UNIVERSIDADE DE SÃO PAULO
Instituto de Física de São Carlos

Aplicação de Redes Neurais em Grafos para predição de crimes

Adaptação de modelo de predição de incidentes

Marvin Mendes Cabral

São Carlos
2024

Trabalho de Conclusão de Curso - Monografia
Marvin Mendes Cabral

Aplicação de Redes Neurais em Grafos para predição de crimes

Adaptação de modelo de predição de incidentes

Trabalho de Conclusão de Curso apresentado ao Instituto de Física de São Carlos da Universidade de São Paulo, para obtenção do título de Bacharel em Física Computacional.

Orientador: Prof. Dr. Luis Gustavo Nonato - Instituto de Ciências Matemáticas e Computação

São Carlos
2024

Resumo

Com o crescimento populacional em grandes centros urbanos, torna-se cada vez mais alarmante a situação da criminalidade. A implementação de técnicas de ciência de dados e aprendizado profundo é cada vez mais aliada à estratégias da segurança pública. Com o advento das redes neurais profundas (Deep Neural Networks - DNN), abre-se a expectativa de metodologias mais eficazes para a análise e predição de crimes, tornando possível o uso de uma grande quantidade de variáveis na construção de modelos preditivos, além disso, passa a ser viável a realização de predições em um nível de granularidade bastante fina, tanto espacialmente quanto temporalmente. A estrutura de uma cidade pode ser representada por um *grafo de ruas*, onde os nós são os segmentos de rua e existem arestas quando há cruzamentos entre estes segmentos, deste modo é possível implementar um tipo específico de DNN para análise e predição de crimes, as chamadas Redes Neurais em Grafos (Graph Neural Networks – GNN), que viabilizam investigar fenômenos no nível de mapas de ruas, fato ainda pouco explorado na literatura. Ainda mais especificamente, o problema necessita de modelos que possam lidar com um grafo espaço-temporal como o gerado para a criminalidade num grafo de ruas. Deste modo foi escolhido e estudado o modelo *EvolveGCN*, da IBM, a fim de adaptá-lo para o contexto de crimes, em específico crimes de furto ou roubo de celulares ou veículos na região central da cidade de São Paulo, no período de 01/jan/2022 até 31/dez/2023. Com as adaptações tanto aos dados quanto ao modelo feitas, foram realizados experimentos com variações dos parâmetros do modelo, seguidas de análises dos resultados nos conjuntos de treino, validação e teste, levando a conclusões interessantes, pois ainda que o modelo não tenha desempenhado bem no problema com as adaptações implementadas, demonstrando não lidar bem com a esparsidade do modo como está, mostrou potencial ao expressar as predições projetadas no mapa da cidade, identificando bem alguns *hotspots* de interesse.

Palavras-chave: Ciência de dados, aprendizado profundo, DNN, GNN, mapa de ruas, grafo espaço-temporal.

1 Introdução

A escalada populacional em grandes centros urbanos torna crescente o surgimento de problemas em diversos âmbitos, como ilhas de calor, poluição, acidentes de trânsito, carência de serviços públicos adequados e criminalidade. Em particular, a questão da criminalidade, tem sido uma preocupação de longa data para a população em diversas metrópoles brasileiras, tais como São Paulo.

O aumento alarmante da violência e criminalidade têm instigado o desenvolvimento de novas estratégias tecnológicas e científicas para enfrentar essa problemática. Nesse contexto, a ciência de dados e o aprendizado de máquina emergem como ferramentas proeminentes, introduzindo abordagens inovadoras para conter a criminalidade e promover a segurança. Essas abordagens [Han et al., 2020, Jin et al., 2020, Wang et al., 2020b] tipicamente empregam modelos que relacionam variáveis externas, como dados de infraestrutura urbana, condições climáticas e indicadores socioeconômicos, como preditoras de crimes.

No entanto, existem duas questões essenciais que são pouco exploradas: a primeira é esparsidade espacial e temporal de crimes, isto é, a granularidade espaço-temporal de crimes é muito inferior se comparada à locais sem ocorrências. A segunda questão relevante está relacionada a como modelar simultaneamente dados temporais (registros climáticos, histórico criminal, etc) e variáveis estáticas (fatores socioeconômicos, características da infraestrutura urbana, etc), ou seja, como fazer o modelo concomitantemente processar dados estáticos e dinâmicos de forma espaço-temporal, de modo a realizar predições precisas e confiáveis.

Outro aspecto importante para modelagem de crimes é a discretização espacial. O uso de grides regulares, embora muito utilizada, dificulta a representação fiel de barreiras urbanas como ruas sem saída, separação física de regiões por meio de rios, lagos, e parques, inviabilizando a imposição de restrições nos modelos preditivos [Han et al., 2020, Jin et al., 2020, Wang et al., 2020b]. Para contornar tal dificuldade, trabalhos recentes empregam discretizações espaciais baseadas em grafos de ruas [Rosser et al., 2017, Zhang and Cheng, 2020] permitindo o tratamento natural da estrutura urbana pelos modelos. Comparativamente, estudos [Rosser et al., 2017, Garcia-Zanabria et al., 2021] indicam um melhor desempenho preditivo com o uso de grafos de ruas, porém, integrar dados dinâmicos e estáticos continua sendo um desafio.

Trabalhos recentes abordam a questão da esparsidade e a modelagem de dados estáticos e dinâmicos por meio de modelos de aprendizado profundo, embora não no contexto de criminalidade. Um bom exemplo de tal estratégia é o modelo proposto por Yu e colaboradores [Yu et al., 2021], que realiza predições de acidentes de trânsito utilizando o mapa viário das cidades como discretização espacial. A correta modelagem preditiva baseada grafos de ruas demanda o uso de modelos de redes neurais em grafos [Bacciu et al., 2020], os quais têm demonstrado um vasto potencial para compreender fenômenos urbanos, como a criminalidade.

Para este trabalho realizamos um estudo de métodos e modelos de aprendizado de máquina e deep learning baseados em grafos (Graph Neural Networks - GNNs). Especificamente, investigamos diversos modelos de GNN capazes de tratar dados espaço-temporais em grafos, como o proposto por Wang e colaboradores [Wang et al., 2021] e por Longa e colaboradores [Longa et al., 2023]. Em particular, investigamos detalhadamente o modelo *EvolveGCN* [Pareja et al., 2020] desenvolvido por uma equipe da IBM, realizando adaptações para o contexto de predição de crimes na cidade de São Paulo.

A seguir, apresentamos conceitos fundamentais e os resultados obtidos com o modelo desenvolvido.

2 Preliminares

Nesta seção falaremos sobre a teoria de grafos e das *Artificial Neural Networks* ou simplesmente *Neural Networks* (NN) que servirão de base para o trabalho desenvolvido.

2.1 Grafos

Um grafo $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ é uma estrutura de dados formada por dois conjuntos: o conjunto de nós ou vértices $\mathbf{V} = \{v_1, \ldots, v_n\}$ e o conjunto de arestas $\mathbf{E} \subseteq \{(v, u) \mid v, u \in V\}$ (do inglês *edges*). As arestas são pares de vértices que representam conexões entre os elementos, como ilustrado na Figura 1.

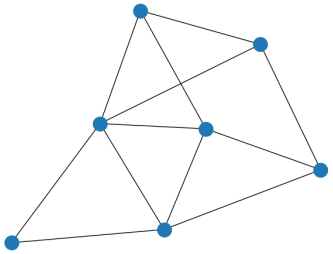


Figura 1: Os vértices são denotados pelos pontos azuis e as arestas pelas ligações entre eles.

Vértices e arestas podem estar associados a *vetores de características* (também conhecidos como features), os quais contém informações específicas relacionadas a cada uma das entidades.

Apesar da simplicidade, é notável que muitos sistemas podem ser descritos ou interpretados como um grafo, tornando a gama de aplicações não só vasta mas também extremamente interdisciplinar, como em ranqueamento de páginas da web [Bianchini et al., 2005], química molecar [You et al., 2018], física de partículas [Shlomi et al., 2020], entre outros.

Algumas definições gerais importantes para grafos são:

- Um grafo é dito não direcionado (ou não dirigido) quando as as arestas (u, v) e (v, u) são equivalentes, isto é, quando o conjunto de arestas define uma relação de simetria. Caso haja direção definida, o grafo é dito direcionado.
- Sendo $u, v \in \mathbf{V}$ dois nós de um grafo, se $(u, v) \in \mathbf{E}$ dizemos que v é *adjacente* a u. No caso de um grafo não direcionado dizemos que u e v são adjacentes.
- Um grafo é dito valorado (ou ponderado) quando uma função atribui um valor a cada aresta.

