ALESON GLEIK SILVA CHAVES

CLASSIFICAÇÃO DE PRODUTOS UTILIZANDO TÉCNICAS FEW-SHOT LEARNING

LAVRAS – MG

2022

ALESON GLEIK SILVA CHAVES

CLASSIFICAÇÃO DE PRODUTOS UTILIZANDO TÉCNICAS FEW-SHOT LEARNING

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Engenharia de Sistemas e Automação, linha de pesquisa em Sistemas Inteligentes, para obtenção do título de Mestre.

Prof. DSc. Bruno Henrique Groenner Barbosa
Orientador

Prof. DSc. Danton Diego Ferreira
Coorientador

LAVRAS – MG

2022

Ficha catalográfica elaborada pelo Sistema de Geração de Ficha Catalográfica da Biblioteca Universitária da UFLA, com dados informados pelo(a) próprio(a) autor(a).

Chaves, Aleson Gleik Silva.
Classificação de Produtos Utilizando Técnicas Few-Shot Learning / Aleson Gleik Silva Chaves. - 2022.
108 p. : il.

Orientador: Prof. DSc. Bruno Henrique Groenner Barbosa.
Coorientador: Prof. DSc. Danton Diego Ferreira.
Dissertação (mestrado acadêmico) - Universidade Federal de Lavras, 2022.
Bibliografia.

1. Comércio Eletrônico. 2. Processamento de Linguagem Natural. 3. Few-Shot Learning. I. Barbosa, Bruno Henrique Groenner. II. Ferreira, Danton Diego. III. Título.

ALESON GLEIK SILVA CHAVES

CLASSIFICAÇÃO DE PRODUTOS UTILIZANDO TÉCNICAS FEW-SHOT LEARNING

PRODUCT CLASSIFICATION USING FEW-SHOT LEARNING TECHNIQUES

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Engenharia de Sistemas e Automação, linha de pesquisa em Sistemas Inteligentes, para obtenção do título de Mestre.

APROVADA em 29 de Abril de 2022.

Prof. DSc. Demostenes Zegarra Rodriguez	UFLA
Prof. DSc. Giovani Bernardes Vitor	UNIFEI

Prof. DSc. Bruno Henrique Groenner Barbosa
Orientador

Prof. DSc. Danton Diego Ferreira
Co-Orientador

LAVRAS – MG
2022

Aos meus pais.

AGRADECIMENTOS

Agradeço primeiramente à Deus por ter possibilitado que o plano fazer mestrado na área de inteligência computacional se tornasse realidade.

Aos meus pais por sempre me apoiarem em todos os meus desafios e por me incentivar a buscar meus sonhos.

À Universidade Federal de Lavras, ao Departamento de Engenharia e ao Programa de Pós-Graduação em Engenharia de Sistemas e Automação (PPGESISA) pela elaboração e oferta de um curso diversificado e atual. O mestrado no PPGESISA foi uma das melhores experiências que tive.

Ao orientador Prof. DSc. Bruno Henrique Groenner Barbosa e ao co-orientador Prof. DSc. Danton Diego Ferreira pela dedicação e pelos bons métodos pedagógicos utilizados na liderança deste projeto. Conseguiram transformar conteúdos que antes eram complexos em informações passíveis de serem compreendidas.

À empresa Omnilogic Inteligência S/A pelo suporte financeiro e à seus colaboradores pela contribuição nos direcionamentos durante o desenvolvimento de uma pesquisa aplicada.

Ao grupo de pesquisa *Artificial Intelligence and Automation* (AIA) pela oportunidade de ter feito parte deste projeto de pesquisa de ferramentas de *machine learning* aplicadas à *marketplaces*. Me concedeu a oportunidade de fazer o que mais gosto, que é desenvolver pesquisa voltada para a aplicação.

Por fim, à todos aqueles que contribuíram para o desenvolvimento desta dissertação o meu sincero agradecimento.

O sucesso é a soma de pequenos esforços - repetidos dia sim, e no outro dia também.
(Robert Collier)

RESUMO

As plataformas de comércio eletrônico (*marketplaces*) recebem diariamente milhares de produtos pertencentes a classes novas que não participaram do processo de treinamento do algoritmo responsável por automatizar a classificação de produtos. O retreinamento com estas classes novas é uma necessidade, pois a categorização incorreta de produtos nos *marketplaces* pode levar o consumidor a experiências desagradáveis no processo de compra. Porém, é difícil a constante atualização do sistema com estes produtos, pois o custo de retreinamento dos classificadores atualmente em operação é elevado devido à grande dimensão das bases de dados. A proposta apresentada neste trabalho é a utilização de classificadores de produtos que utilizam algoritmos do tipo *few-shot learning*, que são capazes de serem treinados com uma ou com poucas amostras por classes. Estes possuem treinamento rápido e necessitam de base de dados em dimensão reduzida. Os algoritmos testados foram: *k*-vizinhos mais próximos (KNN), redes *Matching Networks* (MN) e as redes DPGN (*Distribuition Propagation Graph Network*). Os algoritmos propostos para classificação de produtos utilizam características previamente extraídas a partir do processo de *transfer learning*, exceto para as redes *matching* com *encoder* contendo uma rede Bi-LSTM que recebe dados em linguagem natural extraídos por algoritmos *embeddings*. Os algoritmos foram testados com validação cruzada do tipo *leave one out* e *k-fold*. Também foi realizada a seleção das melhores características da base, possibilitando a redução de dimensão das mesmas, facilitando treinamento das redes neurais com *few-shot learning*. Foram utilizadas duas bases de dados para os testes, uma contendo 34 classes e 394 amostras e outra que possui 312 classes e 3120 amostras. O KNN foi utilizado como *baseline* do projeto e, apesar da simplicidade e não necessidade de treinamento, apresentou resultado satisfatório. As redes *matching* e DPGN ambas apresentaram resultados com 96,85% de acurácia conseguindo superar o KNN utilizando a base de dados com 34 classes e para a base de dados com 312 classes o melhor resultado foi obtido pelas redes *matching* com 93,78% de acurácia. A abordagem proposta de classificação de produtos pertencentes a classes novas traz como contribuição a correta categorização e a manutenção da acurácia exigida em *marketplaces*, sem a necessidade do retreinamento constante dos classificadores atualmente em operação. Isso pode trazer uma redução significativa de custo de uso do servidor em nuvem e melhores experiências de compras para os clientes.

Palavras-chave: Comércio Eletrônico, Processamento de Linguagem Natural, Aprendizado de Máquina, Redes Neurais Artificiais, Few-Shot Learning.

ABSTRACT

E-commerce platforms (marketplaces) receive daily thousands of products belonging to new classes that have not participated in the training process of the algorithm responsible for automating the classification of products. Retraining with these new classes is a necessity, as incorrect categorization of products in marketplaces can lead consumers to unpleasant experiences in the purchase process. However, it is difficult to constantly update the system with these products, because the cost of retraining the classifiers currently in operation is high due to the large size of the databases. The proposal presented in this work is the use of product classifiers that use few-shot learning algorithms, which are capable of being trained with one or few samples per class. These have rapid training and need a small-scale database. The algorithms tested were: k-nearest neighbors (KNN), Matching Networks (MN) and DPGN (Distribution Propagation Graph Network). The proposed algorithms for product classification use characteristics previously extracted from the transfer learning process, except for encoder matching networks containing a Bi-LSTM network that received data in natural language extracted by embedding algorithms. The algorithms were tested with leave one out and k-fold cross validation. The selection of the best characteristics of the data base was also carried out, making it possible to reduce their dimension, facilitating training of neural networks with few-shot learning. Two databases were used for the tests, one containing 34 classes and 394 samples and the other containing 312 classes and 3120 samples. KNN was used as a baseline for the project and, despite its simplicity and no need for training, it presented satisfactory results. The matching and DPGN networks both presented results with 96.85% accuracy, managing to overcome the KNN using the database with 34 classes and for the database with 312 classes, the best result was obtained by matching with 93.78% accuracy. The proposed approach for classifying products belonging to new classes contributes to the correct categorization and maintenance of the accuracy required in marketplaces, without the need for constant retraining of the classifiers currently in operation. This can bring significant cost reduction of cloud server usage and better shopping experiences for customers.

Keywords: E-commerce, Natural Language Processing, Machine Learning, Artificial Neural Networks, Few-Shot Learning

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Com o desenvolvimento dos recursos computacionais e do acesso à internet por grande parte da população, houve um aumento significativo do comércio eletrônico (RISTOSKI et al., 2018). Este mercado está se tornando cada vez mais promissor devido a diversos fatores tal como a praticidade de realizar as compras sem sair de casa.

O número de consumidores adeptos ao modelo de mercado *on-line* tem crescido, tal como o número de vendedores, e consequentemente a quantidade de ofertas nas plataformas estão cada vez maiores, oferecendo uma variedade de produtos. Em grandes plataformas de mercado como Amazon, Walmart, eBay e Magazine Luiza, novas vendas e vendedores são adicionados todos os dias (KRISHNAN; AMARTHALURI, 2019). Isto gera uma grande quantidade de dados, o que requer automatização para que os dados sejam cadastrados. Em geral, os dados encontram-se em linguagem natural não estruturada havendo a necessidade de realizar a transformação para a linguagem estruturada, para viabilizar a categorização de produtos e o processo de busca pelos clientes. Para que o consumidor possa ter uma boa experiência de compra e encontrar todos os produtos que deseja é necessário que a plataforma de *marketplace*[1] esteja muito bem categorizada e os anúncios devem ser padronizados com uma estrutura de texto que atenda ao consumidor de forma satisfatória. O produto deve ter todas as especificações organizadas para facilitar o processo de busca e compra de produtos (RISTOSKI et al., 2018).

A quantidade de produtos é crescente, o que inviabiliza a correção manual de cada produto nos *marketplaces* (KANNAN et al., 2011). Portanto, a transformação de dados não estruturados em dados estruturados se tornou possível com aplicação de ferramentas automatizadas de aprendizado de máquina aplicadas a resolução de tarefas de processamento de linguagem natural (SILVA et al., 2020). Neste processo estão sendo bastante utilizados métodos *embeddings* que extraem as características das palavras (YOUNG et al., 2018). Estas características são agrupadas em vetores que geram padrões para serem utilizados no processo de classificação, que geralmente é realizado por um classificador supervisionado implementado com redes neurais artificiais, como redes convolutivas (*Convolutive Neural Networks* - CNNs) (YAN et al., 2019) e redes neurais recorrentes do tipo *Long Short Term Memory* (LSTM) bidirecionais

[1] *Marketplace* se refere a uma plataforma de comércio eletrônico que reúne diversas marcas e lojas em um só lugar.

(KRISHNAN; AMARTHALURI, 2019). Os classificadores realizam o processo de extração de entidade dos produtos, ou seja, obtêm a classe a que os produtos pertencem.

As redes neurais artificiais aprendem o modelo do ambiente em que estão inseridas, sendo este ambiente representado por uma base de dados que é apresentada a ela no processo de treinamento (SONAWANE; PATIL, 2015). O treinamento é realizado estimando os parâmetros do modelo que minimize o erro entre a saída do sistema e a resposta desejada. Uma peculiaridade destes classificadores é a necessidade de grandes quantidades de dados para serem treinados, portanto, quanto maior o número de amostras e mais diversificados forem os dados, melhor será o resultado do algoritmo (LOURIDAS; EBERT, 2016) (ONG; GUPTA, 2019).

A classificação automática de produtos vem sendo bem solucionada com redes neurais com aprendizado profundo, uma vez que possuem bom desempenho justamente com grande quantidade de dados, que é o caso dos *marketplaces* atuais (FADLULLAH et al., 2017). Mas um problema para estes classificadores é a chegada de classes novas de produtos que não passaram pelo treinamento do classificador, sendo necessário constantemente fazer o retreino com as classes novas. Grande parte destas classes possuem poucas amostras, sendo isso um fator limitante para classificadores fundamentados em redes neurais.

A obtenção de classificadores de produtos que levem em consideração novas classes com poucas amostras é uma tarefa complexa, seja pelo custo computacional de retreinar os modelos existentes de forma a contemplar as novas classes, ou seja pela baixa quantidade de amostras dessas novas classes. Vale ressaltar que o número de classes em comércio eletrônico é da ordem de dezenas de milhares e, portanto, o projeto e treinamento do classificador para tal tarefa é bastante complexo. Dessa forma, alternativas que evitem o retreino dos classificadores ao incorporar o conhecimento de novas classes são muito bem vindas (CHAVES et al., 2021).

Os erros de classificação provocam, dentre outros problemas, a categorização incorreta de produtos nos *marketplaces*, podendo causar experiências desagradáveis no processo de compra. A falta de padrão pode fazer com que novas ofertas não apareçam na página do produto ou apareçam em página errada, devido à não correspondência dos mesmos produtos pertencentes a vendedores distintos (RISTOSKI et al., 2018).

Nesse sentido, técnicas de *Few Shot Learning* (FSL) são opções promissoras por serem projetadas especificamente para lidar com treinamento de modelos com pequenas quantidades de amostras por classe (JADON; GARG, 2020) (WANG et al., 2020). As técnicas *few-shot learning* são inspiradas no aprendizado humano que generaliza bem após ter visto alguns poucos

exemplos e reconhece variações desses conceitos em percepções futuras (LAKE et al., 2011). Pois, uma pessoa pode identificar a presença de uma outra em uma imagem após ter visto ela uma única vez anteriormente.

O contraste das redes neurais de aprendizado profundo (em relação à quantidade de dados) com o aprendizado humano traz grande atenção para a pesquisa *few-shot learning* (YANG et al., 2020). Em virtude disso, vários algoritmos de aprendizado FSL tem surgido, como: Redes Siamesas (KOCH; ZEMEL; SALAKHUTDINOV, 2015), *Memory-Augmented Neural Networks* (MANN) (SANTORO et al., 2016), *Matching Networks* (MN) (VINYALS et al., 2016), *Model Agnostic Meta-Learning* (MAML) (FINN; ABBEEL; LEVINE, 2017), *Graph Neural Networks* (GNN) (GARCIA; BRUNA, 2018) e *Distribution Propagation Graph Network* (DPGN) (YANG et al., 2020).

Desta forma, neste trabalho são propostos classificadores de produtos utilizando arquiteturas com algoritmos do tipo *few-shot learning*, que são capazes de realizar o treinamento de bases contendo somente classes novas com poucas amostras. As novas classes poderão ser classificadas sem a necessidade de realizar o treinamento do classificador com todas as ofertas da base de dados antiga.

Esta abordagem tem como benefício manter uma boa acurácia dos classificadores em operação nos *marketplaces*, mesmo com a chegada de novas classes. Também pode reduzir o uso do servidor em nuvem no treinamento dos modelos, disponibilizando o mesmo para uso em outras aplicações ou reduzindo o valor do contrato. Com esta aplicação não haverá necessidade de constantemente retreinar o classificador com a base dados inteira após a chegada de classes novas que não participaram do treinamento. Este treinamento poderá ser feito utilizando um classificador *few-shot* a parte que utiliza base de dados somente com as classes novas. As classes que o classificador principal em operação detectar como novas, ou seja, não alcançar um limiar de classificação, vão para a redes *few-shot learning* para serem preditas.

Manter uma boa acurácia dos classificadores significa ter a plataforma de *marketplace* bem categorizada, com as especificações de produto organizadas e anúncios padronizados em uma estrutura de texto que atenda às necessidades do consumidor. Isto facilita o processo de busca e compra de produtos, fazendo com que o consumidor encontre todos os produtos que deseja e tenha uma boa experiência de compra.

1.1 Objetivos

O objetivo geral deste trabalho é estudar e implementar algoritmos de aprendizado de máquina do tipo *few-shot learning* para extração de entidades de produtos em bases de dados de *web commerce*, que possuem poucas amostras pertencentes às classes novas.

Os objetivos específicos deste trabalho são:

- Propor uma arquitetura de classificador do tipo *few-shot learning* que funcionará em conjunto com o classificador principal em operação nas plataformas *marketplace*. As classes novas que não tiverem sido treinadas pelo classificador principal serão encaminhadas para este classificador FSL proposto.

- Analisar as características obtidas pelo classificador principal relacionadas com classes novas/desconhecidas, com o intuito de verificar a aplicabilidade de métodos do tipo *transfer-learning*.

- Analisar técnicas mais apropriadas para redução de dimensão das características da base de dados a fim de reduzir custo computacional e facilitar o treinamento das redes neurais do tipo FSL.

1.2 Estrutura do Trabalho

Este trabalho esta estruturado da seguinte forma: O capítulo II contém estudos sobre as diversas tecnologias de aprendizado de máquina envolvidas no desenvolvimento de aplicações de processamento de linguagem natural. No capítulo III são apresentadas a metodologia, descrevendo a implementação das técnicas *few-shot learning* aplicadas a processamento de linguagem natural, e ferramentas computacionais utilizadas no desenvolvimento deste trabalho. No capítulo IV são apresentados os resultados dos algoritmos *few-shot learning* aplicados. Por fim, no capítulo V, as considerações finais são apresentadas.

2 REFERENCIAL TEÓRICO

Neste capítulo será apresentada uma revisão bibliográfica das diversas tecnologias recentes e aplicações no processamento de linguagem natural, mais especificamente empregadas na extração de entidades em base de dados de *web commerce*. Desta forma, serão apresentados, os métodos *word embedding* tal como *GloVe* e *word2vec*, redes neurais, redes neurais recorrentes tal como as LSTM, o algoritmo para seleção de características *Fisher's Discriminant Ratio*, e o algoritmo de redução de dimensão *Principal Component Analysis*. Serão também apresentadas técnicas de aprendizado do tipo *Few-Shot Learning*, ou seja, técnicas de aprendizado onde pode-se utilizar bases de dados com poucas amostras, tal como o algoritmo KNN, redes *matching* e redes neurais de grafos tal como as redes DPGN.

2.1 Comércio Eletrônico

Devido aos avanços dos serviços de Internet e *e-business*, e também ao uso crescente e generalizado da internet houve um grande crescimento do comércio eletrônico e também na quantidade de produtos vendidos.

De acordo com a *Associação Brasileira de Comércio Eletrônico (ABCOMM)* em pesquisa realizada pela *Statista*, o número de consumidores digitais no Brasil em 2018 chegou em 73 milhões. O crescimento do número de clientes a cada ano vem repercutindo também no faturamento do setor, pois de acordo com levantamento feito pelo *Ebit / Nielsen*, o comércio eletrônico brasileiro cresceu 12% em 2018 e 22,7% em 2019. Além disso, em caso excepcional devido à pandemia do covid-19, o crescimento no ano de 2020 apresentou-se ainda mais acelerado, pois de acordo com a ABCOMM a alta foi de 68% em relação a 2019. Continuando nesta onda de crescimento, o comércio eletrônico brasileiro registrou faturamento de R$ 161 bilhões em 2021, sendo uma alta de 27% em relação a 2020, estes dados são da *Neotrust* (empresa que monitora 85% do e-commerce brasileiro). Este momento esta sendo bem aproveitado pelos *marketplaces* que serviram de plataforma de vendas para diversos vendedores pequenos que tiveram suas lojas físicas fechadas durante a pandemia. Essa modalidade já ocupa 78% de participação do comércio eletrônico B2C (*Business-to-Consumer*) segundo o *Ebit / Nielsen*.

O aumento da quantidade de produtos vendidos e do tamanho dos *marketplaces* gera uma grande quantidade de dados. Isto leva à necessidade de obter as melhores práticas para lidar com esta grande quantidade de dados gerados e transformar os textos não estruturados

em informação do produto para garantir sempre a melhor experiência para o cliente (DASHTIPOUR et al., 2019).

As lojas *on-line* dificilmente possuem informações de produtos de forma padronizada. Não há correspondência entre os nomes dos atributos como, marca, modelo, capacidade, cor, tensão, tipo, material e etc., e muitas vezes eles não estão presentes nas descrições dos produtos (KANNAN et al., 2011). Muitas vezes um mesmo produto tem a descrição das especificações de forma diferente por ser realizada por diferentes vendedores (*sellers*). Como os *martketplaces* vendem produtos de diferentes lojas, logo existe a necessidade da estruturação dos dados para obter uma melhor categorização e correspondência entre os produtos destas diferentes lojas, de forma a facilitar que o cliente obtenha o produto desejado (DASHTIPOUR et al., 2019).

Neste contexto, várias aplicações vem surgindo para lidar com esses milhões de produtos que por serem descritos como texto livre, gera um conjunto de dados não estruturados. Vários trabalhos tentam fazer a correspondência destes dados não estruturados com padrões estruturados, proporcionando um melhor atendimento às necessidades dos clientes, com relação à escolha das especificações correta dos produtos, tal como o recebimento de boas ofertas (KANNAN et al., 2011).

De acordo com Akritidis, Fevgas e Bozanis (2018), um dos problemas mais fundamentais é a classificação automática de produtos em uma hierarquia eficiente de categorias. A solução bem sucedida desse problema pode levar a inúmeras aplicações novas, tal como, busca ou consulta de produtos, sugestões de produtos relevantes, recomendações personalizadas e muitos mais.

Devido à sua importância, o problema da classificação automática de produtos em uma dada taxonomia, atraiu a atenção de múltiplos pesquisadores. De acordo Li, Kok e Tan (2018), as plataformas de comércio eletrônico categorizam seus produtos em uma árvore de taxonomia de vários níveis, com milhares de categorias de folhas. Esse trabalho utilizou uma nova abordagem, na qual é realizada a tradução da descrição do produto de um idioma natural para uma sequência de *tokens* para a identificação do produto, representando-o em um caminho da raiz para folha da árvore de taxonomia de produto.

Muitas são as aplicações desenvolvidas para o comércio eletrônico, dentro deste conceito de extração de características, principalmente para a busca semântica, categorização e correspondência de produtos. Tecnologias que vem sendo utilizadas nas plataformas de *marketplaces* tal como na *Amazon, Walmart, eBay* e etc., e também em sistemas conhecidos como

agregadores de produtos, tal como o *Google Product Search e Shopzila* (RISTOSKI et al., 2018; KRISHNAN; AMARTHALURI, 2019).

O trabalho de Ristoski et al. (2018) é uma abordagem que utiliza modelos de linguagem natural e técnicas de aprendizado profundo para a categorização e correspondência de produtos em diferentes *sites*. Os métodos para a extração de características utilizados foram *word2vec* e *paragraph2vec*, além de um modelo *deep learning* (CNN). Esta abordagem, além da integração de produtos, é capaz de fazer o processamento de consultas ou pesquisa, recomendações de produtos e recuperação de informações, o que melhora a experiência de compra dos usuários.

As aplicações de extração de características que utilizam aprendizado profundo tem apresentado resultados promissores em problemas com grandes massas de dados. Pois, de acordo Krishnan e Amarthaluri (2019), na presença de grandes quantidades de dados, os modelos de aprendizado profundo superam significativamente os modelos tradicionais de aprendizado de máquina. Krishnan e Amarthaluri (2019) desenvolveram um trabalho que utiliza dois tipos de redes profundas, as redes *Multi-LSTMs* e as *Multi-CNNs* que apresentaram desempenhos igualmente bons. Ambas foram utilizadas para realizar a extração de características para posterior categorização de produtos.

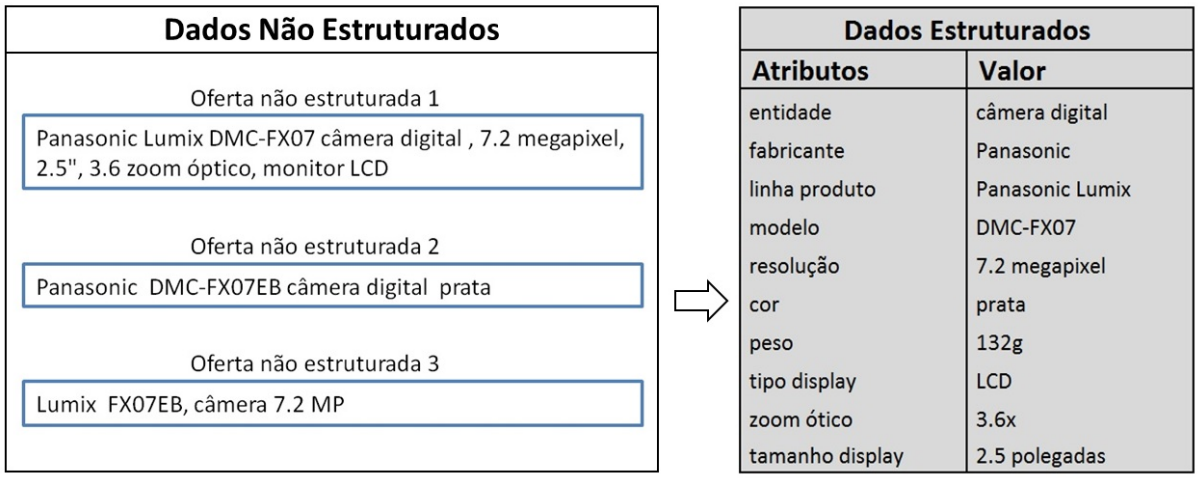
A extração de características de dados não estruturados e a transformação em dados estruturados é um tipo de aplicação que possibilita a implementação de várias outras aplicações automáticas que são bastante utilizadas hoje nas plataformas *on-line*. Neste caso, a aplicação proposta não é exclusividade de comércio eletrônico, mas pode ser facilmente estendida a outras áreas, tal como: para descrever pessoas, eventos, ofertas de emprego, hotéis, lugares, organizações, receitas, filmes, livros, música e muitos outros tipos de aplicações. Assim, é possível a criação de aplicações de catálogos de endereços, agências de viagens, mecanismos de pesquisa, sistemas de recomendação, e etc..

2.1.1 *Marketplaces* e Transformação de Ofertas

O comércio eletrônico das redes de varejo de grande porte eram formados na maioria das vezes somente por uma loja (ou um *seller*), que era o próprio varejista. Porém, muitas dessas plataformas se transformaram em *marketplaces*, onde diversas lojas ou vendedores (*sellers*) podem oferecer seus produtos. Um exemplo que pode-se destacar é a Magazine Luíza que possuía somente a sua loja eletrônica própria e se transformou em um dos maiores *marketplaces* do Brasil com diversos vendedores.

Quando o comércio eletrônico continha somente um *seller*, a correção dos dados dos produtos era realizada manualmente. Porém, com o crescimento da loja digital e também com a formação do *marketplace* fica inviável a correção manual dos dados dos produtos. A quantidade de vendedores é grande (cada um com uma forma diferente de descrever o produto) e, consequentemente, a quantidade de produtos fica muito grande inviabilizando a correção manual de cada produto do *marketplace*. Portanto, a transformação de dados não estruturados em dados estruturados se tornou possível com aplicação de ferramentas automatizadas de aprendizado de máquina aplicadas à resolução de tarefas de processamento de linguagem natural (Seção 2.2). Na Figura 2.1 pode ser visto um exemplo de oferta não estruturada de comércio eletrônico transformada em oferta estruturada. Por meios da extração dos atributos é possível transformar textos não estruturados em textos estruturados possibilitando a padronização das ofertas.

Figura 2.1 – Transformação de ofertas não estruturadas em estruturadas



Fonte: Do Autor (2022)

2.1.2 Extração de Entidades e Atributos em *Marketplaces*

Nesta abordagem serão seguidas algumas definições utilizadas em projetos de extração de entidades e atributos em comércio eletrônico. As entidades representam um tipo de produto, cada entidade pode agrupar um conjunto de atributos que podem ser compartilhados em um mesmo tipo de produto. As entidades ou classes de produtos podem ser uma televisão, celulares, cama, sofá, aparelho de som, computador e etc. Os atributos representam uma característica de determinado tipo de produto, eles diferenciam um produto de outro de uma mesma entidade. Por exemplo, celulares existem de diversas marcas e modelos, do qual cada celular tem seus atributos que o diferencia de outros modelos. Exemplo de atributos: marca, modelo, capacidade,

cor, tensão, linha de produto, tipo, potência, material, recursos, e etc. Os atributos costumam também ser chamados de metadados.

A extração correta das entidades e atributos contribui para a construção de filtros de produtos. O processo de categorização de *marketplaces* é implementado a partir da escolha dos atributos dos produtos, que melhor descreva o nicho dos produtos que se deseja encontrar. A estruturação dos dados também contribui para eficiência da busca semântica, a partir da possibilidade de restrição de retornos de busca que facilita encontrar o produto desejado.

A partir da estruturação dos dados (entidades e atributos) o sistema pode também categorizar os produtos do *marketplace* de forma independente da categoria originalmente atribuída pelo vendedor (*seller*). Essa estruturação também facilita a realização do *matching* que é agrupamento de ofertas de um mesmo produto. A estruturação dos atributos possibilita a padronização dos títulos dos produtos. Cada vendedor digita o título de uma forma, o sistema fica neste caso encarregado da estruturação do título. Na Figura 2.2 pode-se ver um exemplo de um título estruturado. O título pode conter em sua estrutura atributos tal como: entidade, marca, modelo, capacidade, linha de produto, tipo, recursos e etc., de acordo com o padrão do *marketplace*.

Figura 2.2 – Título do produto

Fonte: Site da Amazon

2.2 Processamento de Linguagem Natural

Com o aumento dos serviços de internet, ocorreu um uso extensivo de plataformas digitais, tais como mídias sociais, plataformas de comércio eletrônico, *streamings* e *fintechs*, etc. Todos estes serviços geram uma enorme quantidade de dados, do qual grande parte destes dados são multimodais (vídeos, imagens, textos, números) e não estruturados (KUMAR et al., 2020).

Com isso, ocorreu o aumento da geração de dados em escala exponencial, sendo necessário o tratamento destes dos dados, seja do ponto de vista operacional e também econômico. Como muito destes dados são linguísticos, neste caso o tratamento se faz com de uso das tecnologias de Processamento de Linguagem Natural (PLN), um campo da Inteligência Artificial que vem sendo bastante estudado.

O PLN é uma série de técnicas computacionais que tem por objetivo fazer a análise e representação automática da linguagem humana. Trata-se de uma área de pesquisa que estuda como os computadores podem compreender e manipular a linguagem humana, seja para a compreensão da fala ou de textos escritos em linguagem natural a fim de automatizar processos e realizar tarefas importantes (GOMES; EVSUKOFF, 2017; YOUNG et al., 2018). De acordo com Gomes e Evsukoff (2017), os algoritmos de PLN buscam gerar representações matemáticas significativas para elementos textuais, treinados a partir de conjuntos de dados representativos do problema a ser processado.

Várias áreas vem sendo impactadas com aplicações de PLN tal como a área de *marketing*, vendas, industrial e outras. Dentre as principais tecnologias presentes nas aplicações de PLN pode-se citar: assistentes virtuais (SARIKAYA, 2017), categorização e *matching* de produtos a partir de extração da informação (KRISHNAN; AMARTHALURI, 2019; KANNAN et al., 2011), análise de sentimentos (KARTHIKAYINI; SRINATH, 2018), tradutores (LI; KOK; TAN, 2018), buscadores (YOUNG et al., 2018), corretor de texto (GHOSH; KRISTENSSON, 2017), a aplicação com robôs em linhas de montagem industrial (STENMARK; NUGUES, 2013) e na geração de comportamentos e movimentos de robôs humanoides com base na entrada de linguagem natural (WÄCHTER et al., 2018).

O PLN também tem um papel importante na digitalização da indústria (indústria 4.0, ou internet industrial, e outras nomenclaturas) (XU; HUA, 2017). Por exemplo, na gestão da produção industrial é comum a utilização de indicadores de performance, que é um meio importante para as organizações identificarem oportunidades para melhorarem suas operações. Os indicadores por sua vez são formados por dados em linguagem natural não estruturada, e de-

pende de um esforço manual considerável para obter informações que possam ser monitoradas. Para a digitalização do processo é utilizada a transformação de descrições de indicadores de performance do processo (PPI), de linguagem natural não estruturada para uma notação estruturada, que deve ser alinhada com a implementação subjacente do processo de negócios. Portanto, esta é uma alternativa automatizada que elimina tempo e esforço considerável investido para transformar dados linguísticos em indicadores que possam realmente ser monitorados (AA et al., 2017).

O PLN vem sendo bastante utilizado também na análise de opinião ou de sentimentos. A análise de sentimentos é um sistema computacional automatizado de entendimento e classificação de informações subjetivas de materiais tal como, comentários em sites de comércio eletrônico, postagens e comentários em plataformas de mídia social. A tarefa fundamental na análise de sentimentos é atribuir polaridade (positiva ou negativa) a um determinado texto (DASHTIPOUR et al., 2019).

