataques adversariais.

Para a realização desses testes foram utilizados os componentes do TextAttack para desenvolver um ataque que fosse personalizado de acordo com o contexto do classificador. Esses ataques são formados a partir da perturbação de algumas descrições de produtos, quando temos um conjunto considerável dessas perturbações passamos esses dados para o classificador para que ele os classifique. Os resultados da classificação das perturbações geradas podem ser encontrados na Seção 5.

4 CENÁRIO DA PESQUISA

Nesta seção estão descritos os detalhes relacionados ao cenário da pesquisa, na subseção 4.1 estão todos os detalhes concernentes ao modelo do classificador de produtos e na subseção 4.2 encontra-se as características do ataque realizado.

4.1 Classificador de produtos

Para desenvolver o classificador foram mais de 6,5 milhões de notas coletadas (NF-e e NFC-e) de diferentes bases de dados. A divisão desses dados foi realizada da seguinte forma: 80% foi utilizado para treino, 10% para os testes e os outros 10% para a validação do modelo.

O estudo descrito nesse artigo se restringe a categoria de bebidas, sendo os produtos categorizados entre cerveja, aguardente de cana, bebidas quentes, refrigerante, água mineral e outros (qualquer produto que não se encaixe nas demais categorias). O campo da nota fiscal utilizado como parâmetro para fazer essa categorização foi o NCM.

A confiança geral do classificador é de 97%, sendo 91% para a categoria de bebidas quentes, 94% para a categoria de cerveja, 99% para aguardente, quase 100% para as categorias de água e refrigerante e também de 99% para a categoria de outros.

Foi nesse classificador que fizemos os ataques adversariais utilizando o TextAttack.

4.2 Aplicação do TextAttack

O TextAttack foi aplicado a um modelo pré-treinado do classificador. Dado o cenário único que temos optou-se for realizar um ataque (black-box) personalizado ao modelo. As configurações relacionadas ao ataque, tais como o método de busca e a função objetivo utilizados bem como as transformações e restrições aplicadas podem ser encontradas na Tabela 1 abaixo.

Tabela 1 – Configurações do ataque

Método de busca	Função objetivo	Transformações	Restrições
GreedyWordSwapWIR	UntargetedClassification	WordSwapNeighboringCharacterSwap	LevenshteinEditDistance
		WordSwapQWERTY	RepeatModification
		WordSwapRandomCharacterDeletion	StopwordModification
		WordSwapOrtography	

Fonte: Elaborada pelo autor, 2021.

O método de busca **GreedyWordSwapWIR** primeiramente ordena as perturbações por índices de acordo com a sua importância e só então ele escolhe quais perturbações serão aplicadas dando prioridade as de maior índice, ou seja, as perturbações de maior valor [12]. Outra particularidade desse método é que quando o parâmetro **wir_method** é configurado como **unk**, que foi o caso nesse ataque, ele se torna uma reimplementação do método de busca descrito no artigo Is BERT Really Robust [6] que pode ser encontrado nas referências.

A função objetivo **UntargetedClassification** tem como objetivo diminuir a confiança na categorização correta, ele pode atuar de duas maneiras a depender de como o parâmetro **target_max_score** é configurado, se ele receber um valor float então o objetivo é reduzir o nível de confiança na categoria correta até esse valor passado e todos os ataques que se comportarem dessa maneira serão tidos como ataques de sucesso. Quando nenhum valor é passado para esse parâmetro então o objetivo é mudar a categoria prevista. Neste trabalho, nenhum valor foi passado para o **target_max_score**, o que significa que um ataque tem sucesso quando ele muda a classificação de um produto, fazendo com que um produto que estava sendo categorizado corretamente passe a ser categorizado erroneamente [12].

Foram aplicadas quatro transformações distintas como pode ser visto na Tabela 1, as características delas serão descritas a seguir.

