UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

FÁBIO DE AZEVEDO GOMES

Ferramenta de Análise, Visualização e Previsão de Acidentes de Trânsito no Município de Porto Alegre

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientadora: Profa. Dra. Renata Galante

Porto Alegre
2022

AGRADECIMENTOS

Agradeço à minha família, minha mãe Luciana, pai Herbert e padrasto Emerson por me sustentarem durante minha graduação. À meus colegas de trabalho que entenderam e aceitaram meus períodos de indisponibilidade durante a graduação. Por fim, agradeço a todos os professores e professoras do curso de Ciência da Computação, pela disponibilidade e auxílio que me deram durante todo o curso.

RESUMO

Com o advento de iniciativas por parte de diversos governos para a divulgação de conjuntos de dados pertinentes a tópicos de interesse público de maneira aberta, diversas portas são abertas para o desenvolvimento não-governamental de ferramentas e aplicativos que visam entender e melhorar a situação que os dados refletem. O trânsito de uma cidade é um exemplo destes tópicos que afeta todo cidadão que ali reside, também sendo uma das maiores fontes de problema que acompanha o desenvolvimento de cidades modernas. A prefeitura de Porto Alegre disponibiliza de maneira aberta os registros históricos de acidentes de trânsito que ocorreram dentro de seu território, contendo, entre outras, informações sobre data, hora, local e veículos envolvidos. Outro conjunto de dados abertos que pode ser utilizado em conjunto aos acidentes é o conjunto de dados meteorológicos da cidade, o qual também é disponibilizado. O objetivo deste trabalho é implementar uma aplicação *web* capaz de representar estes dados de maneira visual, em um mapa da cidade, os quais poderão ser consultados e filtrados pelo usuário interativamente. Além disso, a ferramenta permite que seu usuário realize consultas sobre datas futuras, prevendo através de uma rede neural de perceptrons densamente conectada a probabilidade de novos acidentes em determinados pontos da cidade, sugerindo ao seu usuário quais caminhos devem ser evitados caso o mesmo deseje se locomover de um local até outro. Através desta ferramenta pretendemos melhor compreender quais são os fatores que influenciam estas eventuais ocorrências, além de assistir a população em evitar as mesmas.

Palavras-chave: Plataforma Web. acidentes de trânsito. Porto Alegre. visualização. inteligência artificial. encontro de caminhos. base de dados.

Name: A Tool for Analysis, Visualization and Prediction of Traffic Accidents in Porto Alegre

ABSTRACT

With the advent of initiatives by many governments for the disclosure of data sets regarding topics of public interest openly, many doors are opened for the development of non-governmental tools and applications which attempt to understand and improve the situation reflected in the data. The traffic within a city's streets is an example of these topics which affect every member of it's community, also being one of the biggest sources of problems which accompany the development of modern cities. The prefecture of the city of Porto Alegre openly provides historical records of traffic accidents that have occured within it's territory, containing, among others, information about date, time, place and vehicles involved. Another open data set which can be used together with traffic accidents is the meteorological data for the city, which is also openly disclosed. The objective of this work is to develop a web application capable of representing this data in a visual manner in a city map, allowing search queries, filtering and interaction with the user. Furthermore, the tool will allow the user to perform queries regarding future dates, predicting, via a densely connected perceptron neural network, the probability of new accidents in certain points of the city, suggesting which paths should be avoided in case they wish to move around town. Through this tool we intend to better understand which factors influence these accidents, as well as assist the city's population in avoiding them.

Keywords: Web platform. traffic accidents. Porto Alegre. visualization. artificial intelligence. database. pathfinding.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ANTP	*Associação Nacional de Transportes Públicos*
SIMOB	*Sistema de Mobilidade Urbana*
ANPET	*Associação Nacional de Pesquisa e Ensino em Transportes*
ANTT	*Agência Nacional de Transportes Terrestres*
ONTL	*Observatório Nacional de Transporte e Logística*
OSM	*Open Street Map*
ODBl	*Open Data Commons Open Database License*
ConvLSTM	*Convolutional Long Short-Term Memory*
EPTC	*Empresa Pública de Transporte e Circulação*
CCAL	*Creative Commons Attribution License*
INMET	*Instituto Nacional de Meteorologia*
API	*Application Programming Interface*
CRUD	*Create, Retrieve, Update, Delete*
JSON	*JavaScript Object Notation*
HTML	*Hyper Text Markup Language*
REST	*Representational State Transfer*
RNN	*Recurrent Neural Network*
MSE	*Mean Squared Error*

SUMÁRIO

1 INTRODUÇÃO

Veículos automotivos são uma parte essencial do trajeto diário de qualquer cidadão trabalhador brasileiro, seja ele um automóvel pessoal, de transporte público ou fretado. De maneira geral, podemos classificar estes veículos em dois tipos: aqueles que seguem um caminho fixo, como trens e metrô, e aqueles que são pilotados por um só indivíduo e são livres para seguir qualquer caminho, como carros, motos e ônibus.

De acordo com o SIMOB, mantido anualmente pela ANTP(ANTP, 2018), no ano de 2018 por volta de 85% de passageiros de transporte público se encaixam no segundo tipo, sendo ônibus municipal e metropolitano, enquanto que apenas 15% são transportados sobre trilhos. Se incluirmos veículos pessoais, temos que esta razão se aproxima de 93% de veículos nas avenidas e rodovias e apenas 7% em trilhos, podendo afirmar que o transporte nas ruas é a grande maioria.

Em uma pesquisa realizada pela ANTT para o 34º Congresso da ANPET(MIGUEL, 2022) temos uma análise sobre acidentes ferroviários, apontando que entre os anos de 2011 e 2019 ocorreram, em média, 845 acidentes ferroviários por ano. Além disso, o Anuário Estatístico de Transportes de 2019 divulgado pelo ONTL(LOGíSTICA, 2019) revela a média de acidentes de trânsito rodoviário para este mesmo período é de 130 mil acidentes por ano. Com estas informações em mente, temos que a razão de viagens por tipo de transporte vista anteriormente, 93% para 7%, está muito distante da razão de acidentalidade vista aqui, 130.000 para 845, uma diferença de 7 ordens de magnitude.

O transporte nas ruas trás diversas vantagens, como a liberdade de tomar qualquer caminho que o piloto ou motorista desejar e não estar preso a um cronograma universal, porém estas vantagens vêm a custo do controle que cada indivíduo tem sobre a sua segurança no trânsito. No transporte por trilho, diversos sistemas de segurança estão instalados como sensores e freios automáticos, que trazem uma maior confiança para os passageiros de que não haverá perigo durante sua movimentação, além de sistemas capazes de amenizar a severidade de tais eventualidades. Em contrapartida, um motorista de ônibus ou carro depende de fatores que estão fora de seu

controle, como a condição das ruas, a visibilidade e, pior de todos, outros condutores.

Tendo isto em mente, este trabalho tem como objetivo o desenvolvimento de uma aplicação aberta e de simples utilização, capaz de demonstrar ao seu usuário uma análise dos dados de acidentalidade na capital do Rio Grande do Sul, o município de Porto Alegre. Além de permitir que o usuário realize consultas sobre acidentes que ocorreram no período de 2011 a 2018, a aplicação também oferecerá uma estimativa de possíveis acidentes que virão a ocorrer no futuro, através do treinamento uma rede neural de perceptrons, técnica amplamente utilizada para execução de previsões com base em ocorrências passadas coletadas. A interação com esta aplicação é realizada através de uma página *web* acessível por qualquer dispositivo com um navegador de forma a torná-la independente de qualquer plataforma, usufruindo da arquitetura cliente-servidor para servir a página. Avaliamos então os resultados através de comparações entre as previsões realizadas e os acidentes que de fato ocorreram no dia, além de avaliar os caminhos sugeridos quanto à sua qualidade e performance em tempo de execução.

A estrutura deste texto é a seguinte: o Capítulo 2 introduz os conceitos que são abordados neste trabalho, além das tecnologias que são utilizadas para desenvolver a aplicação. O Capítulo 3 traz os principais trabalhos relacionados ao tópico de acidentalidade, realizando uma análise comparativa entre os mesmos e o trabalho atual. No Capítulo 4, realizamos a quebra das funcionalidades descritas, além de como elas foram desenvolvidas e damos uma visão da arquitetura completa da aplicação. O Capítulo 5 apresenta o resultado desta implementação, analisando através de métodos empíricos a qualidade do código, performance e funcionalidades. Por fim, no Capítulo 6, revisamos tudo o que foi desenvolvido, refletindo sobre a adequação das técnicas utilizadas e os resultados do desenvolvimento.

2 CONCEITOS E TECNOLOGIAS

Neste Capítulo apresentamos as tecnologias, aplicações, técnicas e conceitos que foram utilizados no desenvolvimento deste trabalho. Estes conceitos abrangem diversas áreas abordadas no curso como Bancos de Dados, Engenharia de Software, Inteligência artificial, entre outros.

2.1 Fontes de Dados

Esta Seção discorre sobre as fontes das quais os dados utilizados para o desenvolvimento foram retirados.

2.1.1 Registros de Acidentes

Os registros históricos detalhados sobre instâncias de acidentes de trânsito foram retirados do portal de dados abertos da cidade de Porto Alegre(DADOS. . . , 2022), disponibilizados pela EPTC do município de maneira aberta e sobre licença CCAL. Os dados abrangem acidentes do período de 2017 até julho de 2022, totalizando 66445 registros no momento da coleta.

2.1.2 Dados Meteorológicos

Os dados sobre precipitação e umidade foram obtidos do portal do INMET (PORTAL. . . , 2022), disponibilizados de modo aberto conforme a Lei de Acesso à Informação nº 12.527/2011. Os dados têm origem na estação meteorológica número 83967, localizada no centro da cidade de Porto Alegre, cobrindo o período de 2017 até julho de 2022. Um total de 3 medições são efetuadas em cada dia dentro deste período, nos horários de 00:00, 12:00 e 18:00, totalizando 6048 registros climáticos.

2.1.3 Dados Geográficos

De maneira a oferecer a funcionalidade de sugestão de caminhos, foi utilizada a base de dados aberta OSM(OPENSTREETMAPS, 2022), licenciada sobre ODBl, para capturar a rede de ruas da cidade de Porto Alegre na forma de um grafo. A obtenção destes dados foi facilitada através do módulo *python* OSMnx(BOEING, 2017), capaz de interagir com esta base de dados de maneira simples.

2.2 Conceitos

Aqui apresentamos os principais conceitos essenciais para o entendimento da implementação, suas principais características, pontos fortes e fracos e o motivo por trás de sua escolha.

2.2.1 REST API

Uma API REST é uma interface de face externa que adere ao padrão REST(FIELDING; TAYLOR, 2000), permitindo interações com outras aplicações de maneira fácil e intuitiva. Para que uma interface seja considerada *RESTful*, ela deve permitir a identificação e manipulação de recursos de um sistema sem que seja necessário manter informação do estado do cliente, através de mensagens auto-descritivas. Isto significa que o servidor se torna altamente escalável, uma vez que não mantém informações de seus clientes, além de apresentar um aumento na sua velocidade e responsividade. Não só isso, a modularidade do sistema também aumenta, uma vez que estas interfaces agora são independentes de estado, e podem ser reutilizadas livremente entre diferentes objetos.

Interfaces REST modernas geralmente são definidas de maneira a operar sobre o protocolo HTTP, e utilizam como corpo de sua mensagem objetos JSON para transferência de informações. Um dos benefícios mais mencionados para interfaces nesse estilo é a padronização do acesso aos objetos, permitindo que um servidor se comunique com diversos clientes

Tabela 2.1: Mapeamento entre operações CRUD e requisições HTTP

Operação CRUD	Requisição HTTP
Create	POST/PUT
Retrieve	GET
Update	PATCH
Delete	DELETE

independentemente da forma como foram implementados, desde que sigam a interface.

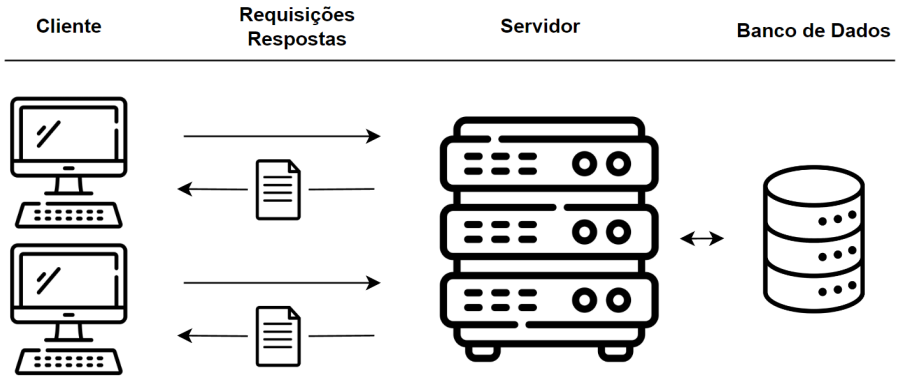
Um uso comum para REST APIs é a definição das operações de Create, Retrieve, Update, Delete (CRUD) utilizadas em praticamente qualquer sistema para gerenciamento dos objetos armazenados, as quais são facilmente traduzidas para operações do protocolo HTTP no qual a comunicação ocorre. A Tabela 2.1 demonstra este mapeamento.

2.2.2 Modelo Cliente-Servidor

O modelo Cliente-Servidor é um padrão de desenvolvimento de aplicações que separa a carga de trabalho da obtenção de um recurso em duas partes: a requisição por parte de um consumidor, denominado cliente, e a resposta por parte do dono do recurso, denominado servidor. A origem deste conceito é nebulosa, porém um dos primeiros registros desta separação foi escrito por Israel et al. (1978), onde o conceito foi aplicado para um sistema de arquivos distribuído. Este conceito se encaixa perfeitamente no modelo utilizado atualmente pela internet, onde máquinas de diversos usuários ao redor do mundo realizam requisições sobre diversos protocolos a servidores que os respondem com o recurso solicitado, conforme mostra a Figura 2.1.

Além de promover este equilíbrio da carga de trabalho este modelo

Figura 2.1: Arquitetura cliente-servidor



Fonte: Elaborado pelo autor

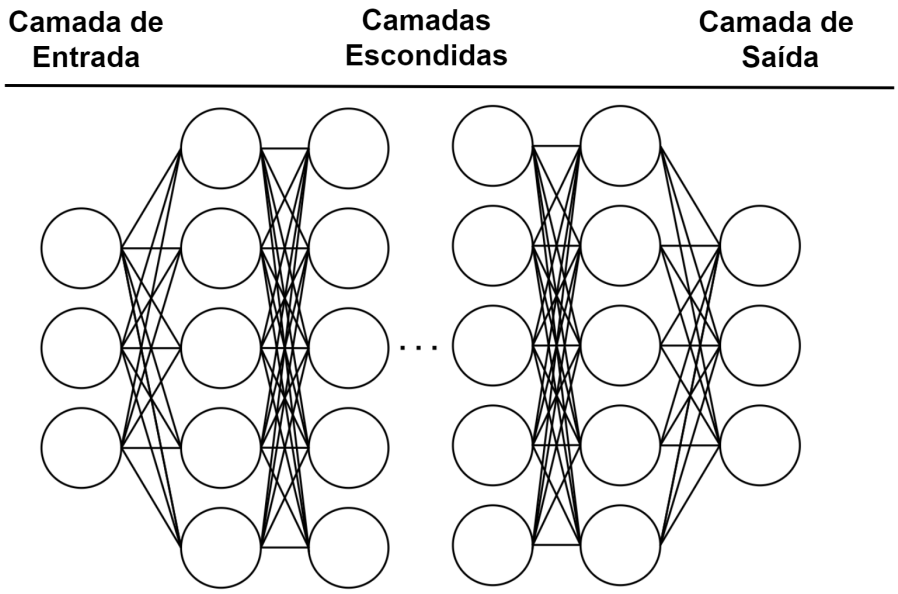
também permite, em conjunto com uma REST API, que um servidor se comunique com quaisquer clientes independentemente da implementação, promovendo uma abstração do recurso de maneira que ele possa ser apresentado a cada cliente da maneira mais adequada. Enquanto um cliente apresenta certos dados na forma de gráfico outro cliente pode apresentar na forma de mapa, e assim por diante, promovendo uma camada de abstração sobre dados transmitidos pela aplicação.

2.2.3 Redes Neurais

Redes Neurais(MCCULLOCH; PITTS, 1943) são um grupo de algoritmos de aprendizado de máquina que mimetiza os processos que ocorrem no cérebro humano. Estes algoritmos procuram simular um conjunto de neurônios individualmente simples, porém capazes de disparar sinapses, ou sinais elétricos, com intuito de se comunicar com outros neurônios. Estes sinais são responsáveis por condicionar os neurônios atingidos, alterando a forma com a qual os mesmos processam sinais subsequentes.

De forma a simular a interação, o algoritmo de Redes Neurais de Perceptrons opera sobre um modelo simplificado deste sistema, onde cada neurônio é representado por uma função denominada função de ativação,

Figura 2.2: Estrutura de uma Rede Neural de Perceptrons



Fonte: Elaborado pelo autor

que pode ser, por exemplo, uma regressão linear. Estes neurônios são então colocados em camadas, as quais propagam os sinais recebidos e processados a camadas subsequentes, gerando uma rede conforme mostra a Figura 2.2.

Cada neurônio neste processo recebe um conjunto de valores de entrada da camada anterior, além de um conjunto de pesos de mesma dimensão. Cada neurônio tem, também, um viés associado a ele. Ao receber estes valores, um nodo desta rede então aplica sua função de ativação considerando todos estes parâmetros, resultando em um valor de saída que é transmitido à camada seguinte. O aprendizado ocorre quando o resultado previsto é comparado com o resultado esperado para aquela entrada e os pesos de cada nodo são atualizados.

2.2.4 *Pathfinding*

A busca de caminhos é um dos problemas clássicos da computação, com uma miríade de algoritmos capazes de, dado um grafo, encontrar o caminho de menor custo quando ele existe. Neste trabalho utilizamos o algoritmo A∗, desenvolvido como parte de um projeto do instituto de pesquisa

de Stanford(SHAKEY, 1972). O pseudo-algoritmo A∗ é apresentado na listagem 2.3, e é adequado para aplicações em tempo real por utilizar uma heurística na determinação da pontuação de cada nodo sem executar a busca.

Figura 2.3: Pseudo-Algoritmo A*

```
openSet := {start}, cameFrom := an empty map
gScore := map with default value of Infinity, gScore[start] := 0
fScore := map with default value of Infinity, fScore[start] := h(start)

while openSet is not empty
  current := the node in openSet having the lowest fScore[] value
  if current = goal
    return reconstruct_path(cameFrom, current)

openSet.Remove(current)
for each neighbor of current
  tentative_gScore := gScore[current] + d(current, neighbor)
  if tentative_gScore < gScore[neighbor]
    cameFrom[neighbor] := current
    gScore[neighbor] := tentative_gScore
    fScore[neighbor] := tentative_gScore + h(neighbor)
    if neighbor not in openSet
      openSet.add(neighbor)
```

Fonte: https://en.wikipedia.org/wiki/A*_search_algorithm

Este algoritmo não garante que o caminho de menor custo seŕa sempre encontrado, a menos que a heurística utilizada seja considerada admissível, isto é, ela nunca superestima o custo de cada ponto até o destino. Como a implementação deste trabalho opera sobre um grafo planar de duas dimensões, sem custos negativos, podemos garantir o caminho de menor custo através de uma heurística de distância em linha reta, visto que é impossível um custo menor que este.

2.3 Tecnologias

Aqui apresentamos um compilado de todas as tecnologias que foram utilizadas no desenvolvimento deste trabalho, separando-as entre as três principais áreas que a aplicação abrange.

2.3.1 Modelagem, limpeza e carregamento de dados

A limpeza e a preparação para o carregamento dos dados ocorre através de um *script* Python(FOUNDATION, 2021), a linguagem de programação mais popular no tempo de escrita(TIOBE, 2022). A escolha desta linguagem se deve à simplicidade de escrita do código, dado que o mesmo é utilizado apenas para o carregamento inicial da base de dados. Os dados são carregados em massa através da `API` descrita na Seção 2.3.2, e posteriormente armazenados na base de dados.

Para armazenamento dos dados, foi escolhido o Sistema Gerenciador de Banco de Dados MongoDB(MONGODB, 2021), um banco de dados não-relacional orientado a documentos JSON(RFC-8259, 2017). Além de proporcionar vantagens de *performance* sobre um banco relacional para armazenamento de dados no estilo documento(MARTINS et al., 2019)(WANG; YANG, 2017), com estrutura variável, o MongoDB oferece funcionalidades para pesquisas geográficas, isto é, apresenta otimizações para consultas, por exemplo, restritas a uma área bem definida, o que é altamente usado para este trabalho.

2.3.2 Desenvolvimento de aplicação base e *website*

A aplicação segue um modelo cliente-servidor, conforme apresentado na Seção 2.2, utilizando a linguagem Java para codificação do servidor. Java é uma linguagem de programação orientada a objeto e de alto nível, que executa código sobre uma máquina virtual específica para cada sistema operacional. A linguagem é a terceira mais utilizada no ano de 2022, além de ser uma das linguagens mais utilizadas na programação de servidores(TIOBE, 2022).