Como a geração de dados tem aumentado em escala exponencial, neste caso a análise de sentimentos tornou-se necessária, principalmente a análise de opinião de clientes a respeito de produtos, a análise de mercados, identificação do gosto do público para a geração de campanhas *marketing* e propaganda exclusivas para determinado grupo de pessoas, opinião de eleitores e em sistema sistemas de recomendação (QIN et al., 2019; DASHTIPOUR et al., 2019).

A análise de opinião pode ser empregada em comentários sobre produtos em *marketplaces* da seguinte forma: se alguém quiser comprar um *laptop*, o comprador pode consultar os comentários *on-line* para entender as limitações do produto tal como duração da bateria, portabilidade, analisando a qualidade e usabilidade do produto. Além disso, entender os problemas de usabilidade enfrentados pelos clientes de *laptops* possibilita ao fabricante uma oportunidade de melhorar ainda mais o seu produto. Os dados contêm informações valiosas que podem ser exploradas pelos compradores para escolher o produto e também pelos vendedores para levar em consideração as necessidades e desejos dos clientes, e gerar recomendações automáticas de produtos (DASHTIPOUR et al., 2019).

A categorização e a especificação correta e uniforme dos produtos das várias lojas contidas nos *marketplaces* contribuem para uma melhor experiência de compra para o cliente. A categorização correta e a padronização das ofertas com suas respectivas especificações é possível por meio da utilização das técnicas de extração de entidades e atributos do produto (KRISHNAN; AMARTHALURI, 2019). A aplicação destas técnicas possibilita o cliente encontrar mais

facilmente o que deseja, e a satisfação pela realização de uma boa compra é repercutida com comentários positivos a respeito do produto.

O avanço ocorrido nos últimos anos na área de PLN aconteceu devido tecnologias de aprendizado de máquina, mas a disrupção efetuou-se com as novas tecnologias proveniente do aprendizado de máquina profundo (*Deep Learning*). Existem diversas abordagens de aprendizado do tipo *deep learning*, mas as que obtiveram maior sucesso em aplicações de PLN são as baseadas em Redes Neurais Convolucionais (CNNs) e principalmente Redes Neurais Recorrentes (RNNs) (GOULARAS; KAMIS, 2019; PHAM; LE, 2018).

Com relação ao futuro do PLN, experimentos mostram que há uma grande margem para melhorias em relação ao desempenho ao adotar arquiteturas paralelas. E também pode-se esperar resultados ainda melhores com o uso de grandes *clusters*, computação em nuvem e *Big Data*. Essas melhorias nas arquiteturas de computação e processamento de dados em larga escala possibilitam o treinamento das redes *deep learning* que estão cada vez maiores e mais sofisticadas, consequentemente gerando resultados melhores (AGERRI et al., 2015). E por outro lado, para casos em que não se disponha de grande quantidade de dados, a tendência é a utilização de técnicas *few-shot learning* (BENDRE; MARÍN; NAJAFIRAD, 2020). Estas técnicas não dependem da quantidade de características, mas da qualidade das mesmas.

2.2.1 Métodos *Word Embeddings*

Os *word embeddings* são um conjunto de técnicas utilizadas para mapear as características das palavras para números reais em vetores densos de pequena dimensão. Estes são capazes de capturar informações contextuais semântica e sintática de conjuntos de dados grandes (YU et al., 2018). Os *word embeddings* trabalham com o conceito de que palavras com significados semelhantes tendem a ocorrer em contextos semelhantes. Estes capturam as similaridades entre as palavras, dos quais as com características semelhantes são representadas em vetores na mesma vizinhança. A medida de similaridade pode ser realizada utilizando métricas de distâncias tal como a do cosseno (YOUNG et al., 2018).

Os *word embeddings* são muito bons em prever uma palavra com base em seu contexto, analogia, devido à captura de similaridades. As palavras similares são projetadas próximas no espaço semântico, por exemplo, as palavras “Brasil” e “Brasília” são mapeadas em vetores próximos. Uma característica interessante dos *embeddings* é que devido à pequena dimensio-

nalidade, são rápidos e eficientes na computação das principais tarefas de PLN (YOUNG et al., 2018).

Word embedding faz o mapeamento da sintática e semântica da linguagem natural utilizando técnicas de aprendizado de máquina e estatísticas (GOMES; EVSUKOFF, 2017). A partir dos vetores de palavras mapeadas são extraídas as características que podem ser utilizadas para as mais diversas aplicações de PLN, tais como, tradução, recuperação de informação, classificação de produtos, assistentes virtuais e análise de sentimentos, e etc.

Dentre os métodos *embeddings* mais utilizados em PLN, tem-se o *word2vec*, os vetores globais (*GloVe*), que são mais indicados para extrair características das palavras.

2.2.2 *Word2Vec*

O *word2vec* é considerado um dos algoritmos mais bem estabelecidos na literatura. Ele é do tipo *word embeddings*, sendo capaz de extrair conhecimento semântico dos textos, ao invés de apenas realizar a análise sintática. Apresenta capacidade de realizar a composicionalidade que ocorre da seguinte forma, ao adicionar dois vetores de palavras, resulta em um vetor que é uma composição semântica de palavras individuais, por exemplo, "*homem*" + "*real*" = "*rei*" (YOUNG et al., 2018; MIKOLOV et al., 2013).

De acordo Mikolov et al. (2013) os modelos do *word2vec* recebem grandes conjuntos de dados textuais (*corpus*) e produzem um espaço vetorial com muitas dimensões, tal que cada palavra do *corpus* é representada por um vetor no espaço. As palavras que compartilhem a mesma semântica são posicionadas próximas. Devido à esta representação no espaço vetorial, o *word2vec* tem a característica de mapear palavras que apresentam similaridade em posições próximas dos vetores, ou seja, possuem valores próximos de caracterização (GOMES; EVSUKOFF, 2017). Portanto, essas relações de similaridade podem ser inferidas a partir do cálculo de similaridade entre seus vetores utilizando métricas de distância tal como a do cosseno:

$$cos(\Theta) = \frac{A.B}{\|A\| \, \|B\|} \tag{2.1}$$

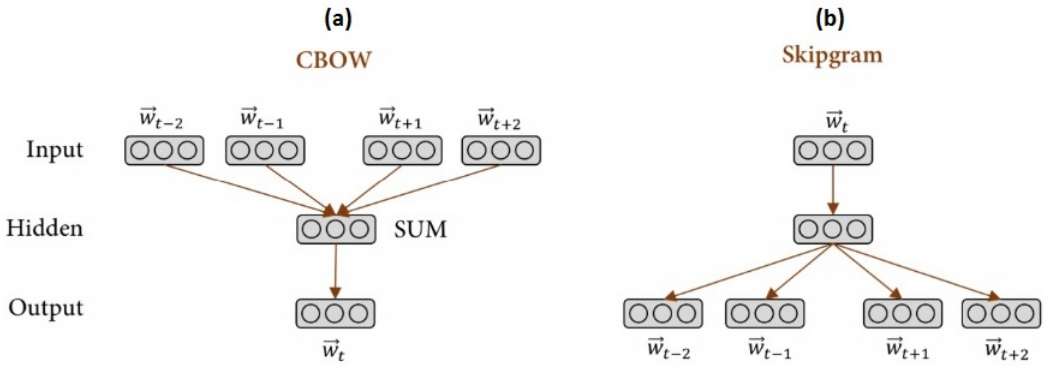
A distância do cosseno realiza a métrica de similaridade entre dois termos considerando o ângulo entre seus vetores. Considerando duas sentenças A e B, a similaridade do cosseno é dada pelo produto escalar destas duas sentenças, dividido pela magnitude do produto das duas sentenças. Um ângulo menor do que 90 graus entre os vetores indica uma maior similaridade entre as sentenças (cos=1). Um ângulo superior a 90 graus significa que os vetores são orto-

gonais entre si (cos = -1). Isso mostra que as sentenças não estão relacionadas (QURASHI; HOLMES; JOHNSON, 2020).

A arquitetura do *word2vec* é similar ao *Feed Forward Neural Net Language Model* (NNLM) proposto por Bengio, Ducharme e Vincent (2000), que consiste de uma rede neural com camada de entrada, projeção, oculta e saída. Esta arquitetura apresenta bons resultados para grandes bases de dados, porém o problema é que ela se torna complexa com a computação entre a camada de projeção e a camada oculta, e também com a *softmax* na camada de saída. Usar esta arquitetura para obtenção de representação de palavras é ineficiente do ponto de vista de custo computacional. Neste sentido, Mikolov et al. (2013) propôs o *word2vec* que é uma rede neural simples que possui somente a camada de entrada, projeção e saída (e também não existe função de ativação presente), ou seja a camada oculta não linear foi eliminada e a camada de projeção foi compartilhada com todas as palavras da janela de contexto. Sendo assim, o *word2vec* é eficiente em termos computacionais devido ao seu modelo ser baseado em redes neurais simples, possui a capacidade de ser treinado rapidamente e produz resultados com uma boa acurácia. Isto o faz eficiente para treinamentos em grandes conjuntos de dados textuais.

O *word2vec* é composto por duas arquiteturas *continuous bag-of-words (CBOW)* e *skip-gram*. Na Figura 2.3 pode ser visto o esboço das arquiteturas. Para o entendimento da Figura 2.3, será citado um exemplo sobre palavras de contexto e a palavra central ou alvo. Considerando a frase: "achei **muito** *interessante* **este** conteúdo". A palavra em itálico "*interessante*" representa a palavra central e as palavras em negrito do lado representam uma janela de contexto de tamanho k=2.

Figura 2.3 – Arquitetura CBOW e Skip-gram



Fonte: Camacho-Collados e Pilehvar (2018)

O *CBOW* faz a previsão da palavra central $w(t)$, a partir do conjunto de palavras dentro de uma janela de contexto de tamanho k. O modelo *CBOW* pode ser visto na Figura 2.3 (a), a

camada de entrada receberá as palavras de contexto ($w(t-2), w(t-1), w(t+1), w(t+2)$), e a saída terá uma palavra que representa a maior probabilidade para a palavra alvo (palavra central $w(t)$). Na camada de projeção tem a função *sum* que é a soma da representação numérica das palavras da janela de contexto. Os dados de entrada da rede são pré-processados normalmente para o tipo *one-hot-encoded* (YOUNG et al., 2018; GOMES; EVSUKOFF, 2017).

O *skip-gram* faz o oposto do *CBOW*, prevê as palavras de contexto dada uma palavra de origem $w(t)$. Conforme pode ser visto na Figura 2.3 (b), na camada de entrada, tem-se apenas uma entrada para a palavra alvo, e em sua saída temos diferentes probabilidades, para cada uma das palavras de contexto possíveis (YOUNG et al., 2018; GOMES; EVSUKOFF, 2017).

Presume-se que as palavras de contexto estejam localizadas simetricamente às palavras de destino com uma distância igual ao tamanho da janela em ambas as direções. Em configurações não supervisionadas, a dimensão do vetor formado pelo *word2vec* é determinado pela precisão da predição (YOUNG et al., 2018). À medida que os vetores de representação das características aumentam de dimensão, a precisão da predição também aumenta até convergir em um mesmo ponto, do qual é considerada a dimensão ideal, sendo a menor dimensão sem comprometer a precisão.

Não se pode dizer que um modelo é melhor do que o outro, isto vai depender do problema que se deseja resolver. O *skip-gram* funciona bem para uma quantidade de dados menor do que necessita o *CBOW*, e tem a capacidade de representar melhor palavras ou frases raras. O modelo *CBOW* consegue representar melhor as palavras mais frequentes e possui a vantagem de treinamento rápido.

2.2.3 *Global Vectors* - GloVe

O *Global Vectors (GloVe)* proposto por Pennington, Socher e D. Manning (2014) é um algoritmo não-supervisionado para obtenção de representações vetoriais de palavras, baseado em estatísticas extraídas de co-ocorrência de palavras. Este é um modelo global de regressão *log-bilinear* descrito como uma combinação dos pontos fortes dos métodos *global matrix factorization* e *local context window*. Estas são as duas principais famílias de modelos de aprendizagem de vetores de palavras (SEOK et al., 2016).

O *GloVe* treina seus vetores baseados em uma matriz de co-ocorrência de palavras. Este indica a frequência com que as palavras do *corpus*[2] são correlacionadas entre si, ou seja, a

[2] *Corpus* são grandes conjuntos de dados textuais.

frequência com que as palavras aparecem em um mesmo contexto (PENNINGTON; SOCHER; D. Manning, 2014). Os linguistas acreditam que palavras que ocorrem em contextos semelhantes tendem a ter significados semelhantes. Com base nestas hipóteses, foram introduzidos conceitos de medidas de associação de palavras e contextos, tal como as frequências de co-ocorrência de palavras (CAO et al., 2019).

O *Local Context Window* conceito utilizado no *skip-gram*, aprende a representação numérica usando palavras adjacentes sendo bom para a tarefa de analogia de palavras devido às estatísticas locais. O contexto local reflete as características semânticas e sintáticas locais de uma palavra. A limitação é que as operações aplicadas a nível local utilizam mal as estatísticas do corpus, pois não possuem uma matriz de co-ocorrência do *corpus* do texto global, dessa forma não conseguem tirar proveito da grande quantidade de repetição nos dados (MENG et al., 2020). Considerando o *skip-gram* o contexto da predição é somente local devido a saída obtida ser formada somente por palavras dentro da janela de contexto em torno da palavra central $w(t)$.

Os métodos que utilizam o *Global Matrix Factorization* aproveitam com eficiência as informações estatísticas de co-ocorrência de palavras. Porém, a sua utilização tem como desvantagem um baixo desempenho na tarefa de analogia de palavras, uma vez que somente a análise das palavras mais frequentes não é suficiente para calcular o grau de similaridade de forma satisfatória. Um modelo que usa o conceito, é o *Latent Semantic Analysis* (LSA), que é bom para resolver problemas de sinônimos combinando variáveis correlatas (PENNINGTON; SOCHER; D. Manning, 2014).

O *GloVe* combina os benefícios dos modelos preditivos baseados em janelas de contextos, com os benefícios de examinar informações de estatísticas globais, proveniente dos métodos de fatoração de matriz tal como o LSA. O *GloVe*, com essa junção de estratégias, se torna um método atrativo para a representação das palavras (QUISPE; TOHALINO; AMANCIO, 2021). Portanto, em relação ao *word2vec*, o *GloVe* apresenta como vantagem que este não depende apenas de estatísticas locais, mas incorpora estatísticas globais (coocorrência de palavras) para obter vetores de palavras (CHAWLA et al., 2019).

2.2.4 Formação de *Embeddings*

A formação de *word embeddings* ou a extração de características segue as etapas de pré-processamento de dados, *tokenização*, *stemming* e extração de características. Na etapa de pré-processamento é realizada a preparação da base para a formação dos *word embeddings* com

as informações que possuem valor para a aplicação. Se a base possuir informações indesejadas para a extração de características, estas são removidas. A preparação dos dados envolve diversas operações de acordo com a necessidade de cada aplicação e de acordo com o conteúdo da base. Por exemplo, se for realizar análise de sentimentos com base do *Twitter*, é necessária a remoção dos dados ou caracteres indesejados presentes no texto. Também são removidas as palavras com menos de três caracteres, tal como, conjunções, preposições, além da remoção de *stop words* (palavras irrelevantes para o processamento).

O pré-processamento das bases de dados de comércio eletrônico deve ser mais cuidadoso com relação à remoção de dados indesejados. O texto não estruturado é formado pelas ofertas, que contém as informações dos produtos, que são formados por palavras e códigos com letras e/ou números que são importantes para especificação do produto e portanto são dados que não podem ser descartados devido possuir valor para a recuperação de informação.

Na etapa de pré-processamento é bastante comum a utilização da *tokenização*, que consiste em separar o texto em *strings* de palavras isoladas. Após realização do pré-processamento podem ser utilizados os métodos *embeddings* de PLN para a extração de características. Os métodos do tipo *word embeddings*, tal como o *word2vec* e *GloVe*, podem então ser utilizados para extração de características de palavras. Os métodos do tipo *deep learning* também podem ser utilizados para a extração de características a partir de caracteres e palavras, tal como as redes LSTM. Estas redes são capazes de aprender a realizar o processamento da linguagem natural lendo caractere a caractere da base de dados até conseguir o entendimento por completo das palavras e do contexto.

As características são dados numéricos que representam a sintática e semântica do texto. No caso da aplicação para comércio eletrônico, como a extração de atributos, os dados numéricos das características contém informações que representam o significado dos atributos das ofertas, tais como: entidades, marca, modelo, capacidade, linha de produto, tipo, material, e etc.. A tarefa de extração de entidades e outros atributos normalmente é realizada com a aplicação de classificadores nos *embeddings* que por sua vez representam as características.

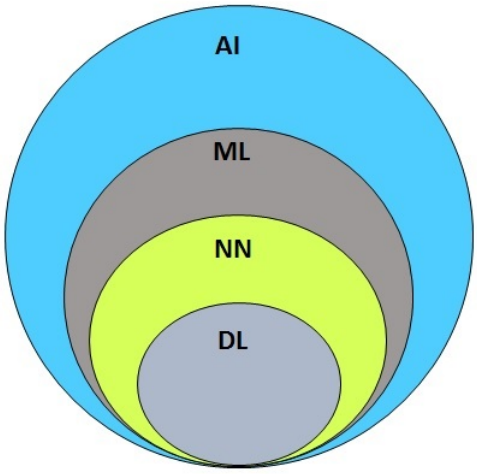
2.3 Aprendizado de Máquina

A ideia do aprendizado de máquina (*machine learning*) é que o computador aprende a executar uma tarefa baseado no conjunto de dados utilizados para treinamento. Quando surge novos dados ele executa a tarefa da mesma forma que aprendeu antes (LOURIDAS; EBERT,

2016). Pode se dizer que a máquina aprende através da experiência, se o seu desempenho mensurável nestas tarefas aumenta à medida que a experiência aumenta na execução das tarefas. Os algoritmos de aprendizado de máquina, reproduz padrões, faz previsões e toma decisões baseada em dados. O desempenho do algoritmo é medido através das previsões e detecções corretas do problema, validadas por um profissional experiente da área analisada (RAY, 2019).

Aprendizado de máquina é um subcampo da inteligência artificial, conforme pode-se ver na Figura 2.4. O aprendizado profundo (*deep learning*) é um subcampo de aprendizado de máquina. O aprendizado profundo foi gerado a partir das redes neurais artificiais que é outro subcampo de aprendizado de máquina. O aprendizado profundo surge da necessidade de treinar redes neurais com uma grande quantidade de camadas ocultas e foi desenvolvido em grande parte a partir do ano de 2006 (ALOM et al., 2018).

Figura 2.4 – Taxonomia da área de aprendizado de máquina, do qual: AI é inteligência artificial, ML é *machine learning*, NN é rede neural, DL é *deep learning*.:



Fonte: Do Autor (2022)

A área aprendizado de máquina não é recente, mas o que mudou com relação ao crescente uso nos últimos anos foi o aumento na capacidade de computação. Como consequência ocorreu também o aumento da geração de dados, que puderam ser capturados e armazenados, os dados atualmente são coletados com uma melhor qualidade, permitindo que o aprendizado de máquina resolva problemas cada vez mais complexos (LOURIDAS; EBERT, 2016)(ONG; GUPTA, 2019). Portanto, quanto maior o número de amostras e mais diversificado os dados, melhor será o resultado do algoritmo.

Uma vantagem do aprendizado de máquina é que o algoritmo é capaz de analisar e aprender rapidamente tarefas que os seres humanos demorariam muito tempo. Podendo ser

iterativos, ou seja, caso seja exposto a novos dados, o modelo é capaz de se adaptar de forma independente. Os campos em que pode ser usado são: reconhecimento facial, reconhecimento de padrões, classificação, processamento de textos e fala, processamento de imagens, automação, entre outros (NAYAK; DUTTA, 2018).

Diferentes métodos tem sido propostos em diferentes categorias de aprendizado, incluindo aprendizado supervisionado, semi-supervisionado e não-supervisionado. Além disso, há outra categoria de aprendizado chamada, aprendizagem por reforço (RL) ou *Deep RL (DRL)*, que frequentemente é abordada no escopo de semi-supervisionado e algumas no escopo de não-supervisionado (AREL; ROSE; KARNOWSKI, 2010).

O aprendizado supervisionado se divide em classificação e regressão. A classificação divide os dados em categoria. A regressão ao invés de categorizar faz a previsão de números. No aprendizado supervisionado, o conjunto de dados do qual se pretende extrair conhecimento já é rotulado, ou seja, os dados possuem as respostas com verdadeiros valores do sistema. Estes são utilizados para estimar o desempenho do aprendizado, comparando-se a saída do sistema com o verdadeiro valor. No aprendizado não-supervisionado, o conjunto de dados não são rotulados, o algoritmo tem a tarefa de identificar as características semelhantes e agrupar estes dados formando *clusters*.

O aprendizado por reforço é usado nos casos em que o problema não está relacionado a dados, mas sim com a interação com ambiente tal qual uma cidade para carros autônomos. A ideia é sobrevivência neste ambiente, como exemplo um robô que aprende por punições e recompensa.

Dentre os inúmeros algoritmos de aprendizado de máquina conhecidos pode-se citar: Gradiente Descendente, Regressão Logística, *Support Vector Machine*, *K-Nearest Neighbor*, Redes Neurais Artificiais, Árvore de Decisão, Aprendizagem Bayesiana, *Naive Bayes*, etc.

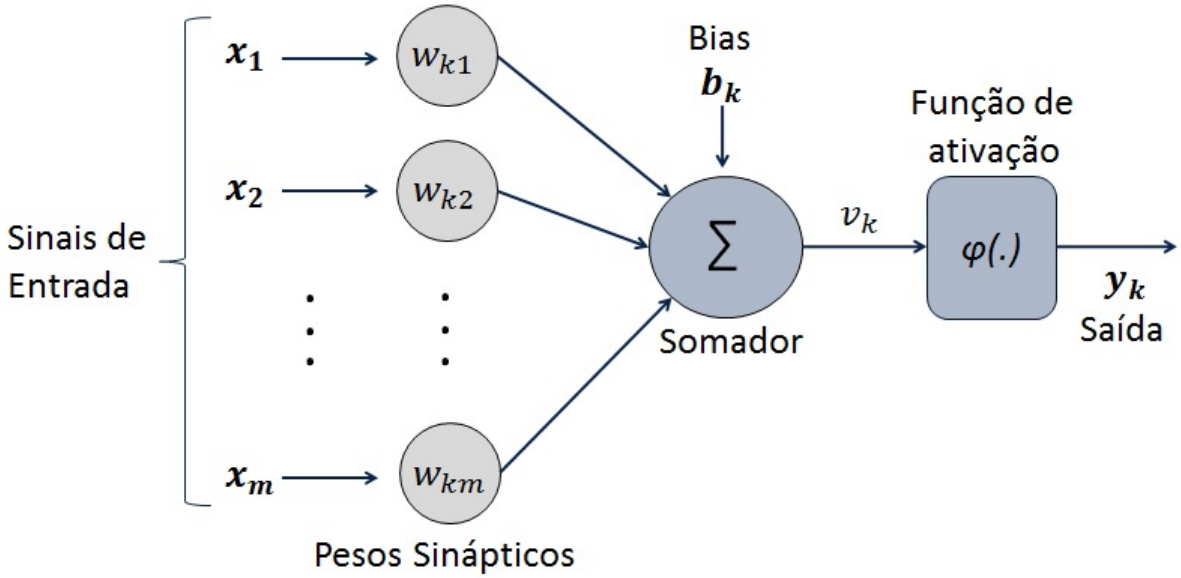
2.4 Redes Neurais Artificiais

As redes neurais artificiais (RNA) tiveram início na década de 40, pelo neurofisiologista *MacCulloch*, do Instituto de Tecnologia de *Massachussetts (MIT)*, e pelo matemático *Waltter Pitts*, da Universidade de *Illinois* (BURGHARDT; GARBE, 2018). O trabalho desenvolvido consistia em um modelo de resistores variáveis e amplificadores representando conexões sinápticas de um neurônio biológico. Desde então foram desenvolvidas diversas técnicas de RNA, até

chegar nas arquiteturas atuais desenvolvidas por sistemas de computador, estas que ganharam muito espaço com desenvolvimento dos recursos computacionais recentes.

Pode-se considerar que as redes neurais são algoritmos de computador que emulam a forma como cérebro resolve problemas. O cérebro é uma grande rede altamente interconectada de neurônios trabalhando em paralelo. Da mesma forma, as redes neurais possuem processamento paralelamente distribuído que é constituído de unidades de processamento simples chamados de neurônios, que tem como característica a capacidade de aprendizado. As conexões entre neurônios são utilizadas para armazenar o conhecimento adquirido, sendo estas chamadas de pesos sinápticos. Na Figura 2.5, é apresentado um modelo de um neurônio artificial simples.

Figura 2.5 – Neurônio artificial



Fonte: Do Autor (2022)

A estrutura de um neurônio é composta pelos sinais de entrada, os pesos sinápticos, uma função de ativação, um somador e uma saída. Sua tarefa é agregar todos os dados de entrada, executar uma função de ativação neles e enviar o resultado para a saída. Isto ocorre através de três operações básicas. O primeiro elemento básico é a sinapse, representada pelos pesos sinápticos, que são valores ajustados durante o processo de aprendizagem. A segunda operação é formada pelo somador que atua como combinador linear realizando a soma das entradas. E por último a função de ativação define se um neurônio está ativo ou não. As funções de ativação mais utilizadas são a sigmoid, tangente hiperbólica e relu (SINGH; SACHAN, 2015).

A saída do neurônio é o produto escalar dos pesos pelo vetor de entradas adicionado ao *bias*, conforme a equação (2.2):

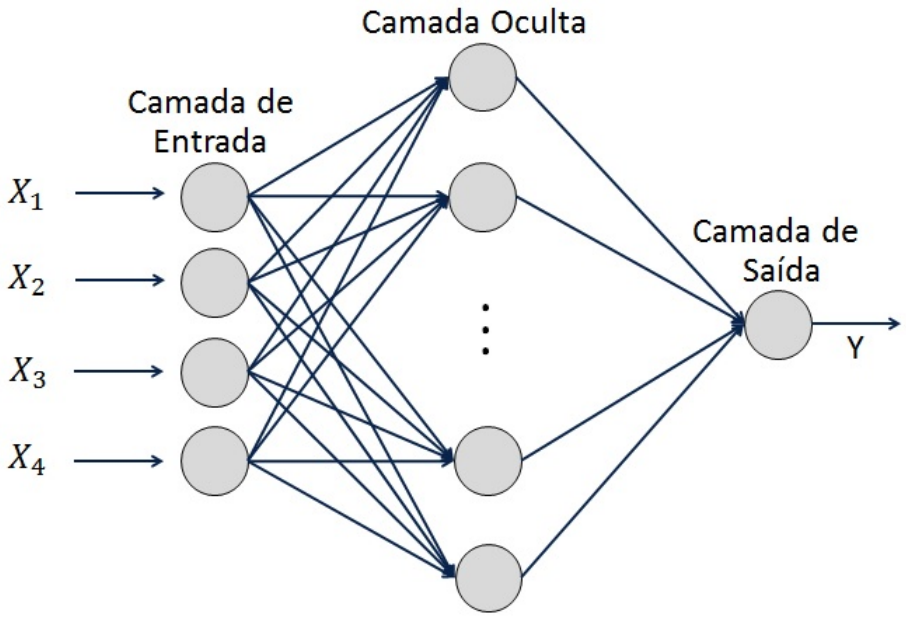
$$z = \mathbf{w}.\mathbf{x} + b \tag{2.2}$$

Sendo que **w** representa vetor de pesos da rede, b o bias e **x** é o vetor de entrada da rede. A saída da função de ativação ou do neurônio é dada para pela equação (2.3):

$$A = g(z) \tag{2.3}$$

Na Figura 2.6 é apresentado um modelo de uma rede neural artificial. A estrutura de um RNA é composta por 3 partes principais: a camada de entrada, a camada escondida ou oculta e a camada de saída. A camada de entrada é designada para receber os dados, estes são encaminhados para as camadas ocultas que tem o papel de extrair as informações. A camada de saída é onde o resultado final é concluído e apresentado. Ao contrário das camadas de entrada e saída a camada oculta não pode ser acessada diretamente. A camada oculta consiste em vários neurônios interconectados entre si e ou entre camadas. A rede que possuiu um camada oculta ou mais camadas ocultas conectadas são chamadas de Redes *Multilayer Perceptron* (MLP) (WIDIASARI; NUGROHO; WIDYAWAN, 2018).

Figura 2.6 – Redes neurais artificiais



Fonte: Do Autor (2022)

As redes neurais artificiais aprendem o modelo do ambiente em que estão inseridas. Este ambiente é representado por uma base de dados que é apresentado a ela. O treinamento

é realizado estimando os parâmetros de pesos do modelo que minimize o erro entre a saída do sistema e a resposta desejada. Este é realizado por um algoritmo de otimização, tal como, gradiente descendente (RAY, 2019). Este processo acontece da seguinte forma: quando os dados são apresentados à rede e as informações são propagadas camada por camada até a saída, tem-se a chamada propagação para frente. A saída obtida pela rede é comparada com o alvo, desta forma o erro é computado. Este erro é então propagado de volta pela rede com a intenção de atualizar os pesos de cada neurônio para que a saída resultante obtida seja mais próxima dos valores da saída esperada. Esta parte do treinamento é chamado de *back propagation*, sendo que, a atualização dos pesos acontece através do cálculo de derivadas parciais em função de cada parâmetro da rede, até que se chegue no desempenho desejado (SONAWANE; PATIL, 2015).

De acordo com Oliveira, Venson e Marcelino (2018) uma rede neural pode ser classificada em duas categorias de acordo com o tipo de interconexão:

- *Feedforward*: o fluxo da rede é apenas da entrada para a saída. Não há conexão entre os neurônios de uma mesma camada. Geralmente é utilizada quando é necessário um nível de modularidade na representação.

- Recorrente: as redes recorrentes apresentam conexões entre dois nós de modo que formem um ciclo dentro da rede. Este tipo de arquitetura é bastante utilizado em aplicações de reconhecimento e PLN.

As arquiteturas de redes neurais mais populares são: Redes *Multilayer Perceptron* (MLP), *Convolutional Neural Network* (CNN), *Recurrent Neural Networks* (RNN), *Autoencoders*, e etc. De acordo com Paterakis et al. (2017), *Multilayer Perceptron (MLP)* são modelos clássicos de redes neurais do tipo *feed forward* que mapeia um conjunto de dados de entrada para um conjunto correspondente de dados de saída, dos quais são utilizados para aprendizado supervisionado. Por ser uma rede do tipo *feed forward* o fluxo de informações nas MLPs é unidirecional, iniciando da camada de entrada em direção à camada de saída. As conexões existem apenas entre camadas consecutivas.

De acordo com Louridas e Ebert (2016), as MLPs são aproximadores universais de funções. Elas podem ser usadas em todo o espectro do aprendizado de máquina: classificação, regressão, agrupamento e redução de dimensionalidade. Porém não funcionam de forma eficiente em problemas de classificação de imagens quando a própria é utilizada como entrada da

rede. Devido as imagens terem grande dimensão de *pixels* (uma imagem de 230 x 230 *pixels* possui 52.900 dimensões.), se aplicada na entrada da rede MLP totalmente conectada resulta em um crescimento exponencial dos pesos da rede, dificultando o treinamento e gerando necessidade de uma grande quantidade de processamento. Desta forma as redes CNNs são mais indicadas para trabalhar com imagens devido possuir arquitetura mais sofisticada para extração de características de dados com muitas dimensões.

A teoria de redes neurais artificiais já existem a mais de meio século, mas as inovações nas arquiteturas e a disponibilidade de recursos de computação, possibilitaram a implementação de RNAs com grande número de camadas ocultas, que passaram a ser nomeadas de redes de aprendizado profundo. Sendo portanto, capazes de aprender a jogar jogos e derrotar humanos, grande capacidade de reconhecer imagens, executar a tradução automática e assim por diante (LOURIDAS; EBERT, 2016). Detalhes sobre a aprendizagem profunda pode ser visto a seguir na seção 2.5.

2.5 Aprendizado Profundo

O aprendizado profundo (*deep learning*) são algoritmos de computadores bio-inspiradas no cérebro humano que por sua vez apresenta uma arquitetura profunda. O cérebro humano organiza as informações de maneira hierárquica, primeiro aprende conceitos mais simples e depois os combina para representar idéias mais abstratas (FADLULLAH et al., 2017). Neste sentido, o aprendizado profundo é formado por redes neurais com inúmeras camadas ocultas, desse modo vem a idéia de profundidade da rede.