O **WordSwapNeighboringCharacterSwap** é responsável por transformar a entrada que recebe trocando de lugar um caractere de uma determinada palavra com seu caractere vizinho. Quando o parâmetro **random_one** é inicializado como *TRUE* então em toda a entrada uma única palavra é modificada tendo dois de seus caracteres trocados de lugar, esse foi o caso utilizado neste trabalho [12].

O **WordSwapQWERTY** se assemelha ao **WordSwapNeighboringCharacterSwap**, no entanto ao invés de trocar um caractere por seu vizinho, ele troca um caractere por outro cuja tecla no teclado *QWERTY* seja adjacente [12].

O **WordSwapRandomCharacterDeletion** quando aplicado a uma entrada exclui de maneira aleatória caracteres de uma palavra, se o parâmetro **random_one** for inicializado como *TRUE* isso significa que um único caractere de uma única palavra será excluído, essa foi a configuração dos ataques aplicado ao modelo do classificador [12].

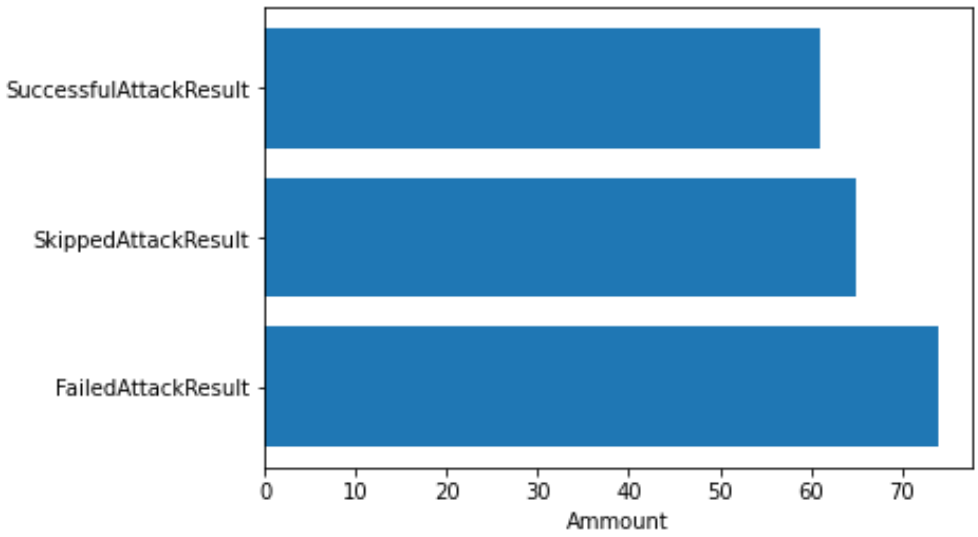
O **WordSwapOrtography** é responsável por mudar a ortografia de uma determinada palavra ao trocar um caractere por outro, difere do **WordSwapNeighboringCharacterSwap** pois o novo caractere não precisa estar contido na palavra original e difere do **WordSwapQWERTY** porque o novo caractere

não precisa estar adjacente ao original, mas pode estar em qualquer posição do teclado [12].

As restrições utilizadas foram a **LevenshteinEditDistance**, a **RepeatModification** e a **StopwordModification**. A primeira usa a distância de Levenshtein para determinar qual a distância entre as transformações [12]. O parâmentro **max_edit_distance** foi inicializado com o valor 2 o que significa que a distância máxima de edição permitida será 2. Já o parâmetro **compare_against_original** que foi inicializado com *TRUE* indica que a comparação realizada para determinar a distância será entre o valor da perturbação e a sua entrada original. A **RepeatModification** serve para impedir que uma mesma palavra sofra mais de uma transformação, essa restrição é bem útil em nosso caso em que quatro tipos diferentes de transformações foram utilizados, assim temos a garantia que cada palavra terá no máximo uma modificação [12]. A **StopwordModification** é uma restrição que impede que *stopwords* sejam modificadas durante as transformações [12].