Uma forma comum de se representar um grafo é através da sua matriz de adjacência. A matriz de adjacência A de um grafo consiste em uma matriz quadrada $n \times n$, sendo n a quantidade de nós do grafo, onde cada entrada $A_{i,j} = w_{ij}$ se existe uma aresta ligando os vértices v_i e v_j, sendo $A_{i,j} = 0$ caso contrário. Um exemplo típico é quando o valor 1 é atribuido a cada aresta do grafo, resultando em uma matriz de adjacência binária.

Em nosso trabalho utilizamos o *grafo de ruas* relativo à região central da cidade de São Paulo. Neste grafo temos que cada nó representa um segmento de rua e as arestas são as esquinas que os conectam.

2.2 Redes neurais

Nesta seção assumimos que estamos tratando de aprendizado supervisionado, ou seja, empregamos dados rotulados no treinamento e avaliação de uma rede neural.

A representação padrão de uma rede neural é feita com um grafo direcionado, neste momento é bom ressaltar que trata-se apenas da representação e que não tem relação nenhuma com as *Redes neurais em grafos*, tópico que será tratado posteriormente neste trabalho. A Figura 2 [Ognjanovski, 2019] ilustra um exemplo típico de uma rede neural densa ou totalmente conectada, onde cada nó está conectado com todos os nós da camada posterior a ele. Observe que existem diferentes tipos de *camadas* (ou *layers*) que uma rede neural pode possuir: a camada de entrada (*input*), as camadas ocultas (*hidden*) e a camada de saída (*output*). Podemos ainda variar o número de nós nas camadas e o número de camadas ocultas conforme a necessidade.

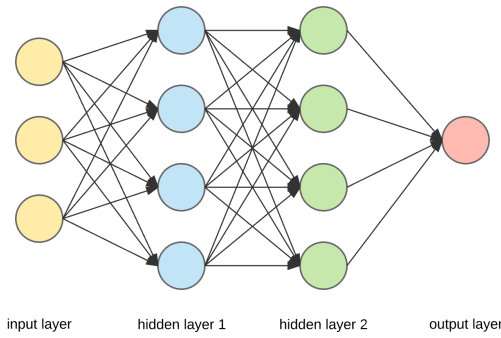


Figura 2: Neural Network Architecture (**Ognjanovski 2019**).

Intuitivamente, o treinamento da rede neural inicia na camada de entrada, por onde os vetores de features chegam até a rede, passando pelas camadas ocultas até chegar na camada de saída. Nas aresta os dados são multiplicados por *pesos*, sendo os resultados agregados em cada um dos nós da camada seguinte da rede. Na saída de cada nó das camadas ocultas é aplicada uma função especial, chamada *função de ativação*, repetindo o processo de multiplicação, agregação e ativação em cada uma das camadas ocultas. Por fim, na camada de saída o resultado obtido é comparado com o *rótulo* de treinamento conhecido, gerando possíveis erros de classificação, que são aferidos por uma *função de perda* (ou *custo*), o qual é propagado na direção contrária do fluxo da rede por um mecanismo chamado de *backpropagation*, que atualiza o valor dos pesos das arestas de modo a minimizar a função de perda. Repetimos esse processo para todos os dados de treino. Para detalhes matemáticos, ver [Aggarwal et al., 2018].

As operações de multiplicação dos pesos e agregação podem ser modeladas como multiplicações matriz-vetor, considerando um vetor de dados e uma matriz de pesos a cada camada. Existe ainda o *viés* (ou *bias*), um parâmetro treinável externo aos dados, que translada o resultado de cada nó. O viés não é multiplicado por nenhum valor de entrada, usualmente não se coloca o viés nas representações para evitar poluição visual, mas consideramos ele sempre presente em nossos exemplos.

É importante notar que em qualquer arquitetura utilizada sempre geramos uma representação *latente* ou *embedding* dos dados, isto é, uma representação dos dados aprendida pelo modelo em questão. A representação latente pode ser dada como uma combinação das saídas de todas as camadas ou simplesmente as saídas da última camada oculta. Tal representação latente é muito útil para realizar agrupamentos e visualizações dos dados.

2.2.1 Funções de ativação

As funções de ativação introduzem um caráter não linear para o modelo, caso contrário estaríamos realizando apenas transformações lineares de uma camada para outra. Assim, é indispensável a existência da não linearidade, aspecto este que traz para as NN sua característica de *caixa preta*. O conceito é inspirado na neurociência, uma vez que para formação de novas memórias e aprendizados os neurônios no cérebro humano são "disparados". As funções de ativação fazem seu trabalho nos nós decidindo se e como a informação é passada para a próxima camada. Existem diversas funções de ativação na literatura [Sharma et al., 2017, Ding et al., 2018, Rasamoelina et al., 2020], vamos nos limitar a descrever brevemente algumas das mais utilizadas. No que segue $z \in \mathbb{R}$ será o valor *pré-ativação*, ou seja, o valor no nó oculto antes da aplicação da função de ativação.

Linear $f(z) = z$**.** Muito simples e barata computacionalmente, esta ativação serve como um canal de passagem para os valores pré-ativação, geralmente utilizada na camada de saída para problemas de regressão.

Sigmóide $f(z) = \frac{1}{1+e^{-z}}$**.** Além da não linearidade acrescentada ao poder representativo da rede, outras vantagens que se destacam são sua diferenciabilidade em todo ponto e sua saída limitada entre 0 e 1, podendo escalar valores pré-ativação que sejam discrepantes dos demais. Porém, esta função possui um alto custo computacional e sofre de um problema conhecido como saturação/anulação do gradiente. A sigmóide é utilizada principalmente na camada de saída para predições de eventos binários.

Softmax $f(z_i) = e^{z_i} / \sum_{j=1}^{N} e^{z_j}$**.** Majoritariamente utilizada para obtenção de uma distribuição de probabilidades de uma gama de classes e/ou atributos. Também possui alto custo computacional, sendo utilizada principalmente na camada de saída para predições de classificação com mais de duas classes , fornecendo a probabilidade predita pela rede para cada uma das classes.

Rectified Linear Unit (ReLU) $f(z) = \max(0, z)$**.** Uma das funções mais inspiradas no funcionamento do cérebro humano, pois caso um neurônio não receba estímulo o suficiente ele simplesmente não ativa. Além de acelerar o treinamento, muitos fatores fazem da ReLU uma das funções mais utilizadas em nós ocultos para adição de não linearidade à rede. Ela tem um baixo custo computacional, não sofre de anulação ou saturação do gradiente já que este é praticamente constante, adicionando ainda esparsidade à rede. As principais desvantagens que esta ativação apresenta são sua não diferenciabilidade em zero e o chamado *Dying ReLU*, fazendo com que o fluxo do gradiente seja nulo a partir de um determinado ponto de treinamento. Alternativas para contornar tais deficiências são a Leaky ReLU ou a PReLU [Sharma et al., 2017, Ding et al., 2018, Rasamoelina et al., 2020], sendo elas significativamente mais custosas computacionalmente.

2.2.2 Funções de perda

Utilizadas para aferir a qualidade das predições realizadas, a escolha das funções de perda dependem do tipo de predição a ser realizada e do tipo dos dados de entrada. Existe uma variedade de funções de perda [Janocha and Czarnecki, 2017, Wang et al., 2020a]. Nesse trabalho, vamos nos ater às mais conhecidas, que por sua vez admitem variações. No que segue, y se refere ao valor do rótulo que queremos prever, pode ser tanto um número real ($y \in \mathbb{R}$), quanto binário ($y \in \{0, 1\}$), e $f(x)$ é a predição dada pela rede neural.

Square Loss $L(y, f(x)) = (y - f(x))^2$**.** Função de perda para problemas de regressão, é diferenciável em segunda ordem e tem o gradiente variável, o que torna a convergência mais rápida. É uma função sensível à outliers, já que os erros são elevados ao quadrado, problema amenizado realizando pré-processamento nos dados. Existem ainda estudos que analisam a utilização desta função para problemas de classificação [Janocha and Czarnecki, 2017].

