UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

LUCAS BENJAMIM CUNHA BANDEIRA

PREDIÇÃO DE TRAJETÓRIAS DE VEÍCULOS A PARTIR DE DADOS DE SENSORES DE TRÂNSITO

QUIXADÁ

2020

LUCAS BENJAMIM CUNHA BANDEIRA

PREDIÇÃO DE TRAJETÓRIAS DE VEÍCULOS A PARTIR DE DADOS DE SENSORES DE TRÂNSITO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Dr. Regis Pires Magalhães

Coorientadora: Prof. Ma. Lívia Almada Cruz

QUIXADÁ

2020

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

B166p Bandeira, Lucas Benjamim Cunha.
Predição de trajetórias de veículos a partir de dados de sensores de trânsito / Lucas Benjamim Cunha Bandeira. – 2020.
60 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Ciência da Computação, Quixadá, 2020.
Orientação: Prof. Dr. Regis Pires Magalhães.
Coorientação: Profa. Ma. Lívia Almada Cruz.

1. Trânsito. 2. Trajetória. 3. Detectores. 4. Sequências (Matemática). 5. Aprendizado do Computador. I. Título.

CDD 004

LUCAS BENJAMIM CUNHA BANDEIRA

PREDIÇÃO DE TRAJETÓRIAS DE VEÍCULOS A PARTIR DE DADOS DE SENSORES DE TRÂNSITO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Aprovada em: _____/_____/_____.

BANCA EXAMINADORA

Prof. Dr. Regis Pires Magalhães (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Ma. Lívia Almada Cruz (Coorientadora)
Universidade Federal do Ceará (UFC)

Prof. Dr. Paulo de Tarso Guerra
Universidade Federal do Ceará (UFC)

À minha mãe, irmã e amigos, por todo o suporte, confiança e amor a mim.

AGRADECIMENTOS

Agradeço a Deus. Sem ele nada seria possível.

Agradeço à minha mãe, Rita, que se dedicou ao máximo para que eu pudesse seguir o meu caminho na educação.

Agradeço à minha irmã, Camila, que sempre me apoiou e me aconselhou nos meus momentos de necessidade.

Agradeço à minha tia, Ambrosina, que é como uma segunda mãe pra mim.

Agradeço às minhas amigas, Dandara e Joycelenne, que me ajudaram a superar uma das fases mais assustadoras da minha vida.

Agradeço aos meus amigos do ensino médio, em especial à Cynthia, Eliada, Emanuela, José Lucas, Lohana, Luana, Marisa, Pablo, Thiago, Thuanny e Suênia, pela amizade e por todas as experiencias que juntas fazem parte da pessoa que sou hoje.

Agradeço à minha tia, Sandra, pelo seu apoio e confiança sobre mim.

Agradeço aos meu melhores amigos, Matheus, Michel e Victor, por sempre me ajudarem, me compreenderem e compartilharem suas experiências comigo.

Agradeço à minha prima, Bárbara, pela amizade e por todos os momentos que compartilhamos.

Agradeço à todos os meus amigos que conheci durante a graduação, em especial à Bárbara, Darliene, Dieinison, Joyce, Karine, Matheus Félix, Paulo Cardoso, Samuel Pinheiro, Tassiane e Ryan, pela amizade, pelos conselhos, e pelos ensinamentos.

Agradeço à todos os meus professores durante a graduação em especial a Lívia, Paulo de Tarso, Regis, Ticiana e Viviane, por serem ótimas pessoas e ótimos profissionais.

Agradeço ao meu orientador, Regis, e à minha coorientadora, Lívia, por me auxiliarem no desenvolvimento deste trabalho.

“Nós só podemos ver um pouco do futuro, mas o suficiente para perceber que há muito a fazer.”

(Alan Turing)

RESUMO

A locomoção em centros urbanos é considerado um dos maiores desafios relacionados a gestão de desenvolvimento urbano, devido ao alto índice de congestionamento. Perante isso, este trabalho propõe e avalia modelos multivariados de trajetórias que predizem um valor que representa um sensor de trânsito que representa o próximo ponto dessa trajetória. Os experimentos foram realizados sobre dados de trajetórias de veículos, geradas por sensores de trânsito da cidade de Fortaleza no Ceará. Os resultados mostraram que o algoritmo *XGBoosting* obteve a melhor performance para essa abordagem geral, já que apresentou os melhores resultados nos quesitos avaliados.

Palavras-chave: Predição de Trajetória. Sensores de Trânsito. Sequência. Aprendizado de Máquina.

ABSTRACT

The locomotion in urban centers is considered one of the biggest challenges related to urban develop management, due to the high rate of congestion. This work proposes and evaluates multivariate models of trajectories that predict a value that represents a traffic sensor in which it is the next point of this trajectory. The experiments were carried out on vehicle trajectory data, generated by traffic sensors in the city of Fortaleza, Ceará. The results showed that the *XGBoosting* algorithm obtained the best performance for this approach, since it had the best results in the evaluated items.

Keywords: Trajectory Prediction. Traffic Sensors. Sequence. Machine learning.

LISTA DE ILUSTRAÇÕES

Figura 1 – Exemplo de uma TSE 18
Figura 2 – Exemplo do funcionamento de um algoritmo de Aprendizado Supervisionado. 19
Figura 3 – Exemplo de um problema de predição de sequência 20
Figura 4 – Exemplo do funcionamento do algoritmo Decision Tree. 22
Figura 5 – Exemplo de representações de dados através de camadas aprendidas por um modelo preditor de dígitos. 24
Figura 6 – Exemplo de funcionamento do treino de uma rede neural. 25
Figura 7 – Exemplo de uma *Simple Recurrent Neural Network* (SimpleRNN) atravéz do tempo. 27
Figura 8 – Exemplo de uma *Long Short-Term Memory* (LSTM) através do tempo. . . . 28
Figura 9 – Exemplo de representação do atributo *dia_da_semana* no formato cíclico . . 32
Figura 10 – Exemplo de uma trajetória onde o modelo preditivo erra o sensor no qual o veículo passa. 34
Figura 11 – Visão geral da metodologia 41
Figura 12 – Exemplo de trajetória com falhas de captura 42
Figura 13 – Histograma das ocorrências dos *Id*s para as classes 43
Figura 14 – Exemplo do funcionamento da função *pad_sequences*. 44
Figura 15 – Exemplo de modelagem de conjunto de dados para os algoritmos de Aprendizado de Máquina. 48
Figura 16 – Arquitetura de Rede Neural que tem como entrada o conjunto de dados *S_DD_DDS* 51
Figura 17 – Importância dos atributos gerados pelo modelo *eXtreme Gradiente Boosting* (XGBoost) da biblioteca *xgboost* e conjunto de dados *S_SD* 54
Figura 18 – Comparação entre os tempos de treinamento e predição dos modelos 55

LISTA DE TABELAS

Tabela 1 – Acurácia dos melhores modelos para os conjuntos de dados e algoritmos . . 52
Tabela 2 – Precisão balanceada obtida para os modelos da Tabela 1 53
Tabela 3 – *F1* balanceado obtido para os modelos da Tabela 1 53
Tabela 4 – *Médias das Distâncias dos erros* em metros dos modelos que obtiveram melhor acurácia 55

LISTA DE QUADROS

Quadro 1 – Matriz de confusão mostrando as categorias referentes a classe gato 33
Quadro 2 – Comparação entre os trabalhos relacionados e o trabalho proposto 40
Quadro 3 – Exemplo de como os dados estão estruturados 42
Quadro 4 – Variação dos valores nos aquivos 46
Quadro 5 – Modelagens dos conjuntos de dados 47
Quadro 6 – Variação dos hiper-parâmetros para os modelos XGBoost gerados pelo *AutoML* 49
Quadro 7 – Algoritmos selecionados 50
Quadro 8 – Valores de entradas e saídas na camada *embedding* para cada atributo . . . 51

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
BLSTM	*Bidirectional Long Short-Term Memory*
DBN	*Dynamic Bayesian Network*
DC	*Dummy Classifier*
DRF	*Distributed Random Forest*
DT	*Decision Tree*
FN	Falso Negativo
FP	Falso Positivo
GBDT	*Gradient Boosted Decision Trees*
GPS	Sistema de Posicionamento Global
LSTM	*Long Short-Term Memory*
MDE	*Média das Distâncias dos Erros*
NB	*Naive Bayes*
PST	*Probabilistic Suffix Tree*
RF	*Random Forest*
RNN	*Recurrent Neural Network*
SimpleRNN	*Simple Recurrent Neural Network*
TSE	Trajetória de Sensores Externos
VN	Verdadeiro Negativo
VP	Verdadeiro Positivo
XGBoost	*eXtreme Gradiente Boosting*
XRT	*eXtremely Randomized Trees*

SUMÁRIO

1 INTRODUÇÃO 15
2 FUNDAMENTAÇÃO TEÓRICA 17
2.1 Sequências e Trajetórias 17
2.2 Aprendizado de Máquina 18
2.2.1 Aprendizado Supervisionado em comparação com Aprendizado Não Supervisionado 19
2.2.2 Problemas de Classificação 20
2.2.2.1 Predição de sequências *20*
2.3 Algoritmos de predição 21
2.3.1 Naive Bayes 21
2.3.2 Decision Tree 21
2.3.3 Random Forest 22
2.3.4 eXtreme Gradiente Boosting 23
2.4 Aprendizado Profundo 23
2.4.1 Redes Neurais Recorrentes 26
2.4.1.1 Simple Recurrent Neural Network *26*
2.4.1.2 Long Short-Term Memory *26*
2.4.1.3 Bidirectional Long Short-Term Memory *27*
2.4.2 Embedding Layer 28
2.4.3 Concat Layer 29
2.4.4 Dense Layer 29
2.5 Conjunto de Dados 29
2.5.1 Conjuntos de Treino, Teste e Validação 29
2.6 Engenharia de Atributos 30
2.6.1 Dados Temporais Discretizados 31
2.6.2 Dados Temporais cíclicos 31
2.7 Métricas de desempenho 32
2.7.1 Matriz de Confusão 33
2.7.1.1 Média das Distâncias dos Erros *33*
2.7.1.2 Acurácia *34*

2.7.1.3 *Precisão, Revocação e F1* *35*
3 **TRABALHOS RELACIONADOS** 37
3.1 ***Trajectory Prediction from a Mass of Sparse and Missing External Sensor Data*** 37
3.2 ***TPRED: A Spatio-Temporal Location Predictor Framework*** 37
3.3 ***MyWay: Location prediction via mobility profiling*** 38
3.4 ***City Scale Next Place Prediction from Sparse Data through Similar Strangers*** 39
3.5 **Análise comparativa dos trabalhos** 39
4 **EXPERIMENTOS E RESULTADOS** 41
4.1 **Coleta dos dados** 41
4.2 **Análise e Pré-processamento dos dados** 42
4.3 **Seleção das técnicas de predição e construção dos modelos preditivos** . . 46
4.3.1 ***Construção dos modelos de Aprendizado de Máquina*** 47
4.3.2 ***Construção das arquiteturas de redes neurais recorrentes*** 50
4.4 **Execução e Análise dos Modelos Preditivos** 52
5 **CONSIDERAÇÕES FINAIS** 57
REFERÊNCIAS 58
ANEXOS 60

1 INTRODUÇÃO

O mundo passa por um processo rápido de urbanização. Estudos preveem que em 2050 a maior parte do mundo será urbanizado e que a população urbana será maior que a população mundial em 2018 (ZHAO; TANG, 2018). Um dos principais desafios associados ao desenvolvimento urbano refere-se locomoção, pois um dos maiores problema de infraestrutura enfrentados por grandes cidades são os recorrentes congestionamentos no trânsito, isso ocorre pelo grande aumento de veículos nestes centros (BRETZKE, 2013). Por exemplo, segundo o IBGE (2018) a cidade de Fortaleza dobrou sua frota de veículos automotivos entre os anos de 2008 à 2018. Portanto, uma das melhores maneiras de contribuir para este quadro, é compreender as complexidades subjacentes aos comportamentos relacionados aos padrões de mobilidade urbana. Assim, isto se torna essencial para a tomada de decisões referentes ao planejamento urbano, à mobilidade sustentável, engenharia de transporte e saúde pública (BATTY *et al.*, 2012).