Após definido o ataque que seria utilizado aplicamos esse ataque aos dados de teste. Um ataque pode obter um dos três resultados quando se tentar aplica-lo, ele pode ter sucesso, pode falhar ou pode nem chagar a ser aplicado. O sucesso ou falha de um ataque é definido pela função objetivo, já a não aplicação de um ataque pode ocorrer em situações em que a perturbação, ao passar pelo método de busca, não atende as restrições e por tanto é descartada. O Gráfico 1 mostra o resultado dos ataques que foram aplicados ao modelo do classificador de produtos.

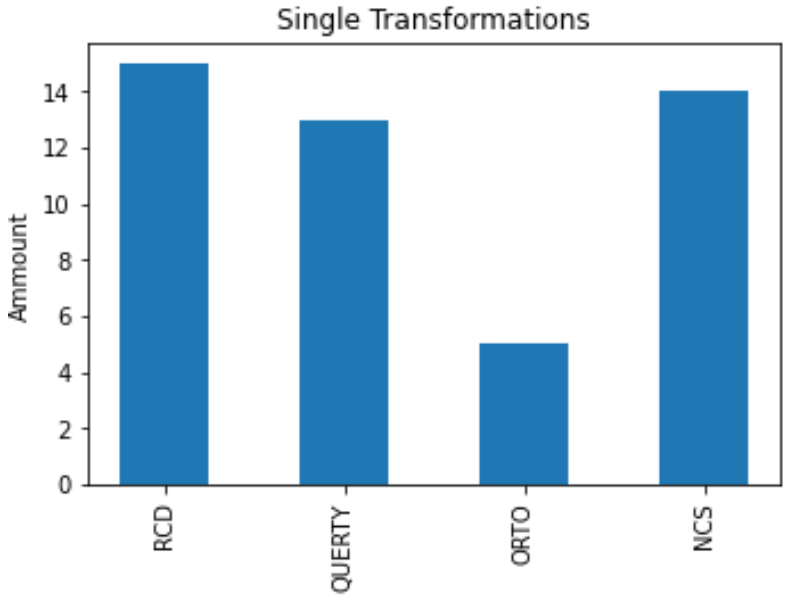
Gráfico 1 – Resultado dos ataques



Fonte: Elaborada pelo autor, 2021.

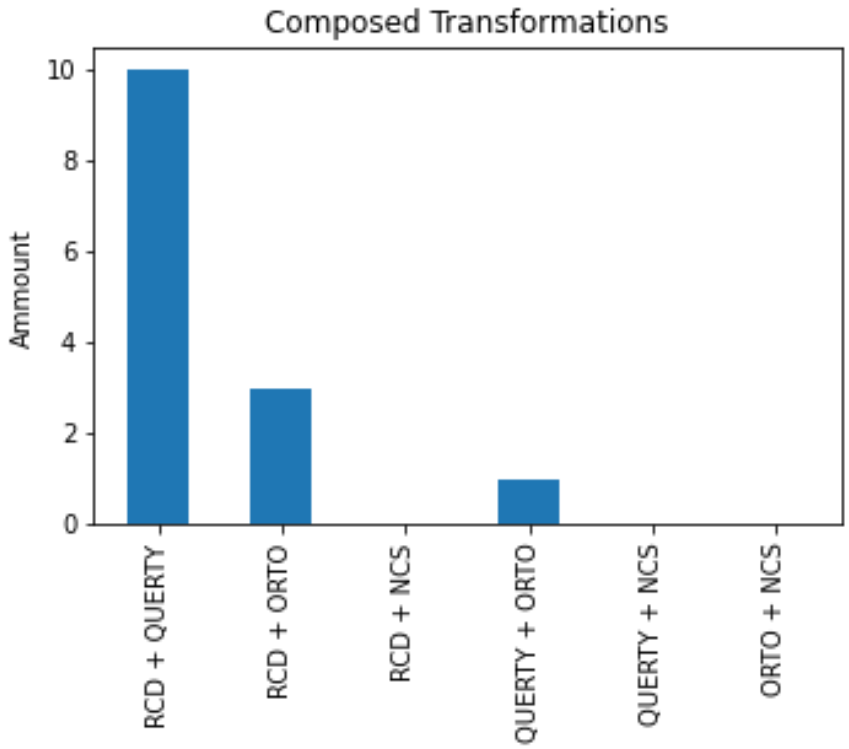
Passando mais especificamente para os ataques que obtiveram sucesso, os Gráfico 2 e 3 nos mostram quais foram as transformações aplicadas a esses ataques. Como visto na Tabela 1 foram utilizados quatro tipos de transformações diferentes. O Gráfico 2 é referente aos ataques em que uma única transformação foi aplicada e, no Gráfico 3 são apresentados os ataques em que houve uma composição de transformações, isso quer dizer que em um mesmo ataque foram aplicadas duas transformações.