Segundo Alom et al. (2018), o aprendizado é um procedimento que consiste em estimar os parâmetros de modelo para que seja gerado um algoritmo de aprendizado que possa executar uma tarefa específica. Em aprendizado profundo estes parâmetros estão contidos nas diversas camadas entre a camada de entrada e a camada de saída, de tal forma que permite a presença de vários estágios de unidades de processamento de informações não lineares com arquiteturas hierárquicas.

O aprendizado profundo se baseia em duas concepções: experiência e a capacidade de enxergar o mundo como uma hierarquia de conceitos, tal que cada conceito é constituído de conceitos mais simples. Desta forma o computador poderá aprender conceitos complicados, construindo-os de forma simples. Essa construção de conhecimento se dá por meio de várias camadas de representações, tal que cada camada do modelo descreve uma parte do conhecimento.

Partindo da idéia de que muitos sinais naturais são hierarquias de composição, é possível representar recursos de nível superior por meio da composição de níveis mais baixos, tal como uma imagem pode ser representada por *pixels* e os textos são compostos por palavras (LECUN; BENGIO; HINTON, 2015).

As redes neurais de aprendizado profundo já existem há um bom tempo, no entanto, independentemente de sua finalidade, arquiteturas profundas não podiam ser treinadas com êxito antes 2006, devido á falta de recursos computacionais e algoritmos mais eficientes (FADLULLAH et al., 2017).

As recentes técnicas de aprendizado profundo tem solucionado problemas de aplicação de uma maneira que nenhuma outra tecnologia jamais o fez. Com um progresso muito relevante nos domínios de aplicações que são considerados difíceis e extremamente desafiadores, tal como reconhecimento de padrões em visão computacional, reconhecimento de fala em PLN e outras aplicações. O avanço é devido a três fatores: ao desenvolvimento de modelos de redes muito mais eficazes, com arquiteturas substancialmente mais complexas; ao desenvolvimento de arquiteturas maciças de computação paralela, como combinações de muitas unidades de processamento gráfico (GPUs) possibilitando um alto processamento e eliminando virtualmente as restrições computacionais e a crescente disponibilidade maciça de bancos de dados (PLOTZ; GUAN, 2018). O aprendizado profundo faz o aproveitamento de grandes bases de dados com grandes capacidades de processamento de forma automática sem a necessidade de esforço humano (YOUNG et al., 2018).

As topologias de redes de aprendizado profundo mais utilizadas, são as redes convolucionais, as redes adversárias e as redes recorrentes. Do qual as redes convolucionais são mais apropriadas para serem utilizadas no processamento de imagens, vídeos, voz e áudio, enquanto que as redes recorrentes são mais aplicadas na análise de textos. As redes adversárias aprendem a imitar uma distribuição de dados e criar novas versões dessas, seja em vídeos, áudios ou imagens.

2.6 Rede Neurais Recorrentes

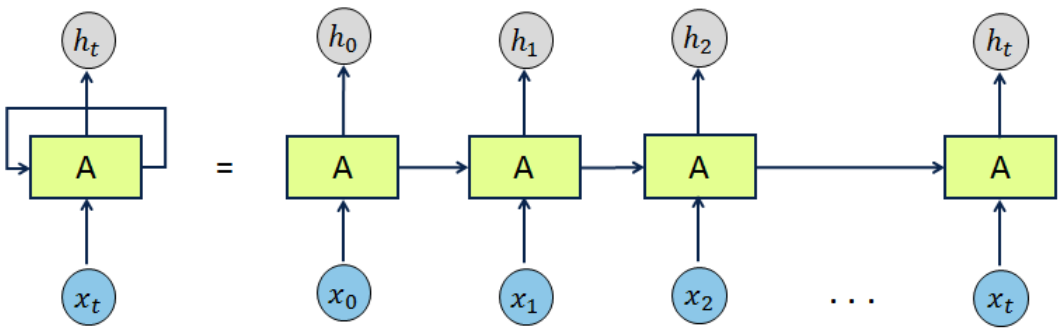
Os trabalhos com modelos de Redes Neurais Recorrentes (RNN) começaram na década de 80, com as Redes de *Jordan* em 1986, a Redes de *Elman* em 1990, as redes LSTM (*Long Short-Term Memory*) em 1997 e redes GRU (*Gated Recurrent Unit*) em 2014. As redes *Elman* foram desenvolvidas com interesse de aplicações em algoritmos de PLN.

As RNNs são muito eficientes em trabalhos que envolvem entradas sequenciais, como reconhecimento de fala e texto, pois ao contrário de uma MLP são capazes de guardar informações ao longo do tempo. Essas redes são muito poderosas e dinâmicas, porém complicadas de serem treinadas com o algoritmo de *back-propagation*. Isso ocorre pois os gradientes retropropagados podem aumentar ou diminuir a cada etapa de tempo, então, ao longo de muitas etapas de tempo, eles explodem ou desaparecem (LECUN; BENGIO; HINTON, 2015). O problema da explosão refere-se ao grande aumento na norma do gradiente durante o treinamento (PASCANU; MIKOLOV; BENGIO, 2013). O desaparecimento do gradiente é um problema caracterizado pela redução do gradiente a medida que ele se propaga ao longo do tempo, pois gradientes pequenos atrapalham o aprendizado da rede (SHERSTINSKY, 2020).

A arquitetura de uma rede neural recorrente pode assumir muitas formas diferentes, sendo que os blocos de uma RNN podem ser conectados aos outros ou podem ser empilhados para formar uma rede profunda. A direção do encaminhamento de dados pode ser para frente, para trás ou bidirecional (CHAI; LI, 2018).

Na sua arquitetura, as RNNs possuem um ou mais laços de realimentação. Estes laços acontecem a partir das ligações entre neurônios que formam ciclos conhecidos como conexões recorrentes, ou seja, apresenta neurônios recorrentes. Nestes *loops* as informações passam de uma camada para a outra, e possibilita as RNNs armazenarem informações ao processar novas entradas. As RNNs possuem capacidade de fazer com que as informações persistam através do tempo, do qual o que ocorreu no passado tem influência em resultados do futuro. As conexões recorrentes permitem que as unidades ocultas mantenham "vetor de estado" que contém implicitamente informações sobre a história de todos os elementos passados da sequência (LECUN; BENGIO; HINTON, 2015).

Figura 2.7 – Redes Neurais Recorrentes



Fonte: Do Autor (2022)

Na Figura 2.7 pode ser vista a arquitetura das redes RNNs. Essas redes apresentam conexões vindas da camada de entrada e das camadas anteriores para as subsequentes nas camadas intermediárias. Elas também apresentam conexões em uma mesma camada, fazendo uso de informações dos neurônios da camada anterior e também até da própria saída. A saída da rede tem dependência da entrada corrente e também das entradas já processadas devido ao armazenamento de informações. O processo de realimentação cria uma espécie de memória, faz com que haja um reforço na atualização dos pesos e isto possibilita à criação de modelos de redes neurais mais complexos.

As redes neurais recorrentes são eficientes para completar informações, prever dados sequenciais, mas apresentam a desvantagem da memória de curto prazo. Se a sequência for muito longa as RNNs tem dificuldades de carregar informações anteriores e consequentemente prever as posteriores (LECUN; BENGIO; HINTON, 2015). Para resolver estes problemas de memória de curto prazo, e também os problemas de desaparecimento e explosão dos gradientes, foram criadas as redes recorrentes LSTM e GRU. Elas possuem mecanismos internos que servem para regular o fluxo de informações, do qual os dados importantes para o contexto são mantidos e o restante são eliminados.

2.6.1 *Long Short-Term Memory*

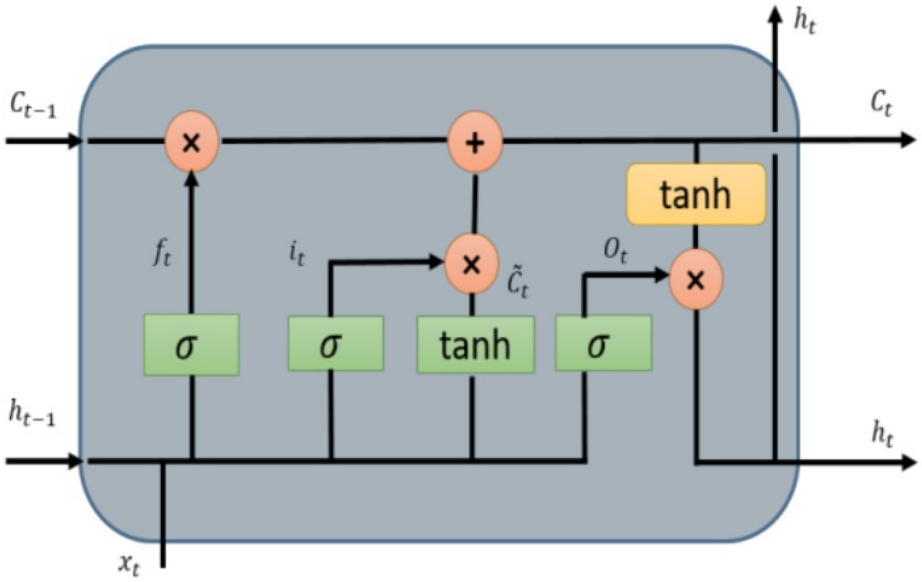
As redes *Long Short-Term Memory* (LSTM) foram propostas por Hochreiter e Schmidhuber (1997) para resolver o problema encontrado pelas RNNs antecessoras no tratamento de sequências longas. Elas possuem uma arquitetura capaz de armazenar informações por um período mais longo de tempo e a capacidade de aprender grandes quantidades de informações relevantes. Além disso, elas fornecem resultados relevantes em muitas tarefas diferentes de PLN, sendo consideradas uma das mais bem sucedidas redes recorrentes.

As RNNs antecessoras das LSTM apresentam alguns problemas na capacidade de armazenar informações. Um dos cenários em que ocorre estes problemas é na predição de palavras em textos, do qual para a predição não há necessidade de informações do início do texto, mas apenas das últimas palavras que fazem parte da construção do contexto. Nestes casos, estes tipos de RNNs não apresentam um bom funcionamento, pois não é dada prioridade para as palavras imediatamente anteriores. Este tipos de RNNs também lidam com explosão e desaparecimento do gradiente quando lidam com grandes quantidades de dados. As redes LSTM tem o objetivo de tratar estes problemas encontrados pelas redes RNNs antecessoras, buscando

oferecer um melhor desempenho nestes cenários onde estas RNNs apresentam problemas, tal como problema de predição de palavras, devido sua capacidade de obtenção das informações relevantes para formar o contexto (LECUN; BENGIO; HINTON, 2015).

Na Figura 2.8 pode ser vista a arquitetura das redes LSTM, em que *x(t)* representa a entrada no tempo *t*, *h(t-1)* representa um estado oculto no tempo *t-1*, *C(t-1)* é o estado da célula no tempo *t-1* e *h(t)* é a saída da rede.

Figura 2.8 – Redes Neurais LSTM



Fonte: Alom et al. (2018)

As redes LSTM se constituem de um conjunto de sub-redes conectadas recorrentemente chamadas de bloco de memória. Os blocos contém células de memórias com auto grau de conexões capazes de armazenar o estado temporal da rede, e também portas (*gates*) que são responsáveis por controlar o fluxo de informações. Estas portas decidem quais informações devem serem armazenadas no estado da célula (SAHIN; DIRI, 2019). De acordo com Chen et al. (2017), a célula de memória *C(t)* é a chave das LSTM. Elas que armazenam as informações de longo prazo enquanto evita problemas de desaparecimento e explosão de gradientes.

As redes LSTM são formadas por três portas: a porta de entrada *i(t)*, a porta de saída *O(t)* e a porta de esquecimento *f(t)*. As três portas trabalham de forma colaborativa no controle do fluxo de informações, pois a porta de entrada controla as informações que são lidas. A porta de esquecimento determina quais informações devem ser descartadas da célula de memória e a porta de saída define quais informações devem ir para saída (CHEN et al., 2017) (YE et al., 2019).

Como as redes LSTM possuem a capacidade de esquecimento das informações que não são relevantes para o momento, elas realizam uma série de verificações nas portas de controle de fluxo do qual realizam a tarefa de permitir ou não a permanência de informação para análise. As portas possibilitam a realização dos ajustes dos pesos e o truncamento da sequência a fim de eliminar informações desnecessárias (SAHIN; DIRI, 2019). Estas operações realizadas pela célula de memória ajudam a aliviar o problema de explosão e desaparecimento do gradiente (QIN et al., 2020) (ZHANG; LIU; LIU, 2017).

A execução do bloco de memória ocorre de acordo com os seguintes passos: o primeiro passo das LSTMs é a tomada de decisão sobre quais informações deve ser esquecidas e quais devem ser lembradas. O segundo passo é definir quais informações devem ser armazenadas no estado da célula, verificando quais valores devem ser atualizados e em seguida são criados novos valores candidatos a serem adicionados. No terceiro passo é realizada a atualização com base nas definições do que esquecer e do que atualizar de acordo com as tarefas realizadas nas etapas anteriores. No quarto passo a célula está com uma versão atualizada e então será decidido o que será a saída da rede.

As redes LSTM possuem entradas fixas, havendo a necessidade de comprimir todas as informações necessárias de uma sentença em um vetor de comprimento fixo. E isto gera problemas para tarefas de processamento de linguagem natural, pois caso as entradas das redes recebam dados que contém muitas informações irrelevantes para contexto da tarefa atual as mesmas podem não ser capazes de selecionar somente os dados importantes. Ocorre também quando as entradas das redes são formadas por sequencias longas, isso faz os vetores de informação de contexto apresentarem influências das últimas entradas processadas, perdendo assim informações iniciais. Neste sentido, o mecanismo de atenção de *Bahdanau* busca resolver estes problemas permitindo que as redes LSTM tenham acesso as informações mais relevantes para o contexto atual incluindo as sequências de entrada com informações iniciais (BAHDANAU; CHO; BENGIO, 2015) (GOMES; EVSUKOFF, 2017).

2.7 *Few-Shot Learning*

Uma pessoa pode identificar a presença de uma outra em uma imagem após ter visto ela uma única vez anteriormente. No entanto, o processo de aprendizagem com a utilização de poucos dados é uma tarefa complexa para as redes neurais tradicionais (JADON; GARG, 2020).

Para o caso de uma pequena base de dados, um algoritmo simples como o KNN (*k* vizinhos mais próximos) pode ter desempenho melhor do que uma rede neural MLP, caso se tenha um bom pré-processamento que leve a uma boa separação entre as classes e a escolha de uma boa métrica para o cálculo da distância (JADON; GARG, 2020).

As redes de aprendizado profundo conseguem um bom desempenho em uma variedade de tarefas, porém, necessitam de uma grande quantidade de dados para aprender (ZHENG et al., 2019). No entanto, existem bases de dados com apenas algumas poucas amostras por classes e isso dificulta o processo de treinamento das redes neurais tradicionais, obtendo desempenho reduzido. Neste caso, é necessário implementar métodos de aprendizagem de máquina específicos para redes neurais que sejam capazes de realizar o treinamento quando se tem poucas amostras por classe, estes métodos são conhecidos como *Few-Shot Learning* ou *One-Shot Learning* (WANG et al., 2020).

Existem vários algoritmos de aprendizado de *Few-Shot Learning* (FSL), alguns dos mais conhecidos são as redes: *Matching Networks* MN (VINYALS et al., 2016), *Model Agnostic Meta-Learning* MAML (FINN; ABBEEL; LEVINE, 2017), *Memory-Augmented Neural Networks* MANN (SANTORO et al., 2016) e as Redes Siamesas (KOCH; ZEMEL; SALAKHUTDINOV, 2015).

2.8 K-Vizinhos Mais Próximos

O *k*-vizinhos mais próximos (KNN) é um dos mais simples e tradicionais classificadores cujo desempenho é competitivo em relação aos mais complexos classificadores da literatura. Trata-se de um dos principais algoritmos utilizados em reconhecimento de padrões, categorização de textos, reconhecimento de objetos, dentre outras aplicações. A grande vantagem do KNN é que o mesmo não depende de treinamento, a sua tarefa se resume em identificar, em um conjunto rotulado, os *k* objetos mais semelhantes ou próximos (com menores distâncias) da amostra não rotulada (TRSTENJAK; MIKAC; DONKO, 2014). Portanto, o KNN é uma opção interessante para conjunto de dados com pequena quantidade de amostras. De fato, o KNN pode ser considerado uma técnica de *few-shot learning* uma vez que, a partir do momento que uma ou mais amostras de uma certa classe nova estiver disponível, tal amostra pode ser prontamente adicionada no conjunto de dados e está apta a ajudar na classificação de outra amostra de sua mesma classe, caso isso seja necessário (CHAVES et al., 2021).

O bom desempenho do KNN depende da escolha apropriada do método de cálculo de distância para cada aplicação em específico (Abu Alfeilat et al., 2019), além da escolha da quantidade correta de vizinhos k (GUO et al., 2003). Normalmente, a distância Euclidiana e do cosseno são as mais utilizadas, porém, existem diversas outras distâncias que também apresentam resultados satisfatórios, como a *Manhattan, Hassanat, Lorentzian*, entre outras (Abu Alfeilat et al., 2019; XU; TIAN, 2015; ALKASASSBEH; ALTARAWNEH; HASSANAT, 2015). No entanto, não é possível definir qual distância apresenta o melhor resultado, pois cada uma tem suas particularidades, sendo que esta escolha deve ser realizada de acordo com cada conjunto de dados.

2.9 Redes *Matching*

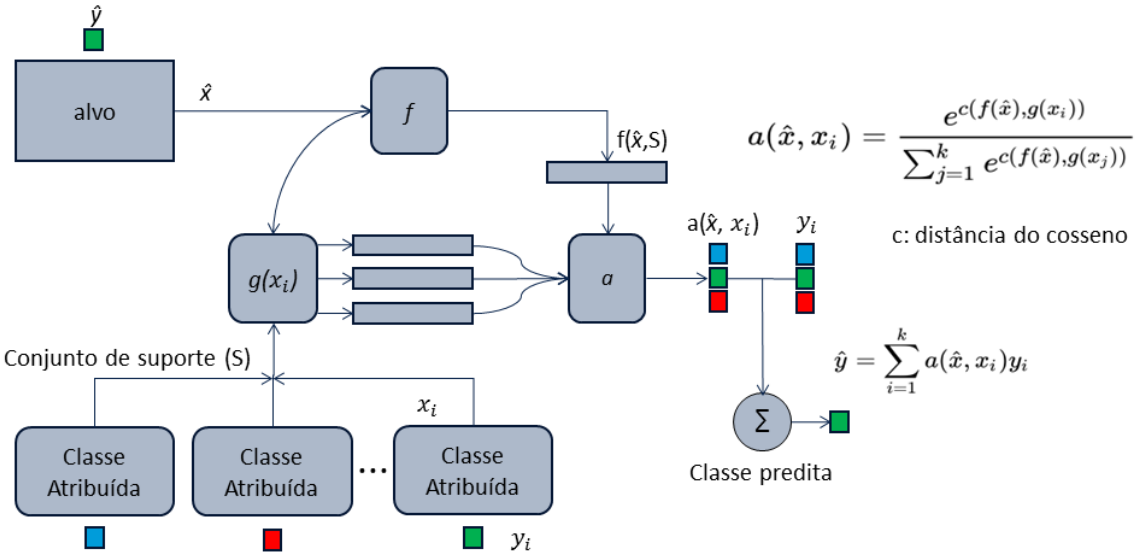
As redes *matching* aprendem a mapear uma pequena base de dados de treino e teste em um mesmo espaço de *embeddings* (tarefa *few-shot*). Elas são treinadas para aprender uma representação com os *embeddings* da pequena base de dados de treino de forma adequada, em busca da minimização dos erros do cálculo de similaridade entre o alvo e os representantes de classe. As redes *matching* funcionam com a ideia de um KNN diferenciável com medida de similaridade do cosseno. Além dos *encoders* que formam os *embeddings* estas redes utilizam um mecanismo de atenção para auxiliar na predição, este que consiste na aplicação da função *softmax* na saída da distância do cosseno (VINYALS et al., 2016).

As redes *matching* consiste em cinco principais etapas: (i) "Extrator de *embeddings*" g, que forma o encoder com os dados das amostras; (ii) "*Embeddings* de contexto completo" f, cujo o objetivo é obter o contexto entre as amostras quando houver, este é formado por uma rede LSTM bidirecional (o uso é opcional); (iii) cálculo de similaridade com a função de distância cosseno c; (iv) mecanismo de atenção[3] a que é dado pela aplicação da função *softmax* na saída da função de cálculo de similaridade c e por último (v) a função de perda entropia cruzada (JADON; GARG, 2020).

A arquitetura com as etapas das redes *matching* descritas anteriormente pode ser vista na Figura 2.9. O "conjunto de suporte" (S) (à esquerda da figura) é o conjunto de dados de entrada, como se fossem os representantes (vizinhos) do KNN sendo que x_i é o vetor de dados, y_i são os *labels*, que são utilizados para realizar o aprendizado supervisionado. O alvo $\hat{x}$ é a amostra a ser predita.

[3] Este mecanismo de atenção é diferente da camada de atenção de *Bahdanau*.

Figura 2.9 – Arquitetura das redes *matching*



Fonte: Do Autor (2021)

Os dados do "conjunto de suporte" formados são encaminhados para o "extrator de *embeddings*" g, que aprende uma nova representação para as amostras. Depois os dados passam pelo "*embedding* de contexto completo" f, ou seja, a saída de g e o alvo $\hat{x}$ passam pela rede *BILSTM* representada por f (SPROAT; JAITLY, 2016) (VINYALS et al., 2016). Conforme a Figura 2.9, abaixo de f pode ser observada a equação $f(\hat{x}, S)$ que é a nova representação dos dados do "conjunto de suporte" (S) e do alvo $\hat{x}$ obtida por f.

O próximo passo antes da aplicação da atenção é o cálculo de similaridade utilizando a função do cosseno. A similaridade é calculada entre a nova representação do "conjunto de suporte" (x_i) e o alvo $\hat{x}$. O mecanismo de atenção $\boldsymbol{a}$ é obtido pela aplicação da função *softmax* na saída da função de similaridade do cosseno conforme a Equação 2.4:

$$a(\hat{x}, x_i) = \frac{e^{c(f(\hat{x}),g(x_i))}}{\sum_{j=1}^{k} e^{c(f(\hat{x}),g(x_j))}} \tag{2.4}$$

em que c é a função que executa o cálculo de similaridade com a distância do cosseno e e representa a *softmax*.

A predição é dada por:

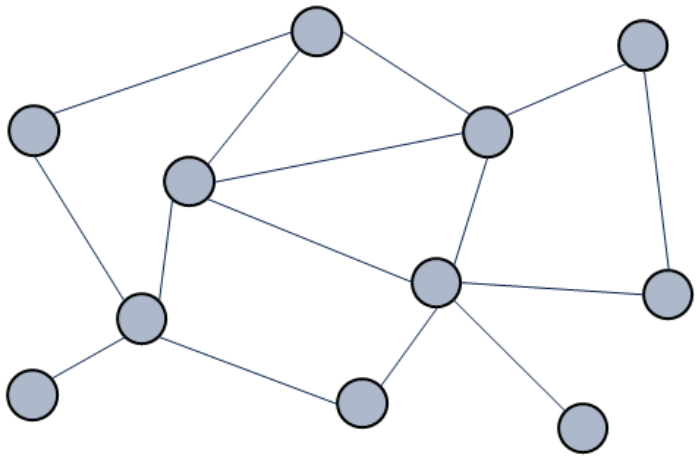
$$P(\hat{y}|\hat{x}, S) = \sum_{i=1}^{k} a(\hat{x}, x_i) y_i, \tag{2.5}$$

que implementa a combinação linear da *softmax* da camada de atenção com o vetor *one hot encoded* dos *labels* y_i. Esta equação é uma combinação linear de probabilidades que determina a qual classe o alvo (*query*) pertence. A função de perda normalmente utilizada é a entropia cruzada.

2.10 Redes Neurais de Grafos

Os grafos são um tipo de estrutura de dados que modela um conjunto de objetos (nós) e seus relacionamentos (arestas). Os grafos são definidos por $G = (V, E)$, sendo que V representa os nós e E as arestas. Os nós podem representar os elementos, ou seja, os dados das amostras, e as arestas representam os relacionamentos. As pesquisas com grafos tem ganhado atenção devido ao poder de representação que possuem, pois podem ser utilizados em um grande número de sistemas, do qual pode-se citar como exemplos: representação de mídias sociais, de forma que os usuários são os nós e as arestas representam as conexões; conexão entre cidades, onde as cidades podem ser representadas como os nós e as rodovias como arestas, e texto e fala que podem ser considerados um grafo representado em forma de linha (ZHOU et al., 2020; GARCIA; BRUNA, 2018). Um exemplo de um grafo pode ser visto na Figura 2.10:

Figura 2.10 – Exemplo de grafo



Fonte: Do Autor (2021)

As redes *Graph Neural Network* (GNN) fazem o aprendizado da representação por meio da agregação das características da vizinhança, ou seja, transformam as características dos nós levando em conta as características dos nós da vizinhança. Essas redes são modelos neurais capazes de capturar a dependência dos nós dos grafos por meio da passagem de mensagens entre estes nós. Essa mensagem pode ser obtida por vários tipos de funções, tal como, similaridade,

média de vizinhos, entre outras. O vetor com os dados das mensagens pode ser passado por uma rede neural, tal como será visto na seção 2.10.1. A rede neural faz a atualização formando uma nova representação para o nó e esse processo é feito em cada nó do grafo, sendo que cada nó coleta as mensagens da vizinhança e agrega as mensagens ao nó. Dessa forma, as redes GNNs conseguem trabalhar com dados com relacionamentos complexos e interdependência de objetos (ZHOU et al., 2020; GARCIA; BRUNA, 2018; KIM et al., 2019).

As redes GNN podem ser aplicadas em variadas tarefas tais como: sistemas de recomendação, visão computacional, otimização combinatorial, química e física, descoberta de drogas e processamento de linguagem natural. Em comércio eletrônico é comum o uso de *Knowledge Graph* para recomendação de produtos e outros. Nos últimos anos, várias variantes de GNNs tem surgido do qual podem-se citar, *Graph Convolutional Network* (GCN), *Graph Attention Network* (GAT), *Graph Recurrent Network* (GRN) e *Gated Graph Network* (GGN), que têm demonstrado desempenhos inovadores em muitas tarefas de aprendizagem profunda (ZHOU et al., 2020).

As redes de grafos também têm sido aplicadas em muitas tarefas *few-shot learning* (GARCIA; BRUNA, 2018). Dentre as redes mais conhecidas pode-se citar as EGNN (*Edge-Labeling Graph Neural Network*) (KIM et al., 2019), GNN (*Graph Neural Network*) (GARCIA; BRUNA, 2018) e as DPGN (*Distribution Propagation Graph Network*) (YANG et al., 2020), que foi escolhida para ser abordada neste trabalho por ter apresentado bom desempenho para bases de dados *few shot learning*, tal como *mini*ImageNet (VINYALS et al., 2016).

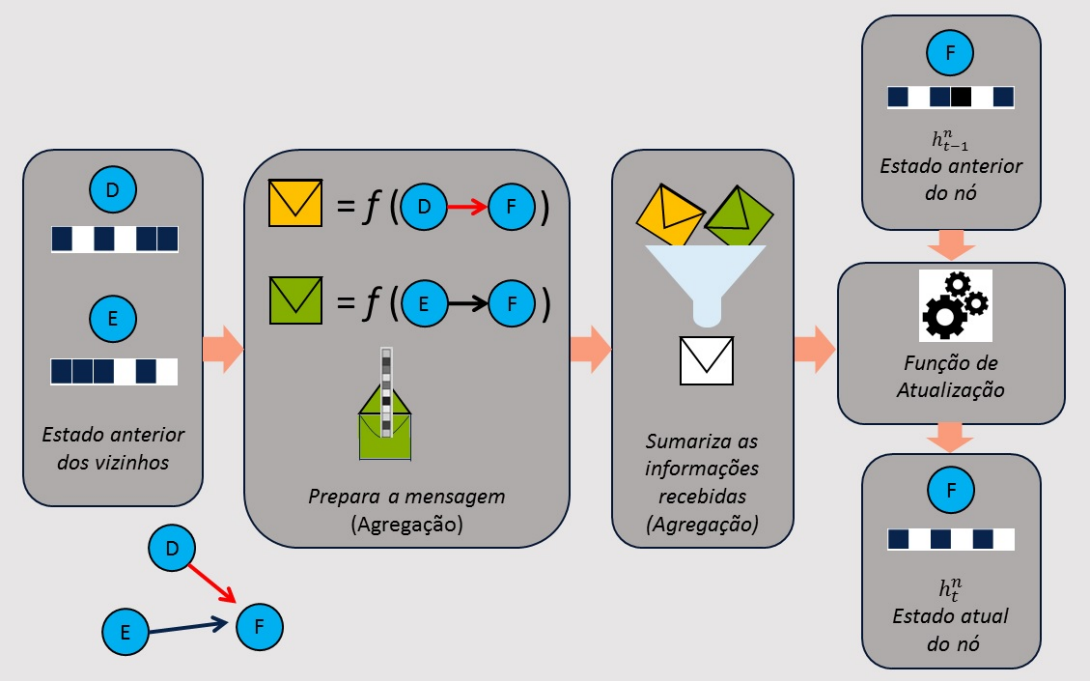
2.10.1 Passagem de Mensagem Neural

As redes GNN podem ser entendidas como um simples algoritmo de passagem de mensagem. Elas usam uma forma de passagem de mensagem na qual mensagens vetoriais são trocadas entre nós e atualizadas usando redes neurais. A passagem de mensagem acontece da seguinte forma: primeiro a rede recebe a representação inicial (*embeddings*); em seguida prepara as mensagens; as informações são então sintetizadas (ou agregadas); e na atualização é formado outro vetor a partir das informações da vizinhança. Estas informações são combinadas com o estado anterior e gera a atualização (estas etapas são realizadas com funções de acordo com a aplicação).

A Figura 2.11 ilustra um exemplo de como acontece o processo de passagem de mensagem. As informações dos nós são representadas por vetores. Considerando F como nó do

estado atual, este recebe as informações dos nós vizinhos D e E utilizando alguma função tal como média, soma, similaridade, e após, sumariza em um único vetor e assim é realizada a agregação. A saída da agregação é concatenada com o estado anterior do nó, e desta forma é realizado a atualização do próximo estado do nó F. Na atualização pode-se utilizar redes neurais tal como MLP, GRU, entre outras.

Figura 2.11 – Passagem de mensagem das redes GNN



Fonte: Do Autor (2022)

Definindo matematicamente a passagem de mensagem tem-se que, durante cada interação da GNN, a camada de *embedding*, representada por $h_u^{(k)}$, correspondente a cada nó u que pertence a V, é atualizada de acordo com as informações agregadas dos nós da vizinhança $N(u)$ no grafo G. A passagem de mensagem é definida de forma geral em agregação e atualização (HAMILTON, 2020; GILMER et al., 2017). A descrição geral da passagem de mensagem se dá pelas equações a seguir:

$$m_{N(u)}^{(k)} = AGREGACAO^{(k)}(h_v^{(k-1)}, \forall v \in N(u)) \tag{2.6}$$

$$h_u^{(k)} = ATUALIZACAO^{(k)}(h_u^{(k-1)}, m_{N(u)}^{(k)}) \tag{2.7}$$

$$z_{(u)} = h_u^{(K)}, \forall u \in V \tag{2.8}$$

em que, *AGREGACAO* e *ATUALIZACAO* são funções diferenciáveis arbitrárias (redes neurais), de forma que cada abordagem usa um tipo diferente de redes neurais, como por exemplo *graph recurrent network* (GRN) utiliza uma rede *Gated Recurrent Unit* (GRU) como função de atualização; $m_{N(u)}^{(k)}$ representa a mensagem agregada dos nós da vizinhança; $h_u^{(k-1)}$ é o estado anterior do nó u. A cada iteração k da GNN, a função *AGREGACAO* recebe o conjunto *embeddings* dos nós e gera uma mensagem $m_{N(u)}^{(k)}$ com base nas informações agregadas da vizinhança $N(u)$. A função *ATUALIZACAO* combina a mensagem $m_{N(u)}^{(k)}$ com o estado anterior $h_u^{(k-1)}$ do nó u e assim obtém a atualização do estado atual $h_u^{(k)}$ do nó. Depois de executar as k iterações da passagem de mensagem das redes GNN, a saída da camada final é utilizada para definir a nova representação dos *embeddings* de cada nó conforme a equação (2.8) (HAMILTON, 2020; GILMER et al., 2017).