Atualmente já existem diversas aplicações que buscam interpretar os padrões de mobilidade urbana. Celulares e veículos rastreiam constantemente a posição de diversos usuários, tornando possível, através de algoritmos de predição, tanto prever suas próximas localizações, quanto fornecer recomendações oportunas, como predições de tráfego e assistência ao motorista (BUCHER, 2017). As tendências recentes na monitoração autônoma adicionam câmeras, radares, e dispositivos de medição a *laser* para vigilância e rastreamento de posição. A utilização das informações desses sensores, principalmente quando combinadas com mapas de ruas, possibilitam a predição da localização e rota de um veículo (BUCHER, 2017).

No atual trabalho são utilizado dados de sensores de trânsito. Estes sensores de trânsito são equipamentos fixados próximos às as ruas em vários pontos da cidade, que capturam a passagem de cada um dos diversos tipos de veículos, como automóveis, motocicletas, ônibus, dentre outros. Então, assumindo que cada captura contenha informação suficiente para detectar unicamente cada objeto seja, possível derivar a sua Trajetória de Sensores Externos (TSE) desse objeto, que é um conjunto de sensores ordenados pelo momento no qual eles capturaram as informações de que este determinado objeto os cruzaram.

No trabalho relacionado Cruz *et al.* (2019) também são utilizados dados de sensores de trânsito para realizar predições de trajetórias. Em Cruz *et al.* (2019) foram realizados experimentos que utilizaram modelos de Aprendizado Profundo que buscam realizar a predição do próximo sensor de transito da trajetória dada uma TSE.

O objetivo deste trabalho consiste na criação de modelos preditivos para estimar

Ids de sensores de trânsito, para prever trajetórias de veículos. Porém, diferente do trabalho de Cruz *et al.* (2019), este trabalho utiliza diversas técnicas de predição, como *Naive Bayes* (NB), *Decision Tree* (DT), *Random Forest* (RF), XGBoost, arquiteturas de redes neurais profundas como *SimpleRNN*, *LSTM* e *BLSTM*, além do uso de variados conjuntos de dados derivados de diferentes combinações dos atributos extraídos do conjunto de dados original.

Portanto, o objetivo principal deste trabalho é descobrir qual a melhor técnica de predição dentre os algoritmos selecionados, além de descobrir quais atributos foram mais significativos para as predições. Além disso, este trabalho traz como contribuição reduzir problemas com locomoção, tendo em vista que é possível compreender certos padrões de movimentação. Outra contribuição é a discussão acerca das técnicas utilizadas, levantando hipóteses dos motivos pelos quais algumas técnicas se saíram melhores que outras para solução do problema abordado.

Os próximos capítulos estão organizados da seguinte maneira: No Capítulo 2, será apresentada a fundamentação teórica, onde há um levantamento de conceitos sobre trajetórias, *Aprendizado de Máquina* e algumas de suas subáreas, além da definição de algumas métricas de avaliação e outros conceitos utilizados neste trabalho; no Capítulo 3, serão apresentados os trabalhos relacionados, com descrição e comparação de projetos e pesquisas que possuem aspectos similares com os propostos neste trabalho; no Capítulo, serão apresentados os procedimentos metodológicos, onde têm-se a descrição das atividades relacionadas ao desenvolvimento deste trabalho; no Capítulo 4, são mostrados os experimentos realizados com base na metodologia adotada, e seus respectivos resultados; no Capítulo 5, são mostradas as conclusões deste trabalho, bem como os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Os conceitos teóricos utilizados neste trabalho são apresentados neste capítulo.

2.1 Sequências e Trajetórias

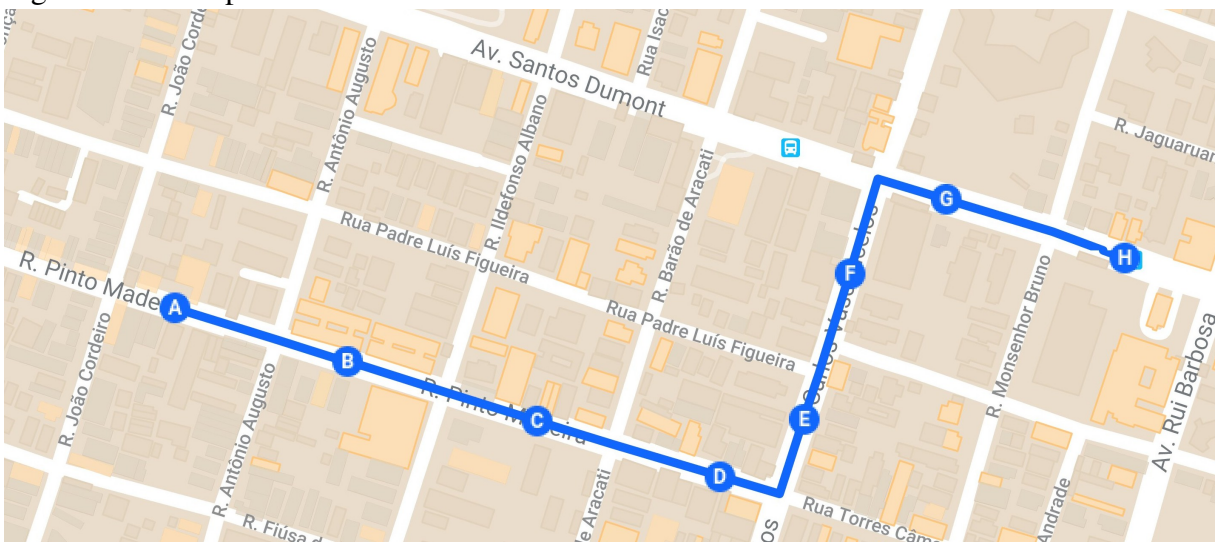
Uma sequência, de forma geral, é um conjunto de elementos onde pode-se especificar um elemento inicial x_1 e definir, a partir de algum critério, um método para obter os próximos elementos $(x_2, x_3, ..., x_n)$, onde, dado qualquer elemento (x_f) possa ser obtido o seu sucessor $(x_{f+1}|\{f \geq 1\})$, podendo assim definir uma determinada ordem aos elementos deste conjunto (BARTLE; SHERBERT, 2000).

Já uma trajetória pode ser definida como uma sequência de pontos espaçotemporais tendo o tempo como critério de ordenação. Cada elemento da sequência é formado por sua localização e o registro do momento referente ao instante em que o elemento foi gerado (ZHENG, 2015; YING *et al.*, 2011). Neste trabalho, cada ponto espaçotemporal das trajetórias serão compostas pela localização dos sensores de trânsito em que os veículos foram detectados e pelo momento em que estes sensores detectaram a passagem destes veículos.

Neste trabalho, as trajetórias são chamadas de TSE, uma TSE pode ser definida como uma sequência de capturas $(c_1, c_2, ..., c_n)$ realizadas por sensores de trânsito através da detecção de um objeto O. Na sequência, através do atributo temporal é possível definir qual é a primeira captura e suas respectivas capturas sucessoras. No entanto, o objetivo deste trabalho é: dado os valores de uma TSE de um determinado objeto O, que são os valores que representam os sensores de trânsito (*ID*s) $(s_i = \{i_1, i_2, i_3, ..., i_n\})$ que pertencem ao conjunto de todos os sensores S, e os valores que representam os atributos temporais $(s_t = \{t_1, t_2, t_3, ..., t_n\})$, seja possível predizer o próximo *Id* (i_{n+1}) da sequencia s_i onde $(i_{n+1}) \in S$.

Na Figura 1 é apresentado um exemplo de uma TSE. Os pontos de A a H representam as posições dos sensores de trânsitos que pertencem a esta trajetória. Primeiramente o sensor A detecta a passagem do veículo que realizou essa trajetória e gera as informações da captura, momentos após, o sensor B realiza o mesmo processo, e assim ocorre sucessivamente até completar a TSE. Então, dado uma TSE como a da Figura 1 estima-se realizar a correta predição do próximo sensor de trânsito desta sequência com intuito de descobrirmos a próxima localização do objeto, escolhendo um detre todos os possíveis sensores de trânsito.

Figura 1 – Exemplo de uma TSE



Fonte: Elaborado pelo autor

2.2 Aprendizado de Máquina

Aprendizado de Máquina é um ramo da *Inteligência Artificial* que permite programar computadores a fim de gerar um outro programa chamado *modelo*. A execução de um algoritmo de aprendizado de máquina constrói um *modelo* preditivo baseando-se em um conjunto de dados de treinamento ou experiências prévias, para que o modelo se adéque melhor ao problema a ser resolvido sem a necessidade de uma programação explícita (GÉRON, 2017; BELL, 2014). Todos os algoritmos de Aprendizado de Máquina consistem em encontrar automaticamente alguma representação dos dados que os tornem mais úteis para uma determinada tarefa (CHOLLET, 2017). Depois que os modelos são gerados, estão aptos a receberem como entrada novas instâncias de dados para que possam ser realizadas predições no futuro (*modelos preditivos*), ou obter conhecimento sobre novos dados (*modelos descritivos*). (RUSSELL; NORVIG, 2016).

Segundo Alpeydin (2010), Aprendizado de Máquina se faz necessário quando escrever diretamente um programa de computador para resolver um dado problema é muito difícil, como quando o algoritmo necessita de muito refinamento manual ou há um conjunto muito grande de regras; ou quando o problema aborda uma diversidade ou quantidade muito grande de dados, como por exemplo em um problema de reconhecimento de fala, nessas situações os algoritmos de Aprendizado de Máquina conseguem facilmente se adaptar ao problema.

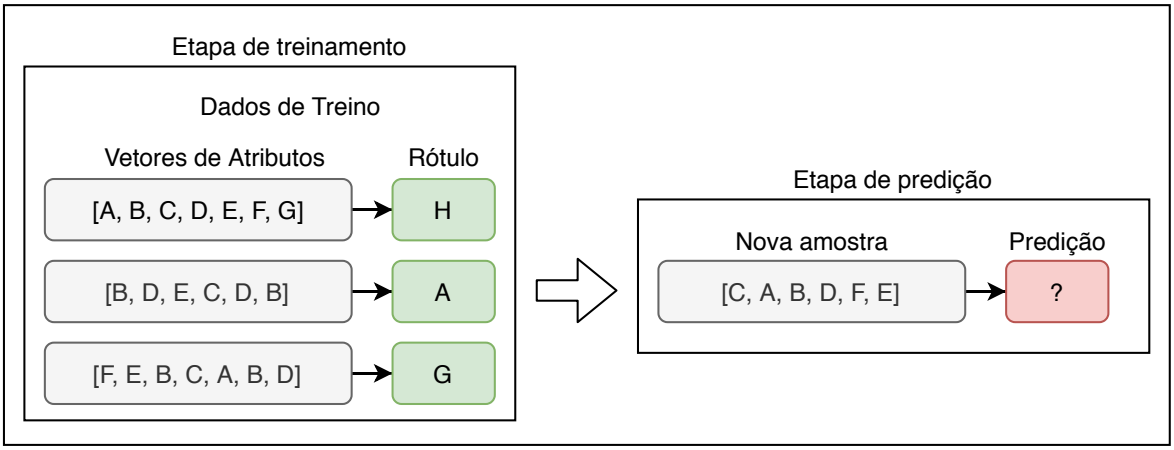
A solução do problema deste trabalho é predizer qual será o próximo sensor de trânsito no qual um determinado objeto irá cruzar baseado nas sequências que representam as TSEs. Então, desenvolver um programa de forma tradicional pode ser altamente complexo,

pois deve ser levado em consideração cada variação das TSEs, portanto as soluções utilizando aprendizado de máquina tendem a ser promissoras.

2.2.1 Aprendizado Supervisionado em comparação com Aprendizado Não Supervisionado

No Aprendizado Supervisionado, os dados de treinamento são compostos por uma coleção de amostras, onde cada amostra possui um par de informações que são um *vetor de atributos* e um *rótulo*. O *vetor de atributos* é um conjunto de características que descrevem à amostra. Já o *rótulo* é um valor que corresponde ao resultado desejado do *vetor de atributos*. Portanto, o objetivo do algoritmo de aprendizado supervisionado é observar os pares de *atributos* e *rótulos* para produzir um modelo que tem *vetores de atributos* como entrada para gerar como saída informações que permitem deduzir quais os *rótulos* para estes *vetores de atributos* (RUSSELL; NORVIG, 2016; BURKOV, 2019).