Gráfico 2 – Transformações únicas utilizadas nos ataques que obtiveram sucesso



Fonte: Elaborada pelo autor, 2021.

Gráfico 3 – Transformações compostas utilizadas nos ataques que obtiveram sucesso



Fonte: Elaborada pelo autor, 2021.

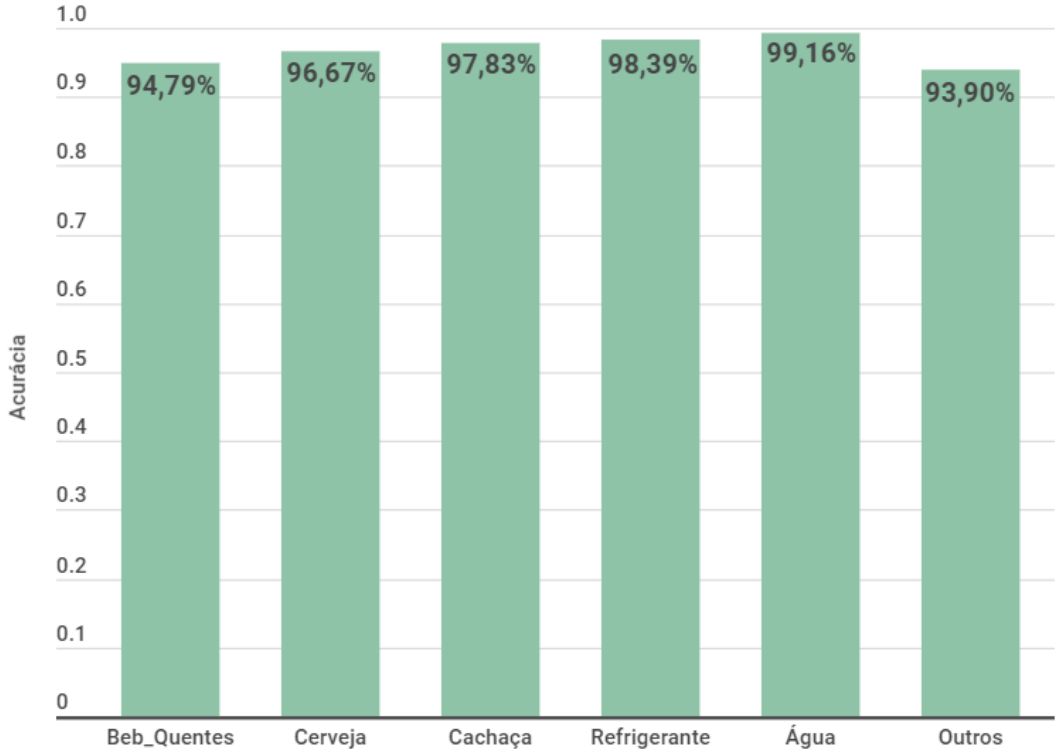
Os Gráficos 1, 2 e 3 são referentes a um recorte com 200 casos de testes, o classificador sofreu ao todo 20000 ataques.