Vale lembrar que as equações de (2.6) a (2.8) são *frameworks* que descrevem de forma geral e abstrata as redes de grafos. Desta forma, para um exemplo mais específico pode-se citar uma das mais básicas redes neurais de grafos que é uma simplificação da rede GNN proposta inicialmente por Merkwirth e Lengauer (2005) e Scarselli et al. (2009). A passagem de mensagem desta rede pode ser definida por:

$$h_u^{(k)} = \sigma(w_{self}^{(k)} h_u^{(k-1)} + w_{neigh}^{(k)} \sum_{v \in N(u))} h_v^{(k-1)} + b^{(k)}) \tag{2.9}$$

em que $w_{self}^{(k)}$ e $w_{neigh}^{(k)}$ são matrizes de parâmetros treináveis e σ denota uma função não linear (o termo *bias* foi omitido por simplicidade). A agregação $m_{N(u)}^{(k)}$ é dada pela soma proveniente dos vizinhos e a atualização é dada pela combinação linear do nó anterior com as informações dos vizinhos representada por $m_{N(u)}^{(k)}$ formando o nó atual $h_u^{(k)}$, conforme as equações a seguir (HAMILTON, 2020):

$$m_{N(u)}^{(k)} = \sum_{v \in N(u))} h_v^{(k-1)} \tag{2.10}$$

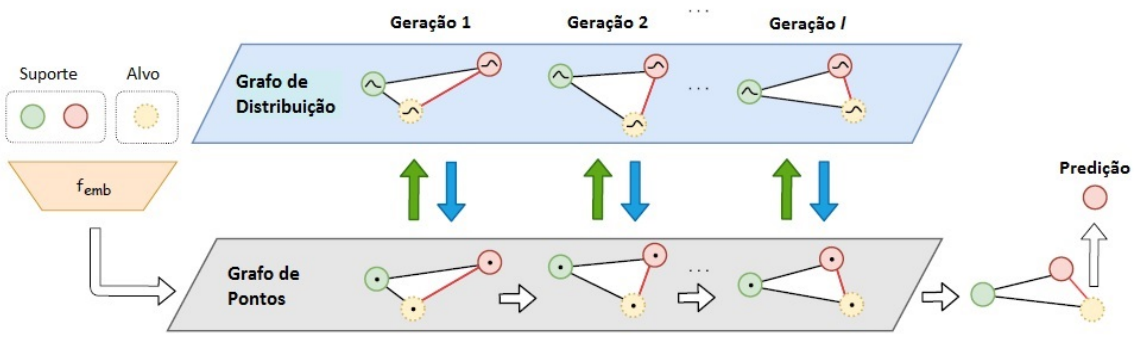
$$h_u^{(k)} = ATUALIZACAO^{(k)}(h_u^{(k-1)}, m_{N(u)}^{(k)}) = \sigma(w_{self}^{(k)} h_u^{(k-1)} + w_{neigh}^{(k)} m_{N(u)}^{(k)}) \tag{2.11}$$

2.11 Redes DPGN

A DPGN (*distribution propagation graph network*) é um tipo de rede de grafo aplicada para a tarefa *few-shot learning*. Esta contém uma dupla arquitetura chamada: "grafo de ponto" (*PG*) e "grafo de distribuição" (*DG*). O "grafo de ponto" consiste na inferência no nível de instância e o "grafo de distribuição" é a inferência no nível de distribuição. Desta forma é processada a representação dos dados em diferentes níveis de forma independente, aproveitando a representação no nível de instâncias e no nível de distribuição. A atualização das redes acontece da seguinte forma: *PG* gera *DG* reunindo a relação $1 \times n$ em todos os exemplos, ou seja, a relação de um nó com toda a vizinhança. Enquanto *DG* refina *PG* através da entrega das relações de distribuição. Desta forma, as relações de *PG* e *DG* são fundidas ao longo das gerações (YANG et al., 2020).

A arquitetura das redes DPGN pode ser vista na Figura 2.12. Nesta figura tem-se o exemplo da DPGN em uma tarefa 2-*way* e 1-*shot*. A formação do *encoder* f_{emb} é similar às redes *matching*, onde são extraídas as características no nível de instância do conjunto de suporte e dos alvos. Estes dados são entregues ao "grafo de pontos" da arquitetura dupla das redes DPGN que realizam ciclos de transformação "ponto para distribuição" (P2D) e "distribuição para ponto" (D2P) ao longo das gerações. A seta verde indica a transformação (P2D) cujo papel é agregar a similaridade de instância para construir a representação no nível de distribuição. Seta azul indica a transformação (D2P) que por sua vez, agrega a similaridade de distribuição a fim de obter características de instância mais discriminativas. No final das gerações a rede DPGN faz a predição do alvo (YANG et al., 2020).

Figura 2.12 – Arquitetura das redes DPGN



Fonte: Adaptada de Yang et al. (2020)

A DPGN consiste de l gerações e em cada geração é atualizado o "grafo de pontos" $G_l^p = (V_l^p, E_l^p)$ e o "grafo de distribuição" $G_l^d = (V_l^d, E_l^d)$ de acordo com as seguintes etapas:

I) As características das amostras extraídas pelo *embedding* são utilizadas para inicializar os nós V_l^p que são utilizados para calcular a similaridade de instância E_l^p. Estas relações de instâncias são enviadas para construir o "grafo de distribuição" G_l^d.

II) As características dos nós V_l^d são atualizadas através da agregação das arestas E_l^p.

III) As características das arestas E_l^d representam as similaridades de distribuição entre as características dos nós V_l^d.

IV) As características das arestas de distribuição E_l^d são entregues para o grafo de pontos G_l^p construindo uma representação mais discriminativa dos nós V_l^p.

Este processo é repetido geração por geração. Em resumo, a atualização da DPGN pode ser expressa como: $E_l^p \rightarrow V_l^d \rightarrow E_l^d \rightarrow V_l^p \rightarrow E_{l+1}^p$ (que corresponde a uma geração, ou seja, é realizado um ciclo de atualização dos nós). Para maiores explicações serão reformuladas as variáveis conforme segue: $E_l^p = e_{l,ij}^p$; $V_l^d = v_{l,i}^d$; $E_l^d = e_{l-1,ij}^d$; $V_l^p = v_{l,i}^p$. A inicialização dos nós $v_{0,i}^p$ é obtida do extrator F_{emb}, do qual para cada amostra x_i tem-se:

$$v_{0,i}^p = f_{emb}(x_i), \tag{2.12}$$

sendo que $v_{0,i}^p \in R^m$ e m denota a dimensão das características dos *embeddings*.

Na Figura 2.13 é mostrada com mais detalhes a agregação P2D (seta verde) e D2P (seta azul), a fim de obter a formalização matemática das redes DPGN conforme Yang et al. (2020). Neste exemplo, tem-se duas classes (2-*way*) e 1 amostra por classe (1-*shot*). A MLP1 é uma rede densa (*leaky relu*) e a MLP2 é uma rede convolucional.

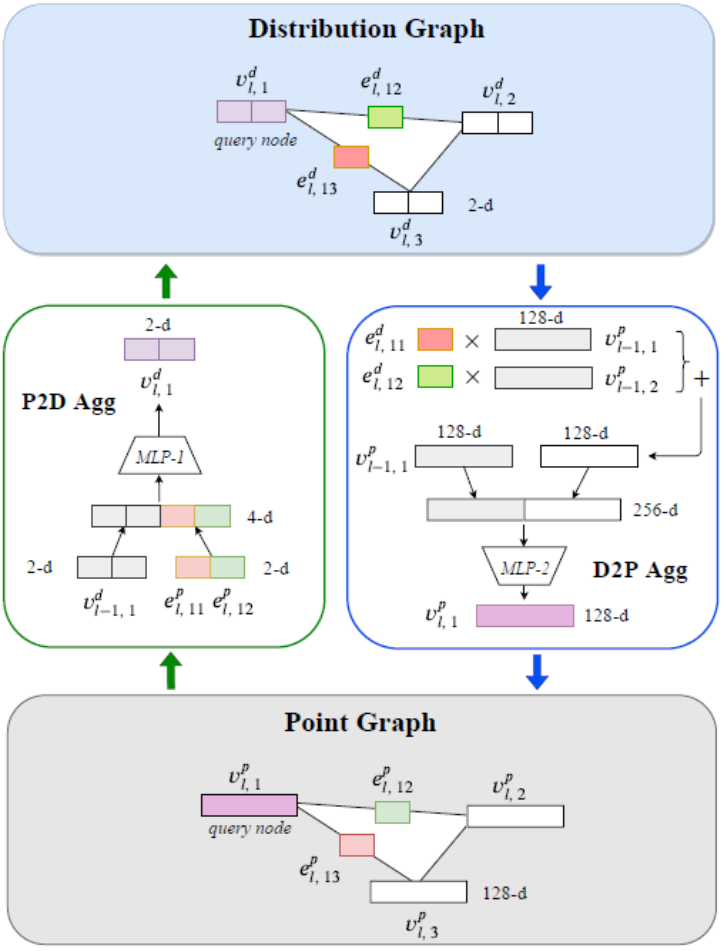
2.11.1 Agregação P2D

Nesta subseção é mostrado como é realizada a agregação P2D pelas redes DPGN. Esta consiste de duas etapas: primeiro é realizado o cálculo de similaridade utilizando os nós do "grafo de pontos" a fim de obter as suas arestas e, depois, estas arestas são utilizadas no processo de agregação com o intuito de obter os nós do "grafo de distribuição".

I) Similaridade de ponto: cada aresta do "grafo de pontos" $e_{l,ij}^p$ representa a similaridade de instância, que para a geração $l > 0$ são dadas por:

$$e_{l,ij}^p = f_{e_l^p}((v_{l-1,i}^p - v_{l-1,j}^p)^2) \cdot e_{l-1,ij}^p \tag{2.13}$$

Figura 2.13 – Arquitetura das redes DPGN



Fonte: Yang et al. (2020)

em que $e^p_{l,ij} \in R$ e $f_{e^p_l} : R^m \rightarrow R$. $f_{e^p_l}$ é uma rede neural convolucional com duas camadas e um conjunto de parâmetros $\theta_{e^p_l}$ e uma camada *sigmoid*. Esta rede transforma a similaridade de instância para uma certa escala.

II) Agregação P2D: uma vez obtido E^p_l, o objetivo é utilizá-lo para agregar as relações de instâncias do "grafo de pontos" G^p_l ao "grafo de distribuição" G^d_l. Desta forma, para $l > 0$, o nó de distribuição $v^d_{l,i}$ é atualizado por:

$$v^d_{l,i} = P2D(||^{NK}_{j=1}\ e^p_{l,ij}\ ,\ v^d_{l-1,i}) \tag{2.14}$$

sendo que $||$ é o operador de concatenação; NK é o número de amostras do "conjunto de suporte". Desta forma, cada nó $v^d_{l,i}$ tem dimensão NK, assim, $v^d_{l,i} \in R^{NK}$; P2D é a uma rede neural MLP com uma camada (relu) e um conjunto de parâmetros $\theta_{v^d_l}$. Esta rede neural de agregação

faz a transformação $P2D : (R^{NK}, R^{NK}) \rightarrow R^{NK}$, ou seja, P2D faz a transformação $R^{2NK} \rightarrow R^{NK}$ nas características das arestas de pontos, concatenadas com o nó de distribuição anterior.

2.11.2 Agregação D2P

Nesta subseção é mostrado como é realizada a agregação D2P pelas redes DPGN. Esta consiste de duas etapas: primeiro é realizado o cálculo de similaridade utilizando os nós do "grafo de distribuição" a fim de obter as suas arestas e, depois, estas arestas são utilizadas no processo de agregação com o intuito de atualizar os nós do "grafo de pontos".

III) Similaridade de distribuição: cada aresta no "grafo de distribuição" representa a similaridade entre diferentes nós de distribuição. Para a geração $l > 0$, as arestas $e^d_{l,ij}$ é dada por:

$$e^d_{l,ij} = f_{e^d_l}((v^d_{l-1,i} - v^d_{l-1,j})^2) \cdot e^d_{l-1,ij} \tag{2.15}$$

tal que $e^d_{l,ij}$ ε R; $f_{e^d_l}$ é uma rede neural convolucional com duas camadas, um conjunto de parâmetros $\theta_{e^d_l}$ e uma camada *sigmoid*. Esta rede transforma a similaridade de distribuição para uma certa escala $f_{e^d_l} : R^{NK} \rightarrow R$; .

IV) Agregação D2P: nesta etapa as informações do "grafo de distribuição" G^d_l flui de volta para o "grafo de pontos" G^p_l no fim de cada geração. Isto acontece da seguinte forma, o nó $v^p_{l,i}$ captura a relação de distribuição através da agregação das características das arestas $e^d_{l,ij}$, conforme a seguinte equação:

$$v^p_{l,i} = D2P(\sum_{j=1}^{T}(e^d_{l,ij} \cdot v^p_{l-1,j}), v^p_{l-1,i}), \tag{2.16}$$

em que $v^p_{l,i} \in R^m$ e T denota o número total de exemplos em um episódio de treinamento; D2P é uma rede convolucional com duas camadas e um conjunto de parâmetros θ_{vp}. Esta rede é responsável pela agregação para formar o novo nó $v^p_{l,i}$. Ela concatena as características computada por $\sum_{j=1}^{NK}(e^d_{l,ij} \cdot v^p_{l-1,j})$ com as características do nó da geração anterior $v^p_{l-1,i}$ e faz a atualização $D2P : (R^m\ ,\ R^m) \rightarrow R^m$. Depois deste processo as características dos nós podem ser utilizadas para computar a similaridade de instância na próxima geração.

2.11.3 Predição

A predição da classe de cada nó pode ser calculada alimentando as correspondentes arestas $e^p_{l,ij}$ em uma função *softmax* na geração final l da DPGN.

$$P(\hat{y}_i|x_i) = softmax(\sum_{j=1}^{NK} e^p_{l,ij} \cdot onehot(y_j)) \tag{2.17}$$

em que x_i é a amostra i e y_j é o alvo da j-ésima amostra do conjunto de suporte; $e^p_{l,ij}$ representa as características das arestas no "grafo de pontos" da geração final; $P(\hat{y}_i|x_i)$ é a distribuição de probabilidades sobre as classes dada as amostras.

2.11.4 Perda do Grafo de Pontos

O erro do "grafo de pontos" na geração l é dado por:

$$L^p_l = L_{CE}(P(\hat{y}_i|x_i), y_i) \tag{2.18}$$

visto que L_{CE} é a função de perda de entropia cruzada; $P(\hat{y}_i|x_i)$ é o modelo de previsão de probabilidades da amostra x_i conforme a equação (2.17).

2.11.5 Perda do Grafo de Distribuição

A fim de aprender uma distribuição de características mais discriminativas foi incorporado o erro do "grafo de distribuição". Este contribui para uma convergência melhor e mais rápida. O erro do "grafo de distribuição" na geração l é definido por:

$$L^d_l = L_{CE}(softmax(\sum_{j=1}^{NK} e^d_{l,ij} \cdot onehot(y_j)), y_i) \tag{2.19}$$

em que $e^d_{l,ij}$ representa as características das arestas no "grafo de distribuição" na geração l.

2.11.6 Perda Total

A função objetivo total é dada pela soma ponderada das perdas do "grafo de ponto" e distribuição:

$$L = \sum_{l=1}^{\hat{l}} (\lambda_p \cdot L^p_l + \lambda_d \cdot L^d_l), \tag{2.20}$$

sendo λ_p e λ_d os pesos definidos para equilibrar as importâncias de cada perda (padrão 1,0 e 0,1 respectivamente) e l é o número total de gerações.

2.12 Algoritmos de Redução de Dimensão e Visualização de Dados

Nesta seção são descritos os algoritmos de redução de dimensão: *Principal Component Analysis* (NG, 2017; KRISHNAN; DUTTA, 2018), *Fisher's Discriminant Ratio* (WANG; LIU; ZHENG, 2007) e o *t-SNE* (*t-Distributed Stochastic Neighbor Embedding*) (MAATEN; HINTON, 2008).

A *Principal Component Analysis* (PCA) é um processo que converte um conjunto de características correlacionadas em um novo um subconjunto de características linearmente não correlacionadas chamadas "Componentes Principais" (PC). Desta forma os "Componentes Principais" são escolhidos afim de encontrar as direções (componentes) que maximizam a variância do conjunto de dados. A PCA é uma técnica de redução de dimensão não supervisionada, pois as amostras semelhantes podem ser agrupadas com base na correlação das características entre elas sem qualquer supervisão (ou rótulos) (KRISHNAN; DUTTA, 2018) (MISHRA et al., 2017).

O objetivo principal da PCA é identificar padrões que permita a redução de dimensão das amostras de uma base de dados com perda mínima de informações. A PCA é uma técnica que faz a projeção do espaço de características de alta dimensão em um subespaço menor buscando as características que sejam capazes de boa representação. Como a PCA utiliza autovetores e autovalores eles possuem direção e magnitude respectivamente. A direção indica em quais eixos principais os dados têm mais variância, e a magnitude significa a quantidade de variância que o "Componente Principal" captura dos dados quando projetados nesse eixo. Uma das principais aplicações da PCA é a redução de custo computacional nos problemas de classificação (NG, 2017) (KRISHNAN; DUTTA, 2018). O algoritmo da PCA consiste das seguintes etapas:

1) As amostras d-dimensionais da base de dados são utilizadas sem os *labels*.

2) Calcula-se o vetor médio d-dimensional, ou seja, a média das características do conjunto de dados.

3) É realizado o cálculo da matriz de covariância do conjunto de dados. A matriz de covariância é calculada para localizar o maior valor de variância do conjunto de dados.

4) É calculado os autovetores, e autovalores correspondentes. Os autovetores com os maiores autovalores indicam a direção de maior variação no conjunto de dados.

5) Através da ordenação dos autovalores os respectivos autovetores também são ordenados. Os k autovetores com os maiores autovalores são escolhidos para forma uma matriz dimensional W onde cada coluna de W representa um autovetor. Dessa forma são escolhidas as melhores características deste novo subespaço.

6) A matriz w é usada para transformar as amostras originais em um novo subespaço. Esta transformação é dada por: $y = W^t * x$. Do qual x é o vetor que representa as amostras e y é a amostra transformada pela matriz de autovetores w para um novo subespaço.

O *Fisher's Discriminant Ratio* (FDR) é usado para quantificar a capacidade de separação de características *features* individuais. Este é um algoritmo simples e rápido de ser calculado, opera apenas offline durante o projeto, sendo portanto excelente para redução de dimensão. É utilizado como um critério de separabilidade de classes, do qual o objetivo é encontrar o eixo que as classes apresentam maior separação. Este eixo indica a direção no qual a projeção dos dados maximiza a separação das classes (WANG; LIU; ZHENG, 2007) (THEODORIDIS; KOUTROUMBAS, 2009).

O algoritmo do FDR atribui a cada característica um critério ou valor que indica a importância de tal característica na separação das classes. Quanto maior este valor, maior é a importância desta característica. Desta forma as melhores características são escolhidas através da ordenação dos critérios. O FDR faz a seleção das características mais relevantes da base, mas não é capaz de eliminar a redundância, neste caso uma solução é seu uso em conjunto com análise de correlação linear afim de eliminar as características redundantes e obter uma base com menor número possível de características.

O algoritmo *t-Distributed Stochastic Neighbor Embedding* (*t-SNE*) é uma técnica de redução de dimensionalidade não-linear e não supervisionada utilizada para representar um conjunto de dados de alta dimensão em um espaço de baixa dimensão. Este possibilita a redução de dimensionalidade dos dados até uma dimensão visível ao olho humano, com duas ou três dimensões, preservando as distâncias de sua dimensão original. Ele possibilita a visualização de dados complexos, sendo possível descobrir com sucesso estruturas ocultas nos dados, tarefas que os algoritmos lineares não são capazes de resolver, tal como a PCA. O algoritmo *t-SNE* calcula uma medida de similaridade entre pares de instâncias no espaço de alta dimensão e no espaço de baixa dimensão. Este procura otimizar essas duas medidas de similaridades usando uma função de custo (LOPES; NETO; MARTINS, 2020) (POTRATZ et al., 2021) (MAATEN; HINTON, 2008). O algoritmo *t-SNE* difere da PCA nas seguintes questões: é uma técnica

de redução de dimensionalidade não-linear; tenta preservar a estrutura local de dados, ou seja, do *cluster*; pode lidar com *outliers*; é um algoritmo não determinístico ou aleatório. O *t-SNE* possui hiperparâmetros tal como perplexidade, taxa de aprendizado e número de interações. A perplexidade é tal como o número de vizinhos mais próximos.

3 MATERIAIS E MÉTODOS

Neste capítulo serão apresentados os materiais e métodos a serem utilizados no desenvolvimento de estudos sobre ferramentas de *machine learning* em problemas relacionados ao comércio eletrônico, para extração de características de textos não-estruturados (presente em *sites* ou digitados por usuários), a partir de técnicas de processamento de linguagem natural (PLN). Esta abordagem abrange aplicações relacionadas à extração de entidades de produtos utilizando técnicas do tipo *few-shot learning*.

3.1 Materiais

Nas subseções a seguir são descritos os materiais utilizados.

3.1.1 Base de Dados

As bases de dados utilizadas são provenientes de plataforma de comércio eletrônico fornecidas por uma empresa[4] parceira sediada em Belo Horizonte que trabalha com tecnologias cognitivas. A primeira base de dados é constituída por 394 amostras (ofertas de utilidades domésticas) divididas em 34 classes. Estas ofertas são provenientes de classes novas que não participaram do processo de treinamento do classificador em operação na empresa. A descrição dessa base de dados pode ser encontrada na Tabela 3.1. Conforme pode ser observado, trata-se de uma base bastante desbalanceada e que possui poucas amostras.

Tabela 3.1 – Base de dados com 34 classes novas

Descrição da Base	**Valores**
Número de classes	34
Número de amostras	394
Classes com 1 Amostra	13
Classes com 2 amostras	5
Classes com 3 a 8 amostras	10
Classes com 20 a 60 amostras	5
Classes com 113 amostras	1
Número de características	1200

Fonte: Do Autor (2020)

As amostras da primeira base foram obtidas a partir da extração dos valores de saída dos neurônios da última camada intermediária da rede neural do classificador de produtos em

[4] Omnilogic Inteligência S/A.

operação na empresa. Como esta camada possui 1200 neurônios, as ofertas em linguagem natural são transformadas em vetores de dados numéricos com 1200 valores, que são considerados como as características de cada oferta para a aplicação dos classificadores *few-shot* propostos nesta Dissertação. Neste caso, como a base contém 394 ofertas têm-se 394 vetores de 1200 características, ou seja, um vetor numérico que representa cada oferta em linguagem natural.

A segunda base de dados é constituída por 3120 amostras (ofertas gerais) divididas em 312 classes. A descrição dessa base de dados pode ser encontrada na Tabela 3.2. As ofertas desta base de dados também são provenientes de classes novas que não participaram do processo de treinamento do classificador em operação na empresa. As amostras da segunda base foram obtidas da mesma forma que da primeira base, a diferença que para a segunda base é utilizado um classificador mais recente que teve sua camada intermediária atualizada e otimizada contendo 1000 neurônios e, portanto, fornecendo 1000 características.

Tabela 3.2 – Base de dados com 312 classes novas

Descrição da Base	**Valores**
Número de classes	312
Número de amostras por classe	10
Número de amostras	3120
Número de características	1000

Fonte: Do Autor (2021)

3.1.2 Ferramentas: Modelos e *Softwares*

Para a implementação dos algoritmos foi utilizada a plataforma Anaconda. Esta é uma plataforma *open source* que contém as IDEs (Ambiente de Desenvolvimento Integrado) *Spyder* e *Jupyter* que foram utilizados na construção dos algoritmos em linguagem *python*. O computador utilizado foi da marca *Dell* (Dell Inspiron, processador Intel I5, 8 GB, HD 1 TB, SSD 240 GB) com sistema operacional *linux*, fornecido pela empresa parceira.

No projeto também foi utilizada a IDE de *python google colab PRO*. A vantagem desta IDE é poder executar os algoritmos na nuvem, não sendo necessária a instalação de bibliotecas e também pode-se utilizar GPU (*Graphics Processing Units*) ou TPU (*Tensor Processing Units*). Ela permite também ter aplicações remotas e com mais de um desenvolvedor ao mesmo tempo, sendo bom para trabalhos em equipe.

Para a implementação dos algoritmos de processamento de linguagem natural e aprendizagem de máquina, foram utilizadas diversas bibliotecas *python*. As principais bibliotecas

são as seguintes: *pandas* que foi utilizada para trabalhar a manipulação das bases de dados; *math* e *numpy* para a realização de cálculos numéricos; *matplotlib* e *seaborn* para plotagem de gráficos; *t-SNE* para visualização de amostras em duas dimensão, *string* para manipulação de strings, *random* para geração de números aleatórios e *tqdm* para criar barra de progresso para uma maior interação com o código.

Para a classificação de entidades via algoritmos *few-shot learning* foram utilizadas bibliotecas *open source*, tais como, *tensorflow*, *keras* e *pytorch*. Estas bibliotecas são as mais utilizadas para a implementação de redes neurais em *python*. Para tarefas como a extração de características via *word2vec* foi utilizada a biblioteca *gensim*. Esta implementação permite a configuração do modelo em diversos parâmetros.

3.2 Métodos

Nas subseções a seguir é abordada a metodologia para a extração de entidades pertencentes às classes novas em base de dados não estruturadas utilizando técnicas *few-shot learning*. Desta forma, serão apresentadas as propostas das arquiteturas de aplicação das redes *few-shot learning* para comércio eletrônico, tal como o algoritmo KNN, as redes *matching*, redes *matching* com *encoder* Bi-LSTM; redes neurais de grafos DPGN e também aplicação de redução de dimensão.

3.2.1 Ferramentas de Análises Estatísticas

Neste trabalho foram utilizados os métodos de validação cruzada *leave one out* e *k-fold* para a avaliação (SCHREIBER et al., 2017). O *leave one out* é um método de computação intensiva onde todas as amostras da base são analisadas. O *k-fold* ao invés de ser utilizado uma amostra por vez tal como o *leave one out*, é utilizado um subconjunto da base de dados para compor o conjunto de teste. No conjunto de treino são utilizados k subconjuntos de mesma dimensão que o conjunto de teste.

Para o cálculo de desempenho em problemas de classificação é bastante comum a utilização de matriz de confusão. Trata-se de uma tabela que permite a visualização do desempenho do algoritmo de aprendizagem de máquina e indica a classificação para cada classe do problema. Ela indica acertos e erros de cada classe comparando com o resultado esperado. Essa matriz corresponde a uma tabulação cruzada entre a classificação predita pelo algoritmo e a classificação real, conforme pode ser visto na Tabela 3.3.

Tabela 3.3 – Matriz de confusão

	Classe predita- A	Classe predita- B
Classe Real- A	Verdadeiro positivo	Falso negativo
Classe Real- B	Falso Positivo	Verdadeiro negativo

Fonte: Do Autor (2020)

Considerando problema de duas classes **A** e **B**, a classe **A** é a que esta sendo buscada. Desta forma, tem-se as seguintes definições:

- Positivo (P) = refere-se à classe **A** que está sendo buscada.
- Negativo (N) = refere-se à classe **B** que não está sendo buscada.
- Verdadeiro Positivo(TP): o sistema previu corretamente a classe **A**.
- Falso Positivo (FP): erro em que o sistema previu incorretamente a classe **A**, sendo que a classe real era a **B**.
- Falso Negativo (FN): erro em que o sistema previu incorretamente a classe **B** sendo que a classe real era a classe **A**.
- Verdadeiro Negativo (TN): o sistema previu corretamente a classe **B**.

Baseado na matriz de confusão, pode-se identificar os problemas de ajustes do sistema, como a análise de desempenho de classificadores em PLN, podendo ser utilizadas as seguintes métricas:

- Acurácia: calcula a porcentagem de acertos e é dada pela razão do total de verdadeiros positivos (TP), mais verdadeiros negativos (TN) e pelo total das amostras. Esta é uma boa indicação geral de desempenho do modelo, porém, falha na detecção de casos específicos. Ela é calculada pela seguinte equação:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

- *Recall*: indica a relação entre as previsões positivas realizadas corretamente e o total das previsões positivas. Dentre todos positivos reais existentes, identifica quais foram preditos corretamente. O *recall* é utilizado em uma situação em que os Falsos Negativos são considerados mais prejudiciais que os Falsos Positivos, sendo calculado por:

$$R = \frac{TP}{TP + FN} \tag{3.2}$$

- Precisão: indica a relação entre as previsões positivas realizadas corretamente e todas as previsões positivas (incluindo as falsas). Indica a eficácia do método, pois dentre as classificações positivas é possível identificar a porcentagem de quais de fato são positivas. A precisão é utilizada em uma situação em que os Falsos Positivos são considerados mais prejudiciais que os Falsos Negativos. Ela é calculada pela seguinte equação:

$$P = \frac{TP}{TP + FP} \tag{3.3}$$

- F1-*score*: utiliza a precisão e *recall* para efetuar o cálculo. Este é realizado por meio da média harmônica da precisão e do *recall*, sendo que atinge seu melhor valor em 1 e a pior pontuação em 0. Este é um valor único que indica a qualidade geral do modelo, de acordo com a seguinte equação:

$$F1 = \frac{2\,P\,R}{P + R} \tag{3.4}$$

Os resultados dos algoritmos *few-shot learning* com validação cruzada *leave one out* foram obtidos utilizando acurácia e o com o *k-fold* foram obtidos utilizando acurácia média e desvio padrão dos *folds*. Os resultados dos algoritmos utilizando o método *k-fold* tiveram as acurácias dos dez *folds* comparadas utilizando-se o teste-*t* do *MatLab*® adotando-se o nível de 5% de probabilidade.

3.2.2 Redução de Dimensão

Nesta seção será utilizado a redução de dimensão motivado com a intenção de facilitar a parametrização e melhorar o treinamento dos *encoders* das redes *few-shot learning*. Sendo assim, serão aplicadas duas abordagens para a redução de dimensão: seleção de características com *Fisher's Discriminant Ratio* (FDR) e transformação das características com *Principal Component Analysis* (PCA).

O algoritmo da PCA apresenta como entradas as características da base de dados e a variância original a ser mantida. É a partir da escolha da variância que o algoritmo obtém o número de características. A escolha da variância é de 0 a 100%, sendo que quanto menor for

a variância escolhida maior é a redução do número de características. Desta forma, o algoritmo define o número de características de acordo com a variância da base, ou seja se a base de dados for mais correlacionada a tendência é que o número de característica retornado seja bem menor do que o tamanho da base. O teste para identificar a variância que apresenta a maior redução de dimensão sem reduzir ou com pouca redução de acurácia foi utilizando o algoritmo KNN com distância do cosseno e k=1. O intervalo de teste para a PCA para a base de dados de 34 classes foi de 70 a praticamente 100% de variância. E para a base de dados com 312 classes foram com 50 a praticamente 100% de variância.

O FDR é um algoritmo ainda mais simples do que a PCA. Considerando um problema de duas classes C1 e C2, o FDR possui como entradas as características destas duas classes (para um problema multi-classe como é o caso deste trabalho é realizada a comparação de todas classes contra todas combinadas de duas a duas). O algoritmo então ordena por relevância as características que indicam uma maior separação entre estas classes. O número de características que apresenta o maior valor de acurácia foi escolhido utilizando o algoritmo KNN com distância do cosseno e k=1. Como o FDR apenas escolhe as características mais relevantes e não elimina a redundância entre as variáveis, foi utilizada a correlação linear em combinação. O FDR foi testado utilizando somente a base de dados de 34 classes com o número de características escolhido variando de 2 a 800. A partir de 800 características já não ocorre mudanças no resultado do KNN. A análise de correlação foi aplicada nas 800 características obtidas por FDR com limiar variando de 65 a 80%.

Neste trabalho também foi utilizado algoritmo de visualização de dados *t-SNE* (*t-Distributed Stochastic Neighbor Embedding*). Como trata-se de um problema de alta dimensão, o objetivo é visualizar as amostras em duas dimensões para verificar se as características obtidas são realmente interessantes para separação das novas classes de produtos. É recomendável usar antes do *t-SNE* outro método de redução de dimensionalidade, tal como a PCA para reduzir a dimensão dos dados afim de suprimir ruídos e acelerar os cálculos de distâncias. Os parâmetros utilizados no *t-SNE* foram: número de componentes igual a dois, distância do cosseno, perplexidade igual a 40 e 800 interações.

3.2.3 Classificadores em Comércio Eletrônico

Nos sistemas supervisionados os classificadores fazem a comparação das predições com a classificação original da base de dados e analisa estatisticamente os acertos. Cada classifi-

cador possui uma forma peculiar de realizar esta tarefa, caso o classificador utilize a etapa de treinamento, os erros são retro-propagados para geração de aprendizado e uma nova representação dos dados. É importante ressaltar que também existem classificadores que não utilizam treinamento, como o KNN, que apenas utiliza métricas de distâncias para fazer a predição.