Dentro desta linguagem, existem diversos *frameworks* para desenvolvimento de aplicações cliente-servidor, porém foi escolhido o Spring Boot(SPRING.IO, 2022) pela sua simplicidade de implementação e customização.

Usufruindo da flexibilidade destas tecnologias, temos duas interfaces da aplicação com o mundo externo: uma interface *web*, na qual o mapa é apresentado e o usuário final pode interagir com as funcionalidades da

aplicação, e uma interface via API disponibilizando operações CRUD ao conjunto de dados de maneira a facilitar a integração desta ferramenta com quaisquer outros sistemas. Esta API oferece os dados no formato JSON.

Para a interface *web*, utilizamos HTML e JavaScript(PLURALSIGHT, 2022), uma linguagem de programação leve suportada na maioria dos navegadores modernos. Para apresentar o mapa através do qual grande parte da interação ocorre, é utilizado o *framework* OpenLayers(OPENLAYERS, 2022), o qual é de código aberto e livre de custo.

2.3.3 Definição e treinamento do modelo preditivo

O modelo preditivo desenvolvido para o trabalho usufruiu das funcionalidades oferecidas pelas bibliotecas *python tensorflow*(TENSORFLOW, 2022) e *keras*(KERAS, 2022), as quais providenciam uma API amigável para a construção e treinamento de modelos de aprendizado de máquina. O modelo foi construído e treinado nesta linguagem, e posteriormente salvo no banco de dados do servidor em formato binário.

A avaliação do modelo pela aplicação em tempo real ocorre com o auxílio da biblioteca de aprendizado de máquina em Java deepLearning4J(DL4J, 2022), a qual é capaz de reconstruir o modelo salvo no banco de dados na forma de um grafo, executando as previsões através do *runtime tensorflow* similar ao utilizado durante o treinamento.

3 TRABALHOS RELACIONADOS

Neste Capítulo, são apresentados os principais trabalhos cujo objetivo também é atacar e analisar acidentalidade no trânsito brasileiro, além de alguns trabalhos e aplicações relacionadas à sugestão de caminhos mediante condições de trânsito.

3.1 Descrição dos Trabalhos

A análise do trânsito é um tópico de interesse não só no Brasil, mas também em cidades de todo o mundo, dado que a mobilidade é um dos principais problemas urbanos atualmente. Aqui trazemos exemplos de trabalhos realizados em cidades, estados e/ou países sobre acidentalidade no trânsito, os quais procuram iluminar suas principais causas ou evidenciar suas áreas de maior ocorrência. Estes trabalhos foram escolhidos devido a sua similaridade com o tópico em questão, acidentalidade, por diferentes razões. O primeiro trabalho trás uma perspectiva das causas principais para os acidentes, o que é valioso na escolha de parâmetros para treino de nossa rede neural. O segundo trabalho explora mais a fundo as técnicas utilizadas para construção da mesma rede, abordando tópicos como a representação dos dados e a arquitetura escolhida.

3.1.1 Analysis of Car Accidents Causes in the USA

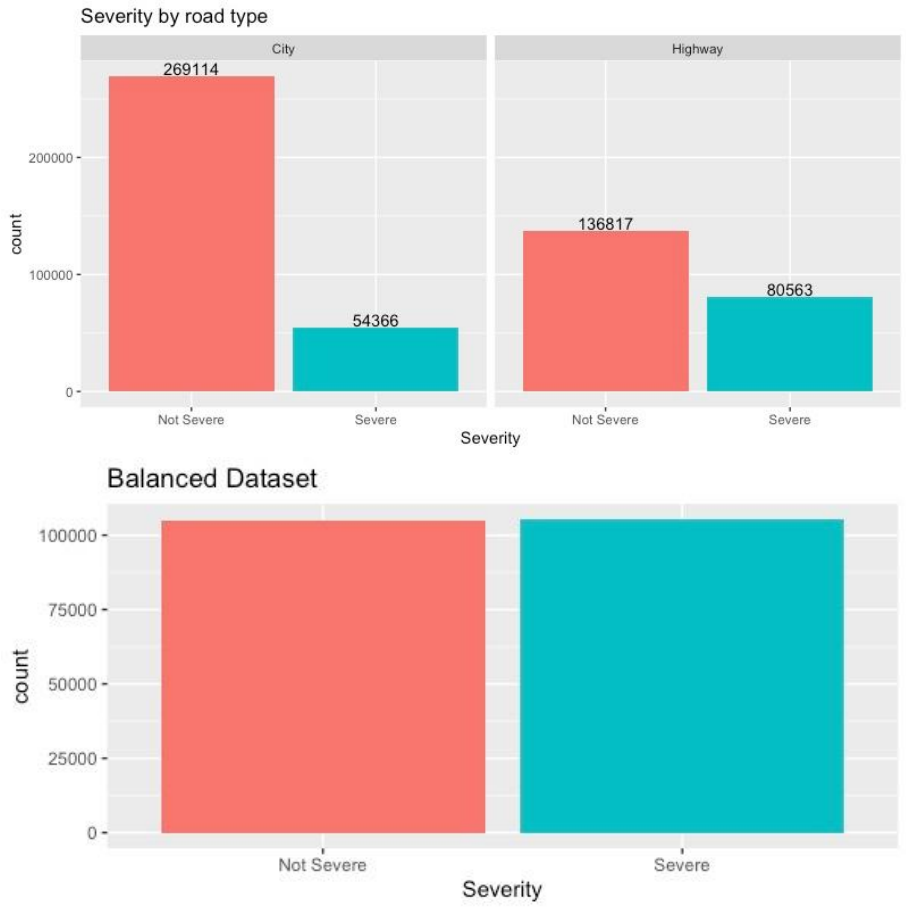
Como tese de mestrado para o *Rochester's Institute of Technology*, Aljaban (2021) dissertou sobre os acidentes rodoviários registrados nos Estados Unidos no período de 2016 a 2020. As principais perguntas feitas pelo autor são:

- O número de acidentes de trânsito está aumentando?
- Onde e quando acidentes de trânsito mais acontecem?
- Quais as possíveis soluções para atacar acidentes de trânsito?

Para responder os questionamentos propostos, o autor opta pela

utilização de técnicas de aprendizado de máquina para derivar relações entre os dados, comparando os resultados obtidos quando utilizando os algoritmos de floresta aleatória(HO, 1995) e Naive Bayes. Além de algumas visualizações, o autor apresenta as transformações que foram executadas sobre os dados de modo a possibilitar a aplicação dos algoritmos, entre elas a divisão dos dados por severidade. O número de acidentes não severos coletados supera em 4 vezes o número de acidentes severos, e portanto o autor revela que realizou um balanceamento dos dados procurando equalizar o conjunto de treinamento e evitar um viés dos resultados. A Figura 3.1 apresenta esta transformação dos dados, com a distribuição antes e depois.

Figura 3.1: Balanceamento do conjunto de dados de treino



Fonte: (ALJABAN, 2021)

O autor conclui a tese explicitando que a *rush hour* é a principal lacuna de ocorrência, seguido pela densidade populacional das cidades onde os mesmos ocorrem. Como possíveis soluções, o trabalho remoto é apresentado como uma alternativa para evitar o tráfego durante este

período, também sugerindo o encorajamento da adoção de veículos auto-pilotáveis, prometendo a redução do fator do erro humano nesta fórmula.

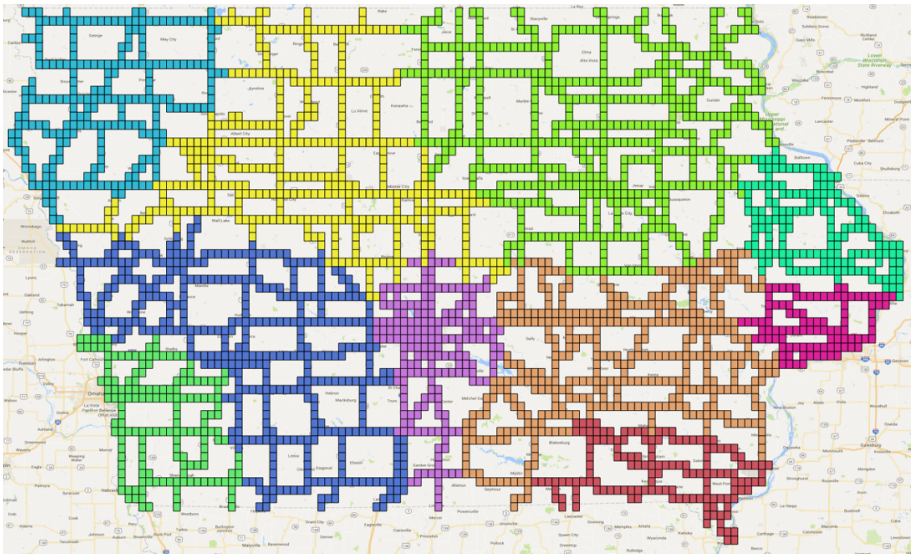
3.1.2 Hetero-ConvLSTM: A Deep Learning Approach to Traffic Accident Prediction on Heterogeneous Spatio-Temporal Data

Interessados na distribuição de ocorrências de acidentes de trânsito em Iowa, Estados Unidos, Yuan et al. (2018) realizaram um estudo sobre a efetividade de utilizar uma rede neural ConvLSTM(SHI et al., 2015) para tentar estimar locais de maior probabilidade de sinistros futuros. Para tal, são utilizados registros de acidentes que ocorrem do ano de 2006 até 2013, além de registros meteorológicos sobre precipitação e até mesmo temperatura.

Os autores exploram a ideia de que as ocorrências estão fortemente ligadas às condições do local e hora em que ocorrem, com ênfase na importância de considerar a heterogeneidade do terreno. Esta suposição parte do princípio de que ruas de cidades, por exemplo, não possuem as mesmas condições de visibilidade e manutenção que ruas de zonas rurais, o que afeta vigorosamente a probabilidade de um acidente. De modo a discretizar este domínio geográfico altamente heterogêneo, os autores tiraram proveito de um algoritmo de agrupamento, ou *clustering*, dividindo a área coberta em um conjunto finito de zonas, conforme a Figura 3.2.

O modelo foi testado durante um período de uma semana, no qual acidentes foram previstos e comparados com os resultados reais. Foram obtidos resultados extremamente positivos, com a ferramenta sendo capaz de prever *clusters* muito próximos aos reais, como apresentado pela Figura 3.3, demonstrando que a utilização de técnicas de inteligência artificial para previsão de acidentes pode ser adequada.

Figura 3.2: Divisão do estado de Iowa em 10 *clusters*



Fonte: (YUAN et al., 2018)

3.2 Aplicações

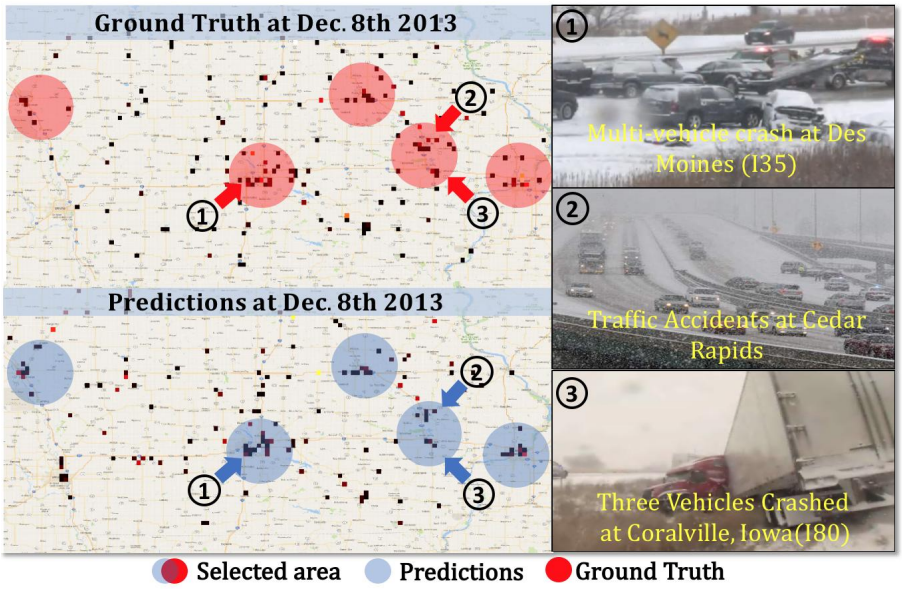
O estudo destas causas é muito importante para que tenhamos foco na proposição e debate de possíveis soluções, porém tão quão importante é o desenvolvimento de tais ferramentas e sua disponibilização ao público. Nesta Seção, temos dois exemplos de aplicações cujo domínio e objetivo é similar ao da ideia proposta pelo autor.

3.2.1 CrashMap UK

O CrashMap UK(CRASHMAPUK, 2022) é uma ferramenta *web* desenvolvida no Reino Unido que utiliza dados de colisão disponibilizados publicamente pelo Departamento de Transporte. A aplicação disponibiliza gratuitamente um mapa *online* no qual acidentes reportados dentro dos últimos 10 anos são apresentados ao usuário geograficamente. A ferramenta realiza a classificação dos acidentes em leves, sérios e fatais, além de permitir a filtragem por tipos de veículos e sinistros.

A ferramenta também permite que, através de um sistema de *subscription*, seus usuários tenham acesso a relatórios com mais detalhes e dados estatísticos sobre os acidentes registrados, tanto individualmente

Figura 3.3: Acidentes previstos e ocorridos durante um período de 7 dias



Fonte: (YUAN et al., 2018)

quanto a granel.

3.2.2 Waze

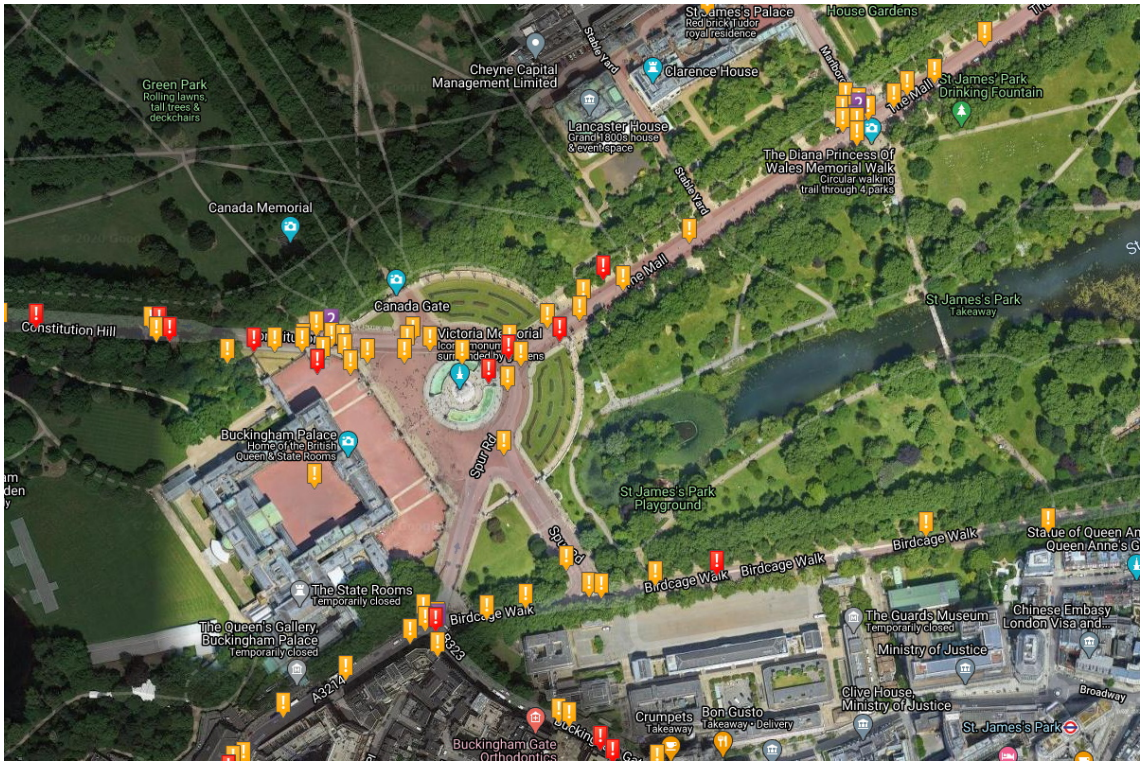
O Waze(WAZE, 2022) é uma aplicação muito popular, utilizada pelo mundo inteiro capaz de fornecer atualizações em tempo real sobre o estado das ruas. Apesar de não tratar especificamente de acidentes de trânsito, o aplicativo é relevante para este trabalho, pois também oferece *feedback* sobre a segurança de logradouros de maneira visual. A ferramenta é única por utilizar os próprios usuários como fonte de dados, recebendo os dados através de relatos de motoristas.

Não só isso, o Waze também é relevante por implementar uma funcionalidade que é explorada neste trabalho, a sugestão de rotas.

3.3 Comparações

Antes de dar início ao lado técnico do trabalho, apresentamos uma análise comparativa da solução proposta com os trabalhos relacionados

Figura 3.4: Visualização dos acidentes no CrashMap UK



Fonte: (CRASHMAPUK, 2022)

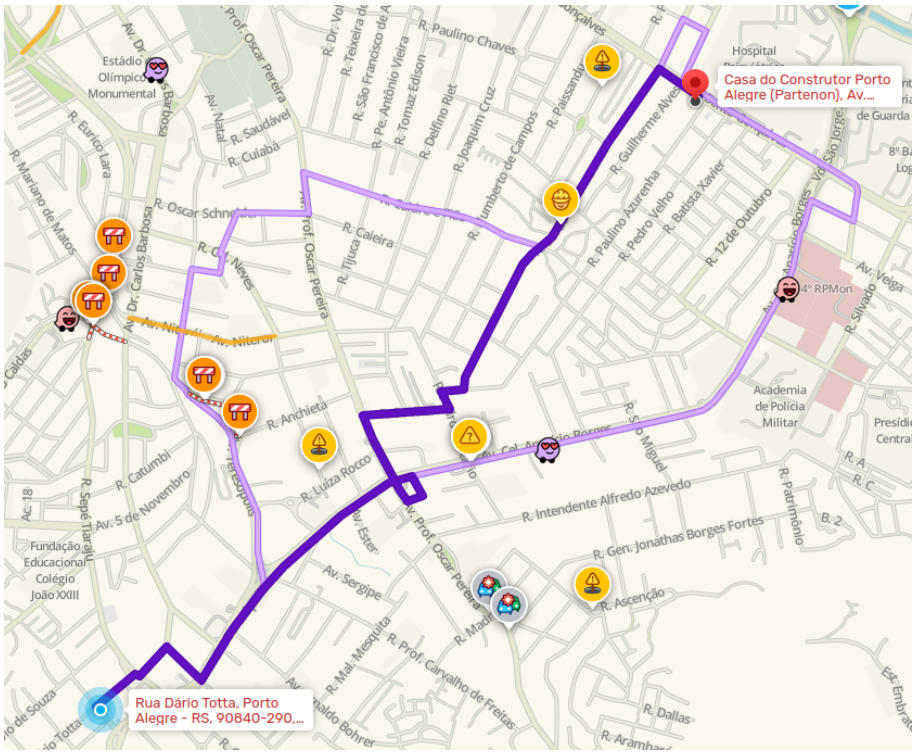
vistos nesta Seção. De maneira a manter a competição "justa", dividimos o nosso trabalho em duas partes, sendo elas a fundamentação teórica e pesquisa e a ferramente em si, realizando comparações com os trabalhos e aplicações separadamente através de duas Tabelas.

3.3.1 Comparação dos Trabalhos Acadêmicos

Os trabalhos relacionados apresentados foram escolhidos de forma a explicitar os diferentes pontos que podem ser explorados em um trabalho desta área. Enquanto Aljaban (2021) procura atacar a fonte do problema, procurando por correlações entre os dados através de algoritmos clássicos de modo a encontrar os principais fatores contribuintes em registros históricos, Yuan et al. (2018) olham para o futuro, procurando prever a ocorrência de novos acidentes através de algoritmos mais modernos, como ConvLSTM.

Ambos os trabalhos tem um foco acadêmico, analisando e processando os dados, apresentando um compilado das informações obtidas. Nosso

Figura 3.5: Visualização de rota no Waze



Rota sugerida pelo aplicativo Waze, apontando acidentes e bloqueios que desfavorecem outras rotas

Fonte: (WAZE, 2022)

diferencial se encontra na implementação da ferramenta preditiva, implementada com base nos resultados destes e outros estudos. A Tabela 3.1 apresenta os pontos que este trabalho traz em contraste com os trabalhos relacionados apresentados.

Tabela 3.1: Comparação entre os trabalhos

Foco	**Aljaban (2021)**	**Yuan et al. (2018)**	**Este Trabalho**
Identificação das principais causas	X		
Proposição de possíveis soluções	X	X	
Predição de novas ocorrências		X	X
Viabilidade de uma ferramenta acessível ao usuário			X

3.3.2 Comparação das Aplicações

Comparada às duas aplicações apresentadas, a ferramenta proposta procura encontrar um "melhor dos dois mundos", tendo tanto as funcionalidades de geo-visualização dos acidentes quanto a sugestão de

rotas. Mantemos a informação individual de cada acidente registrado, além das informações climáticas do momento da ocorrência, e esta informação é disponibilizada ao usuário, suportando consultas específicas por acidentes através das suas características. Adicionalmente, a ferramenta procura integrar ideias obtidas dos trabalhos acadêmicos relacionados às suas funcionalidades, na forma de previsão das ocorrências de acidentes futuros.