5 RESULTADOS E DISCUSSÕES

Para facilitar a identificação da contribuição do uso do TextAttack nos resultados finais do classificador de produtos foi realizada uma comparação entre os primeiros resultados obtidos (sem o uso do TextAttack) e os resultados obtidos quando esse framework foi utilizado.

Neste primeiro cenário a acurácia final do classificador foi de 97%, sendo que individualmente a que apresentou os melhores resultados foi a categoria de água com 99,16%, seguida pela categoria de refrigerantes com 98,39%, a de aguardente (cachaça) com 97,83%, cerveja com 96,67%, bebidas quentes com 94,79% e por último a categoria de outros com 93,90%. Essas informações podem ser visualizadas no gráfico 4 abaixo.

Gráfico 4 – Acurácia do classificador por categoria (sem o uso do TextAttack)



Fonte: Elaborada pelo autor, 2021.

Já quando fizemos a aplicação do TextAttack, descrita na Seção 4.2, percebemos como o classificador está se comportando em relação as entradas que recebe e quais alterações ortográficas podem confundi-lo, a Tabela 2 mostra um recorte desses resultados.

Tabela 2 – Recorte dos resultados do TextAttack

Categoria original	Descrição	Transformações	Nova categoria
Cerveja (100%)	CERV SKOL ML CX	CRV SKOL MI CX **(WSRCD/WSQ)**	Outros (97%)
Cerveja	BOHEMIA GF VD ML CX	BOHEMAI GF VD ML CX	Outros (54%)

(100%)		**(WSNCS)**	
Beb_quentes (97%)	CACHACA TRIUNFO ML	ACHACA TRIUNF ML **(WSRCD)**	Cerveja (53%)
Refrigerante (100%)	GUARANA CHP ANTARCTICA PET ML SHRINK	GARANA CHP ANTARCTICA PET ML SHEINK **(WSRCD/WSQ)**	Cerveja (63%)
Refrigerante (67%)	FANTA LARANJA LT	FANTA LARANJA rT **(WSO)**	Cachaca (43%)
Agua (100%)	AGUA MINDAIA PET BDJ X ML	AGU MINDAIA PET BDJ X ML **(WSRCD)**	Beb_quentes (81%)
Agua (100%)	AGUA MVILLA GAS PET BDJ X ML	AGAU MVILLA GAS PET BDJ X ML **(WSNCS)**	Refrigerante (65%)
Cachaca (70%)	CAMPARI	CAMPAR **(WSRCD)**	Cerveja (38%)
Outros (100%)	CALCIOFARM INJETAVEL ML	CALCIOFARM IJETAVEL ML **(WSRCD)**	Cerveja (53%)
Outros (100%)	FRESH ICE AROMATIZANTE BUCAL MORANGO ML	FlESH ICE AROMATIZANTE BCAL MORANGO ML **(WSO)**	Cachaca (97%)

Fonte: Elaborada pelo autor, 2021.

Na Tabela 2 a coluna **categoria original** é referente a categoria correta do produto dada pelo classificador juntamente com o nível de confiança com que ele foi classificado. A coluna **descrição** possui as descrições originais dos produtos enquanto a coluna **transformações** possui as descrições com as perturbações geradas pelo TextAttack e que foram utilizadas nos ataques, para facilitar a identificação das transformações utilizadas nos ataques, após cada descrição nesta coluna está entre parênteses a abreviação da transformação que foi aplicada. A última coluna (**nova categoria**) diz respeito a classificação das perturbações juntamente com o nível de confiança dado pelo classificador. As cores diferentes possuem dois objetivos, facilitar a identificação da categoria a que a descrição pertence uma vez que cada categoria possui uma cor de identificação única (cerveja – verde, cachaça – azul, bebidas quentes – vermelho, refrigerante – rosa, água – amarelo, outros – laranja), e destacar qual(is) palavra(s) sofreram transformações.