Os sistemas em operação nas plataformas de comércio eletrônico utilizam todas as etapas descritas na Seção 2.2.4 para a classificação de produtos utilizando algoritmos de aprendizado profundo. Estes algoritmos trabalham com milhares de entidades e milhões de ofertas. Porém, não são capazes de classificar produtos pertencentes à classes novas que não participaram do treinamento e nem de bases com poucas amostras. Para as classes novas que possuem ofertas suficientes, estas são acrescentadas à base de dados, onde é realizado novo treino. Como diariamente chegam classes novas, o sistema deve ser constantemente retreinado com todas estas milhões de ofertas. Em razão disso foram propostos métodos de classificação de produtos de bases de dados contendo somente classes novas e com poucas amostras utilizando técnicas *few-shot learning*.

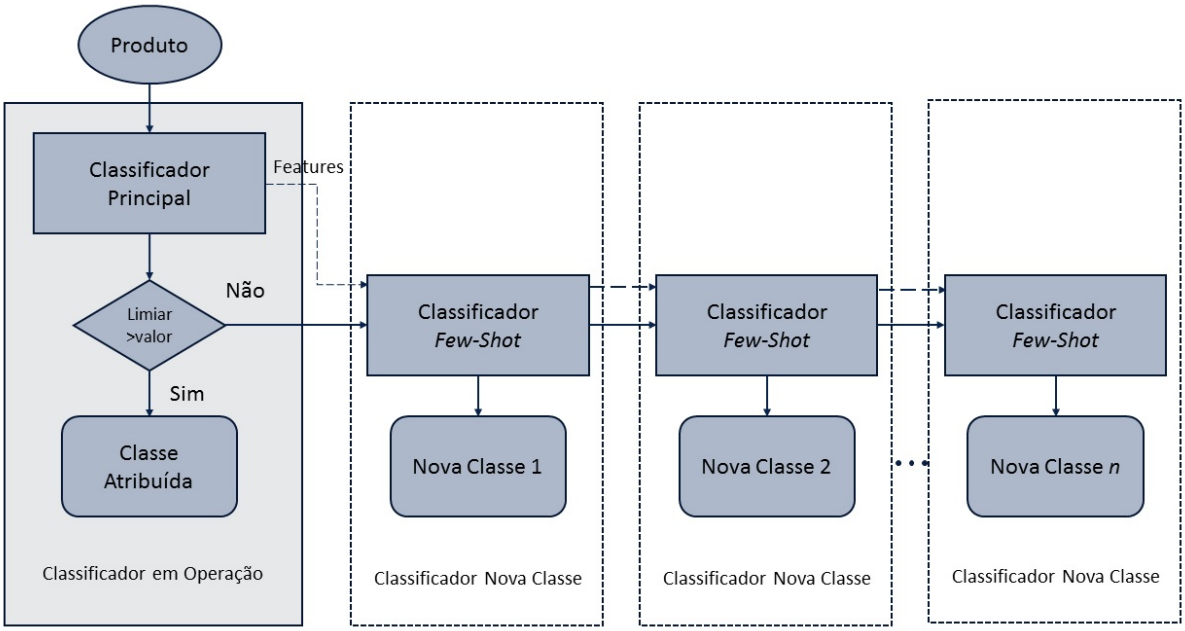
A classificação de produtos empregando as técnicas *few-shot learning* foi realizada utilizando características previamente extraídas da rede da empresa parceira (processo de *transfer learning*), exceto para as redes *matching* com *encoder* Bi-LSTM que foram utilizados dados descritos em linguagem natural.

3.2.4 Classificador *Few-Shot Learning* Proposto

O objetivo deste trabalho é buscar uma solução para a extração de entidades de classes novas de produtos sem a necessidade de retreinar todo o modelo atualmente em operação. A Figura 3.1 contém o esquema do classificador baseado em *few-shot learning* proposto. O classificador principal, representado na Figura 3.1 é o classificador atualmente em operação na empresa parceira, este trabalha com toda a base de dados pré-existente. O classificador *few-shot* (Figura 3.1) é o classificador proposto para realizar a classificação das classes novas que não estavam presentes no processo de treinamento do classificador principal.

Supondo que um produto seja de uma classe não conhecida pelo atual classificador em operação, ou seja, trata-se de uma classe nova, o objetivo é classificá-lo em uma nova classe considerando que esta possua poucas amostras. Como o classificador em operação foi treinado a partir de um número grande de amostras e classes, espera-se que ele seja capaz de extrair características relevantes de uma oferta, mesmo em classes não presentes em seu conjunto de

Figura 3.1 – Modelo de classificação *few-shot learning*



Fonte: Do Autor (2020)

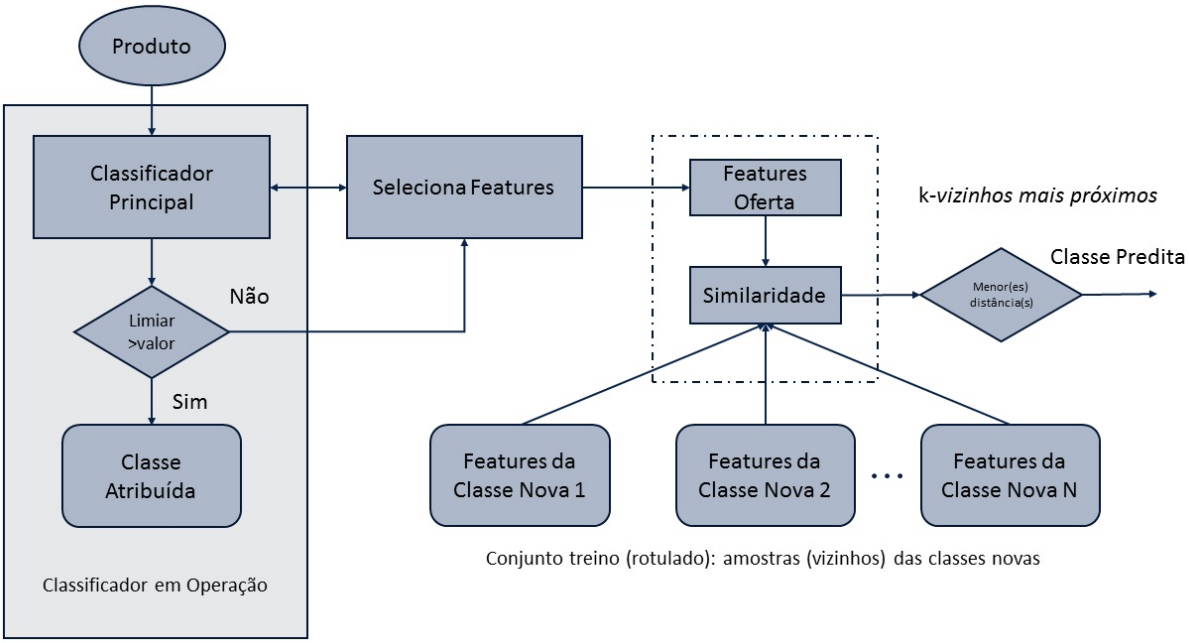
treino. Assim, os novos classificadores são obtidos a partir das características (*features*) extraídas do classificador em operação, não necessitando retreinar este classificador. Além disso, os classificadores para classes novas podem ser mais simples, pois já trabalharão com caraterísticas pré-definidas e supostamente com boa capacidade para separar as novas ofertas.

Nesse sentido, as abordagens estudadas foram o algoritmo KNN, as redes *matching* e as redes DPGN, conforme apresentado nas subseções a seguir.

3.2.5 Aplicação do Algoritmo *k*-NN

O algoritmo KNN é implementado como classificador *few-shot* para os bancos de dados apresentados na Seção 3.1.1. A aplicação do algoritmo KNN no problema em estudo é apresentado na Figura 3.2. Observa-se que, após a extração das características do produto pelo classificador principal, as características da nova oferta a ser classificada é comparada (medida de similaridade) com as características das amostras já existentes no banco de dados e o algoritmo KNN é então implementado. Neste caso, portanto, o KNN tem a vantagem, na fase de projeto, de dispensar treinamento.

Figura 3.2 – *Framework* de aplicação do algoritmo KNN



Fonte: Do Autor (2020)

3.2.5.1 Parâmetros do Algoritmo *k*-NN para a Base de Dados de 34 Classes

A avaliação de desempenho do KNN para a base de dados de 34 classes (Seção 3.1.1, Tabela 3.1) foi pelo método de validação cruzada *Leave-One-Out*. A validação dos experimentos com a utilização do método de validação cruzada do tipo *Leave-One-Out* é aconselhável, pois a base de dados possui poucas amostras.

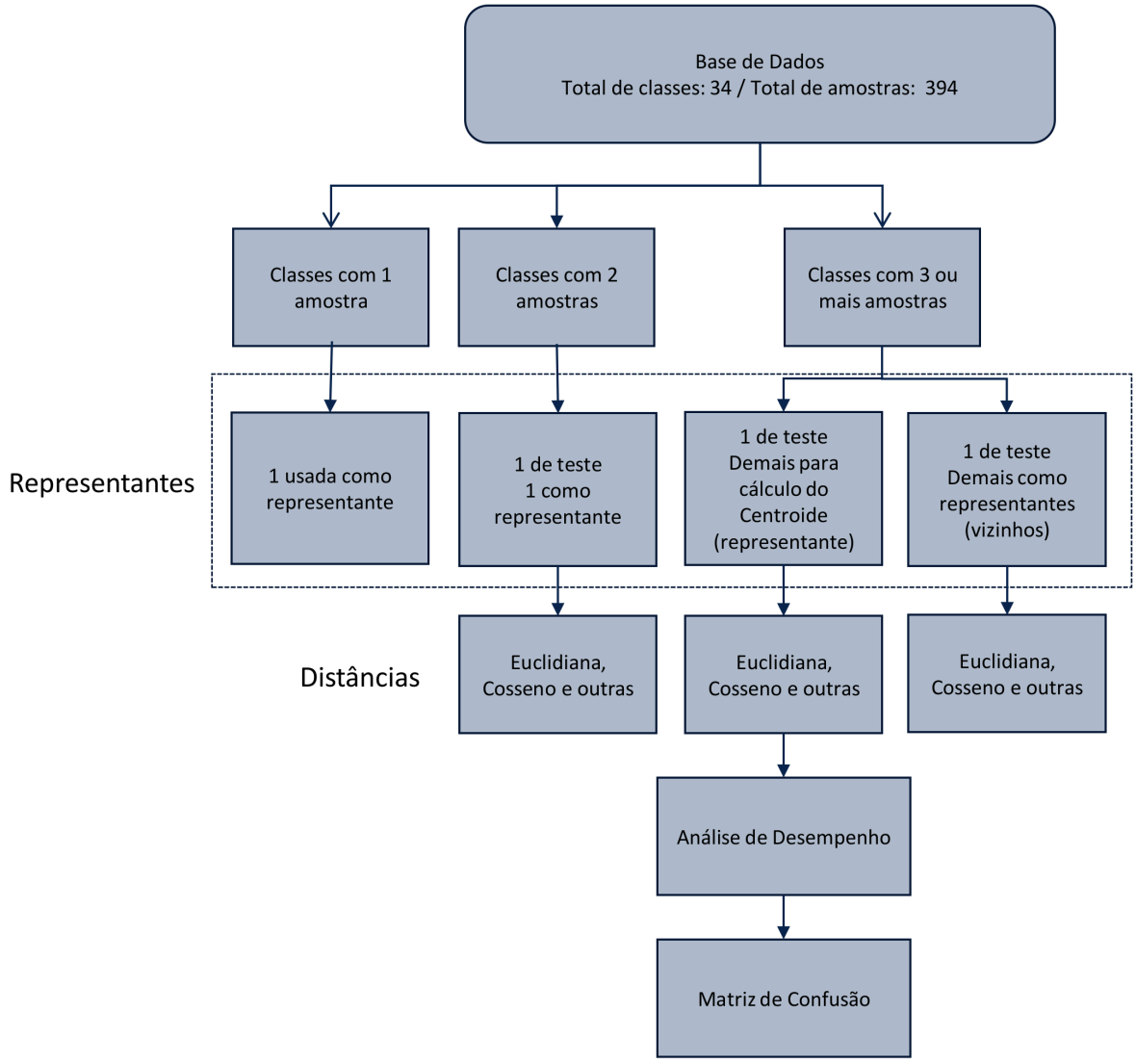
A configuração dos testes executados é apresentada na Figura 3.3. As classes que possuem apenas uma amostra foram colocadas na base de treino apenas para tornar o problema mais complexo. As que possuem duas amostras, uma é usada para teste (*Leave-One-Out*) e a outra como representante. Nas classes com três ou mais amostras, uma é utilizada para teste (*Leave-One-Out*) e as demais utilizadas como representantes ou para o cálculo do centroide que será utilizado como representante da respectiva classe. As distâncias utilizadas foram: *Euclidiana, cosseno, Manhattan, Hassanat, Lorentzian, Clark* e *Mahalanobis*. Também foram testados diferentes valores do parâmetro *k*.

3.2.5.2 Parâmetros do Algoritmo *k*-NN para a Base de Dados de 312 Classes

Os testes do algoritmo *k*-NN para a base de dados com 312 classes (Seção 3.1.1, Tabela 3.2) foram utilizando validação cruzada *k*-fold. Considerando que as classes da base de dados tem 10 amostras, então foi possível de fazer a divisão dos testes em dez *folds*. Então para

cada um dos dez *folds* a base de teste foi composta por 1 amostra de todas as 312 classes (312 amostras) e a base de representantes foi formada pelas 9 amostras restantes de todas as 312 classes (2808 amostras). As distâncias utilizadas foram: *Cosseno, Euclidiana, Manhattan, Hassanat, Lorentzian* e *Clark*. Também foram testados diferentes valores do parâmetro *k*.

Figura 3.3 – Fluxograma do teste de validação cruzada com KNN



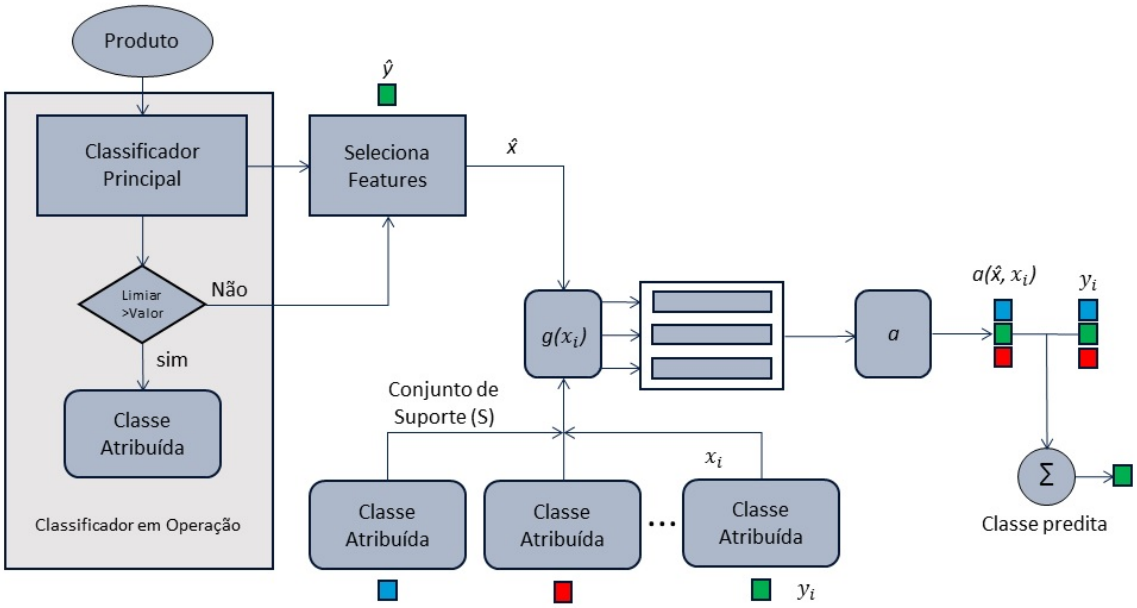
Fonte: Do Autor (2020)

3.2.6 Aplicação das Redes *Matching*

O *framework* de aplicação das redes *matching* pode ser visto na Figura 3.4. À esquerda pode ser observado o classificador principal atualmente em operação e à direita a arquitetura

das redes *matching* e suas respectivas etapas. Embaixo tem-se o conjunto de suporte (*S*) que é o conjunto treino, este, por sua vez, tem a mesma função dos representantes (vizinhos) do KNN.

Figura 3.4 – *Framework* de aplicação das redes *matching*



Fonte: Do Autor (2021)

As classes que o classificador em operação detectar como novas vão para as *matching* para serem preditas. A tarefa de predição segue todas as etapas conforme descrito na seção 2.9, ou seja, passagem dos dados pelos *encoders*, cálculo de similaridade e aplicação do mecanismo de atenção que pondera a predição da amostra pertencente a classe nova. Neste trabalho é utilizado somente o *encoder* g, não foi utilizado o *encoder* f. Este é recomendado para casos onde existe contexto entre as amostras, e não é o caso das amostras das bases de dados utilizadas.

No algoritmo das redes *matching* são utilizados quatro tipos de variáveis como entradas, conforme apresentado na Tabela 3.4. A saída é o *label* do *target x* predito pela rede. Na parametrização das redes *matching*, além dos parâmetros tradicionais tais como os existentes na MLP, existem três parâmetros conforme pode ser visto na Tabela 3.5. Em cada posição do lote é realizado um teste de similaridade entre o "conjunto de suporte" (*suporte set x*) e o alvo (*target x*) com a distância do cosseno. O parâmetro "classes por conjunto de suporte" é o número de classes utilizadas no teste de similaridade realizado em cada posição do lote. O parâmetro "amostras por classe" é bem intuitivo, pois são os números de amostras de cada uma das classes que vão compor o lote, é algo similar ao número de representantes do KNN. Quanto mais "amostras por classe" (representantes de classe) e maior o tamanho do lote, maior é o número de cálculos de distâncias.

Tabela 3.4 – Variáveis das redes *matching*

Entradas	**Descrição**
Suporte Set x	Vetor de dados (vizinhos)
Suporte Set y	Vetor *one-hot encoded* referente aos *labels* do *Suporte Set x*
Target $\hat{x}$	Vetor de dados do alvo (amostra testada)
Target $\hat{y}$	*Label* original do alvo *target* $\hat{x}$

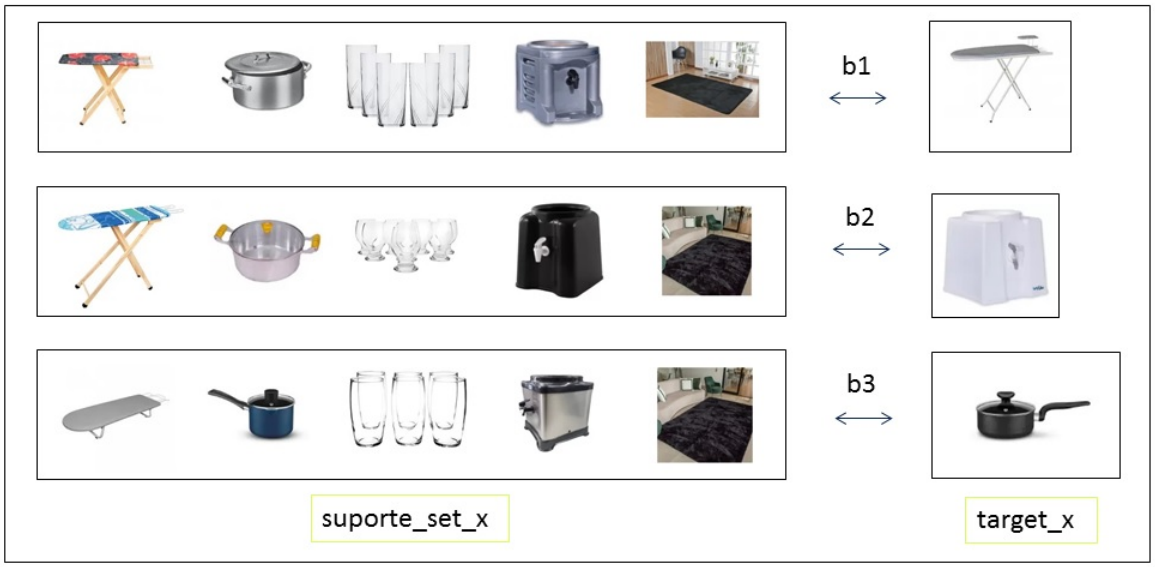
Fonte: Do Autor (2021)

Tabela 3.5 – Descrição dos parâmetros das redes *matching*

Parâmetros	**Descrição**
Classes por conjunto de suporte	Número de classes em cada posição do lote
Amostras por classe	Número de amostras por classe
Lote	Número execuções para atualizar a rede

Fonte: Do Autor (2021)

Figura 3.5 – Exemplo de um lote



Fonte: Do Autor (2021)

Um exemplo de lote de tamanho 3 com as posições representadas por [b1,b2,b3] pode ser visto na Figura 3.5. Neste lote tem 5 classes de produtos diferentes, e cada classe contém somente um representante (1 amostra por classe). Sendo assim, em cada posição do lote são realizados 5 cálculos de similaridades. Cada posição do lote é obtida de forma aleatória com as amostras disponíveis da base.

3.2.6.1 Parâmetros das Redes *Matching* para a Base de Dados de 34 Classes

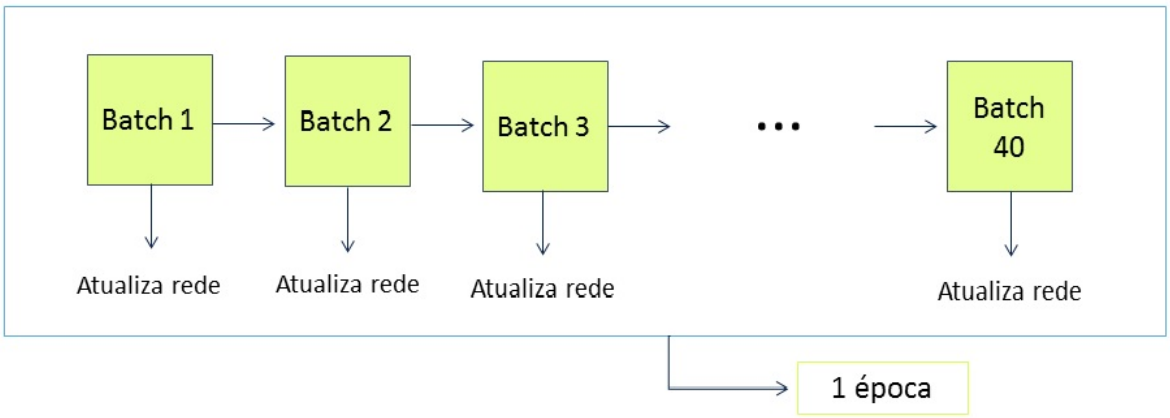
A avaliação das redes *matching* para a base de dados de 34 classes (Seção 3.1.1, Tabela 3.1) foi realizada pelo método de validação cruzada *leave one out* (SCHREIBER et al., 2017). Foram utilizadas 307 características reduzidas por PCA e as 1200 características originais. A escolha dos parâmetros de treinamento foi realizada dentro da faixa descrita na Tabela 3.6.

Tabela 3.6 – Faixa de escolha de parâmetros das redes *matching*

Parâmetros	**Valores**
Intermediária *encoder* (g)	10 a 600
Taxa de aprendizado	0,1 a 0,00001
Classes por conjunto de suporte	5 a 34
Amostras por classe	1 a 111
Lote	2 a 20
Épocas	15 a 400

Fonte: Do Autor (2021)

Figura 3.6 – Época nas redes *matching*



Fonte: Do Autor (2021)

A Figura 3.6 ilustra um exemplo de uma época e os correspondentes lotes. O tamanho de uma época foi considerado de forma que todas as amostras da base fossem abrangidas nos cálculos de similaridades durante o treinamento. Considerando que o lote tem tamanho 10, então sortear 40 lotes já é mais do que suficiente para que as 381 amostras da base sejam abrangidas em uma época. Então uma época é dada pela repetição do sorteio dos lotes 40 vezes. A atualização dos pesos acontece após o término do cálculo de cada um dos lotes. A acurácia é dada pela média dos lotes.

Devido à base de dados ser formada por características previamente extraídas pelo classificador, foi realizada a substituição da rede convolucional (*embedding* g) por uma MLP. Na Tabela 3.7 pode ser vista a configuração da rede MLP que apresentou o melhor desempenho para o treinamento *leave one out*. A escolha do número de neurônios da entrada e da saída é condicionada pelo número de características da base de dados utilizada (307 ou 1200). Apenas uma camada intermediária foi utilizada na arquitetura da MLP. Na Tabela 3.8 pode-se ver os parâmetros das redes *matching* que apresentaram o melhor resultado *leave one out* para a base de 34 classes.

Tabela 3.7 – Parâmetros do *encoder* MLP das redes *matching*

Camadas	**Neurônios (307 carac.)**	**Neurônios (1200 carac.)**	**Função**
Entrada	307	1200	linear
Intermediária	100	100	tanh
Saída	307	1200	tanh

Fonte: Do Autor (2021)

Tabela 3.8 – Parâmetros escolhidos das redes *matching*

Parâmetros	**307 caract.**	**1200 caract.**
Função de erro	CrossEntropyLoss	CrossEntropyLoss
Método de optimização	Adam	Adam
Taxa de aprendizado	0,001	0,0001
Classes por conjunto de suporte	16	16
Amostras por classe	2	2
Lotes	10	10
Épocas	50	80

Fonte: Do Autor (2021)

3.2.6.2 Parâmetros das Redes *Matching* para a Base de Dados de 312 Classes

A avaliação das redes *matching* para a base de 312 classes (Seção 3.1.1, Tabela 3.2) foi realizada pelo método de validação cruzada *k-fold* na mesma condição do algoritmo KNN. Foram utilizadas as 1000 características originais. A escolha dos parâmetros de treinamento das redes *matching* foi realizada dentro da faixa descrita na Tabela 3.9.

Para avaliar os parâmetros, foram realizados testes utilizando o primeiro *fold* da base (o primeiro *fold* que é constituído pela primeira amostra de todas as classes da base de dados, pelo fato de possuir 312 classes o teste é realizado com 312 amostras). Após obtido bons resultados para o primeiro *fold* foram utilizados todos os dez *folds* na obtenção dos parâmetros,

conforme pode ser visto no Apêndice A. Na Tabela 3.10 pode ser vista a configuração da rede MLP que apresentou o melhor desempenho para o treinamento *k-fold*. Apenas uma camada intermediária foi utilizada na arquitetura da MLP. Na Tabela 3.11 pode-se ver os parâmetros das redes *matching* que apresentaram o melhor resultado *k-fold*.

Tabela 3.9 – Faixa de escolha de parâmetros das redes *matching* para a base 312 classes

Parâmetros	**Valores**
Intermediária *encoder* (g)	10 a 1000
Taxa de aprendizado	0,1 a 0,00001
Classes	5 a 312
Amostras por classe	1 a 8
Lote	2 a 20
Épocas	20 a 400

Fonte: Do Autor (2021)

Tabela 3.10 – Parâmetros do *encoder* MLP das redes *matching*

Camadas	**Neurônios**	**Função**
Entrada	1000	linear
Intermediária	450	tanh
Saída	1000	tanh

Fonte: Do Autor (2021)

Tabela 3.11 – Parâmetros escolhidos das redes *matching*

Parâmetros	**Valores escolhidos**
Função de erro	CrossEntropyLoss
Método de optimização	Adam
Taxa de aprendizado	0,0001
Classes por conjunto de suporte	156 e 312
Amostras por classe	2
Lotes	12
Épocas	80

Fonte: Do Autor (2021)

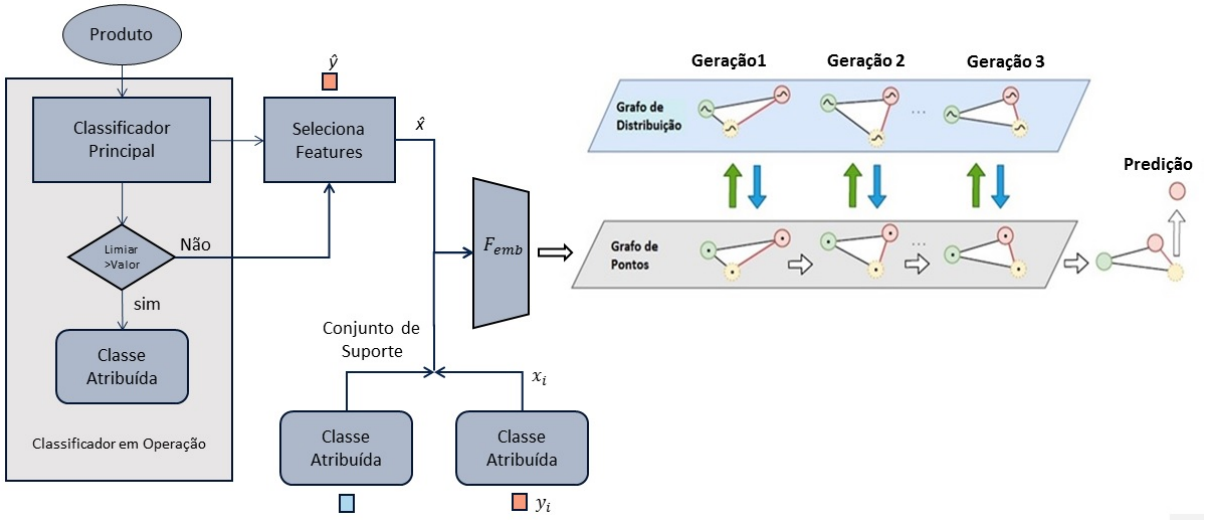
O treinamento das redes *matching* foi realizado utilizando 156 classes e 312 classes em uma mesma época (ou seja, em uma linha de código é executado a função de treinamento com 156 classes (40 lotes) e em outra linha de abaixo no código é executado a função de treinamento com 312 classes (5 lotes por época)). As redes *matching* utilizando o *pytorch* permite essa troca de parâmetros com código em execução. O objetivo de realizar o treinamento desta forma com duas configurações foi reduzir tempo de processamento, pois as *matching* com menos

classes executa bem mais rápido e aprende mais rápido devido à ocorrência de maior número de testes de similaridade. O uso de 312 classes é para que a rede aprenda a fazer os cálculos de similaridade de forma parecida tal como são realizados nos testes da rede.

3.2.7 Aplicação das Redes *DPGN*

A aplicação das redes DPGN para o problema em estudo será conforme o *Framework* da Figura 3.7. À esquerda da figura pode ser observado o classificador principal atualmente em operação e à direita a arquitetura das redes DPGN com seu *encoder* F_{emb} e a dupla arquitetura: grafo de pontos e grafo de distribuição.

Figura 3.7 – *Framework* de aplicação das redes *DPGN*



Fonte: Adaptada de Yang et al. (2020)

Isto posto, o classificador em operação é responsável pela tarefa de extrair as características e identificar as classes novas que serão encaminhadas para o *encoder* das redes DPGN. Os dados do *encoder* são encaminhados para o "grafo de pontos" que faz a operação P2D "ponto para distribuição", ou seja, o "grafo de pontos" processa os dados e os encaminha para o "grafo de distribuição". O "grafo de distribuição" faz operação D2P "distribuição para ponto" do qual a rede de distribuição processa os dados e os devolve para o "grafo de pontos". Este processo é repetido por algumas gerações até que o "grafo de pontos" faz as predições das classes novas testadas.

Nas redes DPGN são utilizadas quatro variáveis de entrada, similar às redes *matching*, conforme descrito na Tabela 3.12. As redes DPGN possuem um *encoder* f_{emb} que, tal como o extrator de *embending* g das redes *matching*, são originalmente uma rede convolucional (CNN)

com 4 camadas. Para simplificar a implementação foi realizada a substituição desta CNN f_{emb} por uma MLP.

Na parametrização das redes DPGN, além dos parâmetros tradicionais tais como os existentes na MLP, existem outros parâmetros que estão listados na Tabela 3.13. Os parâmetros "num way" e "num shot" são equivalentes aos respectivos parâmetros utilizados pelas redes *matching*, "classes por conjunto de suporte" e "amostras por classe". O lote (*batch*) nas DPGN é um pouco diferente do utilizado nas *matching*, pois para cada amostra utilizada como representante tem-se uma amostra da mesma classe para ser o alvo (*query*), por exemplo, se tiver 5 classes no "suporte data" (num way=5), tem-se também 5 classes como "query data".