Em suma, este trabalho procura ser não apenas uma aplicação para o usuário final comum, mas também uma base de conhecimento e desenvolvimento para outros estudos acadêmicos que venham a ser desenvolvidos sobre o tópico de acidentes de trânsito. O foco na previsão de acidentes e sugestão de caminhos também enfatiza este duplo propósito, uma vez que experimenta com os fatores geo-climáticos relevantes para cada uma dessas funcionalidades. A Tabela 3.2 apresenta a comparação entre as aplicações.

Tabela 3.2: Comparação entre as aplicações

Funcionalidade	**Crash Map UK**	**Waze**	**Este Trabalho**
Estatísticas	X	X	X
Dados detalhados sobre acidentes individuais	X		X
Visualização no Mapa	X	X	X
Sugestão de rotas		X	X
Predição de ocorrências			X

4 DESENVOLVIMENTO E IMPLEMENTAÇÃO

Este Capítulo detalha o planejamento e implementação de uma aplicação interativa para visualização e consulta de acidentes de trânsito no município de Porto Alegre, além da sugestão de caminhos entre dois pontos da cidade e a previsão das regiões com maior probabilidade de ocorrências de acidentes para o dia. Esta aplicação pode ser utilizada por órgãos governamentais em conjunto com as bases de dados disponíveis de maneira a realizar estudos sobre a cidade, além de proporcionar um conjunto de funcionalidades que podem ser oferecidas através de um *website* ao público, formando um modelo similar a *B2B* (*Business to Business*). Enquanto que as funcionalidades de consulta de acidentes e as *API* são primariamente utilizadas para realizar os estudos sobre os dados, as funcionalidades de previsão e sugestão de caminhos podem ser utilizadas por estes usuários finais.

Aqui apresentamos a forma como a aplicação foi desenvolvida, incluindo o cronograma planejado, os passos executados em cada etapa e os diagramas de interação e arquitetura da aplicação. Quando falando da implementação, focamos nas três funcionalidades principais da aplicação apresentando os desafios e suas resoluções para cada uma.

O código fonte para a aplicação e todos os *scripts* mencionados neste Capítulo estão disponíveis no repositório deste projeto[1].

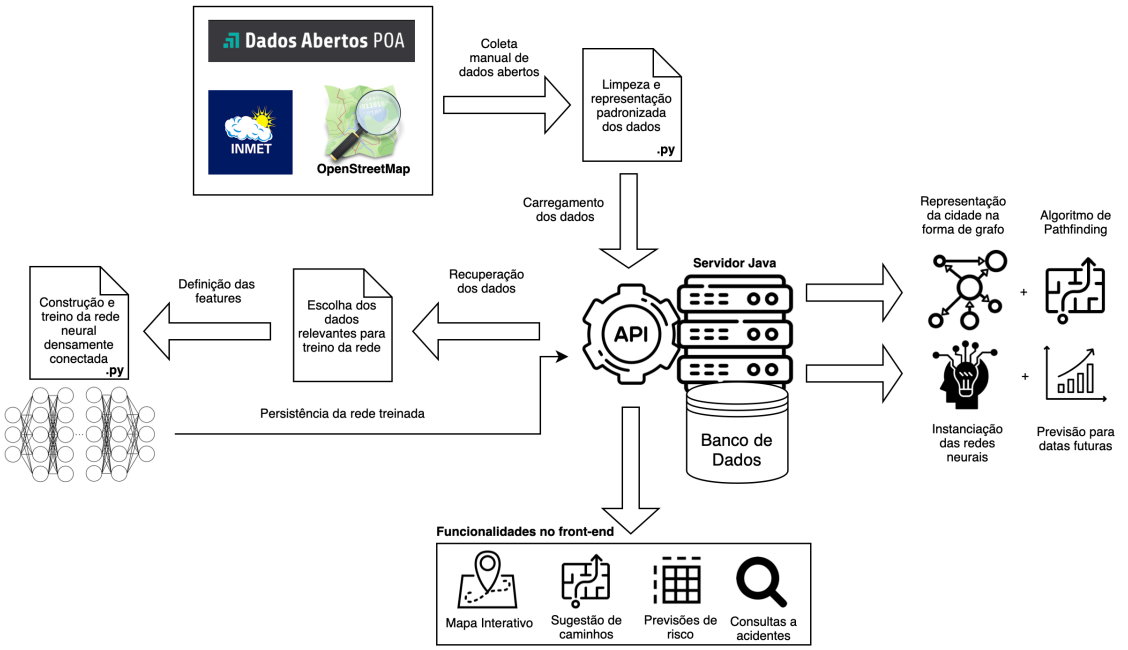
4.1 Visão Geral

Antes de dar início ao detalhamento das funcionalidades da aplicação apresentamos uma visão geral do fluxo de desenvolvimento. Na figura 4.1 está apresentada a visão alto-nível da aplicação como um todo, com as suas funcionalidades principais e o fluxo de atividades.

Inicialmente, os dados foram coletados manualmente das fontes de dados abertos listadas no Capítulo 2 na forma de tabelas *excel* e dados em memória (para o caso dos dados geográficos). Estes dados foram limpos e convertidos para uma representação padronizada para carregamento através

[1] Disponível em <https://github.com/FabioAzevedoGomes/TCC>

Figura 4.1: Visão alto-nível do trabalho desenvolvido



Fonte: Elaborado pelo autor

da API, sendo armazenados no banco de dados do servidor. Com os registros armazenados é iniciado o desenvolvimento das três funcionalidades principais.

Consultas são implementadas para permitir a extração destes dados de maneira padronizada entre objetos, utilizando a abstração de filtros que limitam a busca por acidentes, ruas, interseções e registros climáticos àquelas instâncias relevantes para a requisição.

Os dados armazenados são extraídos novamente através da API, sendo utilizados por um código *python* responsável pela extração das *features* relevantes para previsão de risco de acidentes, além da construção e treinamento de uma rede neural de perceptrons densamente conectada responsável por essa previsão. Esta rede é salva no formato binário *h5*, formato padrão que permite a persistência da arquitetura da rede junto dos seus pesos pós-treinamento, concatenada à lista de regiões utilizadas para o treino e finalmente enviada através de uma API disponibilizada pelo servidor para ser persistida. A rede salva no banco de dados em formato *h5* pode, então, ser reconstruída pelo código *Java*, o qual pode calcular *features* climáticas de uma data específica e realizar as previsões.

Uma camada de abstração é construída sobre os dados geográficos de maneira a permitir o seu acesso através de uma interface de grafo, a qual é utilizada para o desenvolvimento de um algoritmo de busca de caminhos. Esta abstração também inclui uma associação a uma versão da rede neural salva no banco de dados, integrando a funcionalidade de previsão com a sugestão de caminhos.

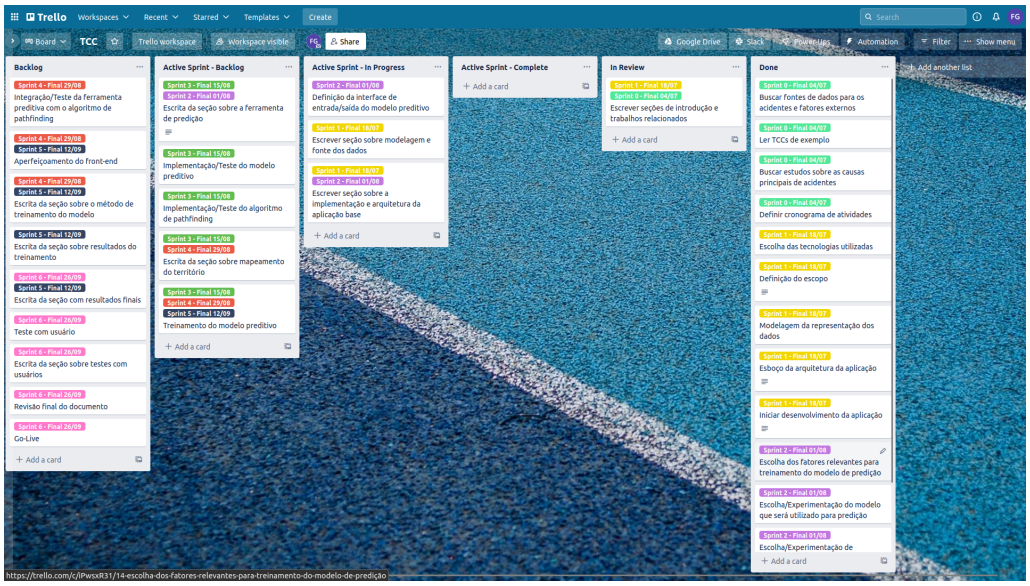
A consulta por acidentes, sugestão de caminhos e previsão de risco podem então ser expostas ao usuário final através de uma página *web* que apresenta todas as informações mantidas de maneira concisa, simples e intuitiva. A API permanece disponível, permitindo tanto interação pelo usuário final através do *front-end* quanto por outros programas e ferramentas que venham a ser desenvolvidas ou integradas.

4.2 Planejamento e Acompanhamento

De maneira a organizar o processo de desenvolvimento do projeto, foi necessária a adoção de metodologias que suportassem as eventuais mudanças de escopo que poderiam surgir ao decorrer do tempo. Para tal, o autor tirou inspiração de técnicas e metodologias ágeis(BECK et al., 2001) de desenvolvimento de *software*, adaptando as mesmas aos *timeframes* da jornada de desenvolvimento.

Como planejamento inicial, o projeto foi dividido em suas atividades principais, descritas através de uma frase ou pequeno parágrafo que capturasse a ideia proposta. Dada esta decomposição, foi feita uma ordenação com base nos requisitos para que cada desenvolvimento pudesse iniciar, separando atividades com dependência e mais ou menos prioritárias. Por fim, estas atividades foram colocadas em um *task board* similar a um *kanban*(OHNO; BODEK, 1998), e divididas em *sprints* com duração de duas semanas - o mesmo período entre cada reunião com a orientadora. A Figura 4.2 apresenta este quadro durante a escrita deste Capítulo.

Figura 4.2: *Task Board* utilizado no desenvolvimento



Fonte: Elaborado pelo autor

4.3 Arquitetura da aplicação

A aplicação segue o modelo cliente-servidor, conforme apresentado na Seção 2, tirando proveito da flexibilidade oriunda da abstração entre o código referente à apresentação e interação com o usuário final e o código responsável pela gerência e processamento dos dados. Para tal, a aplicação foi dividida em dois componentes principais: o *frontend* e o *backend*, as quais se comunicam através de múltiplos *endpoints* REST utilizando uma representação comum dos dados. Esta divisão também facilita a comunicação do servidor *backend* com outros agentes que conheçam esta representação, o que é útil para, entre outras ações, a manutenção e atualização dos dados presentes na base de dados e o versionamento do modelo preditivo.

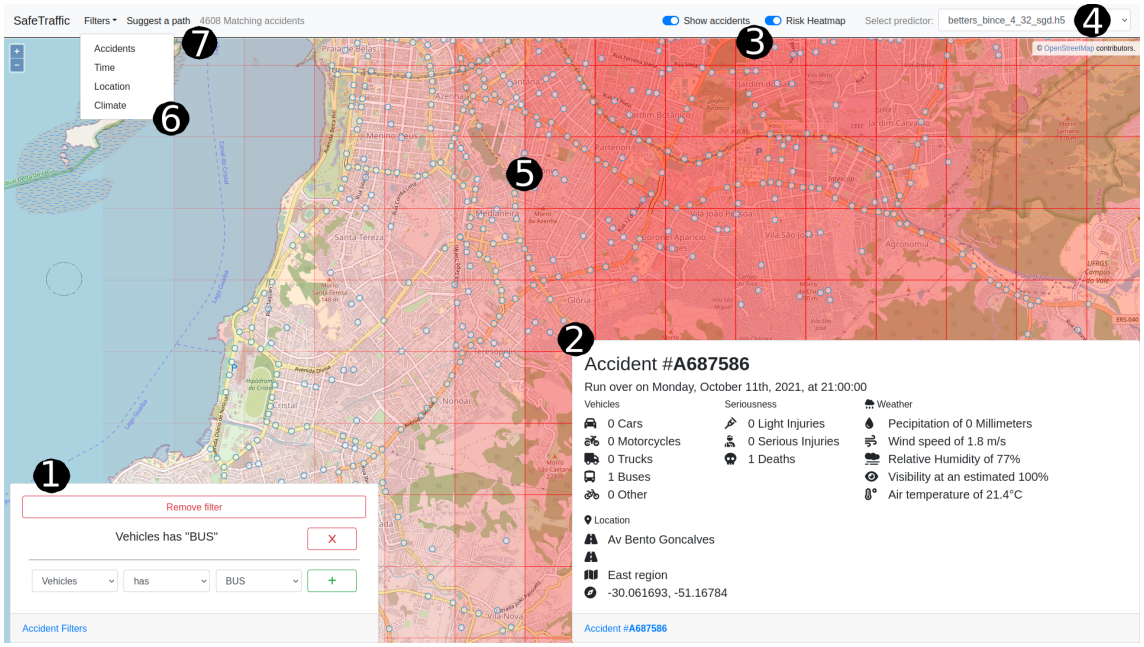
4.3.1 Frontend

Para a interação com o usuário, o autor optou por uma aplicação de "uma tela", na qual todas as funcionalidades seriam disponibilizadas sem a necessidade de troca de contexto. A Figura 4.3 apresenta esta tela, com a visualização de acidentes e mapa de risco ativadas, além de abas de filtro

sobre caraterísticas do acidente e tempo quando ocorreu.

Figura 4.3: *Frontend* da aplicação, contendo:

1. Filtros de acidente ativos, procurando apenas acidentes com ônibus
2. Informações de um acidente específico sendo visualizado
3. Controles de visualização dos acidentes e mapa de risco
4. Seleção da rede neural sendo usada na previsão
5. Mapa interativo, pontos azuis são acidentes e as regiões vermelhas representam o risco previsto.
6. Acesso aos filtros disponíveis para o usuário
7. Botão para iniciar a sugestão de caminhos



Fonte: Elaborado pelo autor

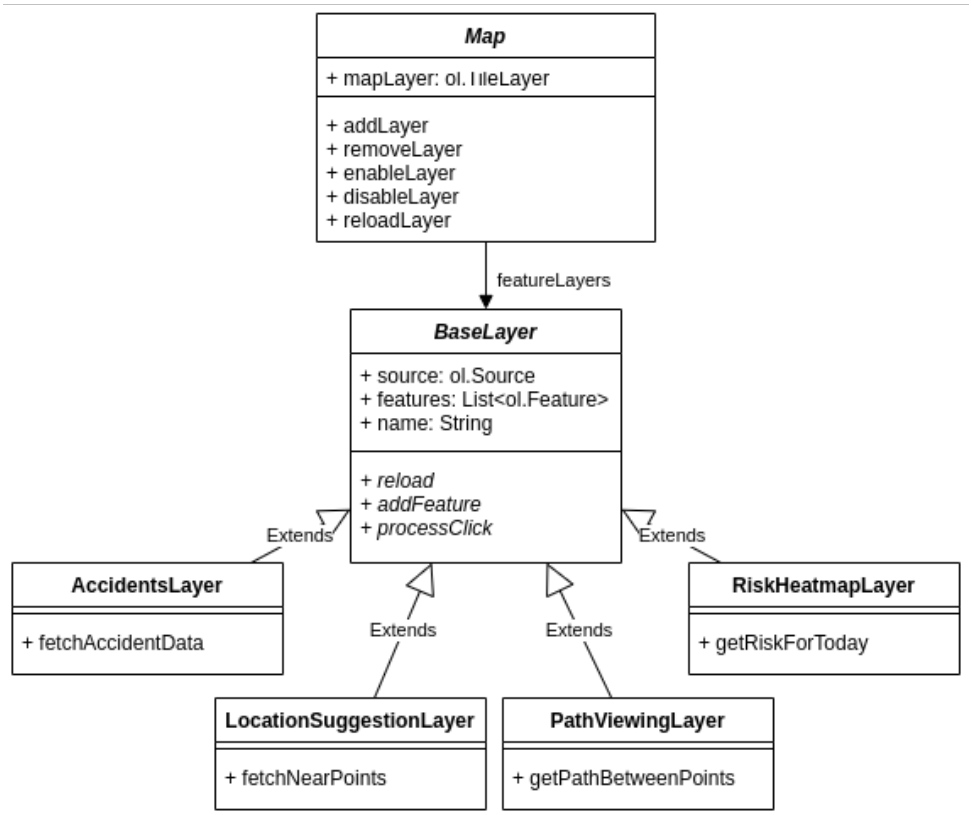
Dado que todas as funcionalidades estão nessa tela, foi necessária a implementação de medidas que mitigassem a desordem e "claustrofobia"da aplicação, e portanto foi definido que o usuário terá a possibilidade de expandir e colapsar todas as telas que forem abertas. Através dessa funcionalidade em uma única tela, é possível apresentar uma quantidade arbitrária de informações, a visibilidade das quais é escolhida pelo usuário.

Como a página foi desenvolvida utilizando apenas `javascript` e `HTML` puros, sem o uso de *frameworks*, não há uma arquitetura elaborada para este código. Existem, no entanto, duas funcionalidades mais complexas que exigem uma organização do código para que sejam suportadas: o mapa com suas camadas e os filtros. Ambos objetos são manipuláveis livremente, com o

usuário podendo adicionar e remover filtros, ou ativar e desativar camadas do mapa.

Foi criada uma classe para gerenciar o objeto do mapa, além de uma classe base abstrata para representar camadas, a qual foi estendida com implementações específicas para cada um dos dados apresentados. A Figura 4.4 apresenta esta configuração do mapa na forma de um diagrama UML.

Figura 4.4: Arquitetura do mapa com suas 4 camadas.



Fonte: Elaborado pelo autor

Cada subclasse de camada tem suas responsabilidades específicas, neste caso a construção de *requests* e comunicação com o *backend*, além da interpretação da resposta recebida e atualização da sua lista de *features*. Cada *feature* aqui representa uma informação a ser apresentada no mapa, como um ponto referente a um acidente, um *tile* do mapa de risco ou um trecho de um caminho sugerido.

Outra responsabilidade destas especificações de camada é o processamento do clique do usuário. Cada camada reage de maneira diferente aos cliques, como, por exemplo, a camada de sugestão de caminhos

salvando as posições de início e fim, ou a camada de acidentes apresentando os dados do acidente selecionado.

A classe mapa, por sua vez, tem a responsabilidade de gerenciar estas camadas, com funcionalidades de adicionar, remover, ativar e desativar as camadas. É através desta classe que a interação do usuário com as camadas acontece.

Para viabilizar a flexibilidade na seleção de filtros, foi utilizada a mesma abstração das camadas, com uma classe de filtro base implementando funcionalidades comuns como adicionar e remover campos e classes específicas para o controle de valores e operações disponíveis para cada campo. Um filtro qualquer foi definido como a combinação de 3 valores: o campo sendo consultado; a operação sobre esse campo; o valor sendo considerado. Desta forma, foram identificados 4 grupos coesos de características sobre a ocorrência de acidentes, e os filtros foram construídos com base nos mesmos:

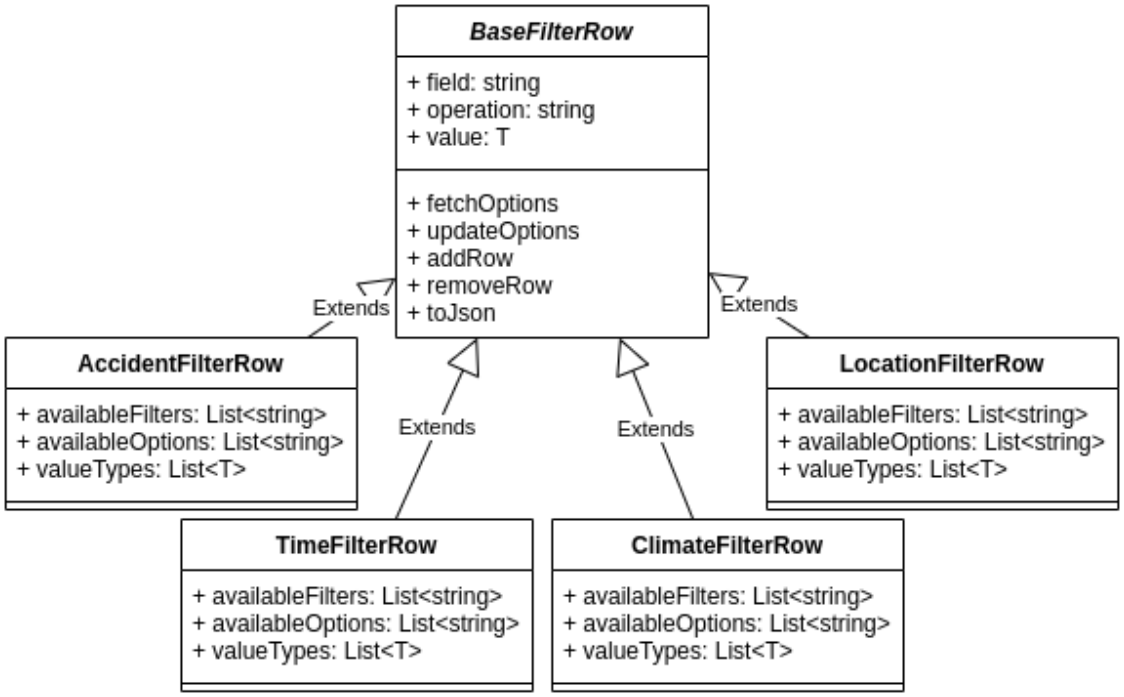
- Filtros de acidentes
- Filtros de clima
- Filtros de localização
- Filtros de data/hora

A Figura 4.5 demonstra, na forma de um diagrama UML, a organização das diferentes classes de filtro, e a Tabela 4.1 explicita os campos, operações e valores disponíveis para cada grupo.