Destacando alguns casos em particular, para a descrição "AGUA MVILLA GAS PET BDJ X ML" o classificador acerta com uma confiança de 100% que ela pertence à categoria de **água**, no entanto quando a transformação é aplicada e dois caracteres trocam de posição, "AGAU MVILLA GAS PET BDJ X ML", o classificador interpreta essa nova descrição como da classe de **refrigerante** com uma confiança de 65%, o que já é um valor considerável principalmente nesse caso em que quase nenhuma perturbação foi realizada; um outro caso bastante crítico ocorre para a descrição "CALCIOFARM INJETAVEL ML" que é inicialmente categorizado corretamente como **outros** com uma confiança de 100%, mas que ao receber uma perturbação mínima onde uma única letra é removida, "CALCIOFARM IJETAVEL ML", o classificador entende essa nova descrição como pertencente à categoria de **cerveja** com uma confiança de 53%. Podemos perceber que pequenos erros ortográficos estão gerando grandes impactos no classificador.

Em nosso contexto é muito importante entender esses detalhes do comportamento do classificador porque como as entradas que ele recebe são descrições de produtos em formato textual onde não há uma padronização da

escrita, além de ocorrerem muitos casos com erros ortográficos, então ter essa compreensão sobre como o classificador está interpretando esses casos facilita muito o processo de aperfeiçoamento do modelo, buscando aumentar não apenas a acurácia, mas também a confiança na categorização.

Foram realizados um total de 20000 testes e o uso do TextAttack nos mostrou que embora o classificador esteja trabalhando bem com as descrições originais, com acurácia e confiança satisfatórias, atualmente ele não está preparado para lidar com descrições que possuam erros ortográficos onde letras são esquecidas e/ou invertidas, erros muito comuns ao se digitar um texto.

6 CONCLUSÃO

Existe uma dificuldade muito grande por parte da SEFAZ-PB em realizar a fiscalização manual das NF-e porque não existe uma padronização das descrições textuais dos produtos e o número das NF-e também são altos. Por isso surgiu a necessidade de automatizar esse processo, como solução para esse problema foi desenvolvido o classificador de produtos.

Qualquer nova ferramenta precisa ser testada para garantir os seus resultados e possibilitar seu aperfeiçoamento, com o classificador de produtos não é diferente. Sendo assim o objetivo desta pesquisa foi apresentar como o TextAttack pode contribuir para o teste de redes neurais que trabalham com PLN aplicando-o ao classificador. Foi escolhido o TextAttack em função das entradas do classificador, uma vez que elas são descrições de produtos são frequentemente encontrados abreviações e erros ortográficos e é preciso compreender como o classificador se comporta ao receber esse tipo de dado.

Foi possível notar que embora a acurácia do classificador seja alta e ele apresente bons níveis de confiança ele não está preparado para lidar com erros gramaticais simples como a omissão de um caractere ou a inversão na posição de dois caracteres dentro de uma palavra. Essa limitação nos mostra a necessidade de retreinar o classificador com mais dados que contenham esses erros para que ele possa aprender a reconhecê-los e classificar esses produtos corretamente.

Por tanto, como trabalhos futuros sugere-se o retreino do modelo do classificador e a aplicação de um novo conjunto de testes para avaliar as melhorias. Podem também ser feitos testes com outras configurações de ataques utilizando o TextAttack para verificar o comportamento da rede ou ainda realizar uma pesquisa comparando os ataques do TextAttack com os de outra ferramenta como por exemplo o CheckList, dessa forma seria possível analisar qual dessas ferramentas se adequa melhor no contexto do classificador.

REFERÊNCIAS

[1] CAPÍTULO 10 – As 10 Principais Arquiteturas de Redes Neurais. **Deep Learning Book**. Disponível em: <https://www.deeplearningbook.com.br/as-10-principais-arquiteturas-de-redes-neurais/>. Acesso em: 19 ago. 2021.