Absolute Loss $L(y, f(x)) = |y - f(x)|$**.** Também utilizada em problemas de regressão, é uma função extremamente robusta à outliers, porém possui a desvantagem de ser não suave na origem. Aqui vale citar a *Huber Loss*, uma combinação por partes das duas funções de perda para regressão já descritas, que busca contornar os problemas de ambas.

Cross entropy Loss $L(P, Q) = P(x)\log(Q(x))$**.** Os argumentos P e Q são respectivamente a distribuição de probabilidade verdadeira e a distribuição de probabilidade predita, enquanto x é um dado de entrada qualquer. Em teoria de informação [Bishop and Nasrabadi, 2006] a grandeza chamada entropia nos diz a quantidade de informação necessária em *bits* para se transmitir um evento de uma distribuição de probabilidade. Na física podemos interpretá-la como um fator de organização do sistema, quanto mais organizados os dados menor a entropia, isto é, menos bits são necessários para transmitir a informação. A entropia cruzada mede o quão próxima é a distribuição Q está da distribuição P.

2.2.3 Backpropagation

Backpropagation é um algoritmo chave no treinamento de redes neurais artificiais. É usado para ajustar os pesos da rede a fim de minimizar o erro entre a saída gerada pela rede e a saída desejada. De modo sucinto, o mecanismo funciona da seguinte forma:

1. Inicialização: Comece com pequenos valores randômicos para os pesos e vieses da rede;

2. Passo no fluxo da rede: Insira dados de entrada pela rede para obter uma saída. Cada camada da rede processa os dados usando seus pesos atuais e funções de ativação.

3. Calculo da perda: Compare a saída da rede com os valores alvo reais usando uma função de perda.

4. Passo fluxo contrário da rede: A partir da camada de saída para a camada de entrada, calcule os gradientes (derivadas parciais) da função perda em relação a cada peso. A regra da cadeia permite propagar o erro para trás através da rede, da camada de saída para a camada de entrada.

5. Atualização dos pesos: Ajuste os pesos e vieses a fim de minimizar a perda usando gradiente descendente da forma:

$$w_{ij} = w_{ij} - \delta \frac{\partial L}{\partial w_{ij}}$$

onde L é a função perda, w_{ij} o peso conectando o nó i ao j e δ é a taxa de aprendizado.

Uma discussão mais detalhada sobre Backpropagation pode ser encontrada em [Haykin et al., 2009].

2.2.4 Arquiteturas clássicas

Existem muitas arquiteturas de NN já consolidadas e algumas destas são utilizadas no modelo aplicado neste trabalho, seja em conceito ou diretamente. Nesta seção discutimos especificamente sobre duas delas, as *Recurrent Neural Networks* (RNNs) e as *Convolutional Neural Networks* (CNNs).

Recurrent Neural Networks. A arquitetura das RNNs tem por base a inserção de dados sequenciais no que chamamos de um *feedback loop*, com isso a rede tem poder representativo sobre as dependências temporais dos dados. O objetivo de uma RNN é criar um espaço latente temporal de dados sequenciais, como textos ou séries temporais. A visualização de uma RNN pode se dar de duas maneiras, como exemplificado na figura 3 [Bhakta, 2023].

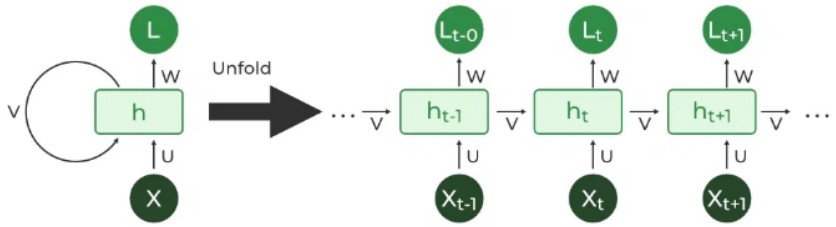


Figura 3: Rede recorrente básica (**Bhakta 2023**).

A primeira (à esquerda) representa diretamente o feedback loop, a outra é uma maneira *desdobrada* de interpretar o loop, como se desenrolássemos o loop para a sequência de dados a serem inseridos, um a um. Como descrita, a rede tem porte para lidar com entradas de tamanhos variados, já que podemos desenrolar as *camadas temporais* conforme a necessidade das entradas. A representação latente temporal é atualizada a cada camada, com informação da sequência até a entrada em questão.

Ainda seguindo a figura, podemos observar a presença de três matrizes de pesos u, v e w, tendo a característica de *compartilhamento de pesos*, isto é, para todos as camadas temporais usamos as mesmas matrizes nas transformações. A atualização das matrizes se dá pelo *backpropagation through time* (BPTT), o fluxo do gradiente é através das camadas temporais. Uma das grandes desvantagens das *vanilla* RNNs está na lenta atualização dos pesos por meio do BPTT, uma vez que precisamos de redes muito profundas para obtermos resultados satisfatórios, um contorno a este problema é truncar o fluxo do gradiente antes do cálculo de todos os gradientes locais, tomando o nome de *Truncated* BPTT. Para a atualização computamos as derivadas da perda em relação aos pesos como se fossem independentes e depois somamos tudo para obter a variação total do peso compartilhado. Algumas variações como as *Bidirectional RNNs* e as *Multilayer RNNs* são comentadas em [Aggarwal et al., 2018, Goodfellow et al., 2016].

Devido a profundidade da rede, as RNNs sofrem de problemas como vanishing ou exploding gradient, penhascos na superfície de perda, dentre outros. Variações surgiram com objetivo de mitigar essas barreiras. Alguns exemplos são as *Echo-State Networks* e as *Gated Recurrent Units* (GRUs) e uma das mais conhecidas, a *Long Short-Term Memory* (LSTM). Nos limitaremos a explicar o funcionamento de uma célula LSTM, mas vale ressaltar que uma GRU é basicamente uma simplificação de uma LSTM, ambas usando da mesma ideia dos *gates*. Uma célula LSTM é como na Figura 4 [İpek, 2021].

Temos um *forget,* um *input* e um *output gate*, o que vem das células anteriores são essencialmente o *cell state* e o *hidden state*, que representam memórias longas e curtas da rede,

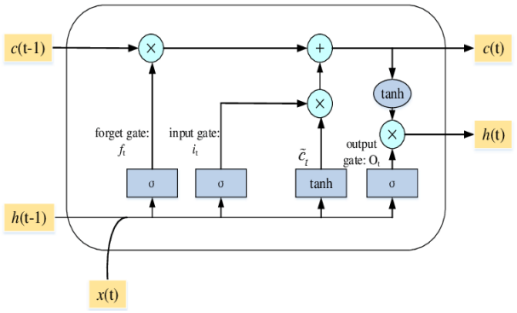


Figura 4: LSTM cell (**Ipek 2021**).

respectivamente, as funções de ativação utilizadas podem ser vistas na imagem. O funcionamento é o seguinte: o forget gate decide quanto da memória longa é mantido, o input gate cria uma memória longa em potencial para se adicionar ao que foi mantido e por fim o output gate atualiza a memória curta com o auxílio de uma memória curta em potencial criada, todos esses passos tem influência da entrada e da memória curta. Embora a imagem não transpareça, antes da passagem pelas funções, os valores são multiplicados por pesos e somados a algum viés, estes parâmetros são compartilhados entre as células LSTM utilizadas. Os problemas com gradiente são grandemente mitigados em relação as vanilla RNNs.

Convolutional Neural Networks. CNNs são um tipo de arquitetura feita para o processamento de dados com topologia em grade (ou *tensores*), tal qual imagens ou uma grade de amostras ao longo do tempo, elas são inspiradas em como o cérebro humano capta e processa imagens. Na matemática a operação de convolução é uma espécie de combinação entre duas funções, podendo ser expressa tanto de forma discreta quanto contínua:

$$(f * g)(t) = \sum_{\tau} f(\tau)g(t-\tau), \quad (f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau.$$

De maneira intuitiva, imaginamos que uma função "desliza" pela outra, captando a forma resultante da composição como uma nova função. Computacionalmente é utilizado de um algoritmo com transformada rápida de Fourier [Brigham, 1988] para realizar convoluções, uma vez que o *teorema da convolução* [Lim, 1990, Stone and Williams, 2008] nos garante que sob condições adequadas a transformada de Fourier de uma convolução é o produto de Hadamard de suas transformadas de Fourier.