Na Tabela, todos os filtros cujo tipo referencia *Enum* tem sua seleção limitada a um domínio de valores, os quais são recuperados do *backend* pelas suas respectivas classes de filtro. Estas classes também mantém um mapa com as operações que são permitidas para cada campo. Como os domínios de valores são tirados do banco de dados diretamente, esta seleção automaticamente é atualizada caso novos acidentes sejam registrados com valores diferentes.

Quando estes filtros são enviados ao *backend*, as condições são unidas através da operação lógica *and*.

Figura 4.5: Arquitetura dos filtros



Fonte: Elaborado pelo autor

4.3.2 Backend

A aplicação foi projetado de acordo com o modelo arquitetural *MVC*(MVC..., 1970), o qual propõe uma separação entre o modelo, o qual representa o dado manipulado, a visão, a qual representa a forma como este modelo é apresentado, e o controlador, o qual media a interação entre as duas partes. O diagrama 4.6 apresenta a visão alto-nível da arquitetura, com as 4 camadas principais onde as classes estão distribuídas.

A camada de *endpoints* é o ponto de comunicação com o mundo externo, responsável por receber requisições por páginas e/ou dados que serão consumidos pelos clientes. Aqui estão presentes as `APIs REST`, além do controlador responsável por servir a página web visível pelo usuário.

A camada de *facades* é responsável pela conversão e validação dos dados fornecidos ou solicitados entre "Model"e "Data". Esta distinção é útil, pois a forma como os dados são estruturados na base de dados (Model) é otimizada para o processamento pelos algoritmos de busca de caminhos e previsão de acidentes, não sendo a mais adequada para visualização pelo

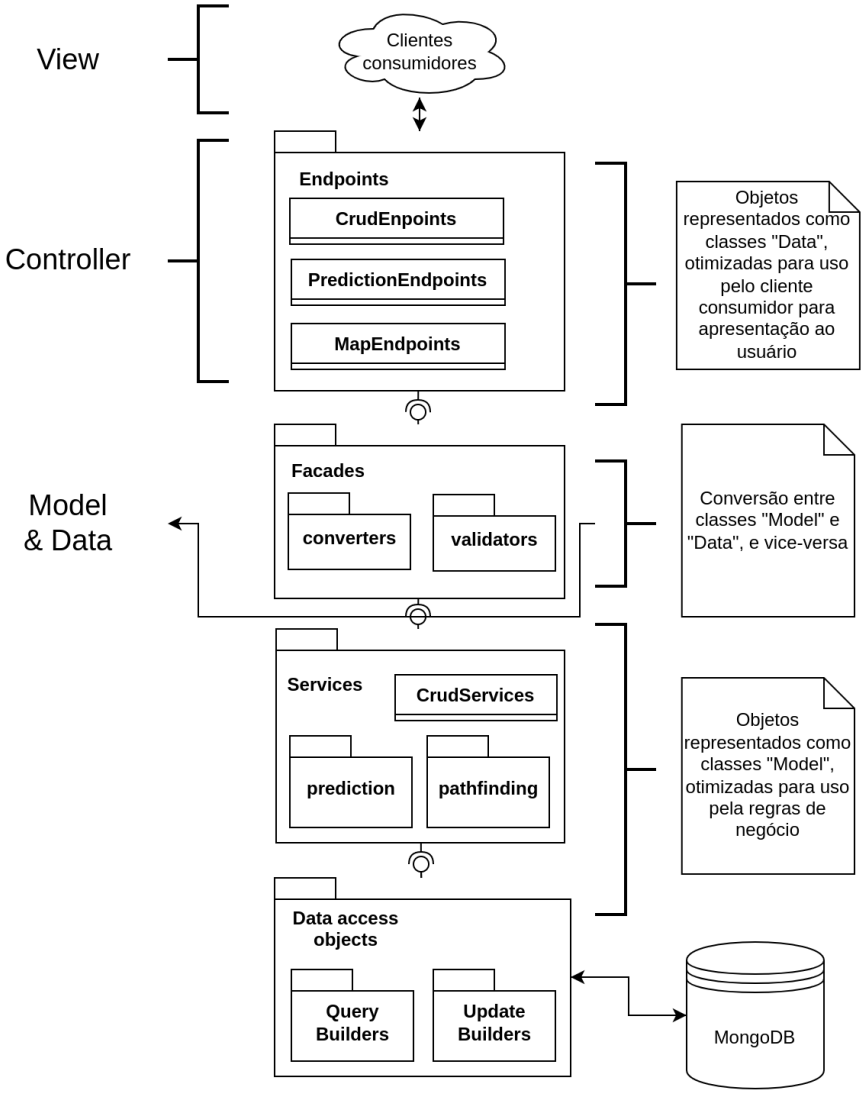
Tabela 4.1: Opções disponíveis para cada filtro

Classe de filtro	Filtros disponíveis		
Acidente	Campo	Tipo	Operações
	ID	String	=, !=
	Tipo	TypeEnum	=, !=
	Veículos	VehicleEnum	$\in$, $\ni$
	Feridos Leves	int	=, <=, >=
	Feridos Sérios	int	=, <=, >=
	Mortes	int	=, <=, >=
Clima	Campo	Tipo	Operações
	Visibilidade	float	=,<=, >=
	Umidade Relativa	float	=,<=, >=
	Precipitação	float	=,<=, >=
	Velocidade do vento	float	=,<=, >=
	Temperatura do ar	float	=,<=, >=
Data/Hora	Campo	Tipo	Operações
	Data	date	IS, BEFORE, AFTER
	Hora	time	IS, BEFORE, AFTER
Localização	Campo	Tipo	Operações
	Rua	StreetEnum	=, !=
	Latitude	float	=, <=, >=
	Longitude	float	=, <=, >=
	Região	RegionEnum	=, !=

usuário (Data). Além disso, nesta camada, certas funcionalidades mais alto-nível são abstraídas na forma de chamadas à camada inferior.

Com os dados validados e traduzidos para uma representação otimizada, a camada de *Service* pode efetuar as transações de negócio que

Figura 4.6: Visão alto-nível da arquitetura da aplicação



Fonte: Elaborado pelo autor

implementam as funcionalidades individuais da aplicação, neste caso, a previsão de acidentes ou traçamento de caminhos. Estas funcionalidades são separadas em módulos, os quais são, assim como os outros componentes, abstraídos por interfaces facilitando a troca de implementações durante o desenvolvimento.

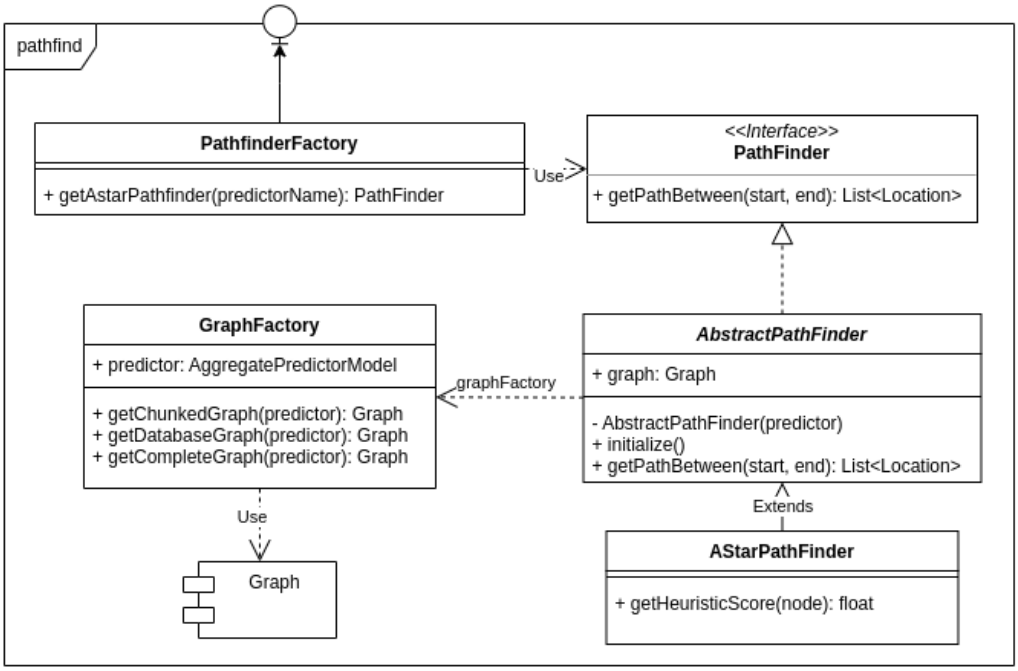
Por fim, a camada de *Data access objects* (*DAO*) é responsável pela construção e execução de *queries* específicas para a base de dados sendo utilizada, neste caso, `MongoDB`. Desta forma, abstraímos a atualização e obtenção de dados da camada de serviço, novamente permitindo a alteração da tecnologia de bancos de dados sem necessidade de alteração da regra de

negócio que opera sobre ela. Para nossa implementação, a separação entre a construção e execução das consultas e atualizações foi necessária de maneira a suportar a implementação dos filtros de consulta selecionáveis pelo usuário.

4.3.3 Módulo *pathfinding*

O módulo de busca de caminhos contém todas as classes responsáveis pelo processamento geográfico do mapa da cidade guardado no banco de dados, a estrutura do qual é visível no diagrama pseudo-UML 4.7. Os objetos deste módulo recebem como parâmetro o nome de um previsor de acidentes, o qual será utilizado para cálculo dos pesos de nodos e arestas pelo grafo como será explorado na Seção 4.5.3.

Figura 4.7: Pseudo-UML do módulo de sugestão de caminhos

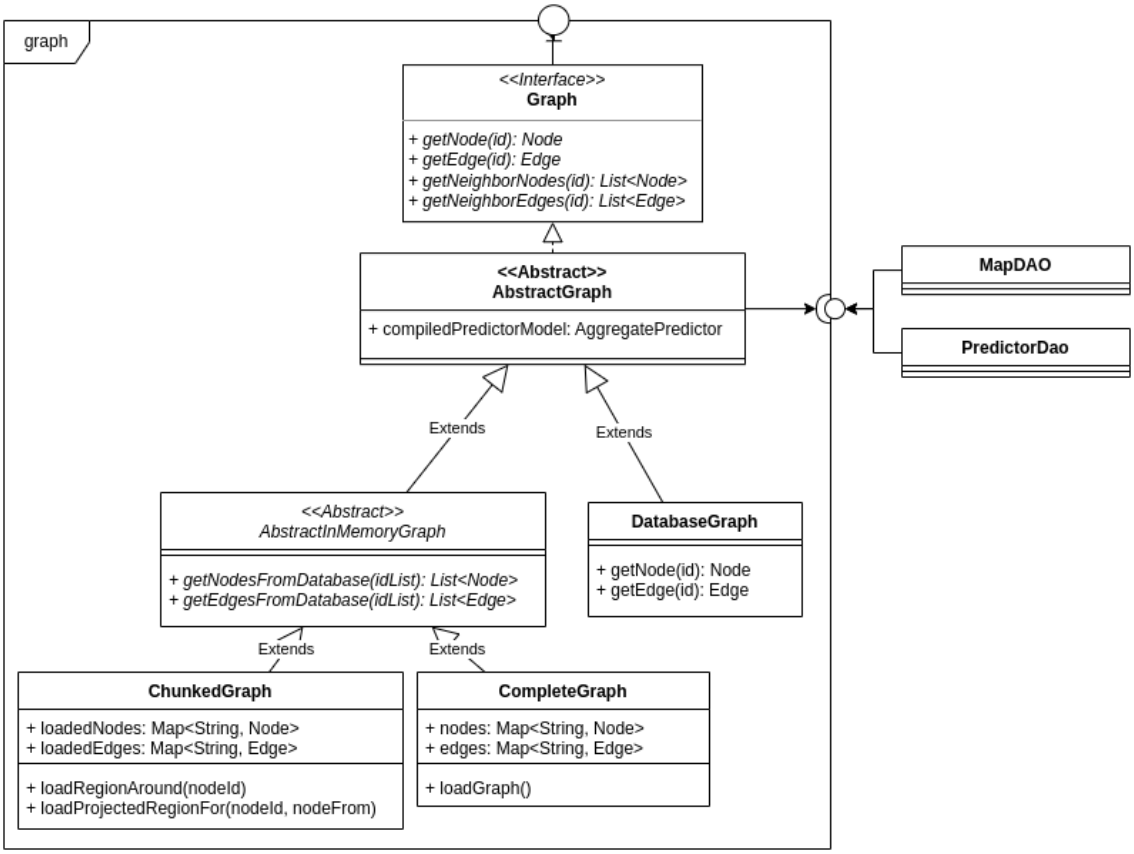


Fonte: Elaborado pelo autor

Aqui foi abstraído o algoritmo utilizado atrás da interface `PathFinder` e classe abstrata `AbstractPathFinder`, a qual oferece a funcionalidade básica de sugestão de caminhos, tirando proveito de uma `GraphFactory` para construção do grafo. Para esta aplicação, foi apenas implementado o

algoritmo A∗, porém este módulo é facilmente estendível para quaisquer outras implementações.

Figura 4.8: Pseudo-UML do módulo de grafos



Fonte: Elaborado pelo autor

Adicionalmente, o grafo também foi abstraído no seu próprio módulo, permitindo a troca entre diferentes implementações. Foram criadas 3 implementações de grafo, as quais foram testadas em termos de *performance* para uso na aplicação. A Figura 4.8 apresenta esta estrutura.

A primeira implementação, `DatabaseGraph`, segue uma abordagem simplificada, onde qualquer busca é executada diretamente sobre o banco de dados. Naturalmente, esta implementação não é adequada, dado que teremos um algoritmo realizando múltiplas consultas por nodos e arestas. As próximas implementações procuram mitigar este problema trazendo o grafo do banco de dados em memória, evitando assim as consultas constantes. A segunda implementação, `CompleteGraph`, busca todo o grafo presente no banco de dados durante sua instanciação, essencialmente removendo a necessidade de consultas durante o processamento pelo algoritmo de

sugestão de caminhos. Pensando na possibilidade de escalação da aplicação, esta abordagem apresenta um grande problema - caso uma área maior seja coberta, este grafo irá aumentar de tamanho, e mais memória será consumida desnecessariamente.

Por fim, chegamos à terceira implementação, `ChunkedGraph`, a qual realiza uma forma de *lazy-loading* de partes (ou *chunks*) do grafo a medida que as mesmas são necessárias. Logo após instanciado, este grafo é vazio, porém à medida que novos nodos ou arestas são requisitados os mesmos são recuperados do banco de dados, em conjunto com todos os outros elementos que se encontram dentro de uma área configurável em torno do ponto solicitado. Esta implementação é ideal para uso por um algoritmo de *pathfinding* em profundidade, uma vez que a expansão durante a execução do algoritmo ocorre sempre em nodos adjacentes ao nodo atual, além de não carregar desnecessariamente partes do grafo que não serão utilizadas.

4.3.4 Módulo *prediction*

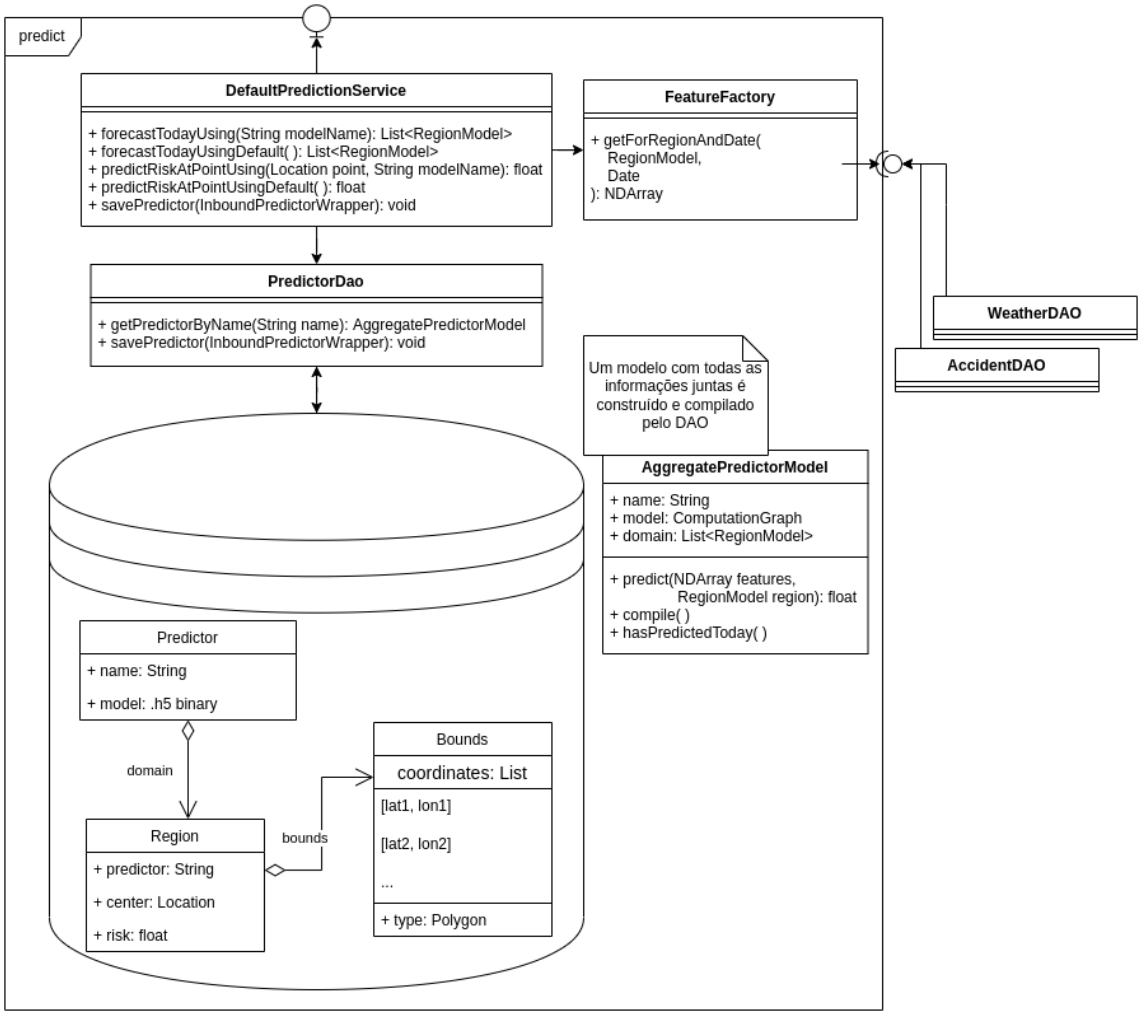
Este módulo é responsável pela coordenação da previsão de risco de acidentes futuros. Para tal, existem duas tarefas necessárias: a criação do modelo e a recuperação das *features* utilizadas na previsão.

O modelo se encontra salvo no banco de dados em formato binário, conforme salvo após o treinamento. Junto do modelo, são salvas a regiões que foram utilizadas para o treinamento, de forma a facilitar a troca em tempo real entre modelos. Cada região é composta por um centro e uma lista de pontos que formam o seu contorno, assim permitindo que as regiões tenham qualquer forma planar. Para simplicidade, o modelo utilizado para este estudo utilizou regiões quadradas, portanto, cada região tem 4 coordenadas que a englobam.

Durante a instanciação, o modelo e suas regiões são recuperados do banco de dados, posteriormente compilado através de chamadas à biblioteca DL4J e salvo na forma de um grafo computacional. Este processo é agregado dentro da classe `AggregatePredictorModel`, que possui todas as informações necessárias para executar a previsão.

A extração de *features* para a data atual é feita em uma classe separada, visto que requer acesso às coleções de acidentes e dados climáticos, chamada

Figura 4.9: Pseudo-UML do módulo de previsão



Fonte: Elaborado pelo autor

`FeatureFactory`. A forma como estas *features* são construídas será explorada em mais detalhes na Seção 4.5.2. A Figura 4.9 apresenta o diagrama pseudo-UML para este módulo.

4.4 Limpeza e Carregamento de dados

O primeiro passo no desenvolvimento de qualquer aplicação envolvendo bancos de dados é a coleta e limpeza das informações que serão utilizadas. Aqui apresentamos as dificuldades encontradas na limpeza de cada uma das três fontes de dados utilizadas, e os métodos adotados para contorná-las.