Tabela 3.12 – Variáveis das redes DPGN

Entradas	**Descrição**
Suporte data	Vetor de dados (vizinhos)
Suporte label	Vetor *one-hot encoded* referente aos labels do suporte data
Query data	Vetor de dados dos alvos (nós testados)
Query label	Label original dos alvos query data

Fonte: Do Autor (2021)

Tabela 3.13 – Descrição dos parâmetros das redes DPGN

Parâmetros	**Descrição**
Gerações	Número de gerações utilizadas pelas DPGN
Distância	Métrica de distância escolhida para obtenção das arestas
Num way	Número de classes em cada posição do lote
Num shot	Número de amostras por classe
Lote (batch)	Número execuções para atualizar a rede
Interações	Número de atualizações
Ajuste de erro	Número de interações para iniciar o ajuste automático de erro
lr-adj-base	Taxa com que o erro vai decaindo no ajuste de erro

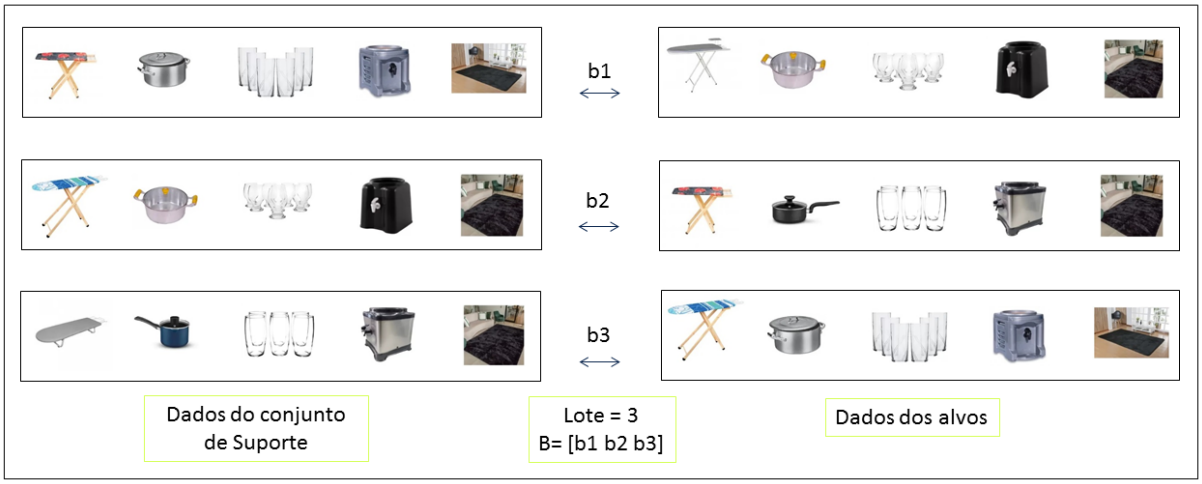
Fonte: Do Autor (2021)

Um exemplo de um lote pode ser visto na Figura 3.8. Este lote é de tamanho 3 conforme representado por [b1, b2, b3], ele possui 5 classes (num way=5) de produtos diferentes e cada classe contém somente um representante (num shot=1). Cada posição do lote é obtido de forma aleatória com as amostras disponíveis da base. A atualização dos pesos acontece após o término do cálculo de cada um dos lotes, sendo que cada vez que a rede é atualizada considera-se uma iteração.

Existe diferença no cálculo de similaridade das redes DPGN em relação às redes *matching*, conforme pode ser visto na Figura 3.9. Nesta figura tem-se um exemplo com 5 classes

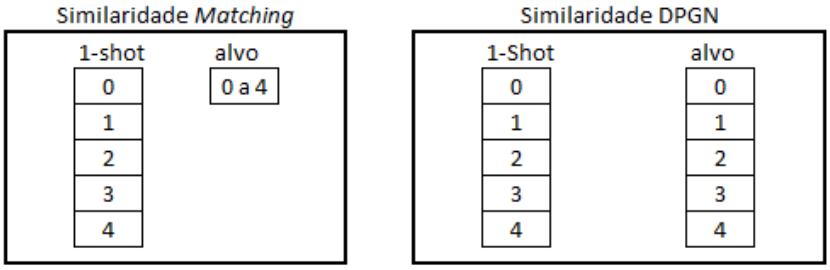
(“num way” = 5 e “num shot” = 1). As redes *matching* só possuem um alvo, ao contrário das DPGN que tem 5 alvos para treinamento com “num way” = 5. Por isso, nesta condição, em uma posição do lote das redes *matching* são realizados 5 cálculos de distância, enquanto nas redes DPGN são realizados 25 cálculos de distância no “grafo de pontos” e 25 cálculos de distância no “grafo de distribuição” em uma única geração, sendo que normalmente são utilizados em torno de 5 gerações. Estes cálculos de similaridade faz com seja consumido muito memória RAM. Este fato não acarreta problemas para a base de dados com 34 classes, que por sua vez possui poucas classes, mas quando se tem maior número de classes tal como na base de dados com 312 classes o consumo de memória RAM se torna prejudicial.

Figura 3.8 – Lote das redes DPGN



Fonte: Do Autor (2021)

Figura 3.9 – Similaridade das redes DPGN (5-way e 1-shot)



Fonte: Do Autor (2021)

3.2.7.1 Parâmetros das Redes DPGN para a Base de Dados de 34 Classes

A avaliação das redes DPGN foi realizada pelo método de validação cruzada *leave one out* utilizando 307 características reduzidas por PCA e as 1200 características originais. A con-

figuração do *encoder* f_{emb} utilizada foi com uma camada intermediária para 1200 características originais e também para as 307 características reduzidas com PCA conforme pode ser visto na Tabela 3.14. A saída do *encoder* foi com 307 neurônios visto que as DPGN funcionam melhor com menor número de características (foi provado na Seção 4.1.3 que com 307 características consegue-se uma boa classificação, então este é um bom valor para ser utilizado na saída). Na Tabela 3.15 pode ser vista a faixa testada e os parâmetros que apresentaram o melhor resultado.

Tabela 3.14 – Parâmetros do *encoder* MLP das redes DPGN

Camadas	**Neurônios (307 carac.)**	**Neurônios (1200 carac.)**	**Função**
Entrada	307	1200	linear
Intermediária	100	100	tanh
Saída	307	307	tanh

Fonte: Do Autor (2021)

Tabela 3.15 – Parâmetros escolhidos das redes DPGN

Parâmetros	**Faixa testada**	**Valores escolhidos**
Função de erro	-	CrossEntropyLoss
Método de optimização	-	Adam
Gerações	2 a 8	5
Distância	$l1$ e $l2$	$l1$
Taxa de aprendizado	0,01 a 0,00001	0,001
Weight decay	0,0001 a 0,0000001	0,000001
Interações	50 a 1600	800
Ajuste de erro	50 a 1000 interações	500 interações
Num way	34	34
Num shot	1 a 8	2
Lote	4 a 20	10

Fonte: Do Autor (2021)

3.2.7.2 Parâmetros das Redes DPGN para a Base de Dados de 312 Classes

Conforme comentado na Seção 3.2.7 é de se esperar que as redes DPGN consumam muita memória RAM devido à forma como os cálculos de similaridades são realizados. E também devido à complexidade da arquitetura das redes que é constituída por: uma MLP no *encoder* e outra para obtenção dos nós do "grafo de distribuição"; três redes CNN de duas camadas sendo uma para obter as arestas de ponto, uma para as arestas de distribuição e uma para obter os nós do "grafo de pontos". E também as redes DPGN necessitam utilizar a mesma configuração do treino para realizar o teste, ou seja, o mesmo número de "classes por conjunto de

suporte" ("*num way*") e número de representantes por classe ("*num shot*") devem ser utilizados tanto no treino quanto no teste. Sendo assim para conseguir testar com 312 classes teria que treinar com 312 classes também. Isto é devido à dimensão do vetor do nó do "grafo de distribuição" ser condicionado pelo número de amostras utilizado no "conjunto de suporte" (Suport data). De maneira que este foi um fator acarretou limitação devido à quantidade de memória RAM utilizada para treinar as redes com 312 classes ser muito elevado. Sendo assim as redes geram o estouro da memória RAM do *google* Colab PRO (o estouro normalmente acontece na inicialização da rede). Desta forma os recursos computacionais não foram suficientes para realizar o treinamento e teste nesta condição.

Portanto o teste das DPGN foi realizado somente com uma parte da base dados. Foi escolhido para teste as 145 classes que o KNN com distância do cosseno (k=1) apresentado na Seção 4.2.2 apresentam erros. Ressaltando, que o teste é fixo com estas 145 classes escolhidas, porém no treinamento o "conjunto de suporte" das redes obrigatoriamente também é com 145 classes, mas que foram sorteadas das 312 classes presentes na base. Em vista disso, em cada posição do lote são utilizadas 145 classes diferentes dentre as 312 classes.

A avaliação das redes DPGN foi realizada pelo método de validação cruzada *k-Fold* utilizando a base de dados da Tabela 3.2. Foram utilizadas as 1000 características originais. A escolha dos melhores parâmetros foi utilizando os dez *folds* conforme pode ser visto na Apêndice A. Nos testes, a configuração do *encoder* f_{emb} das redes DPGN que apresentou o melhor desempenho para o treinamento *k-fold* pode ser observado na Tabela 3.16. Esta rede é uma MLP com uma camada intermediária com 450 neurônios. A entrada da rede MLP possui 1000 características, mas para favorecer o treinamento das DPGN a saída foi reduzida. Pois a saída da MLP é a entrada das DPGN que possuem três redes convolucionais de duas camadas, assim, se esta saída possuir um número muito elevado de neurônios, um elevado consumo de memória RAM será observado. O desempenho também fica comprometido pois maior é o número de parâmetros a serem treinados com uma entrada de dimensão alta.

Na Tabela 3.17 pode ser vista a faixa testada e os parâmetros que apresentaram o melhor resultado até o momento. Como não é possível de realizar o teste das DPGN com 8 representantes de classes tal como foi realizado no KNN, devido a alta requisição de RAM, assim sendo foi obtido a média das amostras da classe para ser o representante. Desta forma o treinamento pode ser realizado por exemplo com 1 "amostra por classe", e o teste também com 1 "amostra por classe" que neste caso é formada pelo centroide do *cluster*.

Tabela 3.16 – Parâmetros do *encoder* MLP das redes *DPGN*

Camadas	**Neurônios (1000 carac.)**	**Função**
Entrada	1000	linear
Intermediária	450	tanh
Saída	128	tanh

Fonte: Do Autor (2022)

Tabela 3.17 – Parâmetros escolhidos das redes DPGN

Parâmetros	**Faixa testada**	**Valores escolhidos**
Função de erro	-	CrossEntropyLoss
Método de optimização	-	Adam
Gerações	5	1
Distância	$l1$ e $l2$	$l1$
Taxa de aprendizado	0,01 a 0,00001	0,001
Weight decay	0,0001 a 0,0000001	0,000001
Interações	50 a 1600	800
Ajuste de erro	50 a 750 interações	500 interações
Num way	145	145
Num shot	6	2
Lote	4 a 20	6

Fonte: Do Autor (2022)

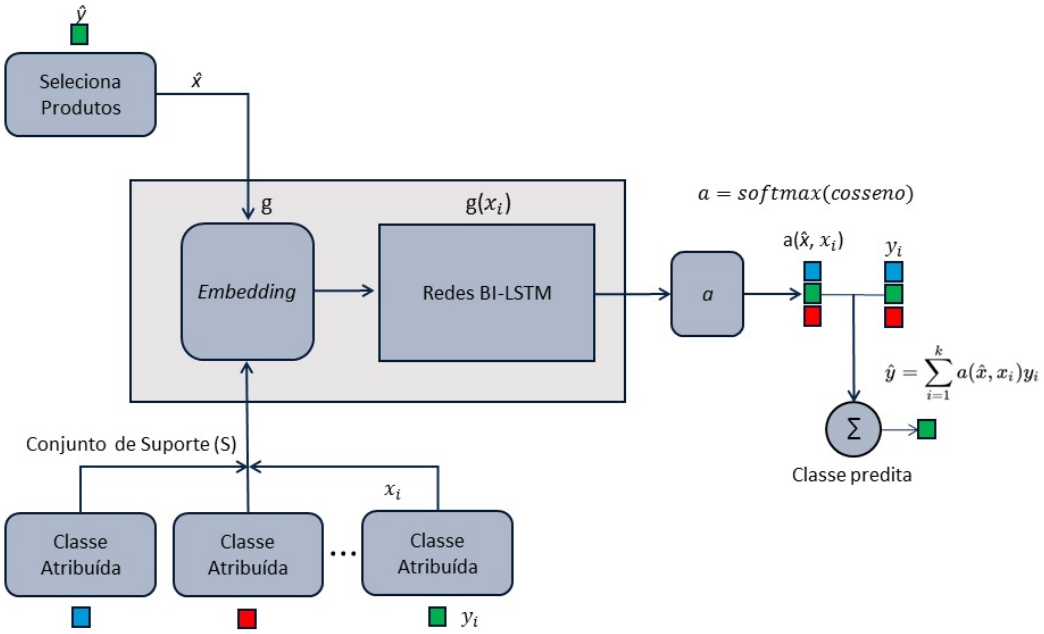
3.2.8 Redes *Matching* Empregando Redes BI-LSTM como *Encoder g*

Outra arquitetura proposta das redes *matching* para o problema de extração de entidades de classes novas de produtos foi utilizando rede Bi-LSTM como *encoder g* (*Matching-Encoder-BiLSTM*). Assim sendo, esta nova arquitetura é similar à apresentada na Seção 3.2.6, a diferença é que ao invés de utilizar uma MLP para receber os dados previamente extraídos por meio do processo de *transfer learning*, será utilizado dados em linguagem natural que serão interpretados pelo novo *encoder g* conforme a Figura 3.10.

O *encoder g* desta arquitetura *few-shot learning* proposta realiza todo o processo desde o recebimento dos dados descritos em linguagem natural e a interpretação dos mesmos. Os dados em linguagem natural antes de serem transformados em dados numéricos pelo *embedding*, passam por um cuidadoso pré-processamento para que sejam obtidas características de boa qualidade. As amostras em linguagem natural depois de pré-processadas são encaminhadas para o *embedding* que as transformam em dados numéricos conforme descrito na Seção 2.2.4. A interpretação destes dados numéricos ficam então a cargo das redes *Bi-LSTM* e camada de atenção de *bahdanau*. Em seguida, as característica obtidas por este *encoder* são então usadas

para realizar a predição da classe testada utilizando cálculo de similaridade conforme é efetuado pelas redes *matching*. A partir desta etapa o processo é idêntico ao descrito na Seção 3.2.6. Portanto, as classes novas vão para as *Matching-Encoder-BiLSTM* para serem preditas executando todas as operações descritas anteriormente.

Figura 3.10 – *Framework* de aplicação das redes *Matching-Encoder-BiLSTM*



Fonte: Do Autor (2022)

A avaliação das redes *Matching-Encoder-BiLSTM g* foi realizada pelo método de validação cruzada *k-fold* na mesma condição do algoritmo KNN. A base de dados utilizada foi a que contém 312 classes e 3120 amostras (Seção 3.1.1, Tabela 3.2). O *embedding* utilizado foi pré-treinado com uma base de dados de comércio eletrônico, sendo capaz de extrair características de palavras e dos caracteres dos textos dos produtos. A obtenção dos parâmetros foi utilizando o primeiro *fold* conforme pode ser visto na Apêndice A. A escolha dos parâmetros de treinamento destas redes foi realizada dentro da faixa descrita na Tabela 3.18. O número máximo de classes utilizado não é maior devido ocorrer estouro de memória RAM do Colab PRO quando aumenta o número de "classes do conjunto de suporte". Este fato é devido ao grande número de parâmetros do *encoder g* com a Bi-LSTM.

Na Tabela 3.19 pode-se ver os parâmetros das redes *Matching-Encoder-BiLSTM* que apresentaram o melhor resultado *k-fold*. Uma época para estas redes foi considerada pela repetição do sorteio dos lotes 100 vezes. O *encoder* com as redes Bi-LSTM que obteve o melhor resultado foi utilizando uma camada intermediária e 150 neurônios. A escolha de utilizar 5 clas-

ses no “conjunto de suporte” foi devido a obtenção de um dos melhores resultados das redes e também ao tempo de treinamento ser bem menor do que utilizando mais classes.

Tabela 3.18 – Faixa de escolha de parâmetros das redes *Matching-Encoder-BiLSTM*

Parâmetros	**Valores**
Intermediária *encoder* (g)	64 a 256
Taxa de aprendizado	0,1 a 0,00001
Classes	5 a 120
Amostras por classe	1 a 8
Lote	2 a 20

Fonte: Do Autor (2022)

Tabela 3.19 – Parâmetros escolhidos das redes *Matching-Encoder-BiLSTM*

Parâmetros	**Valores escolhidos**
Intermediária	150
Função de erro	CrossEntropyLoss
Método de optimização	Adam
Taxa de aprendizado	0,001
Classes por conjunto de suporte	5
Amostras por classe	5
Lotes	12
Épocas	120

Fonte: Do Autor (2022)

4 RESULTADOS

Neste capítulo, são apresentados resultados das técnicas de *few-shot learning* empregadas: o algoritmo KNN, as redes *matching* e as redes DPGN. Os resultados foram divididos em duas partes, primeiro são mostrados os resultados dos algoritmos para a base de dados contendo 34 classes e depois para a base de dados contendo 312 classes.

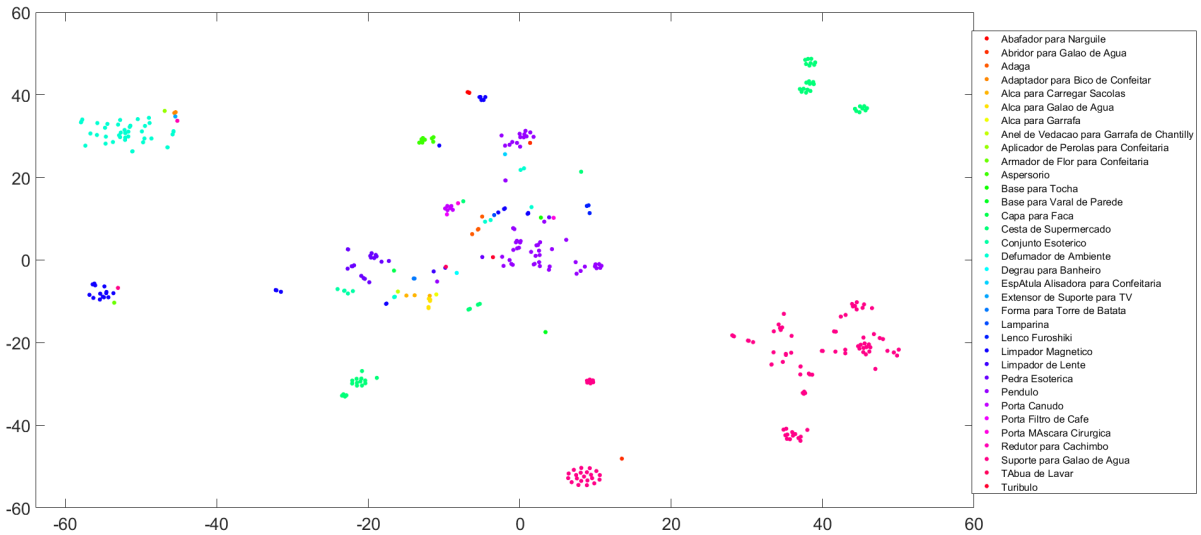
4.1 Resultados para Base de 34 Classes

Nesta Seção são apresentados os resultados para a base de dados contendo 34 classes e 394 amostras conforme mostrado na Seção 3.1.1, Tabela 3.1. Os resultados foram obtidos utilizando o método de validação cruzada *leave one out* e apresentados com a métrica acurácia.

4.1.1 Análise da Base de Dados

A análise da base de dados (Seção 3.1.1, Tabela 3.1) foi realizada a partir do algoritmo de visualização de dados *t-SNE* (*t-Distributed Stochastic Neighbor Embedding*). O objetivo foi visualizar as amostras com 1.200 características em duas dimensões, a fim de averiguar a qualidade das características, se elas são realmente interessantes para separação das novas classes de produtos. A Figura 4.1 apresenta o resultado obtido.

Figura 4.1 – Visualização em 2D do banco de dados com o algoritmo *t-SNE*



Fonte: Do Autor (2020)

Como pode ser observado, há uma boa separação de certas classes, mesmo em duas dimensões. Normalmente, ofertas de uma mesma classe ficam próximas entre si. Interessante observar a existência de pequenos *clusters* de duas ou três ofertas espalhados no espaço de duas dimensões. Apesar de em alguns casos não existir predominância de apenas um *cluster*, geralmente a presença de uma oferta ocorre próxima a de uma outra oferta do mesmo produto, com exceção daquelas classes que possuem apenas uma amostra (13 classes). Essa característica vai conferir ao algoritmo KNN de apenas um vizinho um bom desempenho nesse banco de dados, como será discutido na próxima seção.

4.1.2 O Algoritmo *k*-NN

Os experimentos com o KNN foram inicialmente realizados com 1 vizinho, $k = 1$, com as 34 classes e 394 amostras da base de dados de utilidades domésticas. Apesar do maior custo computacional, o uso de vários representantes por classes obteve melhor resultado do que o uso apenas do centroide mais próximo pelo fato de que algumas amostras testadas estarem mais próximas de outros centroides do que o seu respectivo. A Tabela 4.1 apresenta os resultados obtidos para diferentes medidas de similaridade.

Tabela 4.1 – Valores da acurácia (k=1) para várias distâncias, com uso de centroide ou não

Distâncias	**Manhattan**	**Hassanat**	**Lorentzian**	**Cosseno**	**Eucl.**	**Clark**	**Maha***
KNN	**96,33**	**96,33**	**96,33**	95,80	95,01	94,75	70,35
KNN (cent.)	81,36	77,16	80,05	82,41	**84,78**	76,11	69,00

* Distância de Mahalanobis aplicada para classes com mais de 2 amostras.

Fonte: Do Autor (2020)

As distâncias *Manhattan, Hassanat* e *Lorentzian* apresentaram os melhores resultados, com 96,33% de acerto, seguidas pela distância cosseno, com 95,80% de acerto, uma diferença de apenas 2 amostras a mais de erro, 16 contra 14 amostras. Por outro lado, a distância cosseno é interessante por possibilitar de forma mais direta a elaboração de um limiar de decisão de classificação. Com a distância do cosseno pode ser utilizado um limiar único para todas as classes, no entanto se fosse utilizado as outras distâncias estas dependeriam mais da distribuição dos dados. O resultado do KNN com distância do cosseno (k = 1) pode ser visto de forma detalhada na matriz de confusão no Apêndice B.

A fim de encontrar o melhor valor de k vizinhos, foram escolhidas as seguintes distâncias: Cosseno, Euclidiana e *Manhatan*. Para manter as mesmas condições do cenário em k=1,

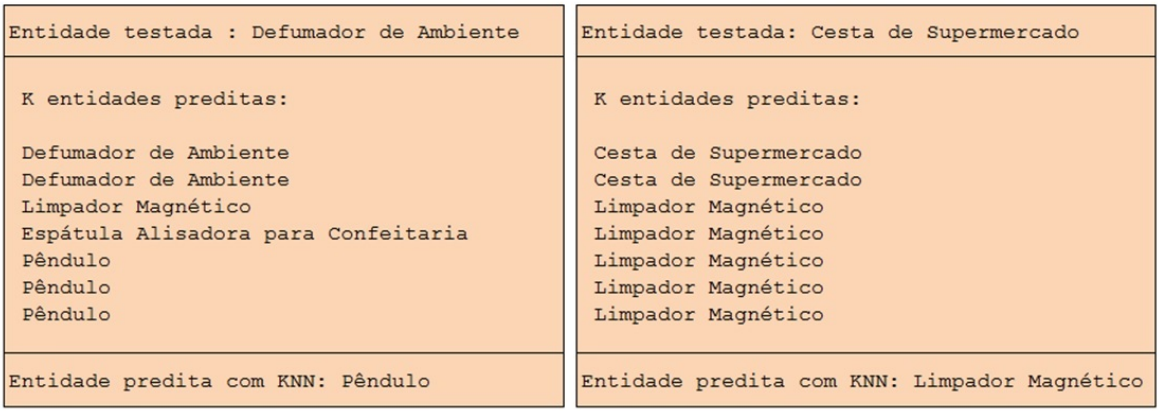
ou seja, realizar os testes com todas as 21 classes, àquelas com número de amostras menor do que k, foram adicionadas cópias de suas amostras até se obter k + 1 amostras totais em cada classe. Os resultados encontrados com o método de avaliação *Leave-One-Out* são apresentados na Tabela 4.2. Pode ser inferido que o valor de $k = 1$ obteve os melhores resultados. Para $k > 2$ os valores de acurácia diminuíram.

Tabela 4.2 – Valores da acurácia KNN (k >1)

Valores (k)	k=1	k=2	k=3	k=4	k=5	k=6	k=7
Cosseno	**95,80**	95,80	94,49	93,96	93,44	93,18	90,81
Euclidiana	**95,01**	95,01	93,96	94,23	92,65	92,13	90,81
Manhatan	**96,33**	96,33	95,01	94,49	93,18	92,65	92,91

Fonte: Do Autor (2020)

Figura 4.2 – Exemplos de erros para $k = 7$



Fonte: Do Autor (2020)

Normalmente é de se esperar que valores de $k > 1$ apresentam maior acurácia, porém, nesta base de dados isto não se confirmou. Conforme apresentado na Figura 4.1, alguns *clusters* não são tão bem definidos na base e pequenos subgrupos são encontrados dispersos na dimensão reduzida, um indicativo de que aumentar o número de vizinhos pode não ser uma estratégia interessante, o que se confirmou com o resultado do KNN (distância do cosseno e $k = 7$).

Na Figura 4.2 são apresentados dois exemplos de erro ocasionados com o aumento do número de vizinhos (KNN com cosseno e $k = 7$). Para os dois casos apresentados na figura, com o uso de apenas um vizinho, a classificação correta seria obtida, mas ao utilizar 7 vizinhos, a maioria deles são de uma classe diferente da classe testada, ocasionando erro de classificação. No entanto, pelo alto índice de acerto em todos os casos apresentados, sempre superior a

90%, pode-se concluir que a maioria das amostras estão bem localizadas em seus respectivos agrupamentos, validando o uso do classificador (em operação na empresa) como extrator de características no processo de *tranfer learning*.

4.1.3 Redução de Dimensão

Com o objetivo de redução de dimensão do problema em estudo (Seção 3.1.1, Tabela 3.1), para posterior aplicação dos métodos de classificação, como o KNN, redes *matching* e DPGN, foram testadas as abordagens descritas na Seção 3.2.2. O resultado do classificador KNN com $k = 1$ e distância cosseno, utilizando características selecionadas pelo FDR, pode ser visto na Tabela 4.3. Enquanto que o resultado obtido a partir do uso da PCA é apresentado na Tabela 4.4.

Tabela 4.3 – Resultados do KNN com número de características (NC) reduzido com FDR

NC	**800**	**600**	**400**	**100**	**50**	**20**	**10**	**5**	**3**	**2**
Resultado	**95,28**	94,23	93,08	91,08	91,34	88,19	89,76	86,35	77,69	69,29

Fonte: Do Autor (2020)

Tabela 4.4 – Resultados do KNN com dimensão reduzida com PCA

Variância(%)	**100**	**99,99**	**98**	**95**	**90**	**85**	**80**	**70**
NC	307	280	107	64	37	24	16	9
Resultado	**95,80**	95,54	95,54	95,28	94,23	93,70	92,39	91,60

Fonte: Do Autor (2020)

O melhor resultado do classificador KNN com $k = 1$ e distância cosseno, utilizando características selecionadas pelo FDR, foi de 95,28% de acurácia para 800 características selecionadas. Enquanto que o resultado obtido a partir do uso de PCA foi 95,80% de acurácia com 307 características transformadas (o que corresponde a mais de 99% da variância dos dados).

Comparando os resultados da abordagem via FDR com os da abordagem via PCA, pode ser observado que o primeiro obteve desempenho inferior. Um possível motivo é que o FDR não elimina as características redundantes e outro motivo é que, diferentemente da PCA, o FDR seleciona as características mais relevantes (reduzindo o total de características utilizadas) enquanto que a PCA as transforma, empregando na sua transformação as 1.200 características originais. Assim, a solução proposta para melhorar o desempenho da abordagem com FDR é o seu uso em conjunto com análise de correlação linear, a fim de eliminar as características redundantes, ou seja, aquelas que apresentarem correlação linear acima de um limiar pré-definido. O

resultado obtido com diferentes limiares de corte de variável via correlação linear é apresentado na Tabela 4.5.

Tabela 4.5 – Resultados do KNN (k=1) com dimensão reduzida (número de características - NC) por FDR e correlação, com diferentes limiares

Limiar	**-**	**0,8**	**0,75**	**0,70**	**0,65**
NC	800*	483	392	308	224
Resultado	95,28	**95,54**	95,28	95,28	94,23

* Características sem aplicação de correlação

Fonte: Do Autor (2020)

A análise de correlação mostrou que existe muita redundância nas características, o que ficou comprovado pelo fato de que com 308 características (limiar 0,7) foi possível reduzir 60% das características da base já reduzida com FDR, mantendo a mesma acurácia. Considerando as 307 e 308 características dos resultados com PCA e FDR mais correlação, respectivamente, constata-se que estes são praticamente equivalentes (PCA com 95,80% e FDR mais correlação com 95,28% de acurácia). A diferença entre estas abordagens (com PCA e com FDR) é que o uso da PCA implica em uma operação de transformação a cada nova amostra, o que não acontece na abordagem com FDR, sendo esta última, portanto, mais simples computacionalmente na fase operacional.

Continuando com a tarefa de redução de dimensionalidade, será abordada a PCA para diferentes distâncias, a fim de analisar se a distância do cosseno continua apresentando bons resultados em comparação com outras distâncias para o caso de dimensões reduzidas e KNN com k=1. O resultado pode ser visto na Tabela 4.6.

Tabela 4.6 – Resultados do KNN com dimensão reduzida com PCA para diferentes distâncias

Distâncias	**Manhattan**	**Hassanat**	**Lorentzian**	**Cosseno**	**Eucl.**	**Clark**
(1200 carac.)	96,33	96,33	96,33	95,80	95,01	94,75
(307 carac.)	**96,33**	93,96	95,54	95,80	95,01	84,51

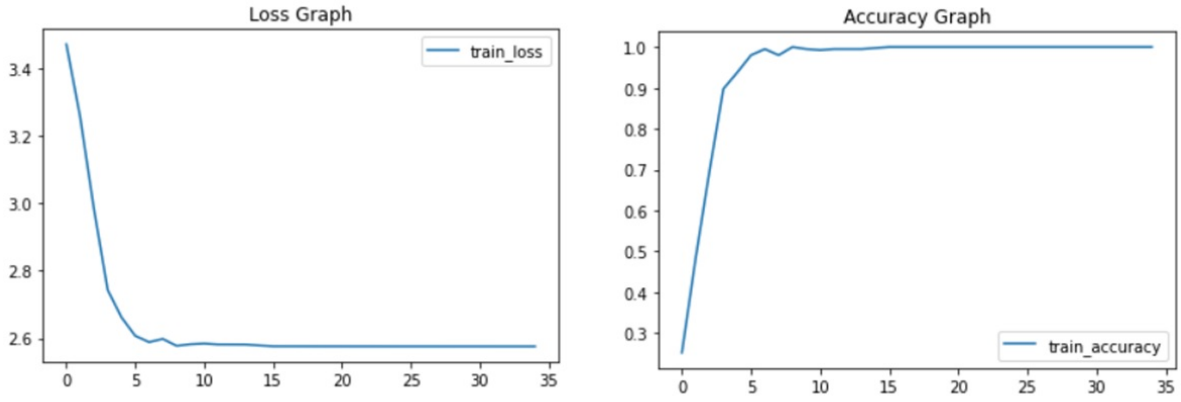
Fonte: Do Autor (2020)

Conforme pode ser observado, algumas distâncias mantiveram a mesma acurácia, enquanto outras apresentaram redução. A distância do cosseno, no entanto, continua sendo uma boa opção também para as 307 características, pois a acurácia é mantida tal como nas 1.200 características originais.