No escopo das redes neurais, além das camadas de entrada e saída, podemos categorizar os tipos de camadas que compõem uma CNN em quatro: Convolução; Ativação; *Pooling*; *Fully Connected* (FC, ou totalmente conectada), ressaltamos ainda que, em geral, nas representações não é mostrada a camada de ativação uma vez que depois de uma camada de convolução temos sempre uma ativação.

Poderíamos descrever cada uma das camadas, todavia é imprescindível notar que uma convolução comum não funcionaria para dados estruturados num grafo, que por natureza constituem uma organização muito mais livre em relação aos grids que as CNNs lidam, por este motivo não nos prolongaremos com a descrição das camadas. Ainda assim, existe uma adaptação geral do conceito para o contexto dos grafos, o que nos leva para ao que é uma *Graph Convolutional Network* (GCNs), a base do que são boa parte das redes neurais em grafos.

3 Metodologia

Iremos discorrer sobre cada uma das ferramentas, técnicas e métodos tanto matemáticos quanto computacionais utilizados no decorrer do projeto. Ao longo dessa seção vamos assumir que temos uma amostra $\{(X_i, y_i)\}_{i=1}^n$ onde $X_i \in \mathbb{R}^d$ é o vetor de features de dimensão d associado ao nó v_i e y_i é um valor de saída que buscamos aproximar através do nosso modelo e este pode ser tanto um número real ($y \in \mathbb{R}$), quanto binário ($y \in \{0, 1\}$).

3.1 Redes neurais em grafos

Tomando como base os conceitos, equações e notações apresentados em [Bacciu et al., 2020], vamos detalhar o funcionamento básico e fundamental por trás das *Graph Neural Networks* (GNNs), em especial falaremos sobre a abordagem convolucional ou *multi-camadas*, a qual é empregada na maioria dos modelos.

Um ponto crucial sobre a maior parte dos modelos de GNN é que após treinados temos no lugar dos nós originais representações destes, chamadas de *estados*. Podemos ainda dizer que o grafo final é o *embedding* que o modelo proporciona, que ao final do modelo será conectado com algum tipo de mecanismo de predição. Estes estados são obtidos com a atualização iterativa do que chamamos de vetores de estado $\mathbf{h}_{v_i}$ para qualquer vértice v_i no conjunto de vértices $\mathbf{V}$ do grafo.

A topologia do grafo pode ser variável, o que traz consigo dois desafios emblemáticos. Primeiro, devemos ser capazes de processar grafos com estruturas e tamanhos diferentes. Segundo, como tipicamente os nós de um grafo não possuem ordenação, o processo iterativo que gera os embeddings deve considerar apenas as relações de vizinhança e troca de informações com base em tais relações.

Uma forma de abordar os problemas mencionados acima é utilizando as chamadas *funções invariantes a permutação* que trabalhem de maneira *local* para "aprender" estruturas *globais* do grafo. Tais funções permitem que um nó atualize seu estado localmente utilizando informação de sua vizinhança. Além disso, tais funções suportam configurações com topologia variável. Uma definição precisa e geral para funções invariantes a permutação é dada no Teorema 4.1 de [Wagstaff et al., 2019]. Porém, exemplos simples de tais funções são a soma, a média ou o produto das features ou estados associadas aos nós e/ou arestas do grafo.

A questão que nos resta para obtermos a generalização de tratamento que desejamos é o de grafos possivelmente cíclicos, pois existe o risco de entrarmos num loop infinito durante o cálculo local dos estados para os nós. Tal questão pode ser contornada fazendo uma atualização iterativa dos vetores de estado. Em termos matemáticos, um vetor de estado associado a um nó v_i na iteração $\ell + 1$, denotado como $\mathbf{h}_{v_i}^{\ell+1}$ é obtido a partir dos estados dos nós vizinhos na iteração anterior ℓ. Em particular, podemos interpretar tal processo iterativo como camadas independentes de uma rede neural. Pode-se mostrar que, com tal interpretação, cada camada de processamento recebe informação de vizinhanças mais distantes de cada nó, permitindo controlar o quão localmente o processamento se dá. A família de modelos que utiliza tal arquitetura é chamada de *feedforward*, devido a natureza do *mecanismo de difusão de contexto*. Outras famílias de modelos são as arquiteturas *recursivas* e as arquiteturas *construtivas*, mas não discorreremos sobre elas aqui.

Imaginemos um grafo do tipo mais geral possível, e que contenha vetores de features tanto para seus nós quanto para suas arestas. Exitem duas maneiras de se expressar uma agregação

de vizinhança na construção dos vetores de estado $\mathbf{h}_{v_i}^{\ell+1}$, o caso discreto e o caso contínuo, cujas fórmulas gerais de agregação são descritas nas equações abaixo, respectivamente:

$$\begin{aligned}
\mathbf{h}_{v_i}^{\ell+1} &= \phi^{\ell+1}\left(\mathbf{h}_{v_i}^{\ell}, \sum_{c_k \in \mathbf{E}} (\Psi(\{\psi^{\ell+1}(\mathbf{h}_u^{\ell}) \,|\, u \in \mathcal{N}_{v_i}^{c_k}\}) * w_{c_k})\right), \\
\mathbf{h}_{v_i}^{\ell+1} &= \phi^{\ell+1}\left(\mathbf{h}_{v_i}^{\ell}, \Psi(\{e^{\ell+1}(\mathbf{a}_{uv_i})^T \psi^{\ell+1}(\mathbf{h}_u^{\ell}) \,|\, u \in \mathcal{N}_{v_i}\})\right), \\
\mathbf{h}_{v_i}^{0} &= X_i.
\end{aligned}$$

Nas duas equações temos $\mathbf{h}_u^{\ell}$ sendo o vetor de estado do nó u na camada ℓ, ϕ e ψ transformações dos dados de entrada, Ψ uma função invariante a permutação, $\mathcal{N}_{v_i}$ ou $\mathcal{N}_{v_i}^{c_k}$ a vizinhança de v_i. Em especial para o caso discreto w_{c_k} é um peso treinável que pondera o quanto cada aresta (rotuladas por c_k) afeta a determinação do novo vetor de estado para o nó que ela incide. Para o caso contínuo, temos a aresta direcionada $\mathbf{a}_{uv_i}$ explicitamente no processo , e mais ainda temos que e é uma rede neural que retorna uma matriz de pesos.

Existem ainda dois mecanismos bastante utilizados durante o processo de agregação de vizinhança, a *atenção* e a *amostragem*. Técnicas de pooling para grafos, que reduzem o custo computacional e auxiliam na descoberta de *comunidades* no grafo também são largamente empregadas. Uma discussão detalhada sobre tais conceitos pode ser encontrada em [Bacciu et al., 2020].

No contexto de aprendizado supervisionado, em específico classificação de nós em grafos, onde rótulos são atribuídos a cada nó no grafo, normalmente se utiliza a entropia cruzada como função perda, dada por:

$$L_{CE}(y, t) = -\log\left(\frac{\exp(y_t)}{\sum_{j=1}^{C} \exp(y_j)}\right),$$

onde y e t são o vetor de saída e a classe alvo, sendo C o número de classes.

A GNN que utilizaremos em nosso trabalho é mais um método de se conseguir um embedding para os nós do grafo. Uma *Graph Convolutional Network* (GCN) pode ser vista como uma aproximação de primeira ordem de filtros espectrais locais em grafos, como mostrado no trabalho que fez popular o termo GCN [Kipf and Welling, 2016], nele é definida a regra de propagação de informação para um grafo, atualizando o estado dos nós da seguinte maneira:

$$H^{(\ell+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}D^{-\frac{1}{2}}H^{(\ell)}W^{(\ell)}\right).$$

Sendo $\tilde{A} = A + I_N$ a matriz direcionada do grafo não direcionado, adicionada de auto-loops pelo acréscimo da matriz identidade I_N, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, e a matriz de pesos treináveis na camada ℓ sendo $W^{(\ell)}$. Na matriz $H^{(\ell)} \in \mathbb{R}^{n \times d}$ a i-ésima linha é o vetor de estado do i-ésimo nó naquela camada $\mathbf{h}_{v_i}^{\ell}$, seu estado inicial é dado pelos features dos nós, isto é, $H^{(0)} = X$, por fim σ é uma função de ativação qualquer que desejemos, comumente a ReLU.