Figura 2 – Exemplo do funcionamento de um algoritmo de Aprendizado Supervisionado.



Fonte: Elaborado pelo autor

Na Figura 2 é apresentado um exemplo de um problema para identificar o próximo elemento de uma sequência. Seus *atributos* incluiriam sequências e seus *rótulos* seriam o próximo elemento desta sequência. Dessa forma, o algoritmo de Aprendizado de Máquina cria o modelo observando o conjunto de dados de treinamento, tornando possível para o modelo realizar predições de *rótulos* para novas sequências (RASCHKA; MIRJALILI, 2017).

No Aprendizado Não Supervisionado o conjunto de dados não possui *rótulos* (GÉRON, 2017). O algoritmo de Aprendizado Não Supervisionado gera um modelo descritivo observando padrões apenas no *vetor de atributos*. O modelo gerado receberá como entrada um *vetor de atributos* e o transformará em um outro vetor ou em valores que serão utilizados para resolver um problema prático. Por exemplo: em um problema de *Clusterização*, o modelo

retorna um valor que representa um id de um *cluster*, que é o agrupamento de entradas similares; ou em um problema de redução de dimensionalidade, onde o modelo gera um novo *vetor de atributos* que tem menos *atributos* que o *vetor de atributos* dado como entrada; ou em um problema de detecção de anomalia, o modelo produz um valor que indica o quão diferente a entrada é comparada com exemplos *típicos* do conjunto de dados (RASCHKA; MIRJALILI, 2017; BURKOV, 2019).

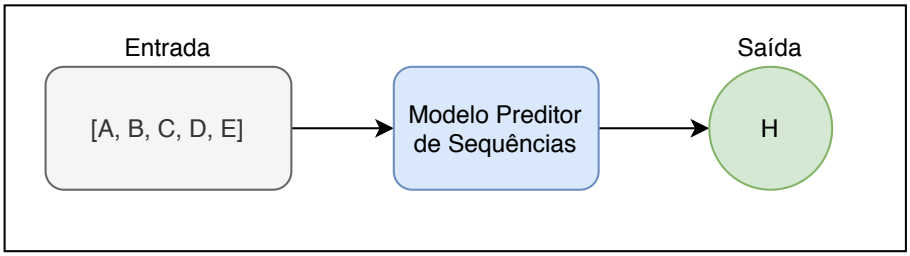
2.2.2 Problemas de Classificação

Problemas de Classificação é uma subcategoria de Aprendizado Supervisionado. Em um problema de classificação, o modelo preditivo tenta predizer uma *classe* dentre um conjunto de classes possíveis. Os valores dos rótulos em um problema de classificação possuem um valor discreto, isso significa que os valores estão compreendidos em um conjunto qualitativo, neste caso, os valores dos rótulos utilizados na etapa de treinamento (RASCHKA; MIRJALILI, 2017). Por exemplo, no problema da imagem 2, o valor que pode ser atribuído à um rótulo em uma predição compreende apenas os valores *H*, *A* ou *G*.

2.2.2.1 Predição de sequências

Segundo Sun e Giles (2001), predições de sequências consiste em predizer elementos de uma sequência com base nos elementos anteriores. Na figura 3 temos um exemplo de um problema de predição de sequências. Considere que o modelo preditor recebe como entrada uma sequência de valores de *A* a *E* que representam os sensores cruzados por um objeto *O*. Neste cenário, a solução ideal para o problema de predição de sequências é obter *H*, tarefa designada ao modelo preditor.

Figura 3 – Exemplo de um problema de predição de sequência



Fonte: Elaborado pelo autor

O modelo é treinado com um conjunto de sequências de treinamento, como sequencia de objetos estruturados, números ou palavras. Uma vez treinado, o modelo é utilizado para

executar predições de sequências. Essa tarefa possui várias aplicações cabíveis, como sugestões de busca de páginas da web, recomendação de produtos para o consumidor, previsão do tempo e previsão do mercado de ações. (GUENICHE *et al.*, 2015; LAIRD; SAUL, 1994)

2.3 Algoritmos de predição

Nesta seção, são descritos alguns algoritmos que são muito conhecidos e capazes de solucionar problemas de classificação (BURKOV, 2019).

2.3.1 Naive Bayes

Os classificadores NB são classificadores lineares conhecidos por serem simples. O modelo probabilístico de classificadores NB é baseado no teorema de Bayes, e o nome *Naive*, que significa ingênuo, vem da suposição de que os atributos de um conjunto de dados são mutuamente independentes. Por serem relativamente robustos, fáceis de implementar, rápidos e precisos, os classificadores NB são usados em muitos campos diferentes. Alguns exemplos incluem o diagnóstico de doenças e a tomada de decisões sobre os processos de tratamento; a classificação de sequências de RNA em estudos taxonômicos; e a filtragem de spam em clientes de *e-mail* (RASCHKA, 2014).

O teorema de Bayes calcula a probabilidade de um evento x_i ocorrer dado que outro evento ω_i ocorreu. No contexto de um problema de classificação, o teorema de Bayes pode ser interpretado como a probabilidade de uma instância x_i pertencer a uma determinada classe ω_i (RUSSELL; NORVIG, 2016). A Equação 2.1 representa o modelo probabilístico de Bayes.

$$P(\omega_i|x_i) = \frac{P(x_i|\omega_i) \times P(\omega_i)}{P(x_i)} \tag{2.1}$$

O algoritmo NB calcula a probabilidade de uma amostra x_i pertencer a uma classe, através do produto entre a probabilidade da classe w_i ($P(w_i)$) e a probabilidade da amostra x_i ser da classe ω_i dado que ela pertença a classe ω_i ($P(x_i|\omega_i)$)

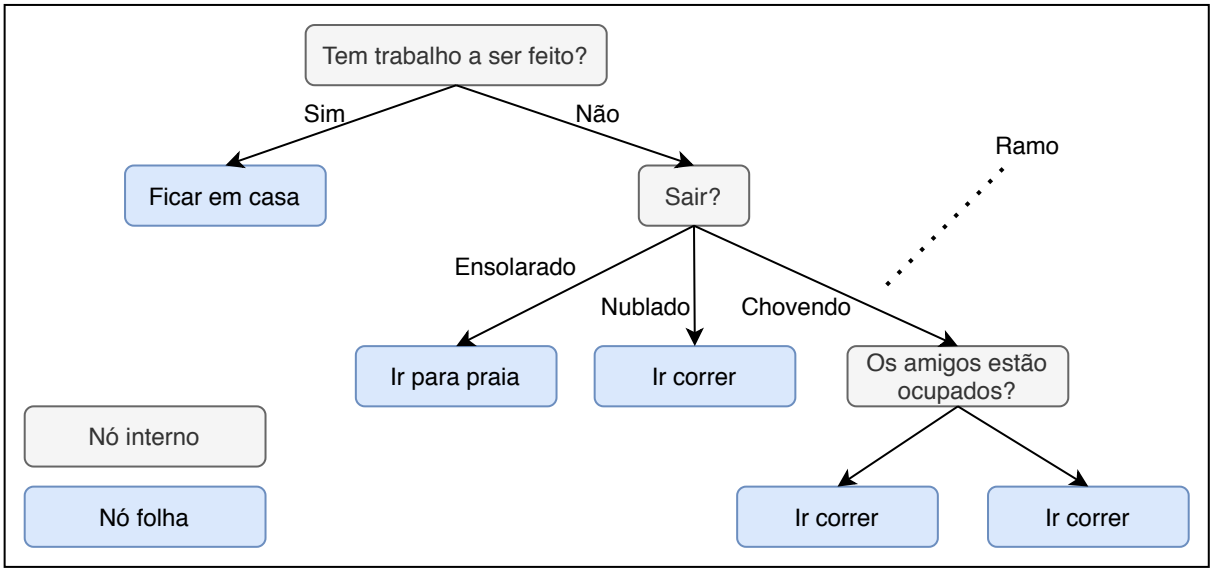
2.3.2 Decision Tree

Classificadores DT podem ser uma boa escolha caso levemos em consideração a interpretabilidade dos dados (RASCHKA; MIRJALILI, 2017; GÉRON, 2017). Um modelo DT

aprende uma série de perguntas para inferir sobre qual classe as amostras pertencem baseado nos atributos do conjunto de dados de treino (RASCHKA; MIRJALILI, 2017).

Através do exemplo ilustrado na Figura 4, podemos entender como um modelo DT pode se comportar. Os nós internos são os atributos do conjunto de dados, os *ramos* são as perguntas aprendidas pelo modelo e os *nós folha* são os *rótulos*. O algoritmo DT inicia-se no nó raiz e vai dividindo as classes dentre os nós através dos atributos que possuem maior ganho de informação através de uma função objetiva, que busca otimizar o aprendizado do algoritmo, e então, o algoritmo toma decisões a partir do nó raiz da arvore até chegar em um nó folha, que representa o rótulo predito pelo modelo. Dessa maneira o modelo DT tenta compreender os padrões do conjuntos de dados construindo essas hierarquias de perguntas. (RASCHKA; MIRJALILI, 2017).

Figura 4 – Exemplo do funcionamento do algoritmo Decision Tree.



Fonte: Elaborado pelo autor

O mesmo conceito pode ser aplicado para dados reais, como no problema de predição de diabetes. Por exemplo, é possível definir um limiar para o atributo que refere-se à glicose no sangue e fazer uma pergunta binária: "A glicose no sangue é $\geq 100_{mg}/dl$?".

2.3.3 Random Forest

Segundo Géron (2017), quando predições são baseadas em um grupo de modelos, os resultados normalmente são melhores do que o resultado do melhor modelo. Uma técnica que utiliza um conjunto de modelos é chamado de *Ensemble Learning*. Por exemplo, treinar um

grupo de modelos DT geraria um *Ensemble* de DT. O RF é um exemplo de *Ensemble Learning*, pois a essência do algoritmo é utilizar múltiplas predições de modelos de DT através de um método *Bagging*.

O método *Bagging* é definido como o uso do mesmo algoritmo para geração de diversos modelos que são treinados não sequencialmente em amostragens aleatórias do conjunto de dados de treino, realizando as predições dada a classe mais provável ou popular dentre todos os preditores (GÉRON, 2017). Dessa forma, o RF é capaz de construir modelos mais robustos que tem um melhor desempenho generalizado e é menos suscetível a *Overfitting*[1] (RASCHKA; MIRJALILI, 2017).

2.3.4 eXtreme Gradiente Boosting

O *Gradient Boosted Decision Trees* (GBDT) é um algoritmo que, como o RF, é um modelo *ensemble*, porém utiliza um método *Boosting*. O método *Boosting* combina modelos de performance ordinária para construir um modelo preditivo mais eficaz. A principal ideia dos métodos *Boosting* é treinar os modelos sequencialmente, onde cada modelo tenta corrigir os seus predecessor. Para alcançar este objetivo, existem diversos métodos *Boosting*, mas um dos mais populares é o *Gradient Boosting* (RUSSELL; NORVIG, 2016).

O modelo GBDT utiliza o método *Gradient Boosting*. Dessa forma, suas predições são baseadas em estatísticas de gradientes de primeira e segunda ordem para cada nó folha das árvores dos modelos (CHEN; GUESTRIN, 2016). O XGBoost é uma biblioteca que foi desenvolvida para ser mais rápida e obter uma melhor performance do que o GBDT, tornando-o um algoritmo mais robusto e eficaz que consegue lidar mais facilmente com uma grande variedade de dados (CHEN; GUESTRIN, 2016).