4.4.1 Acidentes

Os dados históricos de ocorrências de acidentes na cidade de Porto Alegre foram disponibilizados na forma de um arquivo *.csv*, com os registros dispostos horizontalmente e seus valores separados em colunas. O principal desafio desta etapa foi a limpeza da latitude e longitude dos registros dado que esta era a informação mais importante a ser coletada, sem a qual a visualização no mapa não seria possível. De um total de 66.445 registros por volta de 21.000, ou 31%, deles apresentavam um ou mais dos seguintes problemas nas coordenadas geográficas:

- Valores não preenchidos / zerados
- Valores absurdos
- Valores com sinal errado
- Valores sem pontuação das casa decimais

Aqui, valores absurdos são aqueles que não caem dentro da região de Porto Alegre, que para fins deste trabalho foi definida como uma *bounding box* do ponto $(-32°, -53°)$ até o ponto $(-28°, -49°)$. De maneira a manter a integridade dos dados armazenados, o seguinte processamento inicial foi realizado através de um *script* em *python*:

1. Valores não preenchidos foram zerados
2. Todos os valores foram convertidos para *string*
3. Todos os valores tiveram o ponto da casa decimal removido
4. As partes inteira e racional de valores separados por vírgula foram concatenadas
5. Sinal de todos os valores foi definido como negativo
6. Valores absurdos foram zerados
7. Ponto da casa decimal foi adicionado de volta a todos os valores

Para realizar a adição da casa decimal aos valores que não as possuíam, foi identificado que todos estes valores já continham todos os dígitos da latitude ou longitude, apenas sem o ponto. Foi possível então adicionar o ponto da mesma maneira em todos os casos - adicionando após os dois primeiros dígitos do número.

Ao final deste processamento, por volta de 5.000 registros foram normalizados, porém ainda existia uma grande quantidade de registros zerados. Para contornar o problema, foi realizada uma simples cópia de valores de latitude e longitude para estes campos utilizando dados comuns, neste caso, a rua. Todos os registros cujo logradouro 1 e 2 estavam populados, os quais tinham latitude e longitude em outro registro tiveram estes valores transferidos. Isto leva a uma perda de informação, uma vez que o acidente pode ter ocorrido em qualquer altura da rua, porém isto foi preferível à não utilização destes acidentes.

No final de todo o processamento programático, por volta de 1.200 registros permaneciam sem informações de latitude e longitude, e então um processamento manual foi necessário para estes casos. Um esforço foi feito para recuperar as coordenadas destas ruas para cerca de 400 desses registros, porém o autor chegou a conclusão de que, no interesse do tempo, a recuperação dos 800 registros restantes não agregaria valor ao conjunto de dados, uma vez que uma grande porcentagem dos dados já foi recuperada.

Os registros de acidente foram então colocados na estrutura apresentada na Listagem 4.1, e enviados ao servidor através da `REST API` para serem salvos no banco de dados com a mesma estrutura. Durante este primeiro carregamento, as informações de clima não estão presentes, e portanto tiramos proveito do fato de que documentos no *MongoDB* não possuem um esquema fixo, salvando os mesmos sem o objeto `climate` para atualização posterior quando o clima for carregado.

Listing 4.1: Estrutura de um registro de acidente salvo no banco de dados

```
{
  externalId: string,
  address: {
    street1: string,
    street2: string,
    region: RegionEnum,
    location: {
        longitude: float,
        latitude: float
    }
  },
  date: {
```

```
    date: ISODate,
    weekday: int,
    hour: ISODate,
    time_of_day: Long
  },
  fatality: {
    deaths: int,
    light_injuries: int,
    serious_injuries: int
  },
  involvedEntities: [ VehicleEnum... ],
  type: TypeEnum,
  climate: {
    visibility: float,
    relative_humidity_percentage: float,
    precipitation_mm: float,
    wind_speed_ms: float,
    air_temp_celsius: float
  }
}
```

4.4.2 Dados meteorológicos

Os dados meteorológicos utilizados na aplicação também se encontravam no formato *.csv*, porém com uma organização muito maior que os registros de acidentes. Para este caso, não haviam dados incorretos ou inconsistentes na planilha, e apenas foi necessário a extração das informações relevantes, organização na estrutura conforme apresentada na Listagem 4.2 e carregamento pela `API`. Aqui a localização diz respeito à estação meteorológica que foi utilizada para a medição.

Listing 4.2: Estrutura de um registro de clima salvo no banco de dados

```
{
  dateTime: {
    date: ISODate,
    weekday: int,
    hour: ISODate,
    time_of_day: Long
```

```
  },
  visibility: float,
  relativeHumidityPercentage: float,
  precipitationMm: float,
  windSpeedMs: float,
  airTempCelsius: float,
  location: { longitude: float, latitude: float },
}
```

Do lado do servidor, ao receber dados de clima para um certo dia e hora, uma atualização é feita à coleção de acidentes, adicionando os dados de clima para os acidentes adequados. O motivo para esta duplicação de informação - guardando os dados climáticos tanto na coleção de acidentes quanto na coleção de clima - é a facilidade e eficiência no acesso a estas informações em diferentes partes da aplicação. Quando a aplicação mostra informações de um acidente individual na tela, os dados acessados são aqueles presentes no próprio documento do acidente, evitando agregação potencialmente custosa com a coleção de clima. Por outro lado, quando acessando as informações para realização de previsões de acidentes, a coleção de clima é utilizada por conter os dados do dia, sendo considerado na construção das *features*, evitando acessos repetidos e redundantes à coleção de acidentes quando prevendo múltiplas regiões.

4.4.3 Dados geográficos

Como mencionado na Seção 2.1.3, os dados geográficos foram obtidos através de um *script* utilizando o módulo OSMnx, o qual permite o *download* de redes de ruas na forma de um grafo direcional. Este grafo foi carregado na aplicação através de duas coleções, ruas e interseções, também pela *API REST*.

Em um primeiro momento foram persistidas as interseções, as quais consistem de um identificador único e listas, inicialmente vazias, de ruas que as incidem e que delas partem. Esta distinção entre ruas é importante dada a natureza bidirecional do grafo. A localização desta interseção também foi persistida, de maneira a facilitar a busca por interseções e ruas por

proximidade ou em área. A Listagem 4.3 apresenta a estrutura de uma interseção no banco de dados.

Listing 4.3: Estrutura de uma interseção salva no banco de dados

```
{
  externalId: string,
  incomingStreetIds: [ string... ],
  outgoingStreetIds: [ string... ],
  location: { longitude: float, latitude: float },
}
```

Com as interseções no banco de dados, o próximo passo foi o carregamento das ruas, as quais possuem uma estrutura similar, apresentada na Listagem 4.4. Aqui as ruas possuem, além da interseção de origem e destino, um identificador direcional a mais, o qual é composto no formato $\{source\_intersection\_id\} - \{external\_id\} - \{dest\_intersection\_id\}$, necessário dado que a mesma rua pode ser de duas vias, e portanto possuir duas entradas no banco de dados.

Listing 4.4: Estrutura de uma rua salva no banco de dados

```
{
  externalId: string,
  directionalId: string,
  name: string,
  length: float,
  sourceIntersectionId: string,
  destinationIntersectionId: string,
  location: { longitude: float, latitude: float },
}
```

Durante o carregamento das ruas, as listas das interseções previamente carregadas são finalmente atualizadas, cada rua adicionada tendo seu id direcional adicionado à lista correspondente nas interseções.

4.5 Desenvolvimento das Funcionalidades

Nesta Seção se encontram os desafios e escolhas funcionais feitas durante o desenvolvimento de cada um dos 3 pilares principais da aplicação, com detalhes sobre as técnicas e padrões utilizados na implementação.

4.5.1 Mapa Interativo e Consulta de Acidentes

A primeira funcionalidade a ser desenvolvida para a aplicação foi o mapa interativo, através do qual o usuário será capaz de ativar todas as demais. Como peça central da aplicação, foi importante que o mapa e a consulta de acidentes sejam tão genéricos e flexíveis quanto possível, permitindo que o mesmo código seja reutilizado pelas demais funcionalidades.

4.5.1.1 Filtros

Para viabilizar as consultas, foi criada uma classe genérica *FilterWrapper*, a qual abstrai os filtros apresentados na Seção 4.3.1 como uma tupla de 3 valores: campo, operação e valor. Listas de objetos deste tipo são, então, utilizadas por uma classe que segue o padrão *Factory*(FOWLER et

al., 1999) para gerar *queries* específicas à base de dados sendo utilizada, `MongoDB`, realizado através de um mapeamento entre as operações disponibilizadas e critérios Mongo, apresentado na Listagem 4.5.

Listing 4.5: Mapeamento entre as operações e critérios do MongoDB

```
switch (operation) {
    case EQUALS: case IS: case HAS:
        searchCriteria = searchCriteria.is(value);
        break;
    case IS_NOT: case DOES_NOT_HAVE:
        searchCriteria = searchCriteria.ne(value);
        break;
    case GREATER_OR_EQUAL: case AFTER:
        searchCriteria = searchCriteria.gte(value);
        break;
    case GREATER_THAN:
        searchCriteria = searchCriteria.gt(value);
        break;
    case BEFORE: case LESS_OR_EQUAL:
        searchCriteria = searchCriteria.lte(value);
        break;
    case LESS_THAN:
        searchCriteria = searchCriteria.lt(value);
        break;
    default:
        break;
}
```

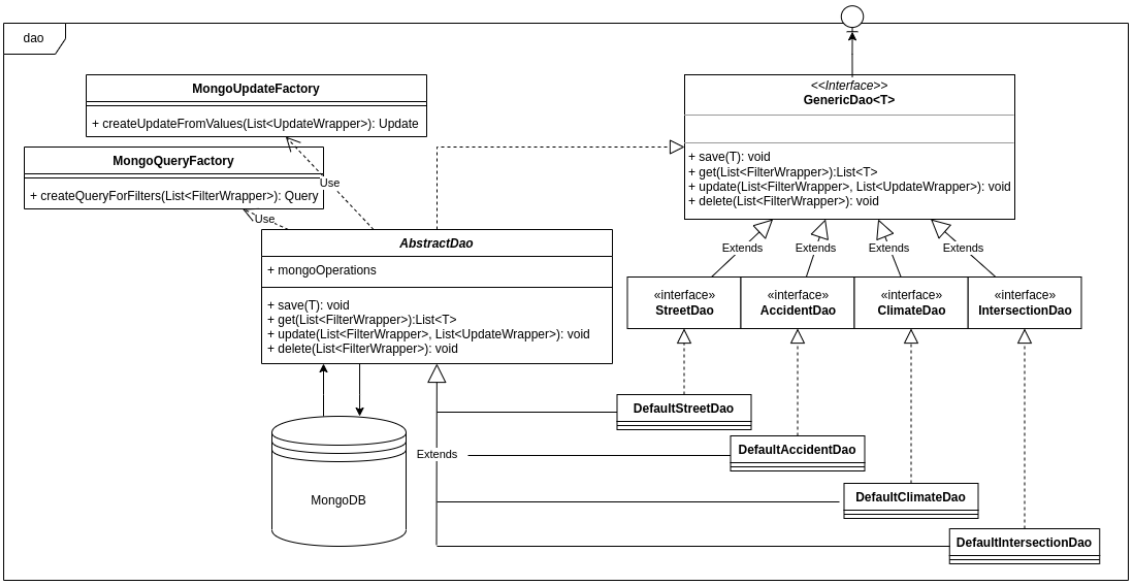
Como estes filtros são genéricos, tal estrutura pode ser utilizada para todos os objetos do domínio como acidentes, clima, etc. com a mesma funcionalidade. Todos os filtros da lista são então concatenados pelo operador lógico `AND`, formando a consulta a ser realizada no banco de dados.

4.5.1.2 Consultas básicas ao banco

As consultas ao banco de dados são realizadas pelas classes *DAO* da aplicação, abstraindo esta camada de acesso aos dados da camada de regras de negócio. Para reduzir o tamanho do código e a repetição desnecessária de código *boilerplate*, foi criada uma classe *AbstractDao*, tirando proveito do *Java Generics* para ter uma única implementação das operações básicas CRUD

compartilhada pelos objetos do domínio. A Figura 4.10 mostra como este *DAO* foi arquitetado.

Figura 4.10: Estrutura do DAO genérico para os objetos do domínio



Fonte: Elaborado pelo autor

Essa estrutura suporta consultas e atualizações simples via filtros, porém para muitos casos foram necessárias consultas mais complexas, geralmente relacionadas à buscas geográficas, as quais foram implementadas nos *DAOs* específicos dos seus objetos.

4.5.1.3 Consultas geográficas

Dado que a aplicação tem seu domínio disposto na forma de um mapa, consultas envolvendo a latitude e longitude também foram utilizadas, tirando proveito das funcionalidades de consulta disponibilizadas pelo *MongoDB*. Duas formas principais de consulta geográfica foram utilizadas - `$geoWithin` e `$geoIntersects`. O operador `$geoWithin` permite que buscas possam ser realizadas por pontos armazenados que se encontram dentro de um determinado polígono, o que será utilizado na sugestão de caminhos para encontrar os pontos iniciais e finais selecionados pelo usuário no mapa. Um exemplo de uso deste operador pode ser visto na Listagem 4.6, a qual procura por todas as ruas e interseções dentro de uma região.

Listing 4.6: Busca por todas as ruas e interseções em uma região

```
protected [...] getGeoPointsForModelRegion(final Region region) {
    final Shape polygon = new GeoJsonPolygon(
        region.getBounds().getCoordinates().get(0).stream().map(list ->
            new Point(list.get(1), list.get(0))).collect(Collectors.toList())
    );
    List<IntersectionModel> intersectionModels = mongoOperations.find(
            Query.query(
                Criteria.where(IntersectionModel.LOCATION).within(polygon)
            ),
            IntersectionModel.class
    );
    List<DirectionalStreetModel> streetModels = mongoOperations.find(
            Query.query(
                Criteria.where(DirectionalStreetModel.LOCATION).within(polygon)
            ),
            DirectionalStreetModel.class
    );
    [...]
}
```

O operador `$geoIntersects` por sua vez permite a consulta reversa - dado um ponto, procuramos o polígono armazenado no banco de dados que o contém. Isto também é utilizado na sugestão de caminhos, de forma a encontrar a região que o contém, de forma a saber o risco associado. Esta busca está apresentada na Listagem 4.7.

Listing 4.7: Busca pela região na qual um ponto se encontra

```
public Query createRegionFromModelIntersectsPointQuery(Location point, String
    modelName) {
    final GeoJsonPoint geoPoint = new GeoJsonPoint(new Point(point.getLatitude
        (), point.getLongitude()));
    return Query.query(
            Criteria.where(Region.PREDICTOR).is(modelName).andOperator(
                    Criteria.where(Region.BOUNDS).intersects(geoPoint))
    );
}
```

Para que essa consulta funcione, as regiões precisam estar armazenadas no banco de dados seguindo o formato `GeoJson`, com o polígono sendo composto por uma `string` que representa seu tipo e uma lista cíclica

de coordenadas. A Listagem 4.8 apresenta essa estrutura para uma região armazenada.

Listing 4.8: Região armazenada com o campo *bounds* no formato GeoJson

```
{
  predictor: 'okers_bin_cross_entropy.h5',
  center: { longitude: -51.26066589355469, latitude: -30.240144729614258 },
  normalizedCenter: {
    longitude: 0.022727273404598236,
    latitude: 0.01515151560306549
  },
  risk: 0.00020454668265301734,
  bounds: {
    coordinates: [
      [
        [ -30.244640350341797, -51.2657585144043 ],
        [ -30.244640350341797, -51.25557327270508 ],
        [ -30.23565101623535, -51.25557327270508 ],
        [ -30.23565101623535, -51.2657585144043 ],
        [ -30.244640350341797, -51.2657585144043 ]
      ]
    ],
    type: 'Polygon'
  }
}
```

4.5.1.4 Mapa

Listing 4.9: Classe do mapa

```
class Map {
    constructor() {
        this.layers = [
            new AccidentsLayer(),
            new LocationSuggestionLayer(),
            new PathViewingLayer(),
            new RiskHeatmapLayer(),
        ];

        this.map = new ol.Map({
            target: 'map',
```

```
12            layers: [new ol.layer.Tile({ source: new ol.source.OSM() })],
13            view: new ol.View({
14                center: ol.proj.fromLonLat([-51.21, -30.04]),
15                zoom: 13
16            })
17        });
18    }
19
20    enableLayer(layer) { [...] }
21    disablelayer(layer) { [...] }
22    centerOnLayer(layer) { [...] }
```

Com todas as funcionalidades de busca e atualização dos dados, viabilizamos a implementação do mapa, criado na interface com o usuário em *javascript*. O mapa é uma classe que engloba todas as diferentes camadas apresentadas, centralizando a criação e o gerenciamento das mesmas. A Listagem 4.9 apresenta esta classe.

4.5.2 Previsão de acidentes

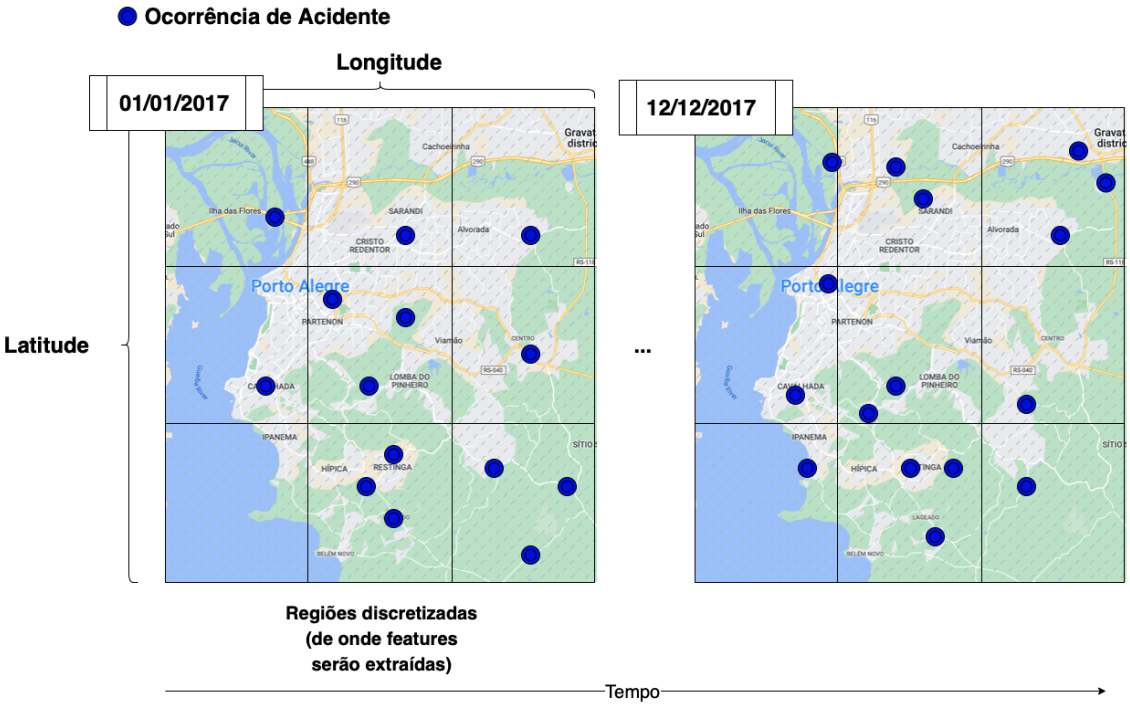
A previsão dos acidentes de trânsito foi planejada como uma espécie de extensão à visualização de acidentes, porém para datas futuras. O objetivo desta funcionalidade é oferecer uma intuição ao usuário sobre quais as áreas mais perigosas da cidade para o dia atual, sem ter conhecimento dos acidentes que irão ocorrer. Esta informação pode ser utilizada tanto funcionalmente pelo usuário, ao visualizar um "mapa de calor"sobre próprio mapa da cidade, quanto programaticamente por outros módulos da aplicação, os quais podem sofrer alteração no seu comportamento considerando estas previsões.

4.5.2.1 Definição do problema e domínio

A previsão será feita, então, por uma rede neural de perceptrons densamente conectada cujo treinamento será realizado com base no amplo banco de dados de registros de acidentes e registros meteorológicos que temos ao nosso dispor, baseando cada previsão individual nas características de clima atuais e no que conhece sobre a região para a qual está prevendo.

Para tal, discretizamos o espaço geográfico da cidade em regiões menores, cujo histórico pode ser analisado individualmente, além de discretizar o espaço temporal em dias. A Figura 4.11 contém uma representação visual deste domínio discretizado. Para cada uma das regiões discretizadas, o objetivo do modelo preditivo será, então, estimar uma probabilidade de ocorrência de acidentes, com valor entre 0 e 1, que indica o quão perigosa a região será naquele dia, com base no conhecimento histórico daquela região.

Figura 4.11: Discretização geo-temporal do domínio dos dados



Fonte: Elaborado pelo autor

4.5.2.2 Definição do modelo

O problema será então resolvido por uma rede neural de perceptrons densamente conectada, a qual tem como *input* um conjunto de *features* F_r da região, além das características geográficas G_r da região, e resulta em um valor no intervalo $[0, 1]$ que representa a probabilidade de ocorrência de acidentes prevista para aquela região. Esta rede foi construída utilizando o *framework keras* através de sua interface funcional, definindo as camadas conforme mostra a Listagem 4.10.