3.2 Modelo EvolveGCN

Adaptamos neste trabalho o modelo *EvolveGCN*, descrito em [Pareja et al., 2020]. A escolha deste modelo se deve em grande parte por sua simplicidade, consistindo na junção direta de *Graph Convolutional Network* (GCN) com uma *Recurrent Neural Network* (RNN) que atualiza os parâmetros da GCN, ou seja, apenas a RNN é treinada. O esquema mostrado

na Figura 5 [Pareja et al., 2020] é uma representação visual de como ocorrem as operações deste modelo, explicadas por partes a seguir.

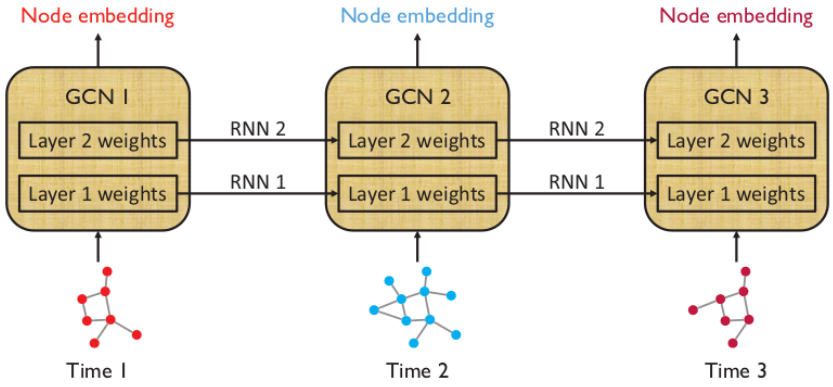


Figura 5: Ilustração esquemática do modelo (**Pareja 2020**).

A entrada do modelo consiste em duas matrizes A_t e X_t a cada passo de tempo t, de dimensões $n \times n$ e $n \times d$ respectivamente, onde n é a quantidade de nós e d a quantidade de features de cada nó. Em outras palavras, A_t é a matriz de adjacência do grafo e X_t a matriz contendo as features de cada nó, respectivamente. Note que a cada instante de tempo um novo grafo é apresentado ao modelo.

O modelo começa a ser aplicado primeiramente passando por uma GCN [Kipf and Welling, 2016] simples com diversas camadas de convolução em grafos. Como o artigo descreve, no tempo t, a camada l recebe a matriz de adjacência A_t e a matriz dos embeddings dos nós $H_t^{(l)}$. Com sua matriz de pesos $W_t^{(l)}$ da camada l, atualiza-se o embedding dos nós para $H_t^{(l+1)}$. Em resumo, na GCN ocorre a seguinte transformação:

$$H_t^{(l+1)} = \text{GCONV}(A_t, H_t^{(l)}, W_t^{(l)}) \coloneqq \sigma(\hat{A}_t H_t^{(l)} W_t^{(l)}),$$

com o estado inicial do embedding dos nós dado por $H_t^{(0)} = X_t$, isto é, as features dos nós. Além disso, σ é uma função de ativação e $\hat{A}$ é a normalização de A_t dada por:

$$\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}; \quad \tilde{A} = A + I; \quad \tilde{D} = \text{diag}\left(\sum_j \tilde{A}_{ij} \right).$$

A matriz de pesos da GCN, $W_t^{(l)}$ (estado oculto), é atualizada por uma RNN, o modelo a princípio propõe duas maneiras disso ocorrer:

1. Levando em consideração o embedding dos nós para a atualização, utilizando de uma *Gated Recurrent Unit* (GRU) como a RNN, este foi o método escolhido para realização dos experimentos neste trabalho. Em termos simples temos a operação

$$W_t^{(l)} = \text{GRU}(H_t^{(l)}, W_{t-1}^{(l)}).$$

Aqui a matriz de pesos é descrita como sendo o estado oculto do sistema dinâmico, uma vez que utilizamos do embedding dos nós e a versão de $W_{t-1}^{(l)}$ no passo anterior para obter um novo estado a ser utilizado. O pseudocódigo da operação é como descrito abaixo:

1: **function** GRU$(H_t^{(l)}, W_{t-1}^{(l)})$

2: $Z_t = \text{sigmoid}(W_Z H_t^{(l)} + U_Z W_{t-1}^{(l)} + B_Z)$
3: $R_t = \text{sigmoid}(W_R H_t^{(l)} + U_R W_{t-1}^{(l)} + B_R)$
4: $\tilde{W}_t = \tanh(W_H H_t^{(l)} + U_H(R_t \circ W_{t-1}^{(l)}) + B_H)$
5: $W_t = (1 - Z_t) \circ W_{t-1}^{(l)} + Z_t \circ \tilde{W}_t$
6: **return** $W_t^{(l)}$
7: **end function**

Onde $\circ$ indica um produto de Hadamard e W_i, U_i e B_i são os parâmetros do modelo, com $i \in \{H, R, Z\}$.

Escolhendo este método de atualização, antes de aplicar a GRU temos de nos preocupar com as dimensões da entrada corresponderem às do estado oculto para podermos realizar as operações. Isto é resolvido com uma função de sumarização que em resumo toma os embeddings de nós mais representativos, um número deles igual ao da dimensão do estado oculto, esta função é detalhada em [Pareja et al., 2020].

2. Levando em consideração apenas o estado anterior da matriz de pesos e utilizando de uma *Long-Short Term Memory* (LSTM) como a RNN. Neste caso a atualização dos pesos é dada por:

$$W_t^{(l)} = \text{LSTM}(W_{t-1}^{(l)}).$$

Neste caso a matriz de pesos é descrita como a saída do sistema dinâmico, e se tratando de uma RNN torna-se a entrada do passo seguinte. Seu pseudocódigo é como segue:

1: **function** LSTM($W_{t-1}^{(l)}$)
2: $F_t = \text{sigmoid}(W_F W_{t-1}^{(l)} + U_F W_{t-1}^{(l)} + B_F)$
3: $I_t = \text{sigmoid}(W_I W_{t-1}^{(l)} + U_I W_{t-1}^{(l)} + B_I)$
4: $O_t = \text{sigmoid}(W_O W_{t-1}^{(l)} + U_O W_{t-1}^{(l)} + B_O)$
5: $\tilde{C}_t = \tanh(W_C W_{t-1}^{(l)} + U_C W_{t-1}^{(l)} + B_C)$
6: $C_t = F_t \circ C_{t-1} + I_t \circ \tilde{C}_t$
7: $W_t^{(l)} = O_t \circ \tanh(C_t)$
8: **return** $W_t^{(l)}$
9: **end function**

Onde $\circ$ indica um produto de Hadamard e W_i, U_i e B_i são os parâmetros do modelo, com $i \in \{C, F, I, O\}$.

Sem a necessidade de uma função auxiliar este método é mais direto, embora não leve em consideração o embedding dos nós para a evolução temporal dos parâmetros.

A saída dos passos apresentados é um embedding dos nós para cada tempo. Com isso, podemos utilizar qualquer tipo de preditor para grafos, adicionando por exemplo uma camada de predição de aresta, um classificador de grafo ou (nosso caso) um classificador de nós. Mais detalhes são apresentados em [Pareja et al., 2020].

3.3 Base de dados: Furtos/Assaltos em São Paulo

Estaremos trabalhando com um grafo de ruas, tomamos a cidade de São Paulo como caso de estudo, tanto pela quantidade de dados disponíveis quanto pela riqueza destes.

O grafo foi obtido em Python, por meio da biblioteca Osmnx [Boeing, 2024]. Essa biblioteca recebe como argumentos o nome de uma região em formato de string, como "São Paulo-SP, Brazil" ou um polígono definido pelas coordenadas geográficas de latitude e longitude dos pontos desejados. Escolhemos restringir nossa análise ao centro de São Paulo, para reduzir o escopo da análise e por ser a região com maior número de ocorrências criminais.

Por padrão o grafo é direcionado com arestas sendo ruas e os nós os cruzamentos entre elas. Seguindo o usual da literatura, processamos estes dados para o grafo ser não direcionado, com os nós sendo os segmentos de ruas e arestas entre eles quando existe um cruzamento.

As features utilizadas até o momento são estáticas e retiradas do site público Geosampa (https://geosampa.prefeitura.sp.gov.br), o qual provê ferramentas para extração de dados variados sobre a cidade.

As features escolhidas para os testes foram pontos de interesse, por exemplo quantos serviços de saúde, cultura, segurança, educação, assistência social e outros se encontram a uma distância pré fixada de um nó onde ocorreu um crime.