2.4 Aprendizado Profundo

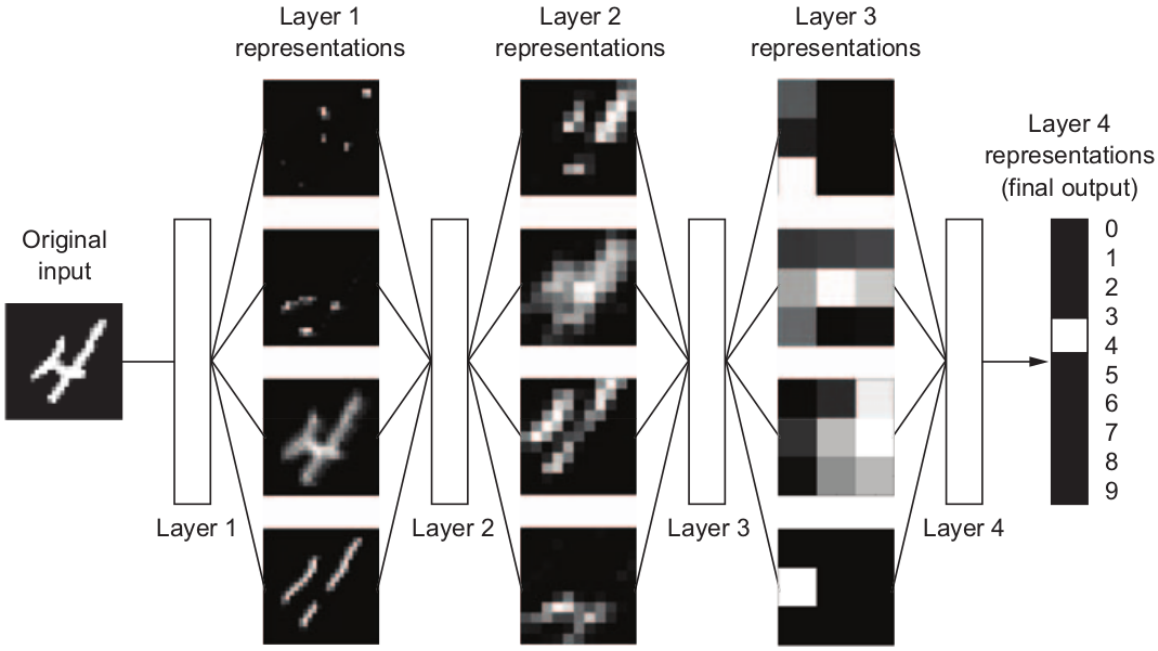
Aprendizado Profundo é uma subárea do Aprendizado de Máquina, que utiliza uma nova abordagem a respeito da forma de encontrar representações de aprendizado a partir dos dados. O Aprendizado profundo combina diversas camadas sucessivas com processamento não linear com objetivo de extrair representações significativas de dados e transformá-las em modelos preditivos (CHOLLET, 2017). Aprendizado Profundo tecnicamente é uma forma do modelo

[1] O Overfitting ocorre quando um modelo se adapta demais para um conjunto de dados e só consegue bons resultados para ele (GÉRON, 2017).

de aprendizado aprender as representações dos dados através de múltiplas etapas. Essas etapas pode ser representadas pelas camadas, onde normalmente, essas camadas aprendem através de modelos chamados de *redes neurais*. A quantidade de camadas está diretamente relacionada com a profundidade da rede, quanto maior o número de camadas mais profunda a rede neural. (CHOLLET, 2017).

A Figura 5 ilustra um exemplo do funcionamento de uma rede neural que aprende representações dos dados para um modelo que reconhece dígitos em imagens. Como podemos observar, a rede transforma a imagem do dígito em representações mais diferentes da imagem original, no entanto, cada vez mais informativa para obtenção do resultado final. Uma *rede neural* pode ser imaginado como uma operação de destilação de informações de vários estágios, onde as informações passam por sucessivos filtros e saem cada vez mais purificada.

Figura 5 – Exemplo de representações de dados através de camadas aprendidas por um modelo preditor de dígitos.

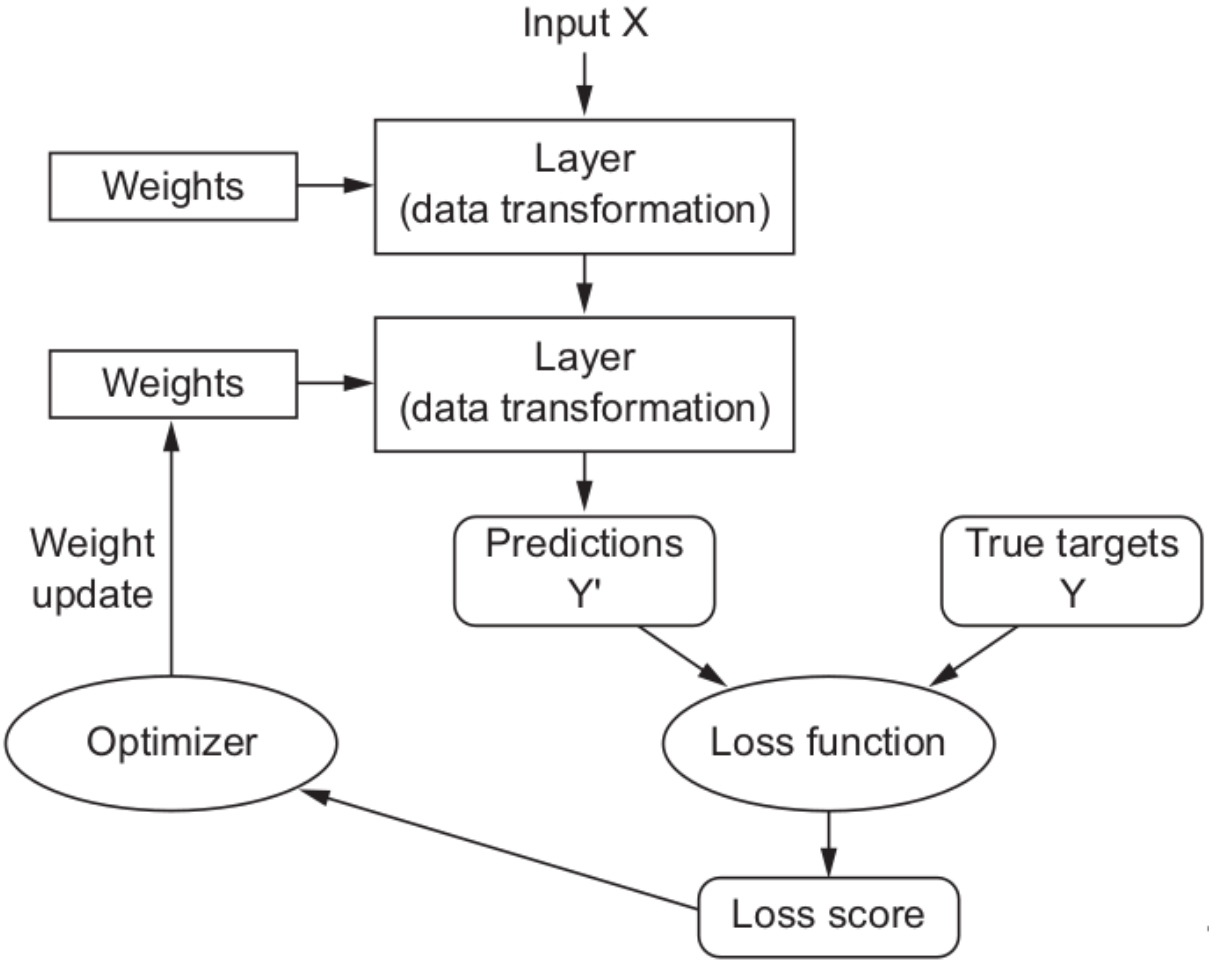


Fonte: Chollet (2017)

O aprendizado realizado pelos algoritmos de Aprendizado Profundo são feitos de forma hierárquica, onde as camadas iniciais aprendem características simples, enquanto as ultimas camadas aprendem características mais complexas formadas pela combinação de informações gerada pela camada anterior (BEZERRA, 2016). Tudo que uma camada faz com seus dados de entradas são guardados no que podemos chamar de pesos da camada, que basicamente é uma porção de números. Então, quando o algoritmo aprende sobres os dados,

significa que ele está tentando encontrar os valores corretos para os pesos de cada camada da rede (CHOLLET, 2017). Então, para isso, é necessário medir o quão errado o modelo prediz comparado com o que é esperado. Portanto, é utilizada uma *função de perda* nas predições para calcular uma métrica chamada *perda*, usada para compreender o quão bem a rede está atuando. Dessa forma, a rede utiliza as informações de *perda* para ajustar os valores dos pesos através de um otimizador, para que o erro seja reduzido no decorrer das iterações (CHOLLET, 2017). A Figura 6 ilustra o funcionamento mencionado acima. Primeiramente é dada a entrada X à rede, em seguida essa entrada é transformada pelas duas camadas, depois a rede realiza uma predição e à compara com o dado real, através da *função de perda* a *perda* é calculada, para que o otimizador a use para ajustar os valores dos pesos para a próxima época.

Figura 6 – Exemplo de funcionamento do treino de uma rede neural.



Fonte: Chollet (2017)

2.4.1 Redes Neurais Recorrentes

As Redes Neurais Recorrentes (*Recurrent Neural Network* (RNN)) são normalmente utilizadas para o processamento de sequências. Elas processam as sequências iterando através de ciclos na rede utilizando os dados sequenciais e mantendo um controle sobre um *estado*. Esse *estado* contém informações dos dados já processados, funcionando como um tipo de memória que permite a rede aprender dependências de longo prazo (CHOLLET, 2017). Note que em exemplos de modelos que precisam predizer qual a próxima palavra em uma frase ou, como no caso deste trabalho, predizer qual o próximo sensor de trânsito que um veículo irá cruzar, é importante conhecer a sequência de elementos anteriores para realizar a predição, por isso o uso de estado se torna essencial (BEZERRA, 2016).

2.4.1.1 Simple Recurrent Neural Network

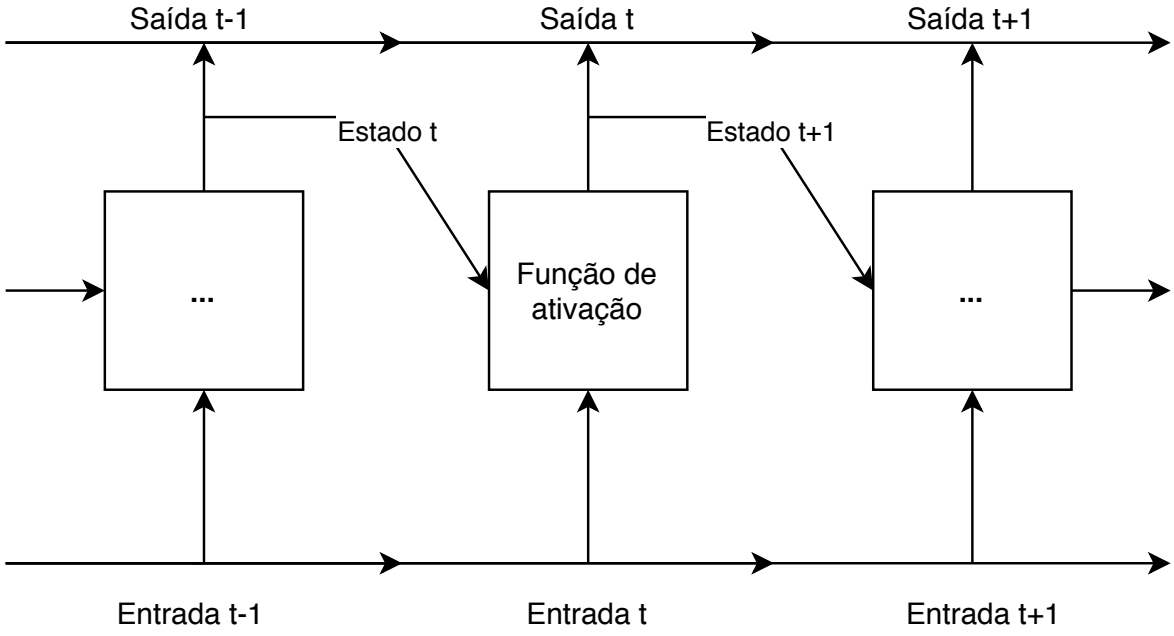
A SimpleRNN é a implementação mais básica de uma RNN, ela itera sobre os dados considerando que cada entrada é um *passo no tempo*, portanto para cada passo no tempo *t* a predição é feita levando em consideração o *estado t* e a entrada *t*, depois, o *estado* do próximo passo (t+1) tem seu valor modificado para que seja correspondente ao resultado obtido no passo anterior (*t*). Isso significa que o comportamento de uma RNN pode ser alterado dependendo das informações processadas no decorrer de suas iterações (CHOLLET, 2017). A Figura 7 mostra como o processo descrito é realizado no decorrer dos passos de tempo.