Listing 4.10: Construção do modelo preditivo em python

```
network_input = layers.Input(shape=np.shape(F + G))

dense = layers.Dense(NEURONS_PER_LAYER, activation="relu")(network_input)
for i in range(LAYER_COUNT - 1):
    dense = layers.Dense(NEURONS_PER_LAYER, activation="relu")(dense)

final_dense = layers.Dense(1, activation="sigmoid")(dense)

final_model = keras.Model(network_input, final_dense)
```

4.5.2.3 Persistência e uso do modelo

Após o treinamento, o modelo é armazenado no banco de dados em formato binário, junto das regiões que foram usadas para o treinamento, conforme apresentado na Seção 4.3.4. O Modelo é então reconstruído através de código Java, sendo necessária a criação de um arquivo temporário em formato $.h5$ devido ao fato de a biblioteca DL4J não suportar a criação de modelo via *InputStream*. O código responsável por essa reconstrução está apresentado na Listagem 4.11.

Listing 4.11: Reconstrução do modelo para avaliação

```
InputStream modelByteStream = new ByteArrayInputStream(modelBytes);

try {
    this.modelCompiled =
        KerasModelImport.importKerasModelAndWeights(modelByteStream);

} catch (final UnsupportedOperationException e) {
    // InputStream not supported by current version, need a file
    String modelTempFile = getTempFile(modelBytes).getAbsolutePath();

    this.modelCompiled = KerasModelImport
        .importKerasModelAndWeights(modelTempFile, false);
}
```

No momento do uso, para cada região na qual se deseja executar uma previsão, as *features* precisam ser coletadas, o que é feito através de chamadas aos DAOs de acidentes e clima, utilizando os filtros geográficos e

temporais disponibilizados na aplicação. Os dados recuperados são então processados de maneira a obter as *features* para aquela região na forma de um *array* N-dimensional, conforme definidas durante o treinamento do modelo, e a previsão é executada utilizando o código apresentado na Figura 4.12.

Listing 4.12: Avaliação do modelo na aplicação

```
public float predict(final INDArray features, final Region region) {
    final Location center = region.getNormalizedCenter();
    return modelCompiled.output(
        Nd4j.concat(0, features, center);
    )[0].getFloat(0);
}
```

As previsões realizadas são então salvas no banco de dados junto à região com um período de validade de 1 dia, de maneira a evitar processamento redundante desses índices de risco, operação que pode ser custosa quando repetida muitas vezes.

4.5.3 Sugestão de caminhos

Para realizar a sugestão de caminhos, foi utilizado o algoritmo A∗, devido ao fato de a aplicação estar realizando esta sugestão em tempo real e necessitar eficiência em termos de tempo de execução. O processamento ocorre no módulo `pathfind`, com integração com o módulo `predict` de maneira a levar a probabilidade de ocorrência de acidentes prevista em conta quando sugerindo o caminho.

A sugestão de caminhos assume o peso de cada nodo como o comprimento da rua, isto é, a distância de uma interseção até outra, desta maneira, dando preferência a caminhos mais curtos. Como heurística de qualidade de um certo nodo, foi utilizada a distância em linha reta do nodo atual até o nodo de destino, calculado através da fórmula de *haversine*, apresentada na equação 4.1, assumindo o raio estimado da terra como $R = 6371.0KM$.

$$d = 2R \arcsin \sqrt{\sin^2 \frac{\Delta\varphi}{2} + \cos\varphi_1 \cos\varphi_2 \sin^2 \frac{\Delta\lambda}{2}} \tag{4.1}$$

Para que o risco seja levado em conta, a probabilidade de ocorrência de acidentes foi dividida em 3 classes, e o peso de cada aresta foi alterado de acordo com a fórmula 4.2, com os parâmetros C_1 e C_2 ditando os limites das diferentes classes de risco, e os parâmetros P_1, P_2 e P_3 indicando a penalidade sobre o comprimento da rua a ser aplicada para cada classe. Na fórmula, r representa o risco estimado para aquela região, e l o comprimento real da rua.

$$W_{risk} = \begin{cases} l * (1.0 + (r * P_1)), & \text{if } 0 < r \leq C_1 \\ l * (1.0 + (r * P_2)), & \text{if } C_1 < r \leq C_2 \\ l * (1.0 + (r * P_3)), & \text{if } C_2 < r \leq 1.0 \end{cases} \tag{4.2}$$

Estes parâmetros foram alterados durante o desenvolvimento de maneira a obter os melhores resultados, finalmente chegando aos valores $C_1 = 0.3$, $C_2 = 0.6$, $P_1 = 1.0$, $P_2 = 3.0$, $P_3 = 10.0$. Variações nestes parâmetros e na integração com o módulo de previsão serão apresentadas na Seção 5.

De forma a minimizar o número de acessos ao banco de dados e ainda obter a informação do risco, a implementação de grafo com carregamento parcial dos nodos foi alterada de maneira a carregar regiões arbitrárias, não limitado apenas a regiões circulares. Assim, durante a avaliação do algoritmo, carregamos as interseções e ruas pertencentes à mesma região, a qual possui um valor de risco único, desta forma aumentando levemente o número de acessos às coleções de ruas e interseções, porém diminuindo consideravelmente o número de acessos às regiões do modelo preditivo.

5 EXPERIMENTOS E RESULTADOS

Neste Capítulo estão apresentados os experimentos realizados e resultados para as diferentes funcionalidades da aplicação, além das mudanças que foram realizadas com base em resultados favoráveis e não favoráveis. Cada resultado está complementado com uma análise dos possíveis fatores ou causas que o trouxeram, com possíveis alternativas e melhoras quando pertinente.

5.1 Escopo do trabalho

Inicialmente o objetivo deste trabalho era realizar a implementação desta aplicação sobre um conjunto de dados de acidentalidade e clima referentes a todo o estado do Rio Grande do Sul, porém durante a primeira etapa do desenvolvimento foi descoberto que tais informações não são divulgadas de maneira aberta ao público. Foi feita uma tentativa de contato com a polícia rodoviária estadual, explicando a natureza do trabalho e o porquê da necessidade de acesso aos registros detalhados, porém o acesso foi rejeitado. Devido a isso, o escopo do trabalho foi reduzido para acomodar os dados que estavam disponíveis, neste caso, limitados à cidade de Porto Alegre.

Apesar desta limitação, a aplicação foi desenvolvida da forma mais genérica e agnóstica de dados possível dado o escopo das funcionalidades, mantendo a operação correta das mesmas frente à expansão do conjunto de dados e facilitando a sua extensão. Métodos de adição, remoção e atualização de todos os componentes do banco de dados são disponibilizados através da API. A adição de novas ruas e interseções automaticamente atualiza o banco com as conexões apropriadas, assim como a adição de registros de clima atualizam os acidentes relacionados. Os modelos preditivos foram abstraídos e persistidos no banco de dados junto de suas regiões de treino, permitindo que quaisquer novos modelos sejam treinados e adicionados ao conjunto existente com mínimas modificações no código, limitadas ao módulo de previsão. Todas as classes utilizadas nos componentes da aplicação são abstraídas por interfaces, facilitando múltiplas implementações futuras.

Em suma, a aplicação é flexível a mudanças, com a redução do escopo ocorrendo primariamente devido à limitação no acesso aos dados necessários. Cabe ressaltar que tal limitação não afetou a qualidade do trabalho, ainda sendo possível o desenvolvimento completo da aplicação e experimentação com diferentes conjuntos de *features* para execução das previsões de acidentes.

5.2 Sugestão de caminhos

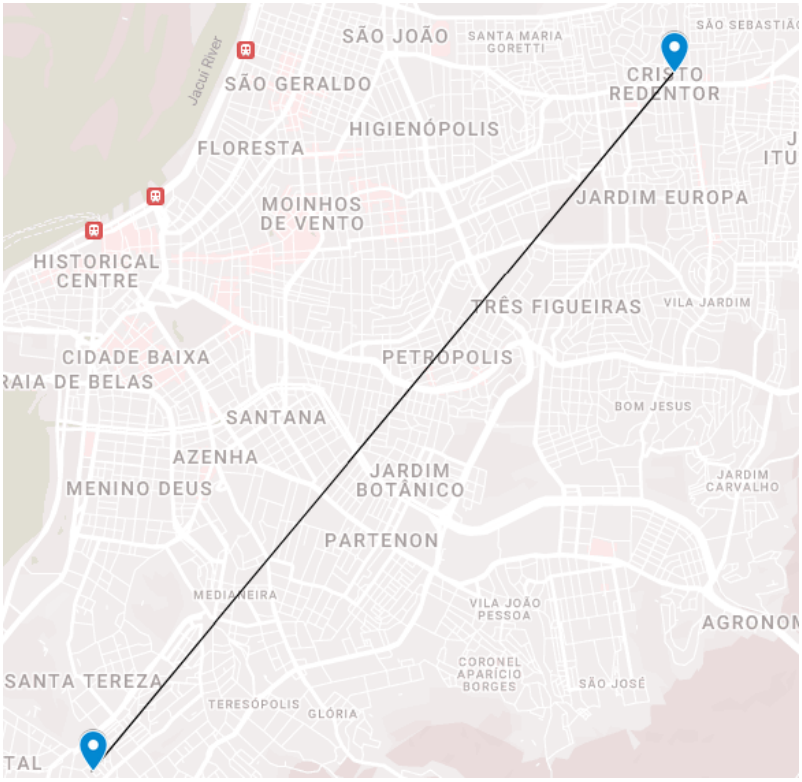
Para a sugestão de caminhos podemos avaliar os resultados obtidos através de dois ângulos, a qualidade do caminho sugerido e a *performance* durante a execução. Enquanto a *performance* pode ser medida de maneira objetiva, através da medição de tempo e memória gasta, a qualidade da solução é relativa, pois os caminhos sugeridos podem ser adequados para um usuário mas não para outro, dependendo do quanto os mesmos estão dispostos a arriscar a ocorrência de um acidente ao tomar um caminho mais curto. A seguir, apresentamos os resultados utilizando diferentes parâmetros de penalização pelo risco e diferentes implementações do grafo, medindo *performance* e apresentando os caminhos sugeridos classificados de acordo com a preferência do autor, porém a qualidade dos mesmos pode ser julgada pelo leitor.

Para a execução com as diferentes implementações de grafo a seguir, os pontos utilizados de início e fim foram $(-30.085148, -51.22842)$ e $(-30.009254, -51.15543)$, respectivamente. Estes pontos estão apresentados no mapa 5.1.

5.2.1 Sugestão de caminhos utilizando `DatabaseGraph`

A primeira implementação é a mais simples, porém também a menos eficiente. O grafo que faz interface direta com o banco de dados realiza uma consulta por cada aresta e nodo solicitado, além de outra consulta pela região à qual o componente pertence de forma a obter a probabilidade de acidente, ou risco, associado. Como mostra a Figura 5.2, esta é uma

Figura 5.1: Pontos de início (SW) e fim (NE) utilizados para os testes



Fonte: https://www.google.com.br/maps

implementação extremamente ineficiente, tomando mais de 20 minutos para encontrar o melhor caminho entre os pontos.

Figura 5.2: *Logs* da *performance* para a previsão utilizando `DatabaseGraph`

```
1 INFO 43417 --- [nio-8080-exec-2] c.e.a.p.algorithm.AbstractPathFinder     :
    Initializing graph took 945407639 nano seconds (0.945407639s)
2 INFO 43417 --- [nio-8080-exec-2] c.e.a.s.impl.DefaultPathFindingService   :
    Path finding took 134305563075 nano seconds (1343.05563075 s)
```

Fonte: Elaborado pelo autor

A fonte desta lentidão certamente está no número de consultas realizadas ao banco. Assumindo E como o número de arestas e N o número de nodos consultados durante a execução do algoritmo, este grafo efetua um total de $2E + N$ consultas ao banco.

5.2.2 Sugestão de caminhos utilizando `CompleteGraph`

A implementação do grafo completo em memória apresenta uma melhoria imensa na *performance*, uma vez que a busca pelo risco associado a cada região ocorre apenas uma vez durante a inicialização do grafo. Podemos ver claramente através da listagem 5.3 que a grande maioria do tempo gasto durante a sugestão dos caminhos se encontra nesta inicialização, com a previsão de fato tomando apenas $1.1s$, por volta de 1.4% do tempo. Neste caso, temos $3R + 2$ consultas ao banco de dados, onde R é o número de regiões associadas ao modelo preditivo, multiplicado por três devido ao fato de serem realizadas consultas pela região, interseções e ruas dentro desta região. As duas consultas adicionais dizem respeito às interseções e ruas que não pertencem a nenhuma região.

Figura 5.3: *Logs* da *performance* para a previsão utilizando `CompleteGraph`

```
1 INFO 43145 --- [nio-8080-exec-1] c.e.a.p.g.inmemory.CompleteHashGraph     :
    Initializing graph took 78294755933 nano seconds (78.294755933 s)
2 INFO 43145 --- [nio-8080-exec-1] c.e.a.s.impl.DefaultPathFindingService   :
    Path finding took 1113028261 nano seconds (1.113028261 s)
```

Fonte: Elaborado pelo autor

Este tempo de execução, no entanto, é inaceitável para uma aplicação com interação do usuário final, muito menos uma aplicação *web*. Como o grafo inteiro é carregado em memória, o custo de memória para grafos maiores, ou até mesmo para o grafo utilizado no caso de múltiplas execuções ao mesmo tempo é exorbitante, dado que uma instância do grafo completo com todas as suas interseções e ruas totaliza 44 megabytes de dados, e portanto esta implementação não é adequada para os objetivos desta aplicação.

5.2.3 Sugestão de caminhos utilizando `ChunkedGraph`

Como esperado, a sugestão de caminhos através da implementação que carrega o grafo a medida que necessário foi a melhor em termos de *performance*, tomando por volta de $1.03s$ para inicialização do grafo (incluindo a região inicial) e $16.24s$ para a previsão do caminho completo, conforme logado pela aplicação na listagem 5.4

Figura 5.4: *Logs* da *performance* para a previsão utilizando `ChunkedGraph`

```
1 INFO 40798 --- [nio-8080-exec-1] c.e.a.p.algorithm.AbstractPathFinder     :
     Initializing graph took 1032040560 nano seconds (1.032040560 s)
2 INFO 40798 --- [nio-8080-exec-1] c.e.a.s.impl.DefaultPathFindingService   :
     Path finding took 16245457486 nano seconds (16.245457486 s)
```

Fonte: Elaborado pelo autor

Esta melhora certamente ocorre devido ao fato de a consulta ao banco de dados em busca do risco de uma região ser a operação mais custosa desse processo, a qual é minimizada por esta implementação, além de cada região ser carregada uma única vez em conjunto com suas ruas e interseções. Aqui temos um total de consultas variável, o qual depende do número de regiões entre os pontos de início e fim, porém temos que, em média, serão executadas por volta de $R/2$ consultas, com a possibilidade de 2 consultas adicionais para nodos e arestas que não pertencem a nenhuma região.

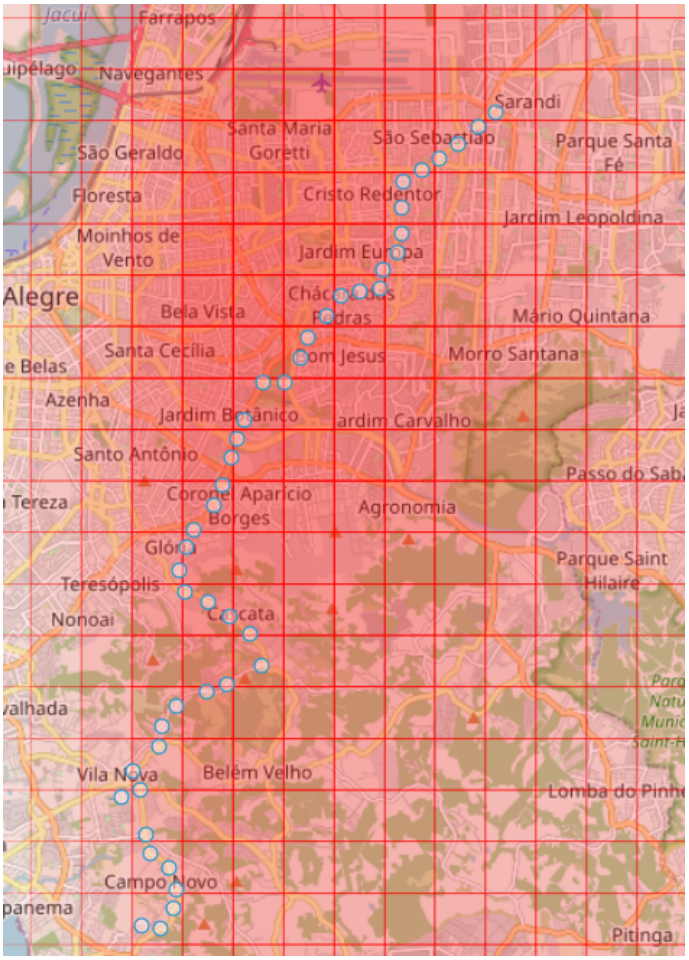
5.2.4 Variações na penalização de ruas pelo risco

Conforme apresentamos na Seção 4.5.3, a penalização no peso de cada rua perante o risco associado à região foi parametrizada, permitindo o teste de múltiplas configurações distintas. A seguir, apresentamos algumas variações nestes parâmetros e na interpretação do risco, mostrando exemplos de caminhos que foram sugeridos pela aplicação.

5.2.4.1 Nenhuma classe de risco, penalidade constante, risco não alterado

Em um primeiro momento, não foram utilizadas classes de risco, e a penalidade no peso de todas as arestas foi simplesmente definida como $l * (1.0 + r)$. Isto apresentava um resultado indesejado, uma vez que em casos onde previsões são muito próximas a sugestão dos caminhos parece não levar em conta o risco associado, prevendo basicamente o caminho mais curto como apresenta a Figura 5.5.

Figura 5.5: Caminho sugerido com peso $w = l * (1.0 + r)$ sem o uso de classes

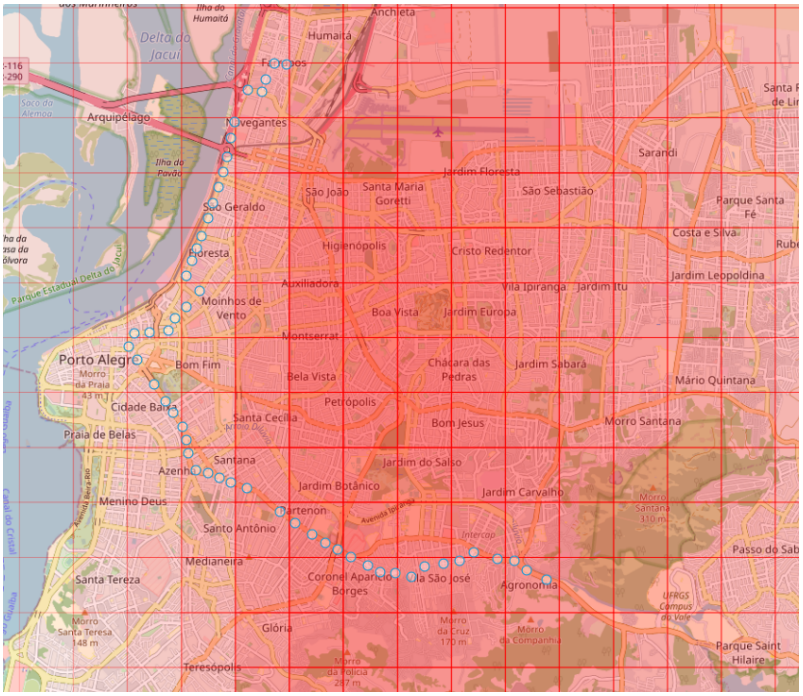


Fonte: Elaborado pelo autor.

5.2.4.2 Nenhuma classe de risco, penalidade constante, risco normalizado

De forma a amenizar o problema de previsões similares para efeitos de cálculo do novo peso, o risco foi normalizado do intervalo $[r_{min}, r_{max}]$ para o intervalo $[0, 1]$, onde r_{min} e r_{max} são o mínimo e máximo risco previstos para o dia, assim esticando os valores e proporcionando um grau maior de diferença entre os pesos. O intuito por trás dessa normalização é tornar o algoritmo flexível às condições do dia para qual está realizando a previsão - se há uma chance alta ou baixa em todas as regiões, é importante poder distinguir quais são mais seguras e quais são mais perigosas, independente do tamanho da rua. A Figura 5.6 apresenta o resultado desta normalização, onde vemos que o melhor caminho passa a ser aquele que evita a região de maior risco.

Figura 5.6: Caminho sugerido com peso $w = l * (1.0 + r)$ e r normalizado para $[0, 1]$



Fonte: Elaborado pelo autor.