[2] CAPÍTULO 34 – O Problema da Dissipação do Gradiente. **Deep Learning Book**. Disponível em: <https://www.deeplearningbook.com.br/o-problema-da-dissipacao-do-gradiente/>. Acesso em: 19 ago. 2021.

[3] CAPÍTULO 51 – Arquitetura de Redes Neurais Long Short Term Memory (LSTM). **Deep Learning Book**. Disponível em: <https://www.deeplearningbook.com.br/arquitetura-de-redes-neurais-long-short-term-memory/>. Acesso em: 11 ago. 2021.

[4] HARTMANN, N. S.; FONSECA, E.; SHULBY, C. D.; TREVISO, M. V.; RODRIGUES, J. S.; ALUÍSIO, S. M. Portuguese word embeddings: Evaluating on word analogies and natural language tasks. **In Proceedings of the 11th Brazilian Symposium in Information and Human Language Technology (STIL 2017)**, Uberlândia, Brasil, arXiv:1708.06025, v. 1, n.1, p. 122–131, ago. 2017.

[5] HOCHREITER, S.; SCHMIDHUBER, J. LONG SHORT-TERM MEMORY, 1997.

[6] JIN, D.; JIN, Z.; ZHOU, J. T.; SZOLOVITS, P. Is BERT Really Robust? A Strong Baseline for Natural Language Attack on Text Classification and Entailment, 2019.

[7] Junior, J. R. F. Redes Neurais Recorrentes — LSTM. **Medium**. Disponível em: <https://medium.com/@web2ajax/redes-neurais-recorrentes-lstm-b90b720dc3f6>. Acesso em: 23 ago. 2021.

[8] LIDDY, E. D. Natural Language Processing. In Encyclopedia of Library and Information Science, 2nd Ed. NY. Marcel Decker, Inc. 2001.

[9] MORRIS, J. X.; LIFLAND, E..; YOO, J. Y.; GRIGSBY, J..; JIN, D.; QI, Y. TextAttack: A Framework for Adversarial Attacks, Data Augmentation, and Adversarial Training in NLP. **Proceedings of the 2020 EMNLP (Systems Demonstrations)**, USA, arXiv:2005.05909, v. 4, n. 1, p. 119-126, nov. 2020.

[10] RIBEIRO, M. T.; TONGSHUANG, W.; GUESTRIN, C.; SINGH, S. Beyond Accuracy: Behavioral Testing of NLP Models with CheckList. **Association for Computational Linguistics (ACL)**, USA, arXiv:2005.04118, v. 1, n. 1, maio 2020.

[11] ROY, D.; GANGULY, D.; BHATIA, S.; BEDATHUR, S.; MITRA, M. Using Word Embeddings for Information Retrieval: How Collection and Term Normalization Choices Affect Performance. **Proceedings of the 27th ACM International Conference on Information and Knowledge Management**, Torino, Italy, p. 1835–1838, out. 2018.

[12] TEXTATTACK Documentation. **Read the Docs**, 2020. Disponível em: <https://textattack.readthedocs.io/en/latest/>. Acesso em: 20 set. 2021.