Segundo Chollet (2017) a SimpleRNN é muito simples para uso em soluções de problemas do mundo real, pois teoricamente ela deveria ser capaz de aprender informações sobre entradas vistas muitos passos de tempo anteriores, porém, na prática, é impossível aprender tais dependências de longo prazo.

2.4.1.2 Long Short-Term Memory

Como mencionado anteriormente, a SimpleRNN não consegue aprender dependências a longo prazo, então, para evitarmos isso, podemos utilizar a LSTM que é uma de suas variações. A LSTM é uma rede que consegue *lembrar-se* de valores por uma quantidade arbitrária de intervalos, salvando as informações para mais tarde, evitando que elementos mais antigos desapareçam gradualmente durante o processamento (CHOLLET, 2017). Para isso, uma rede LSTM adiciona um *fluxo de dados* que transmite informações através dos passos de tempo. Essas

Figura 7 – Exemplo de uma SimpleRNN atravéz do tempo.



Fonte: Elaborado pelo autor

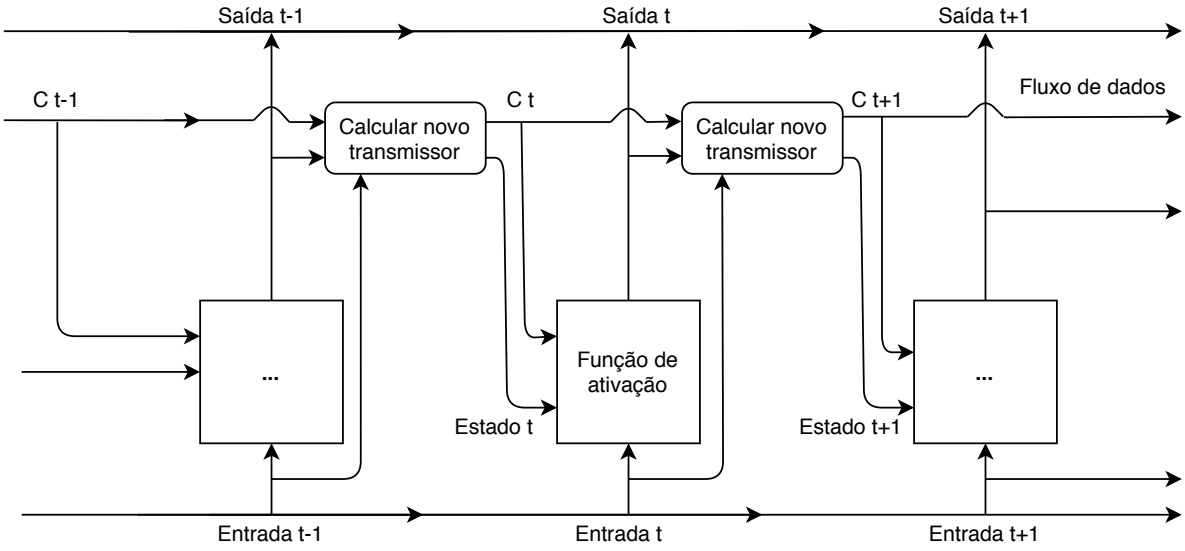
informações serão combinadas com a entrada e com o estado, que por consequência afetarão os estados e os resultados nos próximos passos de tempo (CHOLLET, 2017).

As informações retidas no fluxo de dados são manipulados através de um conjunto de transformações derivadas das informações do resultado do último estado e da entrada no passo de tempo atual. Dessa forma são selecionadas as informações que devem permanecer ou não neste fluxo no tempo atual (CHOLLET, 2017). A Figura 8 apresenta a rede LSTM através dos passos no tempo. No tempo t o modelo recebe a entrada t, o estado t derivado da entrada $t-1$ e os dados do fluxo de dados referentes ao tempo t. Depois disso a saída é calculada e utilizada juntamente com os valores do passo anterior do fluxo de dados e a entrada no tempo t; para gerar os novos valores no fluxo de dados, e assim tornar as informações disponíveis para o próximo passo no tempo.

2.4.1.3 Bidirectional Long Short-Term Memory

Uma *Bidirectional Long Short-Term Memory* (BLSTM) é uma variação da LSTM que pode oferecer melhores performances do que uma LSTM comum para certas tarefas. A BLSTM consiste em usar duas redes LSTM comuns, onde uma processa as sequências da entrada em ordem cronológica e a outra em ordem inversamente cronológica, e então mesclando suas representações. Dessa maneira, uma BLSTM pode capturar padrões perdidos por uma rede LSTM comum (CHOLLET, 2017).

Figura 8 – Exemplo de uma LSTM através do tempo.



Fonte: Elaborado pelo autor

2.4.2 Embedding Layer

No aprendizado profundo, os dados de entrada, normalmente, são representados usando codificação *one-hot* ou *word embeddings*. Levando para o contexto do trabalho atual, a codificação *one-hot* converte as entradas em uma matriz binária, onde as linhas representam os sensores e as colunas denotam as trajetórias. O valor inteiro um (1) indica o sensor usado em uma trajetória, enquanto todas as demais posições dessa linha da matriz são preenchidas com zero (0). A dimensão da matriz é igual ao número de sensores presentes em todas as trajetórias. Desta forma, *one-hot* representa as trajetórias de forma esparsa e desconsidera a ordenação e o relacionamento entre os sensores na trajetória (GRZEÇA *et al.*, 2020).

A codificação *word embedding* é comumente usada em problemas que utilizam texto, por isso seu nome. A *Embedding Layer* é uma camada em uma rede neural que realiza a codificação *word embedding*. Ela consegue aprender automaticamente, a partir dos dados de entrada, as relações sintáticas e semânticas entre as informações e seu contexto. Dessa forma as informações que frequentemente ocorrem em contextos similares são dispostas umas próximas das outras, construindo assim uma representação vetorial densa e de baixa dimensionalidade (CHOLLET, 2017).

No cenário do problema deste trabalho, o uso das *Embedding Layers* são promissoras, pois com a representação *one-hot* a dimensionalidade da entrada ficaria enorme mesmo utilizando a menor das situações. Pois, levando em consideração que será utilizado uma quantidade mínima de capturas para realizar as predições, e que para cada captura seria necessário fazer a

representação *one-hot* do sensor, então, uma matriz de dimensionalidade de aproximadamente 190 X 1 milhão seria necessário para cada captura. Isso teria uma complexidade muito alta para execução do algoritmo. Já com a *word embedding* teríamos dimensionalidades muito menores e com o bônus do significado da ordem não ser perdida.

2.4.3 Concat Layer

A *Concat Layer* é uma camada usada quando a rede possui entradas múltiplas, como sensores de trânsito e informações temporais, e em algum ponto os ramos das entradas precisam ser unidas. A *Concat Layer* concatena as listas de pesos de cada entrada e retorna uma única lista com todos os pesos (CHOLLET *et al.*, 2015).

2.4.4 Dense Layer

Uma *Dense Layer* é uma camada em uma rede neural que está totalmente conectada com a camada anterior, isso significa que cada sinal produzido na camada anterior é dado como entrada para a *Dense Layer* que permite o aprendizado de representações a partir de todas as combinações geradas na camada anterior (RUSSELL; NORVIG, 2016).

2.5 Conjunto de Dados

Um Conjunto de Dados pode ser visto como uma coleção de dados de objetos. Os dados de objetos são descritos por vários atributos que representam as características básicas destes objetos, como por exemplo, a massa de um objeto físico ou o horário no qual um evento ocorreu, variando de objeto para objeto (TAN *et al.*, 2006).

2.5.1 Conjuntos de Treino, Teste e Validação

A única maneira da saber o quão bem um modelo irá performar genericamente sobre novos casos é testá-lo em novos casos. Para isso, dividir o conjunto de dados em um *conjunto de dados de treino* e um *conjunto de dados de teste* pode ser uma solução. Como os nomes sugerem, o conjunto de treino será utilizado para treinar o modelo e o conjunto de teste para testá-lo. Dessa forma, é possível avaliar a performance do modelo para dados nunca visto antes, ou seja, dados que não foram utilizados na etapa de treinamento (GÉRON, 2017). Se o modelo tem uma performance alta para o conjunto de dados de treino mas para o conjunto de dados de

teste uma performance baixa, significa que o modelo está tendo o que chamamos de *Overfitting* no conjunto de dados de treino (GÉRON, 2017).

Dessa maneira, podemos criar vários modelos utilizando variados *hiper-parâmetros*[2] para que possamos encontra um que melhor se ajuste para o conjunto de dados de teste, mas isto pode ser um problema, pois adaptar o modelo para ter melhores resultados para um conjunto de dados, não o torna genérico. Portanto, os dados devem ser divididos em mais um subconjunto, o conjunto de dados de validação. Assim, podemos treinar múltiplos modelos com o conjunto de treino, depois, escolher o modelo que melhor se ajusta no conjunto de dados de validação e, por fim, realizar um teste final com o modelo escolhido utilizando o conjunto de dados de teste para obter uma estimativa da performance do modelo para dados generalizados (GÉRON, 2017; TAN *et al.*, 2006).

2.6 Engenharia de Atributos

Segundo (GÉRON, 2017), a Engenharia de Atributos é uma etapa indispensável para um projeto de Aprendizado de Máquina, pois é nessa etapa que selecionamos o conjunto de atributos que iremos utilizar para realizar o treinamento do modelo. O desempenho de um modelo de Aprendizado de Máquina é proporcional à relevância dos atributos utilizados. Portanto a Engenharia de Atributos engloba:

- Seleção de atributos: seleciona os atributos mais relevantes dentre as atributos existentes.
- Criação de atributos: produção de novos atributos combinando as informações de outros atributos ou de novos dados, pois é possível que haja relevantes padrões ocultos que não podem ser utilizados na forma original (GRUS, 2019).

Todas das decisões tomadas na Engenharia de Atributos são aplicadas na etapa de pré-processamento. O *Pandas*[3] provavelmente é a ferramenta mais usada em ambientes *python*[4] para limpar, transformar, manipular e trabalhar com dados (GRUS, 2019).

[2] São parâmetros do algoritmo de aprendizado de máquina que são definidos antes do treinamento e permanecem constantes durante o treinamento, ou seja, não são afetados pelo algoritmo de aprendizado de máquina(GÉRON, 2017)

[3] O Pandas é uma biblioteca de código aberto, licenciada pela *Berkeley Software Distribution*, que fornece estruturas de dados de alto desempenho e fáceis de usar além de ferramentas de análise de dados para a linguagem de programação *Python* (TEAM, 2020; MCKINNEY, 2010).

[4] Python é uma linguagem de programação interpretada, orientada a objetos e de alto nível, com semântica dinâmica (ROSSUM; DRAKE, 2009).

2.6.1 Dados Temporais Discretizados

Dados de series temporais, frequentemente, são representadas no formato de data, como 1962/07/26 14:30:22, porém não é possível extrair muita informação a partir deste formato, exceto sobre sua ordem cronológica (BESCOND, 2020). Discretizar dados reais em um pequeno número de valores inteiros é frequentemente necessário para construção de modelos de Aprendizado de Máquina, pois torna-se muito útil para a descoberta de padrões significativos (DIMITROVA *et al.*, 2010; MOSKOVITCH; SHAHAR, 2015).

A discretização temporal refere-se ao processo de discretização dos valores de series temporais transformando estes valores em um conjunto números simbólico que representam intervalos de tempo (MOSKOVITCH; SHAHAR, 2015). Imagine que é desejado discretizar os horários do dia de um conjunto de dados que contenha dados de series temporais, então podemos, por exemplo, repartir todos os dias em 24 partes contendo uma hora cada, com o intuito de que cada parte seja representada por um valor inteiro, como por exemplo o valor 0 que representa os horários que vão das 00:00 as 01:00 e assim sucessivamente.

2.6.2 Dados Temporais cíclicos

Horas do dia, dias da semana e meses do ano são exemplos de atributos cíclicos que podem ser extraídos de dados de serie temporal, então transformar estes atributos em uma representação que preserve a essência de ciclo, pode elevar sua relevância significativamente (KALEKO, 2017). No exemplo da discretização dos dados na subseção anterior, o valor 23 representaria o horário das 23:00 horas até as 00:00 e o valor 0 representaria o horário das 00:00 até as 01:00, então a ideia é fazer com que o valor 23 seja representado como um valor próximo ao valor 0.