Uma projeção do grafo de ruas utilizado em cima do próprio mapa da região escolhida é apresentada na Figura 6, para projeções do tipo foi utilizado o módulo Folium [Filipe et al., 2023] do Python.

Figura 6: Grafo de ruas do centro de São Paulo.

A única característica temporal em nosso problema são as ocorrências criminais ao longo do tempo. Os dados de crime foram obtidos do portal da secretaria de segurança pública do estado de São Paulo (https://www.ssp.sp.gov.br/estatistica/consultas). O escopo de casos considerados foi para crimes de subtração (furto ou assalto) de celulares e veículos no período de 01/jan/2022 até 31/dez/2023. Em nossa base, temos um total de 58964 ocorrências e 12 features utilizadas para cada nó. Além disso, temos informações sobre latitude e longitude de cada ocorrência criminal, com as quais podemos realizar projeções dos crimes no mapa a fim de comparar com as predições realizadas. O processamento dos dados foi feito através do módulo Geopandas do Python [Jordahl et al., 2020].

Um dos pontos cruciais sobre o conjunto de dados utilizado é seu desbalanceamento, isto significa que temos muitas amostras de uma das classes (classe majoritária) a serem preditas em comparação a outra (classe minoritária), sendo essas classes não crime e crime, respectivamente. O estudo de dados desbalanceados é uma área ativa de pesquisa e existem diversos métodos para tratar o desbalanceamento [Barella et al., 2021, Fernández et al., 2018, He and Garcia,

2009]. No nosso caso, adotamos o método de variar o peso dado ao erro das classes, dando mais importância ao erro da classe minoritária.

Os dados de crime são registrados no site da segurança pública com precisão de minutos, decidimos agrupá-los mensalmente para aliviar tanto a esparsidade temporal dos dados quanto o desbalanço, com essa partição temos aproximadamente 4.5 vezes mais casos da classe majoritária em relação à minoritária.

Para os experimentos realizados a partição foi de 75% dos dados para treino, 12.5% para validação e 12.5% para teste.

4 Resultados

No nosso caso, temos um problema de classificação binária, onde 1 representa a ocorrência de um crime e 0 a não ocorrência. Para avaliar nosso modelo, utilizamos as seguintes métricas, que são amplamente utilizadas nesse tipo de cenário:

1. **Precisão:** A Precisão indica o número total de previsões para uma classe especificadas como positivas que são atribuídas corretamente. Matematicamente, é definida como $\frac{VP}{VP+FP}$, onde VP é a quantidade de Verdadeiros Positivos e FP os Falso Positivos.

2. **Revocação:** A Revocação é o número total de casos positivos reais que foram previstos corretamente para uma classe. Matematicamente, é definida como $\frac{VP}{VP+FN}$, onde VP é a quantidade de Verdadeiros Positivos e FN os Falso Negativos.

3. **Medida F1:** É a média harmônica das medidas de Revocação e Precisão para uma classe.

4. **Microavg F1:** As medidas Micro Average (aqui abreviadas para Microavg) funcionam como uma medida média global para problemas de classificação multi classes. Simplesmente se somam as quantidades de Verdadeiros Positivos, Falsos Negativos e Falsos Positivos separadamente e para cada uma das classes do problema e se calculam as medidas. Dessa forma temos uma equivalência das medidas de Precisão, Revocação e F1 quando realizamos uma média com o método de Micro Average.

Existem três hiperparâmetros principais na nossa modelagem, que desempenham um papel crucial na performance do modelo. São eles:

1. `class_weights`: lista com dois elementos numéricos reais, na segunda posição indicando o peso da classe alvo, em nosso caso a classe 1, e na primeira o peso da classe 0.

2. `num_hist_steps`: quantidade de passos temporais que o modelo deve considerar para realizar um predição, por exemplo, com nossa partição em meses caso este parâmetro seja igual a três serão desenroladas três camadas temporais da RNN, cada uma para a informação sobre um mês, e a saída será a predição para o mês seguinte, o quarto.

3. `feats_per_node`: Este parâmetro serve principalmente para definir dimensões de entrada e saída entre camadas da rede, podendo reduzir, aumentar ou manter a quantidade de elementos de um vetor de estado conforme o processamento.

Foi realizada uma bateria de experimentos variando estes parâmetros e foi escolhido o *tradeoff* mais vantajoso entre as métricas utilizadas, decidimos ao fim escolher os hiperparâmetros de acordo com a estabilidade do desempenho apresentada no conjuntos de treino e validação. Com isso, para o resultado abaixo discorrido, foram fixados os hiperparâmetros `class_weights` $= [1.0, 10.0]$, `num_hist_steps` $= 10$ e `feats_per_node` $= 15$. É importante ressaltarmos que ao propor uma solução para nosso problema estamos priorizando as métricas para a classe 1, quando há ocorrência de crimes, com isso após treinamento, para a última época de teste temos os valores para as métricas priorizadas como na tabela abaixo:

microavg_f1	class1_precision	class1_recall	class1_f1	mean_errors
0.3505	0.2522	0.9619	0.3997	0.64953

Podemos verificar a evolução destas medidas ao observarmos como elas variam com a passagem das épocas, a Figura 7 mostra essa evolução para a classe 1.

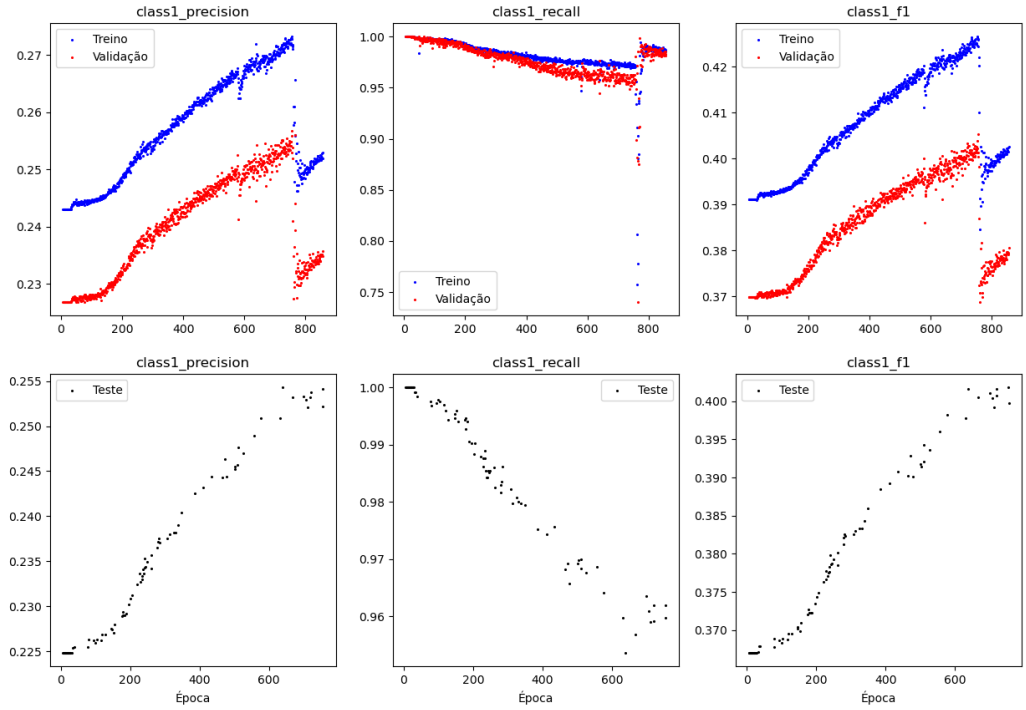


Figura 7: Evolução das métricas de Precisão, Revocação e f1 para classe minoritária. Na imagem de cima, temos essa evolução com a passagem das épocas nos conjuntos de treino e validação, na imagem de baixo é mostrada essa evolução no conjunto de teste.

Aqui fizemos a representação do conjunto de teste separadamente por ser o conjunto que mais nos interessa para avaliação do nosso modelo. Note que para os conjuntos de treino e validação há uma mudança brusca nas métricas por volta da época 750.

Podemos ainda verificar essa evolução para os erros médios (ou `mean_errors`) na Figura 8, com a mesma mudança brusca nos conjuntos de treino e validação.