ANEXO A – ADVERSARIAL ATTACKS FROM THE LITERATURE IN TEXTATTACK

Attack Recipe	**Goal Function**	**Constraints**	**Transformation**	**Search Method**
bae (Garg and Ramakrishnan, 2020)	Untargeted Classification	USE sentence encoding cosine similarity	BERT Masked Token Prediction	Greedy-WIR
bert-attack (Li et al., 2020)	Untargeted Classification	USE sentence encoding cosine similarity, Maximum number of words perturbed	BERT Masked Token Prediction (with subword expansion)	Greedy-WIR
deepwordbug (Gao et al., 2018)	*{*Untargeted, Targeted*}* Classification	Levenshtein edit distance	*{*Character Insertion, Character Deletion, Neighboring Character Swap, Character Substitution*}**	Greedy-WIR
alzantot, fast-alzantot (Alzantot et al., 2018; Jia et al., 2019)	Untargeted *{*Classification, Entailment*}*	Percentage of words perturbed, Language Model perplexity, Word embedding distance	Counter-fitted word embedding swap	Genetic Algorithm
iga (Wang et al., 2019)	Untargeted *{*Classification, Entailment*}*	Percentage of words perturbed, Word embedding distance	Counter-fitted word embedding swap	Genetic Algorithm
input-reduction (Feng et al., 2018)	Input Reduction		Word deletion	Greedy-WIR
kuleshov (Kuleshov et al., 2018)	Untargeted Classification	Thought vector encoding cosine similarity, Language model similarity probability	Counter-fitted word embedding swap	Greedy word swap

hotflip (word swap) (Ebrahimi et al., 2017)	Untargeted Classification	Word Embedding Cosine Similarity, Part-of-speech match, Number of words perturbed	Gradient-Based Word Swap	Beam search
morpheus (Tan et al., 2020)	Minimum BLEU Score		Inflection Word Swap	Greedy search
pruthi (Pruthi et al., 2019)	Untargeted Classification	Minimum word length, Maximum number of words perturbed	*{*Neighboring Character Swap, Character Deletion, Character Insertion, Keyboard-Based Character Swap*}**	Greedy search
pso (Zang et al., 2020)	Untargeted Classification		HowNet Word Swap	Particle Swarm Optimization
pwws (Ren et al., 2019)	Untargeted Classification		WordNet-based synonym swap	Greedy-WIR (saliency)
seq2sick (black-box) (Cheng et al., 2018)	Non overlapping output		Counter-fitted word embedding swap	Greedy-WIR
textbugger (black-box) (Li et al., 2019)	Untargeted Classification	USE sentence encoding cosine similarity	*{*Character Insertion, Character Deletion, Neighboring Character Swap, Character Substitution*}**	Greedy-WIR
textfooler (Jin et al., 2019)	Untargeted *{*Classification, Entailment*}*	Word Embedding Distance, Part-of-speech match, USE sentence encoding cosine similarity	Counter-fitted word embedding swap	Greedy-WIR

AGRADECIMENTOS

Quero agradecer primeiramente a Deus não só pela conclusão deste trabalho, mas por todo o trajeto até aqui. Pela condição de finalizar esse curso e por ter me dado forças para continuar em momentos difíceis. Tenho motivos de sobra para agradecer a Deus pois tem me abençoado e alegrado todos os dias, faltaria espaço para agradecer por cada benção recebida, então concluo agradecendo pela maior delas, a salvação pelo sacrifício de Jesus, porque eu certamente não merecia. Toda glória, honra e louvor seja dada a Deus.

Agradeço também à minha família e amigos que me ajudaram e apoiaram em todos os momentos, por acreditarem em mim e acreditarem que eu um dia receberia meu diploma antes mesmo de eu sequer ter decidido qual curso escolheria.

Agradeço a todos os professores do curso de Computação da UEPB, inclusive àqueles com quem eu não tive oportunidade de estudar, sem vocês nenhum de nós, alunos, teríamos chegado até aqui.

Agradeço em especial à professora Kézia de Vasconcelos Oliveira Dantas, tanto por ter aceitado ser minha orientadora quanto por ter me acompanhado nos estudos e pesquisas desde o início do curso, e por ter sempre me apresentado novas oportunidades.

Agradeço também ao NUTES/UEPB por todas as oportunidades que tive em projetos pois me proporcionaram experiências e aprendizados que vão além do que vemos nas disciplinas durante o curso.

Finalizo agradecendo a todos os meus colegas de curso por terem dividido essa experiência comigo. Agradeço em especial a todos os colegas que se tornaram amigos.