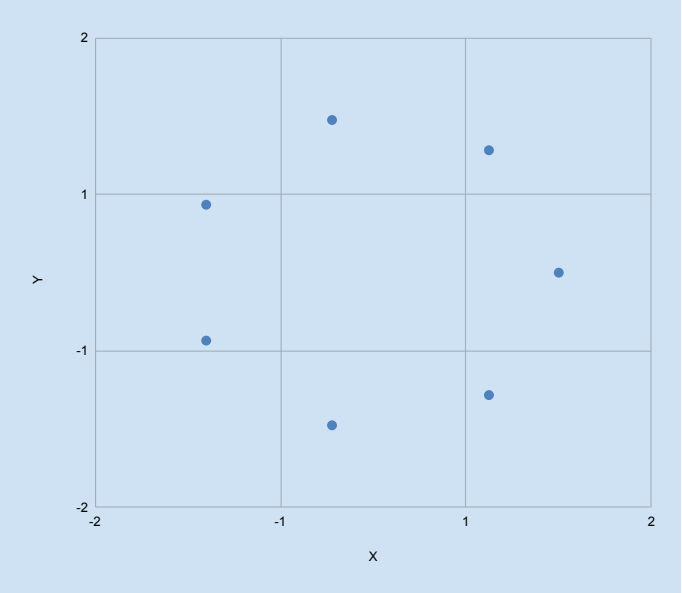
Então, para transformar um atributo cíclico em um atributo que mantenha o aspecto de ciclo, é necessário que o seu valor discretizado seja convertido em coordenadas x e y, associando valores diferentes para cada momento (BESCOND, 2020). As Equações 2.2 e 2.3 mostram como os valores x e y foram obtidos, onde TD_i representa o tempo discretizado atual e TD_{max} representa o tempo discretizado de maior valor.

$$x = \sin\left(TD_i \times \frac{2 \times \pi}{TD_{max}}\right) \tag{2.2}$$

$$y = \cos\left(TD_i \times \frac{2 \times \pi}{TD_{max}}\right) \tag{2.3}$$

A Figura 9 apresenta o exemplo de um atributo que representa o dia da semana transformado para representação de ciclo. Cada ponto no plano representa um dia da semana e estão distribuídos de forma igual, sem apresentar nenhum tipo de tendência, ou aspectos de começo ou fim, determinando assim, a característica de atributo cíclico.

Figura 9 – Exemplo de representação do atributo *dia_da_semana* no formato cíclico



Fonte: Elaborado pelo Autor

2.7 Métricas de desempenho

Para avaliar o desempenho de um modelo de aprendizado de máquina em um determinado conjunto de dados, é necessário, de alguma maneira, medir quão bem suas predições correspondem aos dados reais. Ou seja, precisamos quantificar até que ponto as predições das amostras se aproximam dos valores reais. (JAMES *et al.*, 2013)

Existem diversas métricas de desempenho muito conhecidas, usadas para avaliar o desempenho de modelos, como a *precisão*, *revocação*, *F1* e *acurácia* (RASCHKA; MIRJALILI, 2017), elas serão definidas a seguir, mas antes é necessário entendermos o conceito de *matriz de confusão* para compreensão dessas métricas.

2.7.1 Matriz de Confusão

Uma matriz de confusão é uma matriz quadrada que apresenta o desempenho de um modelo de aprendizado de máquina, relatando contagens categorizadas das predições do modelo baseando-se nos valores reais (RASCHKA; MIRJALILI, 2017). Os valores de uma matriz de confusão estão categorizados em quatro tipos: *Verdadeiro Positivo*, *Verdadeiro Negativo*, *Falso Positivo* e *Falso Negativo* (JAMES *et al.*, 2013). A Tabela 1 apresenta um exemplo da disposição e contagens dessas categorias para a classe Gato em uma matriz de confusão. Segundo Goutte e Gaussier (2005) cada uma das categorias podem ser definidas da seguinte forma:

- Verdadeiro Positivo (VP): Ocorre quando a classe atual é predita corretamente. O Quadro 1 mostra que para a classe gato o modelo previu corretamente 2 gatos.
- Falso Positivo (FP): Ocorre quando a classe atual é predita incorretamente. O Quadro 1 mostra que para a classe gato o modelo previu 1 gato incorretamente.
- Verdadeiro Negativo (VN): Ocorre quando a classe atual é predita negativamente de forma correta. O Quadro 1 mostra que o modelo previu 10 objetos de outras classes corretamente.
- Falso Negativo (FN): Ocorre quando a classe atual é predita negativamente de forma incorreta. O Quadro 1 mostra que o modelo previu 12 objetos de outras classes incorretamente.

Quadro 1 – Matriz de confusão mostrando as categorias referentes a classe gato

		Valores Reais		
		Peixe	Gato	Cachorro
Valores Preditos	Peixe	4 (VN)	6 (FN)	3 (FN)
	Gato	1 (FP)	2 (VP)	0 (FP)
	Cachorro	1 (FN)	2 (FN)	6 (VN)

Fonte: Elaborado pelo autor.

A partir dessas informações podemos definir as métricas de desempenho que serão utilizadas neste trabalho.

2.7.1.1 Média das Distâncias dos Erros

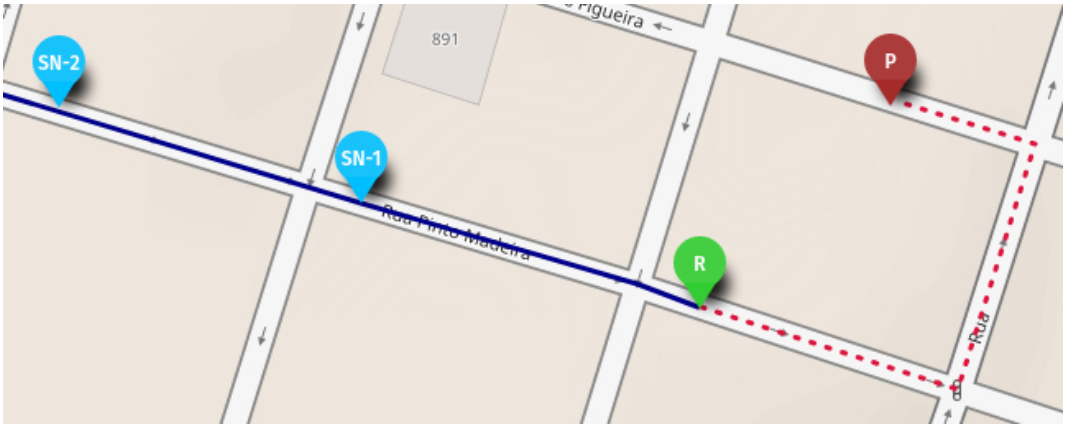
Definida no trabalho de Cruz *et al.* (2019) como *Closeness Error*, A *Média das Distâncias dos Erros* (MDE) é a média entre todas as distâncias da posição do sensor real até a posição do sensor predito, onde a distância é calculada em metros utilizando caminho mais curto

seguindo as leis de trânsito entre as duas posições. A equação 2.4 define a MDE.

$$MDE = \frac{\sum_{i=1}^{j} dist(sr_i, sp_i)}{j} \tag{2.4}$$

Na Figura 10, os pontos posicionados sobre a linha contínua representam os sensores de trânsito que um determinado veículo cruzou, a linha contínua representa o caminho mais curto entre estes sensores, dando uma ideia da trajetória realizada pelo veículo. O ponto *P* representa o sensor que foi predito erroneamente pelo modelo preditivo, e o ponto *R* representa o sensor que capturou de fato a passagem do veículo. A linha pontilhada representa o caminho que será usado para obter a distância entre do ponto real até o ponto predito, esta distância será o valor da *Distância do Erro* para esta instância.

Figura 10 – Exemplo de uma trajetória onde o modelo preditivo erra o sensor no qual o veículo passa.



Fonte: Elaborado pelo autor.

2.7.1.2 *Acurácia*

A *acurácia* é a métrica de avaliação mais utilizada, ela fornece de forma geral a informação sobre quantas amostras foram classificadas corretamente, calculando a soma das predições corretas dividida pelo número total de predições, definida pela equação 2.5 (RASCHKA; MIRJALILI, 2017; SOKOLOVA *et al.*, 2006).

$$A = \frac{VP + VN}{VP + FP + VN + FN} \tag{2.5}$$

Infelizmente a acurácia não atende plenamente às necessidades dos problemas de aprendizado de máquina nos quais as classes não são igualmente importantes e onde vários

algoritmos são comparados, por não fazer a distinção entre a proporção do número de rótulos corretos de diferentes classes (SOKOLOVA *et al.*, 2006). Por este motivo, neste trabalho, também serão utilizadas outras métricas.

2.7.1.3 Precisão, Revocação e F1

A *precisão* é calculada através da razão entre as amostras da classe atual preditas corretamente sobre todas as predições que resultaram nesta classe. (GÉRON, 2017). Ela pode ser definida na equação 2.6.

$$P = \frac{VP}{VP + FP} \tag{2.6}$$

A *revocação* é a razão entre a classe predita corretamente sobre a quantidade de amostras daquela classe (GÉRON, 2017). Ele pode ser definido na equação 2.7.

$$R = \frac{VP}{VP + FN} \tag{2.7}$$

Então, se no processo de ajuste e refinamento do modelo, for utilizado a métrica revocação, significa que os modelos estão sendo adaptados para diminuir o número de FN, pois quanto menor o número de FN, maior a revocação. Já se for utilizada a métrica precisão, os modelos serão adaptados para diminuir o número de FP, pois quanto menor o número de FP maior será a precisão.

A utilização dessas métricas são escolhidas dependendo do tipo de problema que deseja-se resolver. Por exemplo, imagine que um classificador realize predições a respeito de um diagnóstico de diabetes. Então quando um diagnóstico é classificado como positivo significa que o paciente tem diabetes, caso contrário ele é classificado como saudável. Neste caso, provavelmente é requerido um alto índice de revocação, pois se alguém com diabetes for classificado como saudável, acarretará em problemas graves, pois o paciente não irá se tratar, então é necessário que o maior número de diabéticos sejam detectados corretamente. Outro exemplo, seria um classificador de recomendação de usuários para empréstimos. O classificador prediz positivo se um usuário for classificado como um bom pagante e negativo caso não seja. Neste cenário é requerido um alto índice de precisão, pois se algum mau pagante for classificado como um bom pagante acarretará em maiores problemas, então é necessário que o maior número de usuários classificados como bons pagantes sejam realmente bons pagantes.

A métrica *F1* é a média harmônica entre a precisão e a revocação, que diferentemente da média simples, a média harmônica da muito mais peso a valores baixos, portanto os resultados obtidos pela métrica *F1* só será alta se a revocação e a precisão também forem altas (GÉRON, 2017). O *F1-Score* pode ser definido na equação 2.8

$$F1 = 2 \times \frac{P \times R}{P + R} \tag{2.8}$$

3 TRABALHOS RELACIONADOS

Nesta capítulo, é apresentado um breve resumo dos trabalhos relacionados e quais suas relações com o trabalho atual.

3.1 *Trajectory Prediction from a Mass of Sparse and Missing External Sensor Data*

Na pesquisa realizada por Cruz *et al.* (2019) foram criados diversos modelos preditivos baseados no algoritmo RNN para realizar predições de movimentações de veículos a partir dos dados de fotossensores da cidade de Fortaleza - CE. O objetivo do trabalho é prever quais são os próximos sensores de trânsito que os veículos irão transitar. Devido ao alto índice de escassez e incompletude nos dados, o que afetaria negativamente no desempenho dos modelos, foram propostas abordagens que tentam reduzir o impacto deste problema: realizando o preenchimento das trajetórias, adicionando os sensores de trânsito faltantes contidos no caminho mais curto entre os sensores consecutivos dos dados; e *agrupar* os sensores para diminuir o enviesamento e normalizar a representação das trajetórias.