4.1.4 Redes *Matching*

Com a intenção de analisar um dos treinamentos executado pelas redes *matching* com o método de validação cruzada *leave one out* foi realizado a plotagem de acurácia e erro. A Figura 4.3 mostra a evolução do treino para uma amostra utilizando 307 características da base reduzida por PCA. A esquerda da figura pode-se ver a evolução do erro e a direita a acurácia em função das épocas. Conforme pode ser visto, com poucas épocas a *matching* convergiu.

Figura 4.3 – Evolução do treinamento de uma amostra (*leave one out*)



Fonte: Do Autor (2021)

As redes *matching* só conseguem realizar treinamento *leave one out* para classes que possuem 3 amostras ou mais. Uma amostra é separada para o teste, e no treinamento tem-se que realizar o cálculo de similaridade, então é necessário uma amostra para representante de classe e uma amostra para ser o alvo. Como existem 5 classes com 2 amostras, 10 amostras ficariam fora do treino, para evitar que isso acontecesse foi realizado cópia da amostra destas classes do conjunto de treino para ser o alvo.

Tabela 4.7 – Resultados de acurácia em (%) e tempo de treinamento em minutos das redes *matching*.

	307 características		1200 características	
Algoritmo	**Acurácia**	**Tempo**	**Acurácia**	**Tempo**
KNN (dist. cos)	95,80	NA*	95,80	NA*
Matching	**96,85**	1,93 minutos	**96,85**	6,99 minutos

*Abreviações: NA = não aplicável

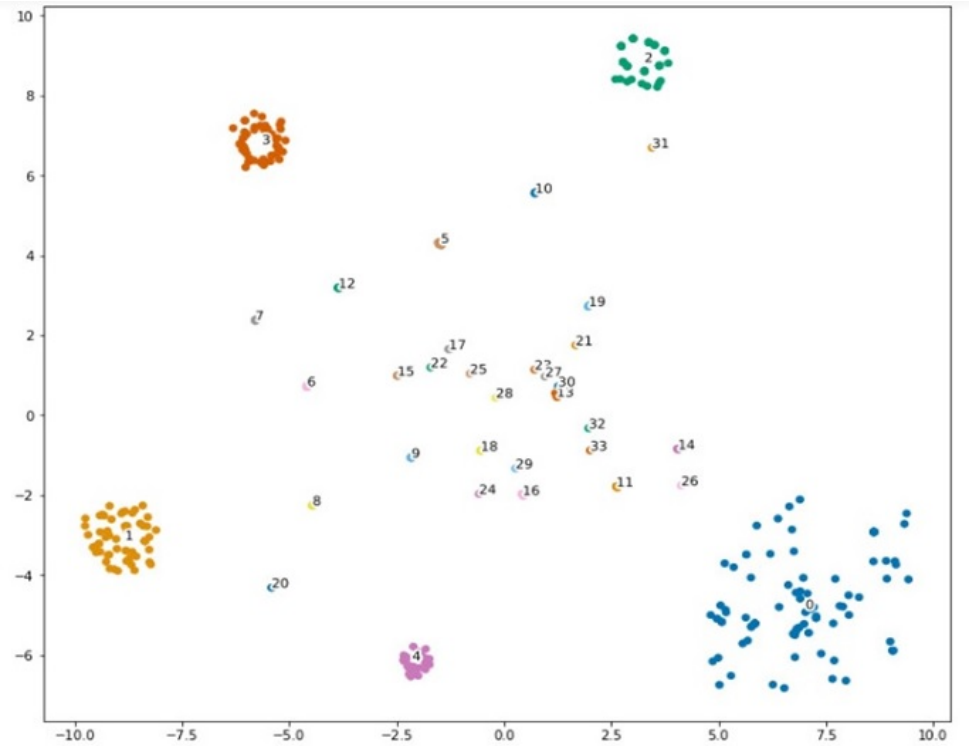
Fonte: Do Autor (2021)

Os resultados da acurácia e o tempo aproximado para realizar treinamento e teste das redes *matching* utilizando a base de dados com 1200 características e a base com 307 características obtidas com a técnica PCA são mostrados na Tabela 4.7, em comparação com os

valores obtidos com o algoritmo KNN. O tempo de processamento (treino e teste) foi obtido por simulação no *google* colab sem a utilização de GPU. Este tempo foi medido considerando apenas 1 dos 381 treinamentos executados com *leave one out*.

Levando em consideração que a maioria das características da base de dados utilizada são de boa qualidade, superar o KNN é uma tarefa difícil, pois o *embedding* das *matching* que é formado por uma rede MLP, precisa ser bem treinado para obtenção de características ainda melhores das obtidas pelo modelo original que já são boas. Tem que ser considerado também que dentre os erros das *matching* existem amostras que são de má qualidade não sendo capazes de serem melhoradas com redes neurais, e também as redes *matching* não adaptaram bem com o treinamento utilizando cópias das 5 classes que apresentam 2 amostras, sendo que os erros que as redes apresentaram para essas amostras representa 1% das amostras testadas. Portanto as redes *matching* obtiveram desempenho satisfatório, conseguindo superar os resultados obtidos com o KNN.

Figura 4.4 – Visualização *t-SNE* das redes *matching* treinada



Fonte: Do Autor (2021)

A utilização de 307 características obtidas por redução com PCA levou a uma rede mais compacta (307 neurônios na entrada e na saída). Com esta configuração, o treinamento convergiu com maior rapidez do que com as 1200 características originais (1200 neurônios na

entrada e na saída), devido ter reduzido o tempo utilizado nos cálculos de similaridade, que interferem bastante no tempo de treinamento.

A fim de analisar a distribuição dos dados, foi utilizado o algoritmo de visualização de dados *t-SNE*. Este foi utilizado para a representação dos 307 *features* obtidos pela PCA em duas dimensões (vide Figura 4.4). A rede conseguiu uma boa separação das classes em relação as amostras originais, conforme a Figura 4.1, que representa os mesmos dados das mesmas classes após serem processados pela rede (na Figura 4.1, foi utilizado 1200 características, para o caso do *t-SNE* com 307 características o resultado é idêntico). É importante observar como a distribuição das características das classes em duas dimensões ficou mais agrupada entre elementos da mesma classe e mais separável em relação às classes distintas do que antes do treino com as redes *matching*, o que mostra o benefício de se utilizar as mesmas.

4.1.5 Redes DPGN

As redes DPGN só conseguem realizar treinamento *leave one out* para classes que possuem 3 amostras ou mais. Para as classes com 1 e 2 amostras foram realizadas cópias das amostras destas classes do conjunto de treino para serem os alvos. No treinamento das DPGN foi necessário acrescentar até as classes com 1 amostra, devido no pré-processamento das redes para cada classe do "suporte data" ser necessário uma amostra desta mesma classe para ser o alvo. E isso complica um pouco o treinamento das redes devido a cálculos de várias similaridades desnecessárias.

A avaliação das redes DPGN foi realizada pelo método de validação cruzada *leave one out* utilizando 307 características reduzidas por PCA e as 1200 características originais. Na Tabela 4.8 pode ser visto os resultados das redes DPGN. Lembrando que tempo de treinamento das DPGN foi obtido com a utilização de GPU, e o tempo das redes *matching* foi utilizando a CPU, ambos do *google* colab. O tempo de treinamento das DPGN foi medido considerando apenas 1 dos 381 treinamentos executados com *leave one out*.

Nos testes para as classes com apenas 2 amostras foi considerado o resultado do KNN que é o *baseline* do projeto, pois as redes não apresentaram bom treinamento utilizando cópias de amostras. À vista disso classes com duas amostras foram responsáveis por 5 dos 12 erros totais apresentados pelas redes DPGN e também as classes com uma amostra complicou o treinamento. Considerando as classes com mais amostras as redes DPGN apresentaram bom desempenho.

Tabela 4.8 – Resultados de acurácia em (%) e tempo de treinamento em minutos das redes *DPGN*.

	307 características		1200 características	
Algoritmo	**Acurácia**	**Tempo**	**Acurácia**	**Tempo**
KNN (dist. cos)	95,80	NA*	95,80	NA*
Matching	**96,85**	1,93 minutos	**96,85**	6,99 minutos
DPGN	96,59	6,72 minutos	**96,85**	6,87 minutos

*Abreviações: NA = não aplicável

Fonte: Do Autor (2021)

Portanto, para esta base de dados que é desbalanceada e possui classes com 1 e 2 amostras, as redes *matching* foram a melhor opção com teste *leave one out*. As DPGN foram melhores somente para classes com mais de 4 amostras, apresentando apenas 5 erros enquanto o KNN apresenta 9 erros para as mesmas classes. Este resultado melhor para classes com maior número de amostras, pode ser devido as DPGN possuir em sua arquitetura redes CNN, que necessitam de maior quantidade de parâmetros comparados com as redes *matching* que possuem apenas uma MLP no encoder g.

4.2 Resultados para Base de 312 Classes

Nesta Seção são apresentados os resultados dos algoritmos *few-shot learning* para a base de dados contendo 312 classes e 3120 amostras conforme descrita na Seção 3.1.1, na Tabela 3.2. Os resultados foram obtidos utilizando o método de validação cruzada *k-fold*, apresentados com a acurácia média e pelo desvio padrão dos 10 *folds*.

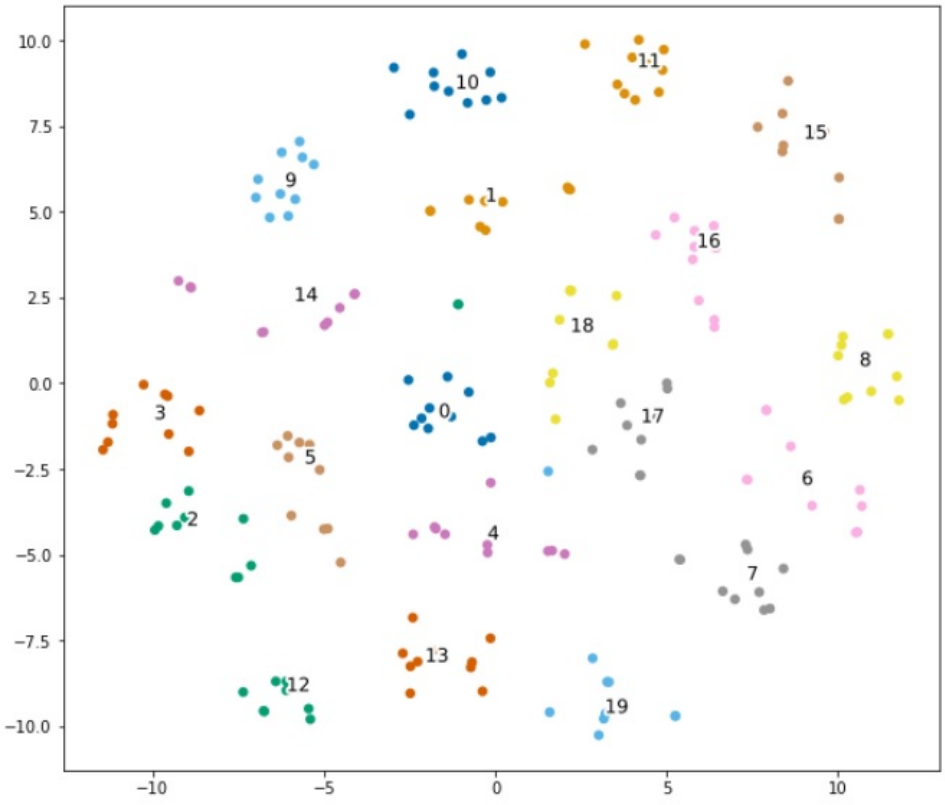
4.2.1 Análise da Base de Dados

A análise da base de dados foi realizada a partir do algoritmo de visualização de dados *t-SNE*. O objetivo foi visualizar as amostras com 1000 características em duas dimensões, a fim de averiguar a qualidade das características, e ver se elas são capazes de obter uma boa separação das classes de produtos contidas nesta base. A Figura 4.5 apresenta o resultado obtido.

Nesta figura foi realizado a plotagem do *t-SNE* para as 20 primeiras classes da base. Não foi utilizada a base inteira afim de obter uma melhor visualização dos dados utilizando um número reduzido de classes. Conforme pode ser observado, as classes estão bem separadas, os *clusters* são bem definidos, com as amostras das mesmas classes próximas entre si, tendo algumas poucas amostras fora de seu *cluster* original. Então pode ser concluído que as caracte-

rísticas extraídas das amostras utilizadas são de boa qualidade, isto fará com que o KNN tenha um resultado competitivo.

Figura 4.5 – Visualização em 2D do banco de dados de 312 classes com o algoritmo *t-SNE*



Fonte: Do Autor (2021)

4.2.2 Algoritmo *k*-NN

Os experimentos com o KNN foram iniciados com 1 vizinho, $k = 1$, com as 312 classes e 3120 amostras da base de dados. A Tabela 4.9 apresenta os resultados obtidos para diferentes medidas de similaridade. A distância *Clark* apresentou o melhor resultado com 92,72% de acurácia, seguida pela distância do cosseno com 92,56% de acurácia. A distância cosseno apresentou bom resultado para esta base, sendo interessante pela possibilidade de implementação de um limiar de decisão de classificação.

Com o intuito de encontrar o melhor valor de k vizinhos, foram escolhidas as seguintes distâncias: *Clark*, Cosseno e *Manhatan*. Os resultados encontrados com o método de avaliação *k-fold* são apresentados na Tabela 4.10. Pode ser inferido que o valor de $k = 1$ obteve os melhores resultados tal como aconteceu para a primeira base, pois para $k > 1$ os valores de

acurácia diminuíram. Em vista disso, será escolhido o resultado do KNN com $k = 1$ e distância do cosseno para ser utilizado para comparação com os outros algoritmos.

Tabela 4.9 – Valores da acurácia (k=1) para várias distâncias

Distâncias	**Clark**	**Cosseno**	**Hassanat**	**Lorentzian**	**Manhattan**	**Eucl.**
KNN	**92,72**	92,56	91,31	90,74	89,97	86,73
Desvio P.	1,30	1,49	1,70	1,83	1,95	1,96

*Abreviações: Desvio P. = Desvio Padrão

Fonte: Do Autor (2021)

Tabela 4.10 – Valores da acurácia KNN (k >1)

Valores (k)	**k=1**	**k=3**	**k=4**	**k=5**	**k=6**	**k=7**
Clark	**92,72**	91,76	91,38	90,48	90,90	90,51
DP. Clark	1,30	1,71	1,72	1,59	1,46	1,49
Cosseno	**92,56**	91,06	91,44	90,58	90,61	90,48
DP. Cosseno	1,49	2,00	1,89	1,93	1,77	1,53
Manhatan	**89,97**	89,55	89,04	87,88	87,79	86,83
DP. Manhatan	1,95	2,28	2,08	1,92	1,48	2,06

*Abreviações: DP. = Desvio Padrão

Fonte: Do Autor (2021)

O desempenho do classificador principal como extrator foi bom, pois por meio de análise de matriz de confusão pode-se perceber que a maioria dos erros de predição apresentados pelo KNN estão distribuídos. A maioria das classes não apresentam erros ou apresentam apenas um erro. E alguns erros apresentados é devido a base possuir alguns *labels* incorretos (dos 234 erros apresentados pelo KNN, 50 são devidos a *labels* incorretos, ou seja, 1,6% da base de dados). O resultado do KNN de mais de 92% é considerando bom também devido ao número de classes testadas ao mesmo tempo, ou seja, (n-ways de 312 classes), que representa para o KNN uma dificuldade de separação bem superior às 34 classes da base anterior, visto que o KNN não apresenta treinamento. Então a combinação do classificador em operação nos *marketplaces* como extrator de características das classes recém chegadas, com o KNN como classificador *few-shot* é uma boa opção visto que as características extraídas são bem discriminativas.

A abordagem proposta com a utilização do algoritmo KNN se mostrou uma boa opção para a tarefa de classificação de produtos com poucas amostras, além da obtenção de um índice de acerto satisfatório sem a necessidade de treinamento ou ajuste de parâmetros de modelos. A partir dos resultados obtidos pelo algoritmo KNN, foi possível construir um *baseline* competitivo para comparações com as redes *matching* e as redes DPGN. Foi importante observar que o

classificador em operação possui ótimas propriedades de extração de características de classes não conhecidas pelo mesmo.

4.2.3 Redução de Dimensão

O objetivo de utilizar redução de dimensão na base de 312 classes (Seção 3.1.1, Tabela 3.2) é analisar a possibilidade de obtenção de um número menor de características de forma que não mude o bom desempenho dos algoritmos *few-shot learning*. O resultado do classificador KNN com $k = 1$ e distância cosseno, utilizando o algoritmo PCA é apresentado na Tabela 4.11. A matriz de projeção foi construída com todas as 3120 amostras da base.

Tabela 4.11 – Resultados do KNN com dimensão reduzida com PCA para a base 312 classes

Variância(%)	**99,99**	**99,9**	**99**	**98**	**95**	**90**	**80**	**50**
NC	996	970	842	760	609	461	287	67
Resultado	**92,56**	**92,56**	92,40	92,5	92,28	92,24	91,92	90,71
Desvio P.	1,49	1,49	1,55	1,53	1,70	1,87	1,64	1,29

*Abreviações: Desvio P. = Desvio Padrão

Fonte: Do Autor (2021)

Conforme pode ser observado, esta base, ao contrário da anterior, não possuiu muita redundância, pois a PCA foi capaz de reduzir somente 3% (1000 p/ 970) mantendo a mesma acurácia de 92,56%. A redução de dimensão nesta base não foi tão viável tal como na base anterior. Um resultado razoavelmente bom foi para a variância de 90%, pois em relação a redução de 53,9% (1000 p/ 461) a perda não foi significativa. No entanto, para esta base, optou-se por manter o número de características originais, sem redução, para treino dos classificadores a seguir.

4.2.4 Redes *Matching*

Nesta Seção são apresentados os resultados das redes *matching* utilizando o método de validação cruzada *k-fold* para a base de dados 312 classes (Seção 3.1.1, Tabela 3.2). Os resultados de acurácia e o tempo aproximado para realizar o treinamento e teste das redes *matching* são mostrados na Tabela 4.12. O tempo de processamento (treino e teste) foi obtido por simulação no *google* colab PRO sem a utilização de GPU. Este tempo foi medido considerando apenas 1 dos 10 treinamentos executados com *k-fold*.

O desempenho das redes *matching* e do KNN foram comparados utilizando o teste-*t*. Como a hipótese nula não foi verificada, ou seja, existe diferença entre as médias dos resultados,

então pode-se concluir que as redes *matching* conseguiram resultado estatisticamente superior ao KNN. Porém, não foi uma tarefa fácil superar o KNN, devido ao desempenho do extrator de características, o que faz com que o KNN apresente resultado satisfatório. Por outro lado, foi observado nas amostras que apresentaram erros, muitos destes erros são devidos à erros de rotulação, sendo este montante equivalente a 1,6% da base de dados. E isso faz com que as características dessas amostras não sejam de boa qualidade, sendo projetadas em outros *cluters*, causando assim erros de predição.

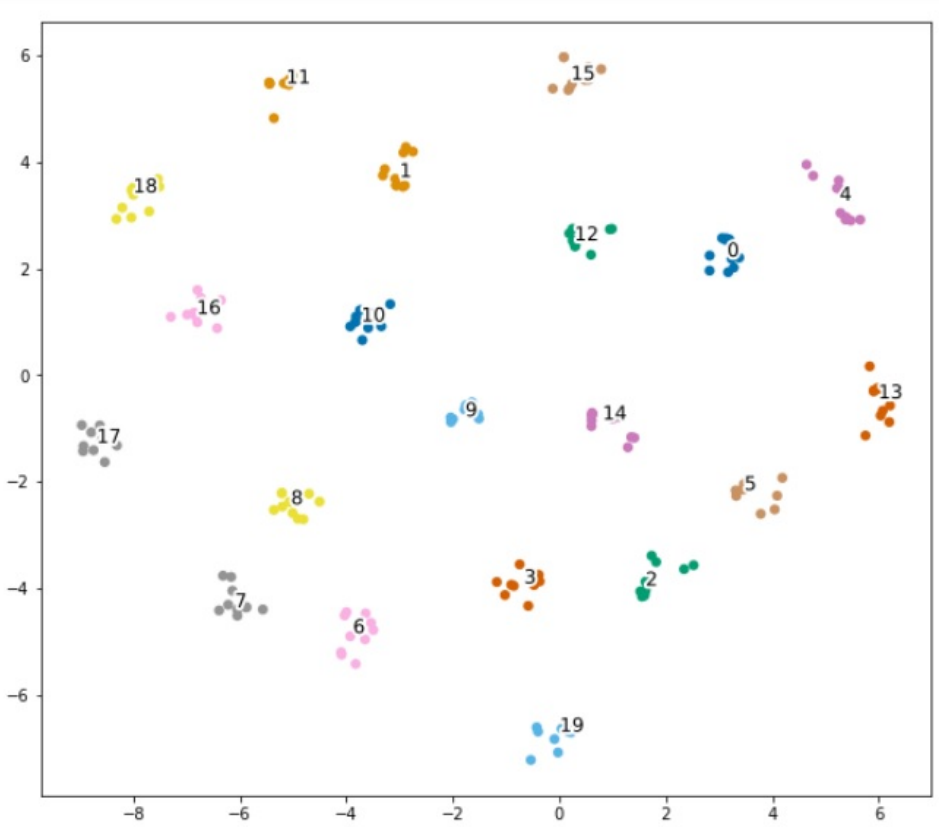
Tabela 4.12 – Resultados de acurácia em (%) e tempo de treinamento em horas das redes *matching*.

Algoritmo	**Acurácia**	**Tempo**
KNN (dist. cos)	92,56 +- 1,49	NA*
Matching	**93,78 +- 1,33**	1,17 horas

*Abreviações: NA = não aplicável

Fonte: Do Autor (2021)

Figura 4.6 – Visualização *t-SNE* das redes *matching* treinada



Fonte: Do Autor (2021)

Dentre os erros, também existem amostras em que o texto de descrição do produto é muito diferente das demais do mesmo *clusters*, e às vezes com texto incompleto. À vista disso,

se houver erros de rotulação ou as características sendo de qualidade inferior, mesmo uma rede neural tal como o *encoder* das *matching* não é capaz de melhorar estas características. Para trabalhos futuros podem ser utilizados algoritmos para fazer o tratamento dos erros de rótulo tal como *confidence learning* (NORTHCUTT; JIANG; CHUANG, 2021).

A fim de analisar a distribuição dos dados, foi utilizado o algoritmo de visualização de dados *t-SNE* nas 20 primeiras classes da base de dados (Tabela 3.2). Este foi utilizado para a representação dos 1000 *features* passados pelas redes *matching* que foram treinadas com as 312 classes e 3120 amostras (vide Figura 4.6). A rede conseguiu uma boa separação das 20 classes em relação as mesmas amostras originais, conforme a Figura 4.5. Conforme pode ser visto os *clusters* estão mais separados uns dos outros e também mais bem definidos, as amostras das mesmas classes estão mais próximas entre si.

4.2.5 Redes *Matching* Utilizando Redes BI-LSTM como *Encoder* g

Nos testes das redes *Matching-Encoder-BiLSTM* foram utilizados os textos dos produtos da base de dados contendo 312 classes (Seção 3.1.1, Tabela 3.2). Isto significa que os dados de entrada do *encoder* Bi-LSTM das redes *matching* agora são descritos em linguagem natural, ao contrário das seções de resultados anteriores que utilizavam características previamente extraídas.

Os resultados de acurácia e o tempo aproximado para realizar o treinamento e teste *k-fold* das redes *Matching-Encoder-BiLSTM* utilizando as amostras de texto da base de dados estão contidos na Tabela 4.13. O tempo de processamento (treino e teste) foi obtido por simulação no *google* colab PRO sem a utilização de GPU. A medida de tempo refere-se a apenas um treinamento dentre os 10 *folds*.

Tabela 4.13 – Resultados de acurácia em (%) e tempo de treinamento em horas das redes *Matching-Encoder-BiLSTM*.

Algoritmo	**Acurácia**	**Tempo**
Matching-Encoder-BiLSTM	82,53 +- 1,64	23 horas
KNN (dist. cos)	92,56 +- 1,49	NA*
Matching	**93,78 +- 1,33**	1,17 horas

*Abreviações: NA = não aplicável

Fonte: Do Autor (2022)

Conforme pode ser observado, os resultados foram bem inferiores aos utilizando dados previamente extraídos pelo classificador principal. A rede principal que extraiu as característi-

cas utilizadas pelo KNN e as redes *few-shot* anteriores, possuem um pré-processamento melhorado, é treinada com o volume de dados enorme, possui uma arquitetura otimizada, o dicionário do *embedding* é recente e mais completo em relação ao utilizado pelas redes *Matching-Encoder-BiLSTM*. Posto isso, este resultado bastante inferior pode ter acontecido devido a alguns fatores tais como a ocorrência de palavras fora do dicionário do modelo pré-treinado conforme pode ser visto na Tabela 4.14 que, por sua vez, dificulta muito o desempenho do algoritmo, pois o *embedding* não realiza uma representação numérica adequada para as palavras faltantes do dicionário. Como existem muitos casos em que o número de palavras faltantes é superior a 50% do texto da amostra, logo fica difícil do *encoder* com as redes Bi-LSTM conseguir interpretar estes dados. Outro possível fator é que o número de amostras da base não seja suficiente para realizar o treinamento das redes Bi-LSTM, em função do alto número de parâmetros dessas redes. Uma solução para melhorar o resultados das redes *Matching-Encoder-BiLSTM* seriam treiná-las com *embeddings* mais completos. Também ficará para trabalhos futuros a utilização de um *encoder* Bi-LSTM pré-treinado com uma base de dados maior.

Tabela 4.14 – Palavras faltantes do dicionário por amostra em (%) e o número de ocorrências na base

Palavras faltantes por amostra (%)	**Número de amostras da base**
30	623
40	340
50	178
60	59
80	6

Fonte: Do Autor (2022)

4.2.6 Redes DPGN

Conforme foi apresentado na Seção 3.2.7.2 foi escolhido para teste as 145 classes que o KNN com distância do cosseno (k=1) apresentado na Seção 4.2.2 apresentam erros.

Na Tabela 4.15 pode ser visto o resultado das redes DPGN. O tempo de treinamento das redes DPGN foi obtido com a utilização de GPU do *google* colab PRO. Este tempo foi medido considerando apenas 1 dos 10 treinamentos executados com *k-fold*. As redes DPGN são bem preparadas para serem treinadas com GPU, pois o treinamento de um *fold* com a utilização de GPU tem duração de 20 minutos, enquanto com a utilização de CPU este mesmo treinamento a duração é 6 horas. Para que a comparação seja realizada na mesma condição das redes DPGN foram realizados testes com o algoritmo KNN com distância do cosseno (k=1) e redes *matching*

com as mesmas 145 classes utilizadas pelas DPGN. A parametrização das redes *matching* foi a mesma descrita na Seção 3.2.6.2.

Tabela 4.15 – Resultados de acurácia em (%) e tempo de treinamento em horas das redes DPGN.

Algoritmo	**Acurácia**	**Tempo**
KNN (dist. cos)	85,79 +- 3,04	NA*
Matching	89,66 +- 2,31	1,2 horas
DPGN	**90,00 +- 1,95**	0,3 horas

*Abreviações: NA = não aplicável

Fonte: Do Autor (2022)

O desempenho das redes DPGN e *matching* foram comparados utilizando o teste-*t*, que mostrou que os resultados foram estatisticamente iguais. Assim sendo as redes *matching* se mostraram a melhor opção para aplicação em comércio eletrônico devido não haver tanta demanda por memória RAM tal como as redes DPGN, permitindo facilmente realizar teste com maior número de classes no conjunto de suporte.

As redes DPGN possui uma arquitetura sofisticada e que apresentou resultados bem competitivos (no início de 2020) em tarefas *few-shot learning*. Entende-se, portanto, que existe o que melhorar nos resultados destas redes, como a escolha da melhor parametrização. É interessante testar outras configurações dessas redes, mas para isso é necessário tratar os problemas de estouro de memória RAM, buscando uma arquitetura mais eficiente. Neste trabalho conforme dito foi utilizado o centroide como representante de classe no teste obtendo-se um bom resultado, uma sugestão que também pode ser utilizada é uso de ensembles e definição de representantes por classes. Isto traz como benefício a possibilidade de realizar treinamento com configurações melhores ou com maior número de classes no conjunto de suporte. Portanto fica como sugestão para trabalhos futuros melhorar o pré-processamento e a alocação de memória RAM das redes afim de reduzir o consumo. E também testar outras redes de grafos tais como as (EGNN) (*Edge-Labeling Graph Neural Network*) (KIM et al., 2019) e GNN (*Graph Neural Network*) (GARCIA; BRUNA, 2018).

5 CONSIDERAÇÕES FINAIS

Visto a necessidade de classificadores de entidades de produtos em tarefas que se dispõe de poucos dados de algumas classes. A abordagem utilizando algoritmos *few-shot learning* (FSL) se mostrou uma boa opção. Estas redes supriram uma lacuna deixada pelas redes tradicionais, estas que são muito utilizadas nos sistemas de comércio eletrônico, porém possuem bom desempenho somente para bases que possuem muitas amostras por classe. Também outro diferencial deste trabalho foi a aplicação das redes FSL para tarefa de processamento de linguagem natural, visto que na literatura as bases normalmente utilizadas são de imagens. Foi possível observar que as características das bases de dados (obtida por *transfer learning*) são boas, do qual facilita o trabalho dos classificadores de entidades, principalmente do KNN que é um algoritmo que não necessita de treinamento. Além disso a utilização das redes *Matching-Encoder-BiLSTM* possibilitou realizar treinamento com dados descritos em linguagem natural.

Foram implementadas as técnicas de *few-shot learning* redes *matching*, redes DPGN, e o algoritmo KNN para duas bases de dados, sendo uma contendo 34 classes e outra com 312 classes. Estes algoritmos utilizaram dados previamente extraídos a partir do processo de *transfer learning*. Também foi implementada as redes *Matching-Encoder-BiLSTM* que utilizou dados descritos em linguagem natural da base de dados com 312 classes. As redes DPGN e as redes *matching* conseguiram superar o KNN mesmo em uma base pequena e desbalanceada tal como a base com 34 classes, obtendo um desempenho satisfatório. As redes DPGN apresentaram dificuldades em predizer classes com muito poucas amostras, já para classes com mais de 4 amostras apresentaram o melhor resultado. Porém para a base de dados inteira, as redes *matching* foram a melhor opção, devido ser uma rede mais simples desempenha bem para base de dados reduzida que contenha poucas amostras. Para classes com duas amostras não foi possível realizar treinamento, e os erros apresentados nestas classes representa 1,27% da base de dados para DPGN e 1% para as *matching*.

As redes DPGN e as redes *matching* conseguiram um bom desempenho para a base de dados maior (312 classes). Apesar das redes DPGN terem obtido um bom resultado utilizando 145 classes, conseguindo superar o KNN, não foi possível realizar o treinamento delas para 312 classes devido ao estouro de memória RAM do *google* colab PRO. As redes *matching* foram treinadas com todas as 312 classes conseguindo superar o KNN e as redes *Matching-Encoder-BiLSTM*, sem ter nenhum problema relacionado a consumo de memória RAM. O resultado das redes *matching* pode ser considerado relevante porque em análise das amostras que o KNN

apresenta erros foi constado que um montante equivalente a 1,6% da base de dados são devidos à erros de rotulação e estas amostras por estarem mal posicionadas podem acarretar outros erros. E também foi observado na base de dados que existem amostras com textos contendo palavras muito diferentes do *cluster* e do nome da classe, e isso pode gerar *outliers*. O extrator das características pode não ser capaz de reconhecer as características de uma amostra inadequada como sendo similar as outras amostras do *cluster*.

As redes *matching Matching-Encoder-BiLSTM* não obteve um bom desempenho, mas há possibilidade de melhorar este algoritmo a ponto de se tornar competitivo com os outros algoritmos FSL que utilizaram dados obtidos por *transfer learning*. A sugestão é utilizar um *embedding* com dicionário atualizado com as palavras contidas na descrição dos produtos da base de dados, e também pré-treinar o *encoder* Bi-LSTM com uma base de dados maior do que a utilizada.

Em virtude disso as redes *matching* se mostraram mais adequada para resolver o problema *few-shot* abordado que é relacionado a *comércio eletrônico*. Pois o problema abordado neste trabalho mesmo que não seja bases de dados grandes tal como as utilizadas pelos classificadores em operação, entretanto o número de classes utilizadas que possuem poucas amostras pode ser mais de mil classes. Então para que as redes DPGN possam trabalhar nesta condição elas devem ser otimizadas, ou senão estas redes vão depender de um sistema com memória RAM robusta para serem treinadas. Também podem ser testadas outras redes de grafos, tais como as redes GNN e EGNN.