A partir deste gráfico podemos concluir que o modelo não está sofrendo de overfitting uma vez que as métricas nos conjuntos de treino e validação andam sempre juntas. Além disso, não temos resultados satisfatórios quando olhamos para os valores das métricas por si só, junto ainda

ao possível vale na superfície de perda que gerou a mudança brusca nas métricas nos conjuntos de treino e validação, todos esses fatores indicam que o modelo como está provavelmente não consegue lidar com a esparsidade dos dados. Isso não descarta o modelo para o estudo do problema ainda, uma vez que podemos implementar diversas outras adaptações à ele.

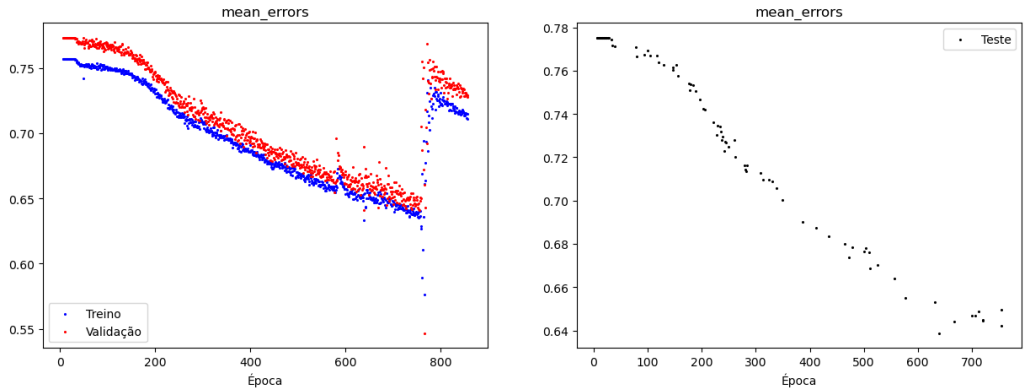


Figura 8: Evolução do erro médio ao longo das épocas para os conjuntos de treino e validação (esquerda) e teste (direita).

Apesar das métricas não estarem tão positivas, visualizando os resultados no mapa de São Paulo (Figura 9), torna-se evidente que o modelo capturou informações significativas. Conforme destacado na Figura 9b, identificamos algumas regiões notáveis no mapa, denominadas A, B, C e D. Essas áreas são notáveis porque coincidem com informações de senso comum sobre locais de alta ou baixa criminalidade no centro de São Paulo. Vamos detalhar cada uma delas:

A. A região **A** é todo o entorno do bairro da **República** em São Paulo, amplamente conhecido por ser perigoso e haver alto risco de roubo ou furto na região.

B. A região **B**, logo acima do bairro da República é uma área que segundo nosso modelo é mais segura, ao averiguarmos a região notamos a existência de alguns bancos como Bradesco, Santander e Itaú nas quadras próximas.

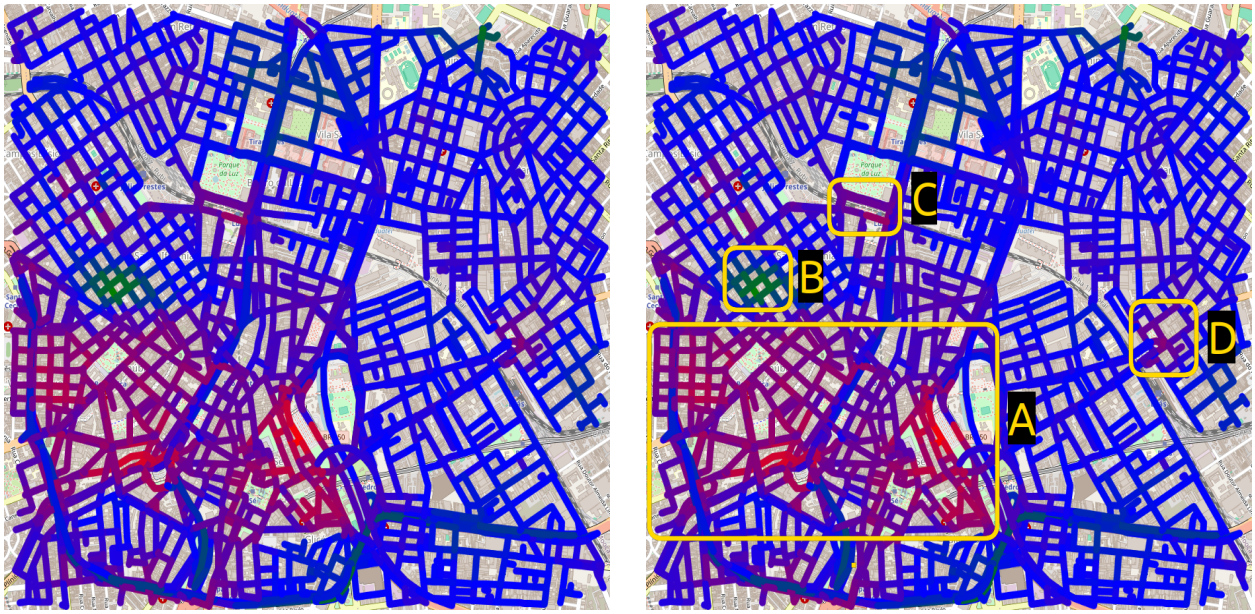
C. A região **C** é todo o entorno da **Estação da Luz** de São Paulo, também conhecida por ser uma região de alta movimentação e perigosa.

D. A região **D** consiste nas proximidades da praça **Largo da Concórdia** em São Paulo, que fica também próxima de um **Lojão do Brás**. Por ser uma região em que o transporte se dá majoritariamente a pé, existem muitos casos de batedores de carteira ou semelhantes, levando a um alto índice de roubos e furtos de celulares na região.

Com este resultado, percebemos que o modelo ainda tem potencial de melhora com possíveis adaptações a serem implementadas, principalmente no contexto de *mapeamento de hotspots*.

4.1 Efeitos dos hiperparâmetros

Nesta seção faremos uma breve explanação sobre como os hiperparâmetros `class_weights`, `num_hist_steps` e `feats_per_node` influenciam nas métricas, algo notado durante os experimentos foi que a combinação entre os parâmetros pode ter algum efeito que ultrapasse a influência individual de cada parâmetro. Para a Figura 10 variamos cada um dos hiperparâmetros



(a) Mapa de calor das predições (b) Regiões notáveis do mapa de calor das predições

Figura 9: Mapas de calor para as predições do modelo na última época de teste.

mencionados e tomamos para cada valor deles a melhor medida da métrica indicada, tudo isso no conjunto de teste dos dados.

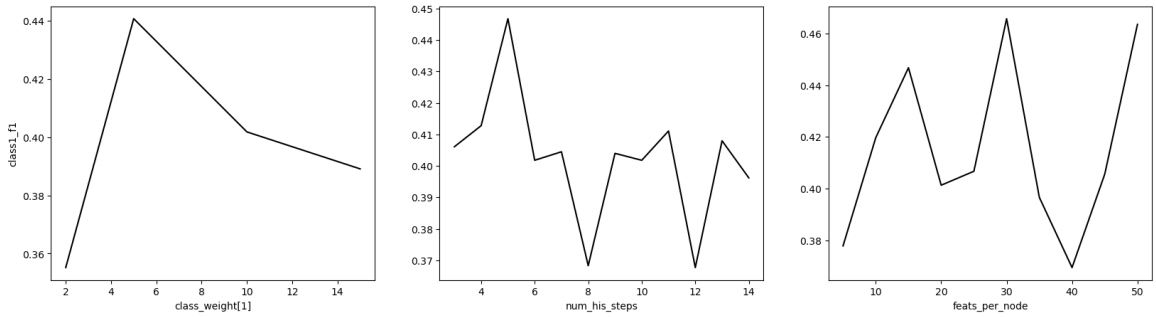


Figura 10: Métrica `class1_f1` no conjunto de teste, variando os hiperparâmetros.

Notamos a alta sensibilidade do modelo a estes hiperparâmetros. Notamos estabilidade para os parâmetros escolhidos no experimento da seção anterior, todavia são necessárias mais análises estatísticas acerca destas variações para se concluir qual o melhor conjunto de hiperparâmetros para o problema.

5 Conclusões e considerações finais

Analisando exclusivamente as métricas, podemos inferir que o modelo não é apropriado para abordar o problema na sua forma atual. No entanto, ao observarmos o mapa de calor, percebemos que há potencial no modelo em relação ao mapeamento de áreas críticas (ou *hotspots*).