No passo de treinamento do modelo, também foi realizado o preenchimento de trajetórias. Como o algoritmo, na fase de treino, tem o rótulo, duas abordagens foram propostas: a *Full Inputation* e a *Next Value Inputation*. A *Full Inputation* tentaria não só predizer o sensor do rótulo como também todos os sensores que estavam entre o ultimo sensor dado no vetor de atributos e o sensor representado pelo rótulo. Já o *Next Value inputation* tentaria predizer apenas o primeiro sensor após o ultimo sensor do vetor de atributos. Dessa forma alguns modelos foram gerados utilizando essas abordagens e foram avaliados utilizando como base de comparação os algoritmos *Cadeias de Markov* de 1^a e 5^a ordem, obtendo um resultado consideravelmente melhor. O modelo que obteve o melhor acurácia foi modelo baseado em *Next Value Inputation* e *Agrupamento* falado anteriormente.

O trabalho atual anseia na resolução do mesmo problema, distinguindo-se na utilização de outras técnicas de predição e na utilização de diferentes combinações de atributos derivados do conjunto de dados original.

3.2 *TPRED: A Spatio-Temporal Location Predictor Framework*

O trabalho de Rocha *et al.* (2016) busca resolver o problema de predição de posições de objetos, com o diferencial de que ele também busca predizer quando estes objetos irão

deixar sua localização atual. Busca-se predizer as posições relevantes dos objetos, porém nos dados utilizados não existem informações sobre a relevância dos lugares para estes objetos, portanto um local é definido como relevante quando um objeto delonga nele uma quantidade de tempo significativa. Para definição dos possíveis locais à predição, uma tabela foi disposta sobre o mapa representando uma divisão em subáreas, onde cada subárea representa uma classe. Dessa maneira, o problema de predizer a próxima localização deste trabalho é um problema de classificação pois tem como objetivo prever a próxima subárea relevante de um objeto dentre as subáreas definidas. Para solução do problema foi implementado um modelo baseado no algoritmo *Probabilistic Suffix Tree* (PST).

Diferenciando-se do trabalho atual, o *TPRED* utiliza dados coletados a partir de trajetórias provindas de dados de Sistema de Posicionamento Global (GPS) para prever as localizações. Outra diferença é que o algoritmo usado é um algoritmo de decisão probabilística e não um algoritmo de aprendizado de máquina. O *TPRED* foca na predição de objetos individuais, enquanto no trabalho atual utilizamos as trajetórias de todos os objetos para realizar as predições. O trabalho citado também diferencia-se na questão da discretização dos dados de localização, pois nele são realizadas predições de subáreas, e no trabalho atual podemos obter a exata coordenada do objeto dado que seja possível saber qual sensor o objeto cruzou.

3.3 *MyWay: Location prediction via mobility profiling*

No trabalho de Trasarti *et al.* (2017) foi proposto o *MyWay*, um sistema de predição que busca explorar comportamentos sistemáticos de usuários para predizer a mobilidade humana de forma exata, ou seja, ao invés de predizer subáreas como no *TPRED*, serão preditas as coordenadas de um usuário. Para isso, foram utilizados dados de GPS sem aplicar nenhum tipo de discretização e três estratégias para realizar as predições: a estratégia *individual*, que utiliza dados de comportamentos individuais do usuário a ser predito para realizar as predições; a estratégia *coletiva* que observa os comportamentos de todos os usuários para realizar as predições; e a estratégia *híbrida* que é a combinação das duas últimas estratégias. A estratégia *hibrida* usa o preditor *individual* definido na estrategia *individual* quando possível. Caso a predição *individual* não esteja disponível será utilizada a predição *coletiva*.

Diferentemente do trabalho atual, o problema que o *MyWay* busca solucionar é um problema de regressão, pois o modelo tenta prever a posição futura de um objeto utilizando coordenadas, e não um conjunto de classes. Outro ponto que este trabalho difere do trabalho atual

é que, como no *TPRED*, ele também leva em consideração trajetórias de um usuário individual para realizar predições deste usuário, enquanto no trabalho atual não utilizamos nenhum tipo de agrupamento para realizar as predições.

3.4 *City Scale Next Place Prediction from Sparse Data through Similar Strangers*

Na pesquisa de Alhasoun *et al.* (2017) foi proposto um *framework* probabilístico baseado em uma *Dynamic Bayesian Network* (DBN) para predizer futuras localizações de usuários a partir de dados gerados de *registros detalhados de chamadas telefônicas*. Os dados utilizados neste trabalho são bastante esparsos e possuem grande incompletude, portanto foi adotada uma estratégia que utiliza os dados de usuários que se comportam de forma semelhante mas que não necessariamente compartilham o mesmo vínculo social para diminuir o efeito causado por este problema. Foram definidas algumas técnicas para realizar o agrupamento dos usuários similares todas estas técnicas examinam as sequencia de aspectos espaciais e temporais.

Similar ao nosso trabalho, o trabalho de Alhasoun *et al.* (2017) leva em consideração as áreas nas quais as torres de telefonia abrangem seu serviço para realizar a classificação das futuras áreas em que os usuários irão estar, enquanto o trabalho atual utiliza sensores de trânsito para predizer a futura localização de um objeto. Ou seja, em ambos os trabalhos são utilizados algum tipo de periférico para definir as predições a serem realizadas.

Diferentemente do trabalho atual, o Alhasoun *et al.* (2017) utiliza dados obtidos de *registros detalhados de chamadas telefônicas*, enquanto o nosso utiliza dados gerados por sensores de trânsito. Além disso, o trabalho citado agrupa as rotinas de usuários similares para realizar as predições.

3.5 Análise comparativa dos trabalhos

A Tabela 2 apresenta as principais diferenças entre os trabalhos relacionados e o trabalho proposto. Todos os trabalhos possuem o intuito de realizar predição de localizações, porém cada um utiliza uma abordagem diferente, por exemplo, lidando com tipos de dados diferentes, como dados de sensores de trânsito, dados de GPS ou dados de chamadas telefônicas; ou diferentes técnicas de predição; ou realizam as predições através de diferentes representações de locais, como através das posições de sensores de trânsito, subáreas discretizadas, coordenadas, ou áreas de coberturas de rede telefônica; ou utilizam ou não algum tipo de agrupamento para

realizar as predições, utilizando dados de usuários individuais, similares ou de todos ao mesmo tempo.

Quadro 2 – Comparação entre os trabalhos relacionados e o trabalho proposto

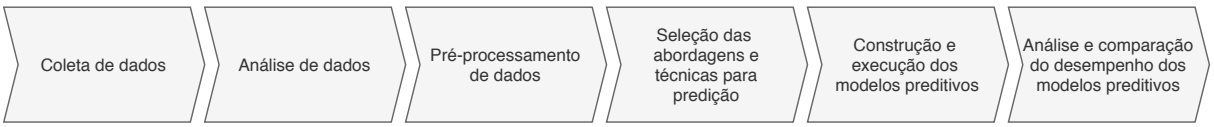
	Cruz et al. (2019)	**Rocha et al. (2016)**	**Trasarti et al. (2017)**	**Alhasoun et al. (2017)**	**Trabalho atual**
Tipo de dado	Sensores de trânsito	GPS	GPS	Registro de chamadas telefônicas	Sensores de trânsito
Técnica	RNN	PST	MyWay	Baseado em DBN	Diversas
Predição	Sensor de trânsito	Subárea e momento do deslocamento	Coordenada	Área da Torre Telefonica	Sensor de trânsito
Discretização de locais	Não	Sim	Não	Não	Não
Agrupamento	Nenhum	Usuário individual	Usuário individual ou Global	Usuários similares	Nenhum

Fonte: Elaborado pelo autor.

4 EXPERIMENTOS E RESULTADOS

Neste capítulo são apresentados os experimentos realizados e os resultados obtidos. Todas as operações e experimentos reportados aqui foram executados em máquinas virtuais do Google sobre as plataformas *Colaboratory*[1] e *Kaggle*[2]. As máquinas virtuais no momento da execução possuíam um processador *Intel Xeon* contendo apenas um núcleo e dois *threads*; 12GB, 16GB ou 25GB de memória RAM; e podendo conter uma *GPU* Tesla P100 da *NVIDEA* com 16GB de memória de vídeo para execução de treinamentos e predições de algoritmos específicos, como o XGBoost e as arquiteturas de redes neurais. A Figura 11 apresenta uma visão geral da metodologia adotada.

Figura 11 – Visão geral da metodologia



Fonte: Elaborado pelo autor.

4.1 Coleta dos dados

O primeiro passo para a realização deste trabalho consistiu em coletar os dados utilizados no trabalho de Cruz *et al.* (2019). Segundo Cruz *et al.* (2019) as TSEs podem assumir formatos altamente diversificados, pois os sensores de trânsito capturam o movimento de vários tipos de objetos, como táxis, entregadores, ônibus e veículos pessoais. Outro aspecto dos dados é que nele existe uma grande esparsidade, visto que os sensores de trânsito ficam fixados em um local e não podem acompanhar o deslocamento dos objetos até o próximo sensor. Por fim, os sensores de trânsito, algumas vezes, falham em capturar a passagem de veículos tornando as TSEs inconsistentes, considerando que veículos realizando a mesma trajetória podem gerar registros diferentes. A Figura 12 mostra o exemplo de uma trajetória que teve falhas em capturar a passagem de um veículo. Os pontos em verde representam os sensores e as linhas vermelhas conectam os sensores que estão em sequência na mesma trajetória. Podemos perceber, que na

[1] É um serviço em nuvem do Google com intuito de incentivar à pesquisa de Aprendizado de Máquina e Inteligência Artificial. O *Colaboratory* permite com que sejam escritos e executados algoritmos em *Python* a partir do navegador sem a necessidade de algum tipo de configuração (GOOGLE, 2020).

[2] É uma comunidade online de cientistas de dados. O Kaggle permite com que os usuários compartilhem conjuntos de dados, aprendam e construam modelos utilizando um serviço em nuvem em um ecossistema voltado para ciência de dados, trabalhem com outros cientistas e participem de competições para solucionar desafios de ciência de dados (KAGGLE, 2019)

Figura 12 existem falhas na captura, pois existem partes da trajetória que o veículo segue na mesma direção, porém alguns sensores que estão sobre a linha vermelha não são utilizados.

Figura 12 – Exemplo de trajetória com falhas de captura

Fonte: Cruz *et al.* (2019)

Segundo Han *et al.* (2011) dados faltantes, discrepantes ou inconsistentes podem afetar negativamente na performance do modelo de aprendizado, e, considerando as características dos dados citadas acima, podemos perceber que eles contém todos esses traços. Portanto há uma explicação caso os modelos não performem tão bem utilizando os dados utilizados neste trabalho.

Os dados estão constituídos em um arquivo de texto, no qual cada linha representa uma trajetória, que por sua vez está estruturada da seguinte maneira: primeiro tem-se o *Id* da trajetória e após um conjunto de tuplas, que representam cada captura de movimento realizada, cada tupla contem o *Id* do sensor de trânsito cruzado pelo veículo pertencente à trajetória e a *marca temporal* que registra o momento da captura. O arquivo de texto possui 825.218 trajetórias. As trajetórias podem ter tamanhos diferentes, tendo como o tamanho mínimo igual a 6 capturas. O Quadro 3 exibe uma breve representação dos dados.

Quadro 3 – Exemplo de como os dados estão estruturados

Id da trajetória	**Id do sensor**	**Marca temporal**	**Id do sensor**	**Marca temporal**	**...**
199884	213	1504275634	18	1504275697	...
21932	95	1504241090	212	150424115220	...
188034	164	1504298043	59	1504298103	...

Fonte: Elaborado pelo autor.