REFERÊNCIAS

AA, H. van der et al. Transforming unstructured natural language descriptions into measurable process performance indicators using Hidden Markov Models. *Information Systems*, Elsevier Ltd, v. 71, p. 27–39, 2017. ISSN 03064379.

Abu Alfeilat, H. A. et al. **Effects of Distance Measure Choice on K-Nearest Neighbor Classifier Performance: A Review**. 2019. 221–248 p.

AGERRI, R. et al. Big data for Natural Language Processing: A streaming approach. **Knowledge-Based Systems**, Elsevier B.V., v. 79, p. 36–42, 2015. ISSN 09507051. Disponível em: <http://dx.doi.org/10.1016/j.knosys.2014.11.007>.

AKRITIDIS, L.; FEVGAS, A.; BOZANIS, P. Effective products categorization with importance scores and morphological analysis of the titles. **Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI**, IEEE, v. 2018-November, p. 213–220, 2018. ISSN 10823409.

ALKASASSBEH, M.; ALTARAWNEH, G. A.; HASSANAT, A. B. A. On Enhancing The Performance Of Nearest Neighbour Classifiers Using Hassanat Distance Metric. v. 9, n. 1, 2015. Disponível em: <http://arxiv.org/abs/1501.00687>.

ALOM, Z. et al. The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches. 2018.

AREL, I.; ROSE, D. C.; KARNOWSKI, T. P. Deep machine learning a new frontier in Artificial Intelligence Research. **IEEE Computational Intelligence Magazine**, n. November, p. 13–18, 2010. ISSN 1556-603X.

BAHDANAU, D.; CHO, K. H.; BENGIO, Y. Neural machine translation by jointly learning to align and translate. In: **3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings**. [S.l.: s.n.], 2015. p. 1–15.

BENDRE, N.; MARÍN, H. T.; NAJAFIRAD, P. Learning from Few Samples: A Survey. In: . [S.l.: s.n.], 2020. p. 1–17.

BENGIO, Y.; DUCHARME, R.; VINCENT, P. A Neural Probabilistic Language Model. **Journal of Machine Learning Research**, n. 6, p. 1137–1155, 2000. ISSN 15324435.

BURGHARDT, F.; GARBE, R. Introduction of Artificial Neural Networks in EMC. In: **2018 IEEE Symposium on Electromagnetic Compatibility, Signal Integrity and Power Integrity, EMC, SI and PI 2018**. [S.l.: s.n.], 2018. p. 165–169. ISBN 9781538666210.

CAMACHO-COLLADOS, J.; PILEHVAR, M. T. From word to sense embeddings: A survey on vector representations of meaning. **Journal of Artificial Intelligence Research**, v. 63, n. March 2019, p. 743–788, 2018. ISSN 10769757.

CAO, X. et al. Calibrating GloVe model on the principle of Zipf's law. **Pattern Recognition Letters**, Elsevier B.V., v. 125, p. 715–720, 2019. ISSN 01678655.

CHAI, J.; LI, A. Deep learning in natural language processing: A state-of- the-art survey. In: . [S.l.]: IEEE, 2018. p. 1–6. ISBN 9789811052095.

CHAVES, A. G. S. et al. Extração de entidades de produtos utilizando técnicas de few-shot learning. In: **SBAI 2021 - XV Simpósio Brasileiro de Automação Inteligente**. Rio Grande, RS, Brasil: Sociedade Brasileira de Automática, 2021.

CHAWLA, K. et al. Pre-trained Affective Word Representations. In: **2019 8th International Conference on Affective Computing and Intelligent Interaction, ACII 2019**. [S.l.]: IEEE, 2019. p. 318–324. ISBN 9781728138886.

CHEN, G. et al. Ensemble application of convolutional and recurrent neural networks for multi-label text categorization. In: **Proceedings of the International Joint Conference on Neural Networks**. [S.l.: s.n.], 2017. v. 2017-May, p. 2377–2383. ISBN 9781509061815.

DASHTIPOUR, K. et al. A Hybrid Persian Sentiment Analysis Framework: Integrating Dependency Grammar Based Rules and Deep Neural Networks. **Neurocomputing**, Elsevier B.V., n. xxxx, 2019. ISSN 0925-2312. Disponível em: <http://arxiv.org/abs/1909.13568>.

FADLULLAH, Z. M. et al. State-of-the-Art Deep Learning: Evolving Machine Intelligence Toward Tomorrow's Intelligent Network Traffic Control Systems. **IEEE Communications Surveys and Tutorials**, v. 19, n. 4, p. 2432–2455, 2017. ISSN 1553877X.

FINN, C.; ABBEEL, P.; LEVINE, S. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In: **34th International Conference on Machine Learning, ICML 2017**. [S.l.: s.n.], 2017. v. 3, p. 1856–1868. ISBN 9781510855144.

GARCIA, V.; BRUNA, J. Few-Shot Learning With Graph Neural Networks. In: **arXiv**. [S.l.: s.n.], 2018. p. 1–13. ISSN 23318422.

GHOSH, S.; KRISTENSSON, P. O. Neural Networks for Text Correction and Completion in Keyboard Decoding. v. 14, n. 8, p. 1–14, 2017. Disponível em: <http://arxiv.org/abs/1709.06429>.

GILMER, J. et al. Neural message passing for quantum chemistry. In: **34th International Conference on Machine Learning, ICML 2017**. [S.l.: s.n.], 2017. v. 3, p. 2053–2070. ISBN 9781510855144.

GOMES, D. d. S. M.; EVSUKOFF, A. G. Processamento De Linguagem Natural Em Português E Aprendizagem Profunda Para O Domínio De Óleo E Gás. In: . [S.l.: s.n.], 2017. p. 1–25.

GOULARAS, D.; KAMIS, S. Evaluation of Deep Learning Techniques in Sentiment Analysis from Twitter Data. **2019 International Conference on Deep Learning and Machine Learning in Emerging Applications (Deep-ML)**, IEEE, p. 12–17, 2019.

GUO, G. et al. KNN model-based approach in classification. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 2888, n. January, p. 986–996, 2003. ISSN 16113349.

HAMILTON, W. L. **Graph Representation Learning Hamilton**. [S.l.: s.n.], 2020. v. 14. 1–159 p. ISSN 19394616.

HOCHREITER, S.; SCHMIDHUBER, J. Long Short-Term Memory. **Neural Computation**, v. 9, n. 8, p. 1735–1780, 1997. ISSN 08997667.

JADON, S.; GARG, A. **Hands-On One-shot Learning with Python Learn**. [S.l.: s.n.], 2020. v. 1. 1–136 p. ISSN 1098-6596. ISBN 9788578110796.

KANNAN, A. et al. Matching unstructured product offers to structured product specifications. **Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**, p. 404–412, 2011.

KARTHIKAYINI, T.; SRINATH, N. K. Comparative Polarity Analysis on Amazon Product Reviews Using Existing Machine Learning Algorithms. **2nd International Conference on Computational Systems and Information Technology for Sustainable Solutions, CSITSS 2017**, IEEE, p. 1–6, 2018.

KIM, J. et al. Edge-labeling graph neural network for few-shot learning. **Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition**, v. 2019-June, p. 11–20, 2019. ISSN 10636919.

KOCH, G.; ZEMEL, R.; SALAKHUTDINOV, R. Siamese Neural Networks for One-shot Image Recognition Gregory. In: **32nd International Conference on Machine Learning**. [S.l.: s.n.], 2015. v. 37. ISSN 00071447.

KRISHNAN, A.; AMARTHALURI, A. Large Scale Product Categorization using Structured and Unstructured Attributes. In: **Conference on Knowledge Discovery and Data Mining, August 04–08, 2019, Anchorage, Alaska. ACM, New York, NY, USA**. [s.n.], 2019. p. 9. Disponível em: <http://arxiv.org/abs/1903.04254>.

KRISHNAN, M.; DUTTA, D. A Study of Effectiveness of Principal Component Analysis on Different Data Sets. In: **2017 IEEE International Conference on Computational Intelligence and Computing Research, ICCIC 2017**. [S.l.]: IEEE, 2018. ISBN 9781509066209.

KUMAR, A. et al. Hybrid context enriched deep learning model for fine-grained sentiment analysis in textual and visual semiotic modality social data. **Information Processing and Management**, v. 57, n. 1, 2020. ISSN 03064573.

LAKE, B. M. et al. One shot learning of simple visual concepts. In: **Proceedings of the 33rd Annual Conference of the Cognitive Science Society**. [S.l.: s.n.], 2011. p. 6.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature**, v. 521, n. 7553, p. 436–444, 2015. ISSN 14764687.

LI, M. Y.; KOK, S.; TAN, L. Don't Classify, Translate: Multi-Level E-Commerce Product Categorization Via Machine Translation. p. 1–15, 2018. Disponível em: <http://arxiv.org/abs/1812.05774>.

LOPES, M. A. D. S.; NETO, A. D. D.; MARTINS, A. D. M. Parallel t-SNE Applied to Data Visualization in Smart Cities. **IEEE Access**, v. 8, p. 11482–11490, 2020. ISSN 21693536.

LOURIDAS, P.; EBERT, C. Machine Learning. **IEEE Software**, IEEE Computer Society, v. 33, n. 5, p. 110–115, 2016. ISSN 07407459.

MAATEN, L. van der; HINTON, G. Visualizing data using t-SNE. **Journal of Machine Learning Research**, v. 9, p. 2579–2605, 2008.

MENG, Y. et al. Unsupervised Word Embedding Learning by Incorporating Local and Global Contexts. **Frontiers in Big Data**, v. 3, n. March, p. 1–12, 2020.

MERKWIRTH, C.; LENGAUER, T. Automatic generation of complementary descriptors with molecular graph networks. **Journal of Chemical Information and Modeling**, v. 45, n. 5, p. 1159–1168, 2005. ISSN 15499596.

MIKOLOV, T. et al. Efficient estimation of word representations in vector space. In: **1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings**. [S.l.: s.n.], 2013. p. 1–12.

MISHRA, S. P. et al. Technical Note Multivariate Statistical Data Analysis- Principal Component Analysis (PCA). **International Journal of Livestock Research**, v. 7(5), p. 6–78, 2017. ISSN 02126567.

NAYAK, A.; DUTTA, K. Impacts of machine learning and artificial intelligence on mankind. In: **Proceedings of 2017 International Conference on Intelligent Computing and Control, I2C2 2017**. [S.l.: s.n.], 2018. p. 1–3. ISBN 9781538603741.

NG, S. C. Principal component analysis to reduce dimension on digital image. In: **Procedia Computer Science**. [S.l.: s.n.], 2017. v. 111, p. 113–119. ISSN 18770509.

NORTHCUTT, C. G.; JIANG, L.; CHUANG, I. L. Confident learning: Estimating uncertainty in dataset labels. **Journal of Artificial Intelligence Research**, v. 70, p. 1373–1411, 2021. ISSN 10769757.

OLIVEIRA, G. de;; VENSON, R.; MARCELINO, R. Redes neurais aplicadas no desenvolvimento de chatbots: uma análise bibliométrica. In: **ARTEFACTUM – REVISTA DE ESTUDOS EM LINGUAGEM E TECNOLOGIA**. [S.l.: s.n.], 2018. ISBN 8801697096107.

ONG, Y.-S.; GUPTA, A. AIR 5 : Five Pillars of Artificial Intelligence Research. In: **IEEE Transactions on Emerging Topics in Computational Intelligence**. [S.l.]: IEEE, 2019. v. 3, n. 5, p. 411–415. ISSN 2471-285X.

PASCANU, R.; MIKOLOV, T.; BENGIO, Y. On the Difficulty of Training Recurrent Neural Networks. In: **Proceedings of the 30 th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013**. [s.n.], 2013. p. 1310–1318. Disponível em: <http://proceedings.mlr.press/v28/pascanu13.pdf>.

PATERAKIS, N. G. et al. Deep learning versus traditional machine learning methods for aggregated energy demand prediction. In: **2017 IEEE PES Innovative Smart Grid Technologies Conference Europe, ISGT-Europe 2017 - Proceedings**. [S.l.]: IEEE, 2017. v. 2018-Janua, p. 1–6. ISBN 9781538619537.

PENNINGTON, J.; SOCHER, R.; D. Manning, C. GloVe: Global Vectors for Word Representation. In: **Proceedings ofthe 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), October 2014, Doha, Qatar. 2014 Association for Computational Linguistics**. [S.l.: s.n.], 2014. v. 19, n. 5, p. 1532–1543. ISSN 00047554.

PHAM, D. H.; LE, A. C. Exploiting multiple word embeddings and one-hot character vectors for aspect-based sentiment analysis. **International Journal of Approximate**

Reasoning, Elsevier Inc., v. 103, p. 1–10, 2018. ISSN 0888613X. Disponível em: <https://doi.org/10.1016/j.ijar.2018.08.003>.

PLOTZ, T.; GUAN, Y. Deep Learning for Human Activity Recognition in Mobile Computing. **IEEE Computer Society**, v. 51, n. 5, p. 50–59, 2018. ISSN 00189162.

POTRATZ, G. L. et al. Automatic lithofacies classification with t-sne and k-nearest neighbors algorithm. **Anuario do Instituto de Geociencias**, v. 44, n. 1, p. 1–14, 2021. ISSN 19823908.

QIN, H. et al. Hotel classification based on online review data. **ICNC-FSKD 2018 - 14th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery**, IEEE, n. 2013, p. 264–270, 2019.

QIN, Y. et al. Improving Sequence Modeling Ability of Recurrent Neural Networks via Sememes. **IEEE/ACM Transactions on Audio Speech and Language Processing**, v. 28, p. 2364–2373, 2020. ISSN 23299304.

QUISPE, L. V.; TOHALINO, J. A.; AMANCIO, D. R. Using virtual edges to improve the discriminability of co-occurrence text networks. **Physica A: Statistical Mechanics and its Applications**, Elsevier B.V., v. 562, p. 125344, 2021. ISSN 03784371. Disponível em: <https://doi.org/10.1016/j.physa.2020.125344>.

QURASHI, A. W.; HOLMES, V.; JOHNSON, A. P. Document Processing: Methods for Semantic Text Similarity Analysis. In: **INISTA 2020 - 2020 International Conference on INnovations in Intelligent SysTems and Applications, Proceedings**. [S.l.: s.n.], 2020. ISBN 9781728167992.

RAY, S. A Quick Review of Machine Learning Algorithms. **Proceedings of the International Conference on Machine Learning, Big Data, Cloud and Parallel Computing: Trends, Prespectives and Prospects, COM-IT-Con 2019**, IEEE, p. 35–39, 2019.

RISTOSKI, P. et al. A machine learning approach for product matching and categorization. **Semantic Web**, v. 9, n. 5, p. 707–728, 2018. ISSN 15700844.

SAHIN, C. B.; DIRI, B. Robust Feature Selection with LSTM Recurrent Neural Networks for Artificial Immune Recognition System. **IEEE Access**, v. 7, p. 24165–24178, 2019. ISSN 21693536.

SANTORO, A. et al. Meta-Learning with Memory-Augmented Neural Networks. **33rd International Conference on Machine Learning, ICML 2016**, v. 4, p. 2740–2751, 2016.

SARIKAYA, R. The technology behind personal digital assistants: An overview of the system architecture and key components. **IEEE Signal Processing Magazine**, IEEE, v. 34, n. 1, p. 67–81, 2017. ISSN 10535888.

SCARSELLI, F. et al. The graph neural network model. **IEEE Transactions on Neural Networks**, v. 20, n. 1, p. 61–80, 2009. ISSN 10459227.

SCHREIBER, J. N. C. et al. Técnicas de validação de dados para sistemas inteligentes: Uma abordagem do Software SDbayes. In: **XVII Colóquio Internacional de Gestão Universitária**. [S.l.: s.n.], 2017. p. 1–18. ISBN 9788578110796. ISSN 1098-6596.

SEOK, M. et al. Named entity recognition using word embedding as a feature. **International Journal of Software Engineering and its Applications**, v. 10, n. 2, p. 93–104, 2016. ISSN 17389984.

SHERSTINSKY, A. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. **Physica D**, Elsevier B.V., v. 404, p. 132306, 2020. ISSN 01672789. Disponível em: <https://doi.org/10.1016/j.physd.2019.132306>.

SILVA, F. C. e et al. Classificador fuzzy-genético aplicado ao processamento de linguagem natural. **Anais do XXIII Congresso Brasileiro de Automática**, v. 2, 2020.

SINGH, G.; SACHAN, M. Multi-layer perceptron (MLP) neural network technique for offline handwritten Gurmukhi character recognition. In: **2014 IEEE International Conference on Computational Intelligence and Computing Research, IEEE ICCIC 2014**. [S.l.]: IEEE, 2015. p. 1–5. ISBN 9781479939725.

SONAWANE, J. S.; PATIL, D. R. Prediction of heart disease using multilayer perceptron neural network. In: **2014 International Conference on Information Communication and Embedded Systems, ICICES 2014**. [S.l.]: IEEE, 2015. ISBN 9781479938346.

SPROAT, R.; JAITLY, N. RNN Approaches to Text Normalization: A Challenge. 2016. Disponível em: <http://arxiv.org/abs/1611.00068>.

STENMARK, M.; NUGUES, P. Natural language programming of industrial robots. **2013 44th International Symposium on Robotics, ISR 2013**, IEEE, p. 1–5, 2013.

THEODORIDIS, S.; KOUTROUMBAS, K. **Pattern Recognition**. San Diego, California, USA: Elsevier Inc., 2009. 948 p. ISBN 9781597492720.

TRSTENJAK, B.; MIKAC, S.; DONKO, D. KNN with TF-IDF based framework for text categorization. In: **Procedia Engineering**. Elsevier B.V., 2014. v. 69, p. 1356–1364. ISSN 18777058. Disponível em: <http://dx.doi.org/10.1016/j.proeng.2014.03.129>.

VINYALS, O. et al. Matching networks for one shot learning. **Advances in Neural Information Processing Systems**, p. 3637–3645, 2016. ISSN 10495258.

WÄCHTER, M. et al. Integrating multi-purpose natural language understanding, robot's memory, and symbolic planning for task execution in humanoid robots. **Robotics and Autonomous Systems**, Elsevier B.V., v. 99, p. 148–165, 2018. ISSN 09218890. Disponível em: <https://doi.org/10.1016/j.robot.2017.10.012>.

WANG, S.; LIU, C. L.; ZHENG, L. Feature selection by combining fisher criterion and principal feature analysis. **Proceedings of the Sixth International Conference on Machine Learning and Cybernetics, ICMLC 2007**, v. 2, n. August, p. 1149–1154, 2007.

WANG, Y. et al. Generalizing from a Few Examples: A Survey on Few-Shot Learning. **ACM Comput. Surv.**, v. 1, n. 1, p. 1–34, 2020. Disponível em: <http://arxiv.org/abs/1904.05046>.

WIDIASARI, I. R.; NUGROHO, L. E.; WIDYAWAN. Deep learning multilayer perceptron (MLP) for flood prediction model using wireless sensor network based hydrology time series data mining. In: **Proceedings - 2017 International Conference on Innovative and Creative Information Technology: Computational Intelligence and IoT, ICITech 2017**. [S.l.: s.n.], 2018. v. 2018-Janua, p. 1–5. ISBN 9781538640456.

XU, D.; TIAN, Y. A Comprehensive Survey of Clustering Algorithms. **Annals of Data Science**, Springer Berlin Heidelberg, v. 2, n. 2, p. 165–193, 2015. ISSN 2198-5804.

XU, X.; HUA, Q. Industrial Big Data Analysis in Smart Factory: Current Status and Research Strategies. **IEEE Access**, v. 5, p. 17543–17551, 2017. ISSN 21693536.

YAN, W. et al. Deep auto encoder model with convolutional text networks for video recommendation. **IEEE Access**, IEEE Access, v. 7, p. 40333–40346, 2019. ISSN 21693536.

YANG, L. et al. DPGN: Distribution propagation graph network for few-shot learning. In: **Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2020. p. 13387–13396. ISSN 10636919.

YE, Q. et al. River Water Quality Parameters Prediction Method Based on LSTM-RNN Model. In: **Proceedings of the 31st Chinese Control and Decision Conference, CCDC 2019**. [S.l.]: IEEE, 2019. p. 3024–3028. ISBN 9781728101057.

YOUNG, T. et al. Recent trends in deep learning based natural language processing. **IEEE Computational Intelligence Magazine**, IEEE Computational Intelligence Magazine, v. 13, n. 3, p. 55–75, 2018. ISSN 15566048.

YU, L. C. et al. Refining Word Embeddings Using Intensity Scores for Sentiment Analysis. **IEEE/ACM Transactions on Audio Speech and Language Processing**, IEEE, v. 26, n. 3, p. 671–681, 2018. ISSN 23299290.

ZHANG, S.; LIU, S.; LIU, M. Natural language inference using LSTM model with sentence fusion. In: **Chinese Control Conference, CCC**. [S.l.: s.n.], 2017. p. 11081–11085. ISBN 9789881563934. ISSN 21612927.

ZHENG, Y. et al. Principal characteristic networks for few-shot learning. **Journal of Visual Communication and Image Representation**, Elsevier Inc., v. 59, p. 563–573, 2019. ISSN 10959076. Disponível em: <https://doi.org/10.1016/j.jvcir.2019.02.006>.

ZHOU, J. et al. Graph neural networks: A review of methods and applications. **AI Open**, Elsevier Ltd, v. 1, n. December 2020, p. 57–81, 2020. ISSN 26666510. Disponível em: <https://doi.org/10.1016/j.aiopen.2021.01.001>.

APÊNDICE A – Tabelas com Testes das Redes

Neste capitulo é mostrado a obtenção da parametrização das redes *few-shot learning*.

Parametrização das Redes Matching para a Base de Dados de 312 Classes

Nesta Seção é apresentado a obtenção da parametrização das redes *matching* para a base de dados descrita na seção 3.1.1, na Tabela 3.2. Para avaliar os parâmetros foram realizados testes a primeiro momento utilizando o primeiro *fold* da base, logo após a obtenção de bons resultados com os parâmetros escolhidos foram realizados testes de avaliação dos parâmetros com todos os 10 *folds*. Foi escolhido trabalhar com o primeiro *fold* somente para economizar tempo no momento da parametrização a fim de possibilitar a realização de vários de experimentos. Na Tabela 1 pode ser visto a escolha do número de camadas da MLP *encoder g* das redes *matching*. A configuração inicial utilizada no treinamento para a obtenção do número de camadas foi: 30 classes por conjunto de suporte, 5 amostras por classe, lote de 12, 50 épocas, função tangente hiperbólica e taxa de aprendizado 0,001.

Uma vez obtido o número de camadas, o próximo passo foi escolher os melhores parâmetros das redes *matching* conforme pode ser visto na Tabela 2. Os parâmetros que apresentavam bons resultados foram sendo fixados. Também foi percebido que realizar testes de 10 em 10 épocas contribui para encontrar as épocas que geralmente as redes *matching* apresentam melhores resultados. Após obtido o melhor resultado das redes *matching* para o primeiro *fold*, foram realizados treinamentos para os dez *folds* conforme pode ser visto na Tabela 3.

Tabela 1 – Escolha do número de camadas do *encoder* MLP das redes *matching* utilizando o primeiro *fold*.

config.	**camadas**	**neurônios**	**Acurácia (%)**
1	4	100	56,09
2	3	100	75,00
3	2	150	80,45
4	2	200	81,09
5	2	300	84,29
6	1	300	88,14
7	1	400	89,42
8	1	500	**91,35**
9	1	600	88,46

Fonte: Do Autor (2021)

Tabela 2 – Parâmetros das redes *matching* para o primeiro *fold*.

config.	classes	shot	lote	época	lr	inter.	saída	função	acurácia (%)
1	156	2	12	200	0,0001	450	1000	tanh	93,59
2	**150**	2	12	**150**	0,0001	**350**	1000	tanh	93,59
3	**200**	2	12	150	0,0001	350	1000	tanh	93,91
4	**312**	2	12	150	0,0001	350	1000	tanh	94,55
5	**156 e 312**	**1**	12	**80**	0,0001	**450**	1000	tanh	93,59
6	156 e 312	**2**	12	**120**	0,0001	450	1000	tanh	94,23
7	50 e 312	**8**	12	**50**	0,0001	**440**	1000	tanh	94,87
8	**156 e 312**	**3**	12	**100**	0,0001	440	1000	tanh	95,19
9	156 e 312	**4**	12	100	0,0001	440	1000	tanh	95,19
10	**50 e 312**	**2**	12	**100**	0,0001	**450**	1000	tanh	95,51
11	**100 e 312**	2	12	100	0,0001	450	1000	tanh	94,87
12	**156 e 312**	2	12	60	0,0001	**430**	1000	tanh	94,87
13	**156 e 312**	2	12	60	0,0001	**460**	1000	tanh	94,87
14	**156 e 312**	2	12	60	0,0001	**475**	1000	tanh	94,55
15	156 e 312	2	**6**	**110**	0,0001	**450**	1000	tanh	94,87
16	156 e 312	2	**8**	**170**	0,0001	450	1000	tanh	94,55
17	156 e 312	2	**24**	**50**	0,0001	450	1000	tanh	95,19
18	156 e 312	2	12	**60**	0,0001	450	1000	tanh	95,51
19	156 e 312	2	12	**200**	0,0001	450	**450**	tanh	93,27
20	156 e 312	2	12	**70**	0,0001	450	450	tanh	94,55
21	156 e 312	2	12	**70**	0,0001	450	**600**	tanh	94,87
22	156 e 312	2	12	**100**	0,0001	450	**800**	tanh	94,87
23	156 e 312	2	12	80	**0,001**	450	1000	tanh	94,23
24	156 e 312	2	12	**80**	0,0001	450	1000	tanh	**95,83**
25	156 e 312	2	12	**100**	0,0001	450	1000	**relu**	94,55
26	156 e 312	2	12	**90**	0,0001	450	1000	**leaky-relu**	95,51
27	156 e 312	2	12	**60**	0,0001	450	1000	tanh	95,51

*Abreviações: config. = configuração, shot = número de amostras por classe, lr = taxa de erro, inter. = neurônios da camada intermediária

Fonte: Do Autor (2021)

Parametrização das Redes *Matching-Encoder-BiLSTM* para a Base de Dados de 312 Classes

A obtenção da parametrização das redes *Matching-Encoder-BiLSTM* para a base de dados com 312 classes (Seção 3.1.1, Tabela 3.2) pode ser visto na Tabela 4. Para avaliar os parâmetros foram realizados testes utilizando o primeiro *fold* da base. Os testes das redes *Matching-Encoder-BiLSTM* foram realizados com 120 épocas e taxa de aprendizado 0,001 que é um valor que apresentou bons resultados nas redes *matching* para a base de dados utilizada.

Tabela 3 – Parâmetros das redes *matching* para os 10 *folds* utilizados.

config.	classes	shot	lote	época	lr	inter.	saída	função	Acurácia (%)
1	156 e 312	2	12	200	0,0001	450	500	tanh	91,99
2	156 e 312	2	12	200	0,0001	450	**1000**	tanh	92,85
3	156 e 312	2	12	**70**	0,0001	450	1000	tanh	93,46
4	**100 e 312**	2	12	**100**	0,0001	450	1000	tanh	93,43
5	**156 e 312**	**2**	12	**80 e 90**	0,0001	450	1000	tanh	**93,78**
6	156 e 312	2	12	**80 e 90**	0,0001	450	1000	**leaky-relu**	93.14
7	**50 e 312**	**8**	12	100	0,0001	450	1000	tanh	90,45

*Abreviações: config. = configuração, shot = número de amostras por classe, lr = taxa de erro, inter. = neurônios da camada intermediária

Fonte: Do Autor (2021)

Tabela 4 – Parâmetros das *Matching-Encoder-BiLSTM* para o primeiro *fold*.

config.	classes	shot	lote	neurônios	acurácia (%)
1	5	5	10	64	78,88
2	**12**	5	10	64	78,53
3	**26**	5	10	64	79,49
4	**65**	**2**	10	64	80,77
5	**130**	**1**	10	64	81,41
6	**5**	**5**	10	**100**	82,69
7	**12**	**5**	10	**100**	**83,01**
8	**52**	**2**	10	100	**83,01**
9	**5**	**5**	**12**	**150**	82,37
10	**20**	5	12	150	82,05
11	**50**	**2**	12	150	**83,01**
12	**120**	**1**	12	150	81,73
13	**30**	**4**	12	150	81,73
14	**5**	**5**	12	**256**	82,65
15	**12**	**5**	12	256	81,73

*Abreviações: config. = configuração, shot = número de amostras por classe

Fonte: Do Autor (2022)

Parametrização das Redes DPGN para a Base de Dados de 312 Classes

Nesta Seção é apresentado a obtenção da parametrização das redes DPGN para a base de dados com 312 classes (Seção 3.1.1, Tabela 3.2). Para avaliar os parâmetros foram realizados testes utilizando os dez *folds*, a taxa de aprendizado foi 0,001, a distância utilizada foi $l1$. Na Tabela 5 pode ser observado a avaliação dos parâmetros para as 145 classes testadas, que são as que o KNN (cosseno, k=1) apresenta erros no teste com validação *k-fold* com 312 classes. Como a DPGN realiza o teste com mesmo número de classes utilizadas no treinamento, então não tem como variar o número de classes. Nesta Tabela alguns parâmetros tal como número de

gerações e lote das DPGN foram mantidos fixos devido a necessidade de reduzir demanda por memória RAM.

Tabela 5 – Parâmetros das redes DPGN para o primeiro *fold* utilizando 145 classes.

config.	**classes**	**shot**	**lote**	**Iterações**	**geração**	**inter.**	**saída**	**função**	**acurácia (%)**
1	145	1	6	500	1	**1000**	128	tanh	87,52
2	145	1	6	500	1	**600**	128	tanh	87,86
3	145	1	6	500	1	**450**	128	tanh	88,14
4	145	1	6	**800**	1	450	**200**	tanh	89,31
5	145	1	6	**1200**	1	450	200	tanh	89,03
6	145	1	6	**600**	1	450	200	tanh	89,24
7	145	**2**	6	**700**	1	450	200	tanh	89,17
8	145	**2**	6	**400**	1	450	**128**	tanh	**90,00**

Fonte: Do Autor (2022)

APÊNDICE B – Matriz de Confusão do *k*-NN com a Base de Dados com 34 Classes

A partir do resultado apresentado pela distância do cosseno para k=1 (Seção 4.1.2), foi construída a matriz de confusão para uma melhor visualização de como os erros estão distribuídos entre as classes. A Figura 1 mostra essa matriz de confusão onde as linhas representam as classes preditas e as colunas as classes testadas.

Figura 1 – Matriz de confusão para distância do cosseno

Preditas \ Testadas		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	Suporte para Galão de Água	112	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	Pêndulo	0	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
2	Cesta de Supermercado	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	Defumador de Ambiente	0	0	0	47	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	Limpador Magnético	1	0	0	1	30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	Pedra Esotérica	0	0	0	0	0	18	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
6	Aspersório	0	0	0	0	1	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	Porta Canudo	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	1	0
8	Lenço Furoshiki	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0
9	Conjunto Esotérico	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0
10	Alça para Galão de Água	0	0	0	0	0	0	0	0	0	0	5	1	0	0	0	0	0	0	0	0	0
11	Alça para Carregar Sacolas	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0
12	Adaga	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0
13	Forma para Torre de Batata	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0
14	Base para Varal de Parede	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0
15	Abafador para Narguilé	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0
16	Abridor para Galão de Água	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	Adaptador para Bico de Confeitar	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
18	Limpador de Lente	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
19	Porta Máscara Cirúrgica	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	Degrau para Banheiro	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
	Corretos	112	59	50	47	30	18	8	6	5	5	5	3	3	3	3	3	0	2	1	0	2
	Erros	1	1	1	2	2	2	0	0	0	0	0	1	1	0	0	0	2	0	1	2	0
	Total amostras/Classe	113	60	51	49	32	20	8	6	5	5	5	4	4	3	3	3	2	2	2	2	2
	Porcentagem (Corretos)	99,1	98,3	98	95,9	93,8	90	100	100	100	100	100	75	75	100	100	100	0	100	50	0	100
	Porcentagem (Erros)	0,9	1,7	2	4,1	6,2	10	0	0	0	0	0	25	25	0	0	0	100	0	50	100	0

Fonte: Do Autor (2020)

As classes que possuem somente 1 amostra não foram apresentadas na matriz de confusão, mas elas foram responsáveis por 6 dos 16 erros de predição obtidos. Na linha "Total amostras por classes" é possível observar que a base de dados com 34 classes é bastante desbalanceada e também existem muitas classes com menos de 10 amostras e ao mesmo tempo tem-se uma com 113 amostras.