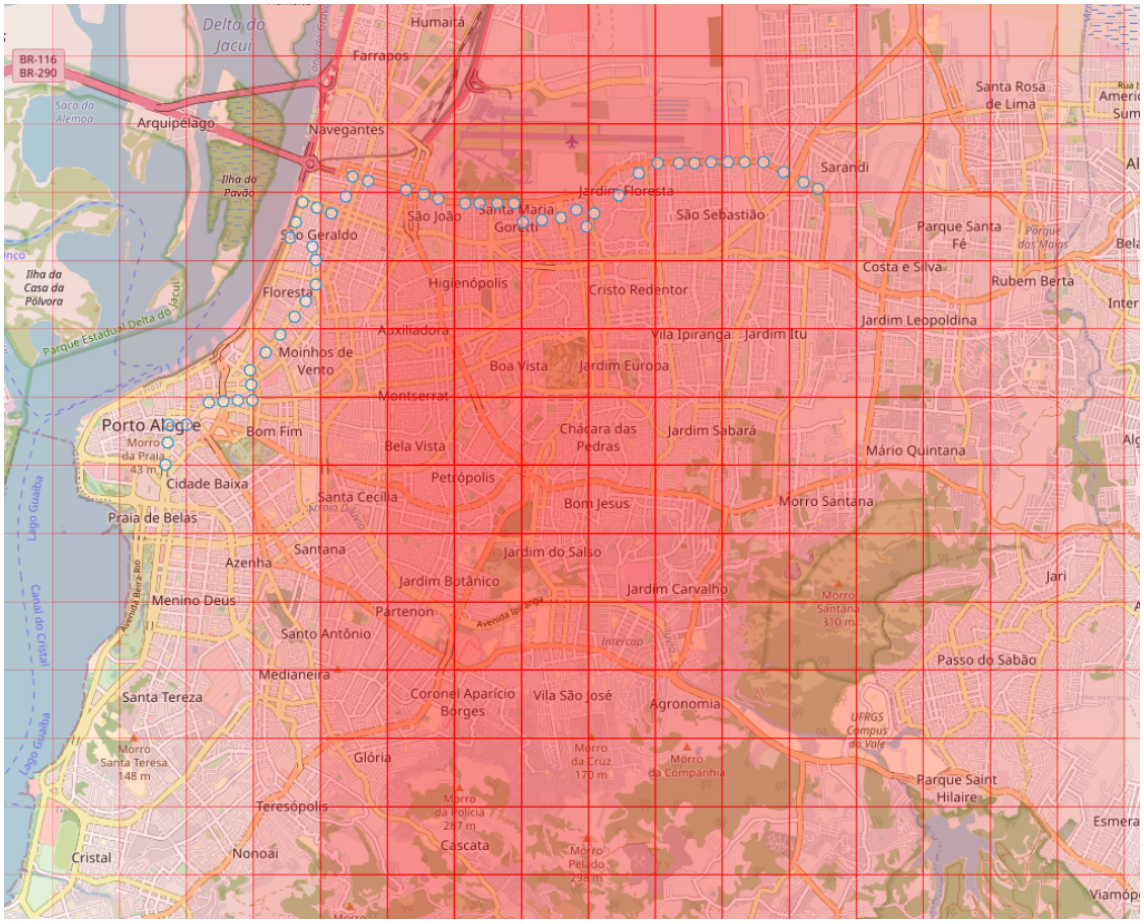
5.2.4.3 Diferentes classes de risco, risco normalizado

Por fim, chegamos à configuração que, com base nos experimentos, é a mais adequada. Com o risco normalizado agora, podemos dividir seu domínio em 3 porções de tamanho similar, com a última classe sendo um pouco maior de forma a captar mais regiões. A penalização para cada região pode ser alterada separadamente, aumentando a medida que o risco aumenta, sendo muito baixa para a classe de menor risco e muito alta para a classe de maior risco. A Figura 5.7 apresenta o caminho sugerido entre dois pontos separados por uma região de risco. É claramente perceptível que o algoritmo evitou a área central, de maior risco, em favor de uma caminho mais longo, porém com menor risco associado.

5.3 Previsão de acidentes

O previsor de caminhos foi a funcionalidade que sofreu o maior número de alterações durante o seu desenvolvimento, constantemente

Figura 5.7: Caminho sugerido com $C_1 = 0.3$, $C_2 = 0.6$, $P_1 = 1.0$, $P_2 = 3.0$, $P_3 = 10.0$



Fonte: Elaborado pelo autor.

passando por alterações de parâmetros de treino, interface de entrada e saída, arquitetura, funções e métricas de perda, entre outros. Podemos julgar a qualidade destas variações através dos mesmos métodos utilizados para o algoritmo de sugestão de caminhos, avaliando o tempo necessário para o treinamento e previsão, além da qualidade da solução. Por fim, também podemos analisar as previsões para datas que não foram utilizadas no conjunto de treino, comparando visualmente as previsões com os acidentes que ocorreram de fato naquele dia assim determinando a qualidade daquela previsão.

5.3.1 Treino da rede

O treino da rede ocorreu através do *framework* `Keras`, com o critério de parada da rede sendo definido como o número de épocas de treino, com uma época representando uma passada completa pelo conjunto de dados de treino. Este critério foi escolhido devido a falta de uma máquina potente para o treino, o que torna cada seção de treino da rede muito lenta, levando mais de duas horas para completar cinco épocas. Durante a maior parte do desenvolvimento o critério de avaliação para as redes treinadas foi a entropia cruzada binária, através da qual somos capazes de avaliar objetivamente a qualidade dos resultados. Esta fórmula retorna um valor entre 0 e 1 representando a acurácia dos resultados previstos quando comparados aos resultados verdadeiros do conjunto de teste. Uma avaliação inicial da rede foi feita baseada nesta métrica, com

5.3.2 Saídas esperadas

A primeira escolha importante para o treinamento da rede foi a saída esperada, a qual deve representar com precisão o risco associado àquela região. Este risco é definido, principalmente, como a chance do usuário sofrer um acidente de trânsito enquanto transita por aquela região. De qualquer forma, foram experimentadas diferentes saídas utilizando mais informações do que a simples ocorrência ou não de um acidente naquela região naquele dia.

A primeira opção de saída considerada foi a simples probabilidade de ocorrência de um acidente naquela região-dia, doravante chamada de Y_{prob}. Y_{prob} é um número no intervalo $[0, 1]$, derivado do conjunto de testes de maneira simples e binária, com região-dias em que houve pelo menos uma ocorrência tendo probabilidade esperada 100%, e as demais tendo probabilidade esperada 0%.

A segunda opção de saída foi definida como um índice de risco, calculado através da fórmula 5.1, com T_x representando o número total de mortes, feridos sérios ou feridos leves naquela região-dia.

$$Y_f = \frac{3 * T_{deaths} + 2 * T_{serious\_injuries} + 1 * T_{light\_injuries}}{6} \tag{5.1}$$

O objetivo deste cálculo é dar um maior índice de risco às regiões onde um grande número de acidentes ocorreu ou regiões com mais acidentes sérios, porém isso traz a dificuldade da limitação deste valor. Y_f se encontra no intervalo $[0, \inf)$, visto que não existe um limite na quantidade ou seriedade de acidentes que podem acontecer. Isto nos força a definir um limite superior a partir do qual todos os valores serão considerados iguais e, portanto, implica em uma perda de informação.

A terceira e última opção expande a ideia do índice de risco, porém ataca o problema do intervalo através da normalização dos valores totais de mortes, feridos sérios ou feridos leves tomando o máximo entre os 3 como valor esperado. A perda de informação ainda ocorre, porém agora ela está relacionada aos 3 totais, facilitando a definição destes limites através de uma breve racionalização. O último índice de risco é calculado então de acordo com a fórmula 5.2

$$Y_m = max(min(1.0, \frac{T_{deaths}}{L_{mortes}}), min(1.0, \frac{T_{serious\_injuries}}{L_{serious\_injuries}}), min(1.0, \frac{T_{light\_injuries}}{L_{light\_injuries}})) \tag{5.2}$$

Assumimos que a existência de uma morte imediatamente representa o máximo risco, portanto definimos o limite de mortes L_{mortes} como 1. Para o número de feridos sérios utilizamos $L_{serious\_injuries} = 2$ dado que em um acidente em geral envolve dois veículos, e portanto contabilizamos feridos sérios para ambos. Por fim, definimos $L_{light\_injuries}$ como 4 de forma a ainda contabilizar feridos leves sem gerar valores exagerados, dado que a média de feridos leves entre acidentes que os tiveram é por volta de 1.1.

5.3.3 Conjunto de *features* exploradas

Como entrada de qualquer rede neural é importante definir um conjunto de *features* que serão utilizadas para obter as previsões, as quais idealmente influenciam o resultado real, ou pelo menos têm alguma relação

com o mesmo. Nos experimentos a seguir, realizamos diversos embaralhamentos das *features* utilizadas, sempre em busca de melhorar a qualidade e eficiência da previsão, e portanto primeiro descrevemos quais são elas, o motivo por trás de sua escolha, como foram extraídas e como foram normalizadas.

5.3.3.1 Features de clima

Iniciando pelo clima, temos à nossa disposição informações de visibilidade, umidade relativa, precipitação, velocidade do vento e temperatura. Cada uma dessas informações pode ser utilizada como uma *feature* válida, uma vez que todas afetam, de uma forma ou outra, as condições de trânsito. Deste conjunto foram escolhidas 3 *features* que afetam mais diretamente a possibilidade de ocorrência de acidentes.

A primeira *feature* escolhida foi a visibilidade (F_v), dado que este é o fator de maior importância enquanto se está dirigindo um veículo. Com visibilidade baixa existe um risco muito mais elevado de colisões e atropelamentos, além de choques e abalroamentos. Este valor já se encontra no intervalo $[0, 1]$ por representar uma porcentagem, e portanto não foi necessária normalização alguma.

A segunda característica utilizada foi a precipitação (F_p) pelo fato de a chuva ser outra causa bem conhecida de acidentes de trânsito. No conjunto de dados analisado temos os valores de precipitação entre o intervalo $[0, 78.6]$, em milímetros por hora, e portanto foi necessária a sua normalização. De acordo com o instituto meteorológico americano (FACT. . . , 2012), podemos classificar a intensidade da chuva em quatro categorias com base na sua precipitação em milímetros por hora, com a classe mais intensa, chuva violenta, sendo atribuída a precipitações maiores que $50mm/h$ e a terceira, chuva pesada, atribuída a precipitações maiores que $10mm/h$. Como a média de precipitação no conjunto de dados disponibilizado é de $1.3mm/h$, com apenas 250 registros ultrapassando $10mm/h$, optamos por definir o valor máximo para precipitação como 10. A fórmula 5.3 mostra como a precipitação foi normalizada.

$$F_p = min(1.0, \frac{precipitation}{10}) \tag{5.3}$$

A terceira e última *feature* climática utilizada foi a umidade relativa do ar (F_h), dado que a mesma é utilizada por ferramentas de previsão meteorológica para determinar como o tempo irá se comportar no futuro próximo. Esta *feature* pode então ser utilizada pelo modelo como uma intuição do comportamento futuro esperado, possivelmente auxiliando nas previsões. Esta característica também representa uma porcentagem, e portanto não requer normalização.

5.3.3.2 Features de acidentalidade

Outro conjunto importante de *features* sobre uma região é seu histórico de acidentalidade. Se muitos acidentes ou acidentes muito sérios ocorreram ali, a chance de isto ocorrer novamente é mais alta do que uma região que nunca teve ocorrências. Tomamos como característica que representa esse risco para ser alimentada ao modelo, a própria saída esperada, o índice de risco associado (F_r). Experimentos ocorreram com as 3 formas de cálculo apresentadas na Seção 5.3.2. Aqui é importante ressaltar que esta *feature* só pode ser utilizada quando calculada para datas passadas, já que não conhecemos os acidentes da data atual para calculá-la, e tendo este valor a previsão seria redundante. O uso dessa *feature* portanto se limita ao modelo `LSTM` que será apresentado na Seção 5.3.5.

5.3.3.3 Features temporais

Um terceiro conjunto foi utilizado como entrada para o modelo cujo objetivo é transmitir a informação temporal em um dado momento. Duas *features* fazem parte deste conjunto, o dia da semana (F_w) e a hora do dia (F_t). O dia da semana é simplesmente um valor no intervalo $[0, 6]$ com 0 representando domingo, enquanto que a hora do dia é um número entre $[0, 82800000]$, representando a hora 0 até 23. Ambas estas características foram utilizadas devido ao fato de acidentes serem mais comuns durante dias da semana, e durante horas específicas do dia como "hora do rush", por exemplo. As equações 5.4 e 5.5 mostram a normalização das mesmas.

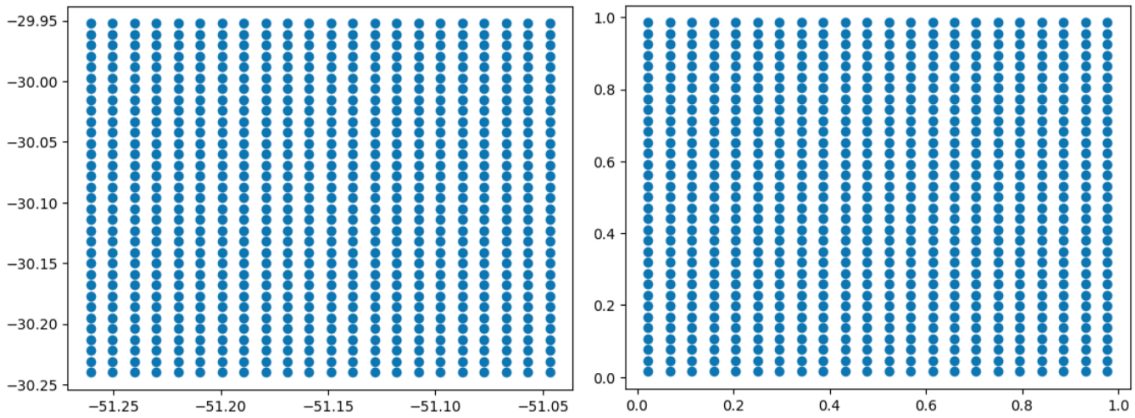
$$F_w = \frac{weekday}{6} \tag{5.4}$$

$$f_t = \frac{time\_of\_day}{82800000} \tag{5.5}$$

5.3.3.4 Features geográficas

Por fim, cada região possui um conjunto de características que não dependem de nenhum fator, e são invariantes no tempo. Este conjunto é a latitude e longitude do centro, e precisa ser utilizado pelo modelo de maneira a endereçar a região para qual está sendo realizada a previsão. A latitude (G_{lat}) e longitude (G_{lon}) foram normalizadas como a distância percentual dentro ao extremo sudoeste do terreno completo estudado, como está apresentado na Figura 5.8.

Figura 5.8: Centros das regiões reais (esquerda) e normalizados (direita)



Fonte: Elaborado pelo autor.

5.3.4 Definição das regiões

Através dos registros de acidentes disponíveis foram obtidos os limites que definem o terreno da cidade de Porto Alegre, através das máximas e mínimas latitude e longitude observadas nos pontos de acidentes. A motivação para essa escolha de definição foi capturar todos os acidentes no terreno de maneira a utilizar o máximo possível do conjunto de dados disponibilizados durante o treinamento. Como mencionado na Seção 4.5.2.1, o terreno foi então dividido em R regiões quadradas, com o critério de divisão das regiões sendo o tamanho desejado de cada região S_r em quilômetros.

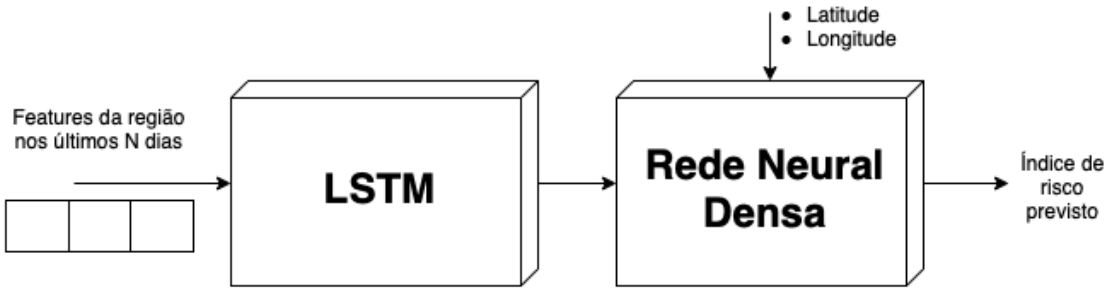
5.3.5 Modelo LSTM

O modelo LSTM foi a primeira arquitetura utilizada, dado que o domínio do problema aparenta se encaixar muito bem em uma série temporal, área na qual LSTMs são muito eficientes(REN et al., 2017). Inicialmente, o problema foi descrito como:

- Dadas as *features* F de uma região r nos últimos N dias, qual será o risco do dia atual?

Esta definição de problema leva a um *feature set* com formato (N, F), e com um *output* esperado unitário, que representa de alguma forma o risco associado com aquela região.

Figura 5.9: Representação do *pipeline* da rede LSTM concatenada à rede densa



Fonte: Elaborado pelo autor.

Dado que existem qualidades de regiões que não se alteram em curto prazo ou de forma alguma, como por exemplo a latitude, longitude, e quaisquer características geográficas, a saída da rede LSTM foi concatenada ao conjunto de *features* invariantes (no tempo), e utilizada como entrada de uma segunda rede, a qual é uma rede neural de perceptrons densamente conectada. A saída desta rede então é utilizada como a previsão para aquele dia e aquela região. A arquitetura da rede completa está apresentada na Figura 5.9.

A rede então passa a possuir duas entradas, uma no formato (N, F) e outra no formato (G), onde G é o número de *features* geográficas. Para este estudo $G = 2$, dado que a única informação geográfica que temos sobre a região é a latitude e longitude do seu centro. Como consequência desta entrada, temos que o tempo de processamento para executar uma previsão é

indesejado, uma vez que a construção das *features* é muito custosa, requerendo múltiplas consultas ao banco de dados ou um processamento demorado pela aplicação.

5.3.6 Modelo tradicional densamente conectado

A segunda arquitetura utilizada simplifica o modelo `LSTM`, removendo este passo e mantendo apenas a rede neural densamente conectada. Esta mudança foi feita após uma observação mais detalhada do conjunto de dados de acidentes, através da qual foi identificado que as ocorrências estão esparsamente dispostas ao longo do tempo. A ocorrência de acidentes muitas vezes eram precedida por muitos dias sem acidentes, assim como períodos curtos com acidentes nem sempre levavam à mais ocorrências.

Neste modelo utilizamos simplesmente as características climáticas, geográficas e temporais do momento atual para realizar a previsão, simplificado o treinamento e uso da rede. Temos que o conjunto de entradas da rede tem então o formato $(G + F)$, abandonando a dimensão temporal.

5.3.7 Configurações da rede

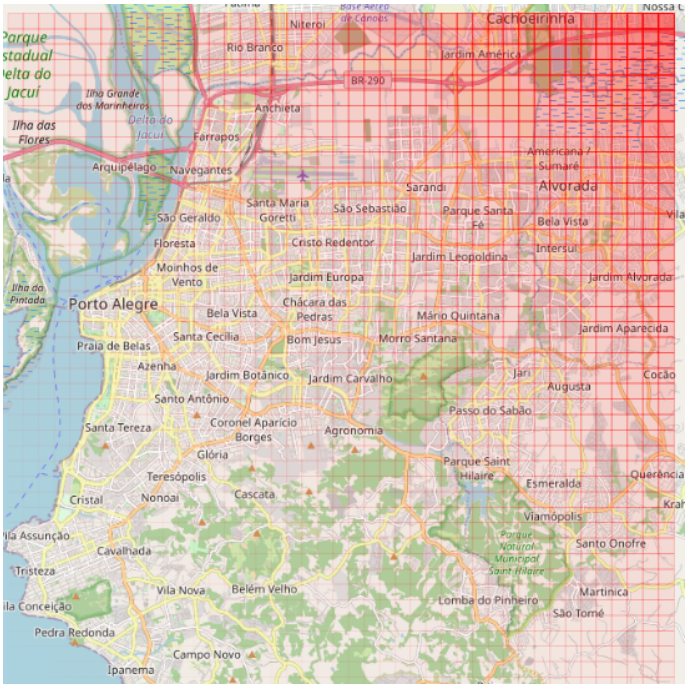
Com as duas arquiteturas da rede e definições de entradas e saídas podemos realizar comparações sobre a *performance* e qualidade das suas previsões visualmente através de gráficos e plotagens no mapa. Em seguida apresentamos diferentes configurações utilizadas em conjunto com suas previsões, incluindo os parâmetros e *features* consideradas. Utilizamos para todos os testes a seguir a data 07 de Julho de 2022, a qual não foi, em nenhum momento, utilizada para o treino da rede, de maneira a evitar a má interpretação dos resultados devido a *overfitting*. Aqui não estão presentes todos os testes realizados, apenas aquelas configurações que obtiveram resultados mais peculiares e que proporcionaram uma direção na alteração dos parâmetros.

5.3.7.1 LSTM com todas as features

Este teste foi realizado utilizando a arquitetura LSTM, com todas as *features* apresentadas na Seção 5.3.3 sendo alimentadas à rede. Restam então dois parâmetros que podem ser variados: o tamanho de cada região (S_r) e o número de épocas de treino (E). Adicionalmente, a arquitetura da rede pode ser alterada variando número de camadas (L_x) e neurônios por camada (S_l) de cada parte da rede. Como valor esperado utilizamos o segundo cálculo de índice de risco, Y_f, onde o valor corresponde a uma média ponderada do número total de mortes, ferimentos sérios e ferimentos leves na região para aquele dia. O período anterior observado para cada previsão é composto pelos últimos 7 dias, ou seja $N = 7$.

O primeiro experimento foi realizado utilizando $S_r = 0.5km$, treinado por 10 épocas ($E = 10$). A rede estava configurada de forma que havia apenas uma camada para cada parte ($L_{lstm} = 1$, $L_{dense} = 1$) com 32 neurônios por camada ($S_l = 32$). A métrica *MSE* foi utilizada como função de perda, com o modelo tentando minimizar o erro quadrático médio entre suas previsões e os valores reais.