Como trabalhos futuros, listamos a continuação de testes com a variação dos hiperparâmetros e análises estatísticas em cima disso, a implementação de outras adaptações ao modelo como a introdução de métricas próprias para problemas em grafos no lugar de métricas gerais como revocação e precisão, mudança de paradigma de predição de crimes para mapeamento de hotspots, dentre outras. Por fim pretendemos realizar experimentos agrupando os dados diariamente e utilizando da cidade toda, atacando ainda mais a esparsidade do problema.

Referências

[Aggarwal et al., 2018] Aggarwal, C. C. et al. (2018). Neural networks and deep learning. *Springer*, 10(978):3.

[Bacciu et al., 2020] Bacciu, D., Errica, F., Micheli, A., and Podda, M. (2020). A gentle introduction to deep learning for graphs. *Neural Networks*, 129:203–221.

[Barella et al., 2021] Barella, V. H., Garcia, L. P., de Souto, M. C., Lorena, A. C., and de Carvalho, A. C. (2021). Assessing the data complexity of imbalanced datasets. *Information Sciences*, 553:83–109.

[Bhakta, 2023] Bhakta, S. S. (2023). Stacked rnns in nlp. https://www.geeksforgeeks.org/stacked-rnns-in-nlp/.

[Bianchini et al., 2005] Bianchini, M., Gori, M., and Scarselli, F. (2005). Inside pagerank. *ACM Transactions on Internet Technology (TOIT)*, 5(1):92–128.

[Bishop and Nasrabadi, 2006] Bishop, C. M. and Nasrabadi, N. M. (2006). *Pattern recognition and machine learning*. Springer.

[Boeing, 2024] Boeing, G. (2024). Modeling and analyzing urban networks and amenities with osmnx.

[Brigham, 1988] Brigham, E. O. (1988). *The fast Fourier transform and its applications*. Prentice-Hall, Inc.

[Ding et al., 2018] Ding, B., Qian, H., and Zhou, J. (2018). Activation functions and their characteristics in deep neural networks. In *2018 Chinese control and decision conference (CCDC)*, pages 1836–1841. IEEE.

[Fernández et al., 2018] Fernández, A., García, S., Galar, M., Prati, R. C., Krawczyk, B., and Herrera, F. (2018). *Learning from imbalanced data sets*, volume 10. Springer.

[Filipe et al., 2023] Filipe, Anema, F., Story, R., Gardiner, J., Journois, M., Bird, A., Lima, A., Cano, J., dbf, Oefelein, M., Leonel, J., Baker, J., Sampson, T., Rump, H., Reades, J., Welsh, B., Kong, Q., Komarov, O., Crosby, A., Harris, G., Dumas, R., KRIEF, D., Kato, D., Mendoza, D., Wilson, N., Nogueira, T. P., kenmatsu4, Furtado, L., and Patil, A. (2023). python-visualization/folium: v0.15.1.

[Garcia-Zanabria et al., 2021] Garcia-Zanabria, G., Raimundo, M. M., Poco, J., Nery, M. B., Silva, C. T., de Abreu, S. F. A., and Nonato, L. G. (2021). Cripav: Street-level crime patterns analysis and visualization. *IEEE Transactions on Visualization and Computer Graphics (online first)*.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

[Han et al., 2020] Han, X., Hu, X., Wu, H., Shen, B., and Wu, J. (2020). Risk prediction of theft crimes in urban communities: an integrated model of lstm and st-gcn. *IEEE Access*, 8:217222–217230.

[Haykin et al., 2009] Haykin, S. S. et al. (2009). Neural networks and learning machines/simon haykin.

[He and Garcia, 2009] He, H. and Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284.

[İpek, 2021] İpek, A. (2021). Comparison of short-term electricity load forecasting using different deep learning methods. *Avrupa Bilim ve Teknoloji Dergisi*, pages 616–623.

[Janocha and Czarnecki, 2017] Janocha, K. and Czarnecki, W. M. (2017). On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*.

[Jin et al., 2020] Jin, G., Wang, Q., Zhu, C., Feng, Y., Huang, J., and Zhou, J. (2020). Addressing crime situation forecasting task with temporal graph convolutional neural network approach. In *2020 12th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*, pages 474–478. IEEE.

[Jordahl et al., 2020] Jordahl, K., den Bossche, J. V., Fleischmann, M., Wasserman, J., McBride, J., Gerard, J., Tratner, J., Perry, M., Badaracco, A. G., Farmer, C., Hjelle, G. A., Snow, A. D., Cochran, M., Gillies, S., Culbertson, L., Bartos, M., Eubank, N., maxalbert, Bilogur, A., Rey, S., Ren, C., Arribas-Bel, D., Wasser, L., Wolf, L. J., Journois, M., Wilson, J., Greenhall, A., Holdgraf, C., Filipe, and Leblanc, F. (2020). geopandas/geopandas: v0.14.4.

[Kipf and Welling, 2016] Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

[Lim, 1990] Lim, J. S. (1990). *Two-dimensional signal and image processing*. Prentice-Hall, Inc.

[Longa et al., 2023] Longa, A., Lachi, V., Santin, G., Bianchini, M., Lepri, B., Lio, P., Scarselli, F., and Passerini, A. (2023). Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities. *arXiv preprint arXiv:2302.01018*.

[Ognjanovski, 2019] Ognjanovski, G. (2019). Everything you need to know about neural networks and backpropagation. https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a.

[Pareja et al., 2020] Pareja, A., Domeniconi, G., Chen, J., Ma, T., Suzumura, T., Kanezashi, H., Kaler, T., Schardl, T., and Leiserson, C. (2020). Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 5363–5370.

[Rasamoelina et al., 2020] Rasamoelina, A. D., Adjailia, F., and Sinčák, P. (2020). A review of activation function for artificial neural network. In *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 281–286. IEEE.

[Rosser et al., 2017] Rosser, G., Davies, T., Bowers, K. J., Johnson, S. D., and Cheng, T. (2017). Predictive crime mapping: Arbitrary grids or street networks? *Journal of quantitative criminology*, 33(3):569–594.

[Sharma et al., 2017] Sharma, S., Sharma, S., and Athaiya, A. (2017). Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316.

[Shlomi et al., 2020] Shlomi, J., Battaglia, P., and Vlimant, J.-R. (2020). Graph neural networks in particle physics. *Machine Learning: Science and Technology*, 2(2):021001.

[Stone and Williams, 2008] Stone, H. S. and Williams, L. (2008). On the uniqueness of the convolution theorem for the fourier transform. *NEC Labs. Amer. Princeton, NJ. Online: http://citeseer. ist. psu. edu/176038. html, accessed on*, 19.

[Wagstaff et al., 2019] Wagstaff, E., Fuchs, F., Engelcke, M., Posner, I., and Osborne, M. A. (2019). On the limitations of representing functions on sets. In *International Conference on Machine Learning*, pages 6487–6494. PMLR.

[Wang et al., 2020a] Wang, Q., Ma, Y., Zhao, K., and Tian, Y. (2020a). A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, pages 1–26.

[Wang et al., 2020b] Wang, Y., Ge, L., Li, S., and Chang, F. (2020b). Deep temporal multi-graph convolutional network for crime prediction. In *International Conference on Conceptual Modeling*, pages 525–538. Springer.

[Wang et al., 2021] Wang, Z., Jiang, R., Cai, Z., Fan, Z., Liu, X., Kim, K.-S., Song, X., and Shibasaki, R. (2021). Spatio-temporal-categorical graph neural networks for fine-grained multi-incident co-prediction. In *Proceedings of the 30th ACM international conference on information & knowledge management*, pages 2060–2069.

[You et al., 2018] You, J., Liu, B., Ying, Z., Pande, V., and Leskovec, J. (2018). Graph convolutional policy network for goal-directed molecular graph generation. *Advances in neural information processing systems*, 31.

[Yu et al., 2021] Yu, L., Du, B., Hu, X., Sun, L., Han, L., and Lv, W. (2021). Deep spatio-temporal graph convolutional network for traffic accident prediction. *Neurocomputing*, 423:135–147.

[Zhang and Cheng, 2020] Zhang, Y. and Cheng, T. (2020). Graph deep learning model for network-based predictive hotspot mapping of sparse spatio-temporal events. *Computers, Environment and Urban Systems*, 79:101403.