Figura 5.10: Previsão para a primeira configuração da rede.



Fonte: Elaborado pelo autor.

Apesar do modelo atingir um erro quadrado médio na casa de $1e^{-6}$, a Figura 5.10 apresenta a previsão realizada pela rede para o dia, a qual aparenta seguir uma gradiente na qual o ponto nordeste é o mais perigoso, e o nível de perigo cai a medida que as regiões se afastam deste ponto. Esta previsão certamente não segue nenhuma lógica, com o modelo aparentemente utilizando apenas a latitude e longitude como valores *pass-through* na definição do risco. Adicionalmente, a maioria das regiões que foram previstas como mais perigosas não apresentavam nenhuma ocorrência de acidente no conjunto de dados inteiro, reforçando a noção de que esta era uma péssima previsão.

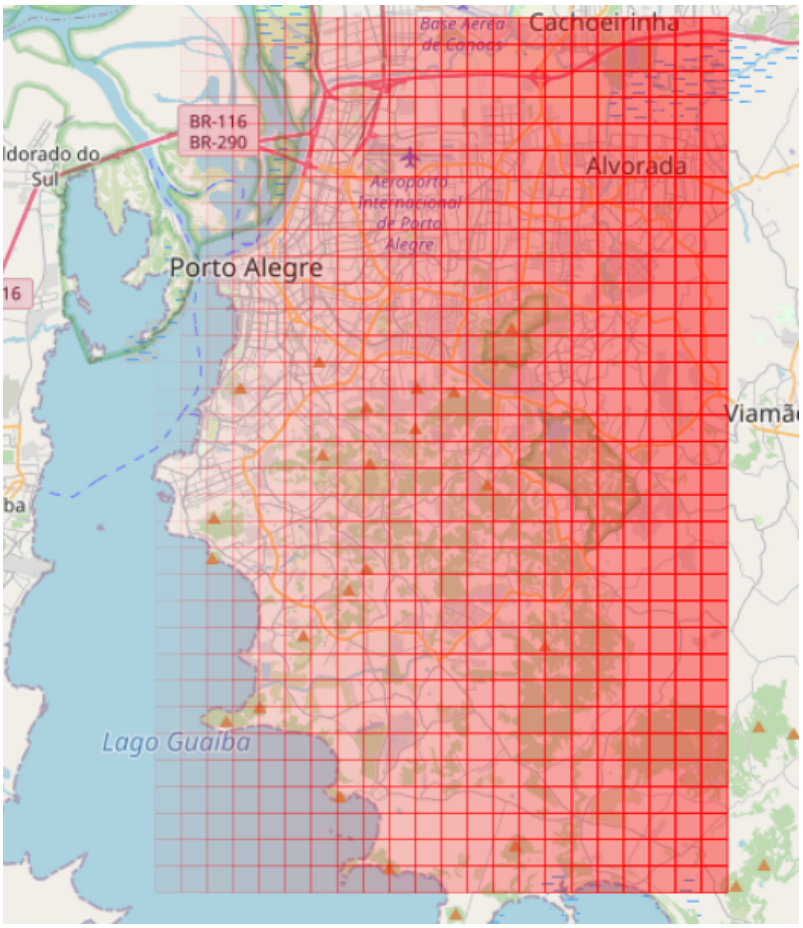
Devido ao tamanho pequeno das regiões tivemos um número total de regiões $R = 2838$, o qual é muito elevado e afetou seriamente a *performance*. Outro problema foi o número elevado de *features*, o qual precisou ser computado 7 vezes para cada dia de cada região. Com estes números, temos que o conjunto de *features* é computado mais de 19 mil vezes para realizar uma previsão completa para o território, resultando em um tempo de previsão maior que 15 minutos.

5.3.7.2 LSTM com parte das features

A segunda configuração atacou o problema do tempo de processamento, reduzindo o número de regiões ao utilizar $S_r = 0.5$ e cortar grande parte das *feature*. O seguinte conjunto de *features* foi utilizado para esta configuração: $(F_{lat}, F_{lon}, F_r, F_h, F_v, F_p, F_h)$. Também foi alterada a saída esperada da rede para o terceiro cálculo, Y_m, além de aumentar o número de camadas da LSTM e da rede densa para 4. A função de perda e número de dias analisados não foram alterados. A Figura 5.11 apresenta a previsão realizada.

Apesar da redução no tempo de processamento para em torno de 5 minutos ainda temos um resultado indesejado na previsão, novamente seguindo uma espécie de gradiente sem qualquer lógica associada. Neste ponto, todos os fatores apontam à conclusão de que a arquitetura LSTM não é adequada para este problema, dado que independentemente da configuração utilizada de *features*, função de perda, número de camadas, *outputs* esperados, a rede sempre retornava uma previsão gradiente, tomando algum

Figura 5.11: Previsão para a segunda configuração da rede.



Fonte: Elaborado pelo autor.

extremo aparentemente aleatório como mais perigoso.

5.3.7.3 Rede somente densa com o conjunto completo de features

Com o componente `LSTM` removido, a nova rede começou a trazer resultados muito melhores, tanto no quesito de *performance* quanto no quesito de qualidade. Desta forma, a rede não mais se baseia em ocorrências de acidentes em um curto período de tempo antes do dia atual, e sim nas características do próprio dia, sendo apenas treinada com as ocorrências anteriores.

A primeira configuração apresentada utiliza novamente um conjunto completo de *features*, com exceção do índice de risco Y_r, porém agora com uma diferença no resultado esperado. O modelo agora prevê a primeira fórmula apresentada, Y_p, a qual simplesmente indica a probabilidade de

ocorrência ou não de acidentes. Outra mudança importante realizada foi a da fórmula de perda utilizada para o treinamento.

A provável causa das gradientes visíveis nas previsões anteriores foi o fato de a rede estar aprendendo a prever valores que são próximos da média dos resultados calculados para o *dataset*, desta forma minimizando o *MSE*, métrica utilizada, porém prejudicando a qualidade da previsão. Como forma de evitar este comportamento, a fórmula de perda foi alterada para a entropia cruzada binária (*Binary cross entropy*), a qual é calculada de acordo com a fórmula 5.6.

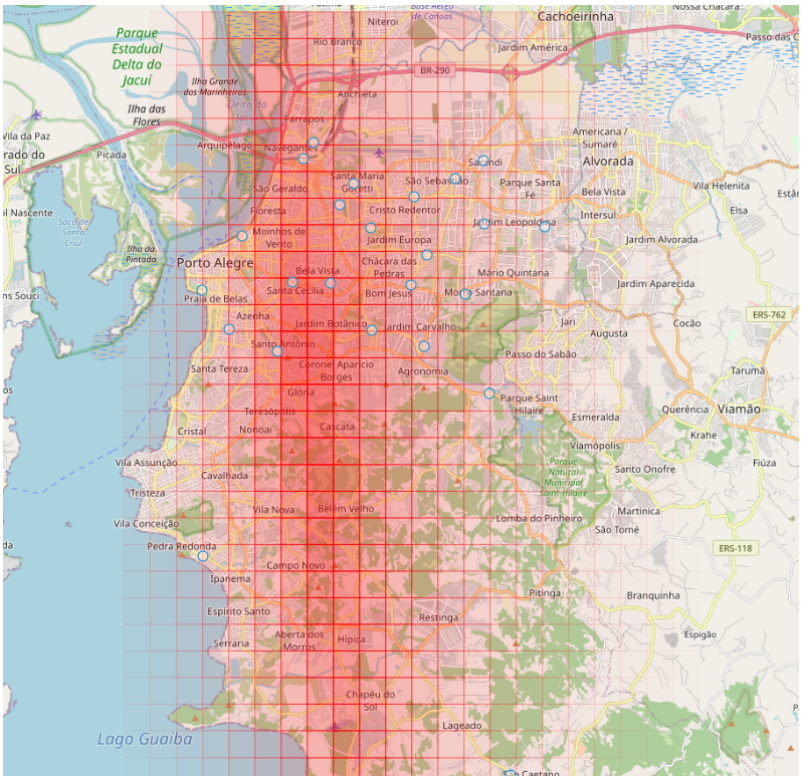
$$loss = -(Y_{true} \log(Y_{pred}) + (1 - Y_{true}) \log(1 - Y_{pred})) \tag{5.6}$$

O objetivo por trás dessa troca foi a forte penalização de previsões incorretas perto da média, devido ao fato de nosso conjunto de dados ser fortemente desbalanceado, tendo muito mais resultados 0 do que 1, puxando a média para perto de zero. Na entropia binária cruzada a classe que se deseja prever, neste caso, é 1 e portanto previsões muito longe deste valor implicam em uma perda maior.

Com estes fatores em mente um novo modelo foi treinado por 5 épocas $E = 5$ utilizando a arquitetura densa, com apenas 1 camada ($L_{dense} = 1$) e 10 neurônios ($S_l = 10$), a qual apresenta um resultado muito mais realista quando comparado aos anteriores, apresentado na Figura 5.12. Claramente podemos ver que o modelo apontou a região central da cidade como mais perigosa, com uma faixa de risco indo da região norte até a região sul. Além disso, a Figura apresenta os acidentes que de fato ocorreram naquele dia como os pontos azuis, os quais realmente se encontram na região norte central da cidade.

Expandindo este modelo foram feitas alterações no número de camadas, esperando obter resultados melhores com a complexidade adicional da rede. Observamos, no entanto, que para quaisquer configurações com mais de 4 camadas os resultados da rede perdiam qualidade, com a previsão de risco sendo mais diluída pelo território e menos assertiva, similar ao problema anterior da gradiente.

Figura 5.12: Previsão para a terceira configuração da rede.



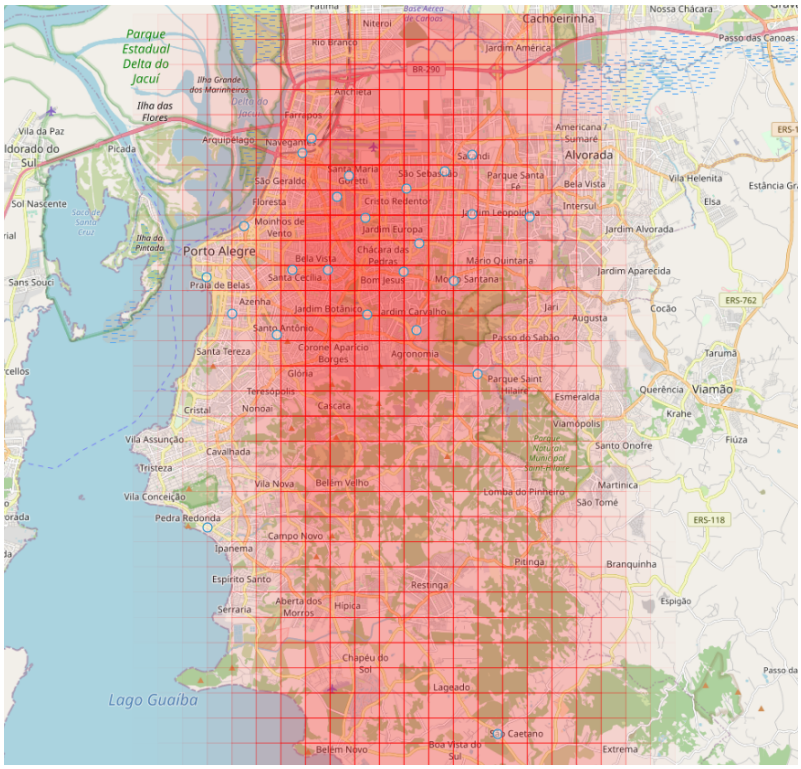
Fonte: Elaborado pelo autor.

5.3.7.4 Rede densa com número de features reduzido

Por fim, foi concluído com base nas configurações anteriores que a quantidade elevada de *features* poderia estar prejudicando a qualidade do resultado e portanto reduziu-se o conjunto a aquelas cuja relação com as ocorrências de acidentes era mais forte. O conjunto final de *features* utilizados para o treino da rede foi então definido como $(F_{lat}, F_{lon}, F_v, F_p, F_t)$. Este conjunto de *features* atinge um bom resultado sem afetar a *performance* da previsão, a qual foi reduzida de 5 minutos no modelo `LSTM` para em torno de 10 segundos.

Finalmente, foi treinada uma rede com 4 camadas ($L_{dense} = 4$) de 32 neurônios cada ($S_l = 32$) por 5 épocas ($E = 5$), os resultados da qual podem ser vistos na Figura 5.13. Esta rede é mais agressiva em suas previsões, prevendo a existência de risco em muitas das regiões do território, porém acerta corretamente o *cluster* de regiões no norte da cidade onde os acidentes de fato ocorreram.

Figura 5.13: Previsão para a quarta configuração da rede.



Fonte: Elaborado pelo autor.

6 CONCLUSÃO

Neste trabalho desenvolvemos uma aplicação capaz de armazenar, gerenciar, analisar e visualizar não somente acidentes de trânsito que ocorreram na cidade de Porto Alegre nos últimos 5 anos, mas também as condições climáticas que influenciaram a sua ocorrência. Também desenvolvemos uma ferramenta capaz de aprender com essas informações passadas de forma a prever regiões com maior possibilidade de ocorrências futuras. Além disso, providenciamos a implementação de um algoritmo de busca de caminhos que leva estas previsões em conta na sua execução, evitando regiões de risco de maneira a manter um viajante por esse percurso mais seguro.

Como resultado dessa implementação, tivemos a oportunidade de avaliar quais fatores de risco e fatores geo-climáticos realmente tem alguma relação com ocorrências de acidentes, além de realizar um estudo sobre quais métodos são mais apropriados para lidar com estes fatores quando se deseja executar previsões. Também aprendemos sobre como essas previsões podem ser integradas a outras aplicações e sistemas, tirando proveito de tecnologias de bancos de dados e inteligência artificial para tornar esta integração mais simples e fácil.

Este trabalho é, por natureza, incompleto, uma vez que tem como objetivo providenciar uma plataforma para alavancar estudos sobre o tópico de acidentes de trânsito. Desta forma, existem diversos pontos de extensão e melhoria na aplicação, tanto de um ponto de vista de implementação quanto de um ponto de vista de estudos acadêmicos. O modelo implementado se limita à cidade de Porto Alegre e a dados meteorológicos e de acidentalidade, mas nada impede que um conjunto de dados maior seja utilizado e um novo modelo seja treinado para o mesmo. Múltiplos modelos também podem ser armazenados separadamente no banco de dados, portanto também é possível que mais de um modelo preditivo seja utilizado ao mesmo tempo, separados, por exemplo, por regiões.

Outro ponto de extensão é a integração com sistemas de previsão de tempo e a execução automática da previsão dos acidentes, permitindo que a ferramenta seja útil como um componente de *backend* de algum outro

sistema, sem interação alguma com o usuário. Por fim, também é possível utilizar a aplicação com um foco maior no seu uso comercial, permitindo que miríade de parâmetros envolvidos na construção dos algoritmos e do modelo sejam customizáveis pelo usuário, ou até mesmo por algum sistema automático que se ajusta dependendo das preferências do usuário.

Como mensagem final, esperamos que este trabalho possa ser utilizado como referência ou base por quaisquer outros estudos relacionados ao tópico de segurança no trânsito, que apenas se torna mais importante com o passar do tempo.

REFERÊNCIAS

ALJABAN, M. Analysis of car accidents causes in the usa. **Rochester Institute of Technology Scholar Works**, 2021.

ANTP, A. N. de T. P. **Sistema de Mobilidade Urbana - SIMOB**. 2018. Disponível na Internet: <http://www.antp.org.br/relatorios-a-partir-de-2014-nova-metodologia.html>.

BECK, K.; BEEDLE, M.; BENNEKUM, A. van; COCKBURN, A.; CUNNINGHAM, W.; FOWLER, M.; GRENNING, J.; HIGHSMITH, J.; HUNT, A.; JEFFRIES, R.; KERN, J.; MARICK, B.; MARTIN, R. C.; MELLOR, S.; SCHWABER, K.; SUTHERLAND, J.; THOMAS, D. **Manifesto for Agile Software Development**. 2001. Disponível na Internet: <http://www.agilemanifesto.org/>.

BOEING, G. Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. **Computers, Environment and Urban Systems**, v. 65, p. 126–139, 2017. ISSN 0198-9715. Disponível na Internet: <https://www.sciencedirect.com/science/article/pii/S0198971516303970>.

CRASHMAPUK. 2022. Disponível na Internet: <https://www.crashmap.co.uk/>.

DADOS Abertos POA. 2022. Disponível na Internet: <http://datapoa.com.br/>.

DL4J. 2022. Disponível na Internet: <https://deeplearning4j.konduit.ai/>.

FACT sheet No. 3 – Water in the atmosphere. 2012. Disponível na Internet: <https://web.archive.org/web/20120114162401/http://www.metoffice.gov.uk/media/pdf/4/1/No._03_-_Water_in_the_Atmosphere.pdf>.

FIELDING, R. T.; TAYLOR, R. N. **Architectural Styles and the Design of Network-Based Software Architectures**. Thesis (PhD) — University of California, Irvine, 2000. AAI9980887.

FOUNDATION, P. S. **Python**. 2021. Disponível na Internet: <https://www.python.org/>.

FOWLER, M.; BECK, K.; BRANT, J.; OPDYKE, W.; ROBERTS, D. **Refactoring: Improving the design of existing code**. Boston, MA: Addison Wesley, 1999.

HO, T. K. Random decision forests. **AT&T Bell Laboratories**, 1995.

ISRAEL, J. E.; MITCHELL, J. G.; STURGIS, H. E. Separating data from function in a distributed file system. Em: . [S.l.: s.n.], 1978.

KERAS. 2022. Disponível na Internet: <https://keras.io/>.

LOGíSTICA, O. N. de Transporte e. **Anuário Estatístico de Transportes 2010 – 2019**. 2019. Disponível na Internet: <https://ontl.epl.gov.br/wp-content/uploads/2021/02/Anuario-2010-2019-Completo.pdf>.

MARTINS, P.; ABBASI, M.; Sá, F. **A Study over NoSQL Performance**. [S.l.: s.n.], 2019. 603-611 p. ISBN 978-3-030-16180-4.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The Bulletin of Mathematical Biophysics**, v. 5, n. 4, p. 115–133, 1943.

MIGUEL, D. A. A. **ACIDENTES FERROVIÁRIOS NO BRASIL: ANÁLISE COMPARATIVA COM A UNIÃO EUROPEIA**. 2022. Disponível na Internet: <https://www.anpet.org.br/anais34/documentos/2020/Aspectos\%20Econ\%C3\%B4micos\%20Sociais\%20Pol\%C3\%ADticos\%20e\%20Ambientais\%20do\%20Transporte/Gest\%C3\%A3o\%20do\%20Transporte\%20Ferrovi\%C3\%A1rio\%20e\%20Hidrovi\%C3\%A1rio/4_220_AC.pdf>.

MONGODB, I. **Mongo DB**. 2021. Disponível na Internet: <https://www.mongodb.com/>.

MVC XEROX PARC 1978-79. 1970. Disponível na Internet: <https://folk.universitetetioslo.no/trygver/themes/mvc/mvc-index.html>.

OHNO, T.; BODEK, N. **Toyota production system: beyond large-scale production**. [S.l.]: Productivity press, 1998.

OPENLAYERS. 2022. Disponível na Internet: <https://openlayers.org/>.

OPENSTREETMAPS. 2022. Disponível na Internet: <https://www.openstreetmap.org/>.

PLURALSIGHT. **JavaScript**. 2022. Disponível na Internet: <https://www.javascript.com/>.

PORTAL INMET. 2022. Disponível na Internet: <https://portal.inmet.gov.br/>.

REN, H.; SONG, Y.; LIU, J.; HU, Y.; LEI, J. A deep learning approach to the prediction of short-term traffic accident risk. **ArXiv**, abs/1710.09543, 2017.

RFC-8259. 2017. Disponível na Internet: <https://datatracker.ietf.org/doc/html/rfc8259>.

SHAKEY. 1972. Disponível na Internet: <https://www.sri.com/hoi/shakey-the-robot/>.

SHI, X.; CHEN, Z.; WANG, H.; YEUNG, D.-Y.; WONG, W.-k.; WOO, W.-c. Convolutional lstm network: A machine learning approach for precipitation nowcasting. Em: **Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1**. Cambridge, MA, USA: MIT Press, 2015. (NIPS'15), p. 802–810.

SPRING.IO. **Spring Boot**. 2022. Disponível na Internet: <https://spring.io/projects/spring-boot>.

TENSORFLOW. 2022. Disponível na Internet: <https://www.tensorflow.org/>.

TIOBE. **TIOBE Index for July 2022**. 2022. Disponível na Internet: <https://www.tiobe.com/tiobe-index/>.

WANG, R.; YANG, Z.-M. Sql vs nosql: A performance comparison. University of Rochester, 2017.

WAZE. 2022. Disponível na Internet: <https://www.waze.com/live-map/>.

YUAN, Z.; ZHOU, X.; YANG, T. Hetero-convlstm: A deep learning approach to traffic accident prediction on heterogeneous spatio-temporal data. Em: **Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining**. New York, NY, USA: Association for Computing Machinery, 2018. (KDD '18), p. 984–992. ISBN 9781450355520. Disponível na Internet: <https://doi.org/10.1145/3219819.3219922>.