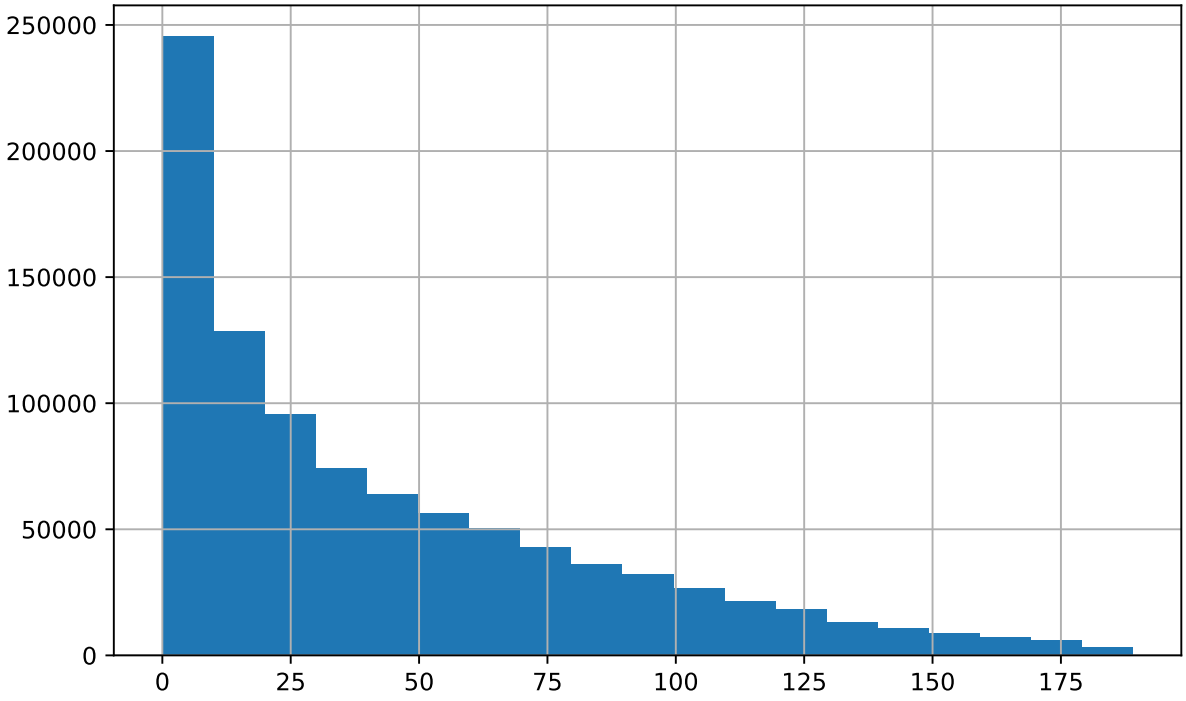
4.2 Análise e Pré-processamento dos dados

Nesta etapa, os dados coletados devem ser analisados para que seja possível compreender sua estrutura e organização, com intuito de realizar um levantamento sobre quais atributos

possam ser utilizados e produzidos; e verificar se existe alguma irregularidade nos dados como, dados faltantes ou fora de contexto. Após isso, é necessário processar os dados para que: se adéquem à entrada dos modelos preditivos; para que problemas detectados possam ser tradados; e para que novas combinações de atributos, derivadas do dado original possam ser geradas.

Uma das primeiras particularidades encontradas nos dados é que os *Id*s dos sensores variam de 1 a 272, porém apenas 190 *Id*s deste intervalo estão sendo usados. Além disso, as ocorrências dos *Id*s para cada classe é desbalanceada. O histograma da Figura 13 mostra o desbalanceamento dentre as distribuições das classes, as classes estão ordenadas em ordem decrescente, e cada barra deste histograma representa 10 classes. Note que existem distribuições de classes com menos de 20 mil ocorrências e outras distribuições com mais de 200 mil ocorrências. Para ser mais exato, a classe que possui mais ocorrências possui 52629 ocorrências, enquanto a classe com menos ocorrências possui apenas 152 ocorrências, apresentando assim uma alta taxa de desbalanceamento.

Figura 13 – Histograma das ocorrências dos *Id*s para as classes



Fonte: Elaborado pelo autor

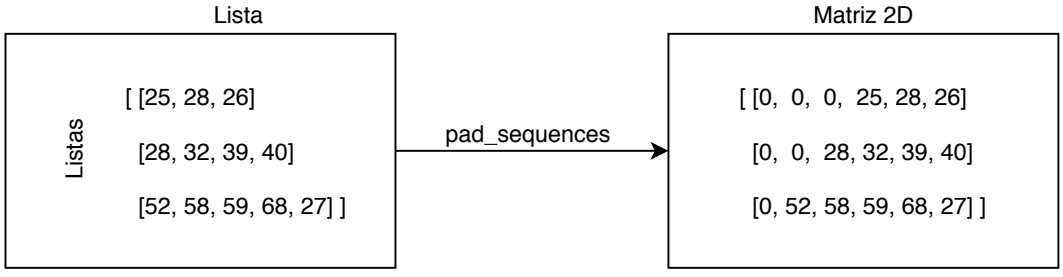
Outra característica encontrada, foi que os registros do conjunto de dados estavam armazenados em ordem cronológica. Então a primeira etapa do pré-processamento consiste em aleatorizar a ordem das linhas do conjunto de dados para que os subconjuntos de dados mencionados na seção 2.5 contenham dados mais genéricos e o modelo consiga treinar com

exemplos de dados que possuam maior amplitude temporal.

Em seguida, foi necessária a extração dos dados referentes as sequências de sensores e marcas temporais contidas no arquivo de formato *txt* e registrá-las em dois novos arquivos no formato *csv* [3], uma para cada, para que pudessem ser carregados em um *DataFrame* [4] do *Pandas* em um ambiente *Python*.

Para que esse processo fosse realizado, os dados foram separados em duas listas de listas. A primeira lista armazena as informações dos sensores, onde cada lista contida nesta lista representa uma trajetória. E a segunda lista funciona de forma semelhante, porém armazenando informações sobre as marcas temporais. Posteriormente, o tamanho máximo de uma trajetória foi limitado à 10 capturas, transformando o restante da trajetória em novas trajetórias caso o seu tamanho fosse maior que 6, totalizando 942.042 trajetórias. Por fim, como o tamanho das trajetórias variam de 6 à 10 capturas, foi necessário utilizar a função *pad_sequences* da biblioteca *Keras*, esta função faz com que as listas de uma lista se transformem em uma matriz 2D, preenchendo posições das listas menores que a maior lista com valores nulos no seu inicio ou fim (CHOLLET *et al.*, 2015). Foi optado pelo preenchimento de valores nulos no inicio das listas. A Figura 14 mostra o funcionamento da função *pad_sequences*. A função *pad_sequences* foi utilizada pois tanto a função que gera o arquivo em formato *.csv* quanto os modelos de aprendizado utilizados neste trabalho exigem que todas as entradas possuam o mesmo tamanho. Os dois novos arquivos gerados foram *sensores.csv* e *marcas_temporais.csv*.

Figura 14 – Exemplo do funcionamento da função *pad_sequences*.



Fonte: Elaborado pelo autor.

O próximo passo é extrair as informações do arquivo "*marcas_temporais.csv*". Como uma marca temporal é um valor inteiro que representa um determinado momento, essas representações precisam ser transformadas em seus respectivos atributos temporais, que são: dia, mês,

[3] *csv* significa *comma-separated-values* (valores separados por vírgulas). Isso significa que os campos de dados indicados neste formato de arquivo normalmente são separados ou delimitados por uma vírgula (SHAFRANOVICH, 2005)

[4] Estrutura de dados principal dos pandas. Contém eixos rotulados (linhas e colunas). Permite operações aritméticas alinhadas nos rótulos de linha e coluna (TEAM, 2020; MCKINNEY, 2010).

ano, hora, minuto e segundo. Por exemplo, o número 1504275634 representa as 11:20:34 do dia primeiro de setembro de 2017. A partir dessa transformações foram gerados mais três arquivos: *dia_discretizado.csv*, *semana_discretizada.csv* e *dia_da_semana.csv*.

O arquivo *dia_discretizado.csv* armazena informações sobre em qual fragmento do dia aquela captura ocorreu, discretizando os dias em intervalos de 30 minutos, ou seja, 24 horas repartidas em 48 partes. Cada parte ficou sendo representada por um número, como o número 28 que representa o intervalo de 14:00 às 14:30. Dessa forma, os modelos de aprendizado podem encontrar determinados padrões, pois da forma anterior as possibilidades de um padrão ser encontrado seriam muito difíceis já que os valores seriam sempre crescentes e dificilmente se repetiam. O arquivo *semana_discretizada.csv* foi construído de forma similar ao *dia_discretizado.csv*, porém, ao invés de apenas o dia ser repartido, a semana inteira foi repartida, cada dia em 48 partes, totalizando 336 partes. Por fim, o arquivo *dia_da_semana.csv* armazena os valores que diz respeito à qual dia da semana a captura foi realizada. Resumindo, todos os arquivos armazenam sequências de números inteiros, onde o 0 representa um valor nulo, devido a função *pad_sequences*, e os outros valores representam um determinado intervalo de tempo.

A ultima alternativa de extrair mais significado dos dados para os modelos de aprendizado consiste em obter valores *cíclicos* que derivam dos dados que possuem atributos temporais. A ideia é fazer com que o modelo entenda que os valores dos dados temporais formam um ciclo. Por exemplo, o valor 48 do arquivo *dia_discretizado.csv*, que representa o intervalo das 23:30 às 00:00, é muito distante do valor 1, que representa o próximo intervalo de 00:00 às 00:30. Portanto usaremos a transformação mencionada na seção 2.6.2 para obter os valores do X e do Y e armazená-los cada um em novo arquivo para cada um dos nossos três arquivos que contêm atributos temporais (*dia_discretizado.csv, semana_discretizada.csv, dia_da_semana.csv*)

Separamos os dados dessa maneira para que pudessem ser testadas diversas combinações de atributos de forma facilitada. Pois, para os modelos de aprendizado de máquina, foi necessário apenas utilizar uma função que mescla a combinação de atributos desejada; já para as arquiteturas de aprendizado profundo, os arquivos foram utilizados do jeito que estão (separados). O Quadro 4 apresenta os possíveis valores contido nas sequencias armazenadas nos arquivos.

Por fim, o ultimo pré-processamento a ser realizado, diz respeito à obtenção da matriz de distâncias, que será utilizado para obter o MDE mencionado na seção 2.7.1.1. As informações necessárias para obtermos a matriz de distâncias estavam contidos em dois aquivos

Quadro 4 – Variação dos valores nos aquivos

Arquivo	**Valores**	**Conjunto**
sensores.csv	0 a 272	$\mathbb{N}$
dia_discretizado.csv	0 a 48	$\mathbb{N}$
semana_discretizada.csv	0 a 336	$\mathbb{N}$
dia_da_semana.csv	0 a 7	$\mathbb{N}$
dia_discretizado_ciclico_y.csv	-1 a 1	$\mathbb{R}$
dia_discretizado_ciclico_x.csv	-1 a 1	$\mathbb{R}$
semana_discretizada_ciclica_y.csv	-1 a 1	$\mathbb{R}$
semana_discretizada_ciclica_x.csv	-1 a 1	$\mathbb{R}$
dia_da_semana_ciclico_y.csv	-1 a 1	$\mathbb{R}$
dia_da_semana_ciclico_x.csv	-1 a 1	$\mathbb{R}$

Fonte: Elaborado pelo autor.

no formato *csv*. O primeiro arquivo (*all_road_distances.csv*) contêm três colunas e 73712 linhas, a primeira e a segunda coluna possuem valores que representam os *IDs* dos sensores de trânsito. Já a terceira coluna possui a distância em metros entre sensores representados da primeira coluna para a segunda coluna. O segundo arquivo (*sensor2id.csv*) possui duas colunas. A primeira coluna contêm a representação dos *IDs* dos sensores de transito referentes aos valores dos *IDS* utilizados no arquivo *all_road_distances.csv*, e a segunda coluna são os *IDs* que representam os sensores de trânsito de fato.

Então, para obter a matriz de distâncias, primeiramente foram substituídos os valores das colunas que continham as representações dos *IDs* no arquivo *all_road_distances.csv* pelos *IDs* dos sensores, recuperados no arquivo *sensor2id.csv*, utilizando a função *map* do *python*. Em seguida, foi utilizado a função *pivot*[5] do *pandas* para que o formato da matriz fosse transformada. Inicialmente a matriz possuía uma linha para cada possível combinação de sensores em sua primeira e segunda coluna, após a transformação, a matriz possui uma linha e uma coluna para cada sensor (272 linha e 272 colunas), onde a posição linha-coluna possui o valor da distancia entre o sensor da linha até o sensor da coluna. Por fim, foi gerado o arquivo *distâncias.csv* que armazena esta matriz de distâncias.

4.3 Seleção das técnicas de predição e construção dos modelos preditivos

O tipo de dado e o tipo de problema a ser resolvido ditam qual modelo de classificação deve ser escolhido. Na prática, é sempre recomendado comparar diferentes modelos para cada conjunto de dados especificado e considerar os desempenhos das predições, bem como a eficiência computacional (RASCHKA, 2014).

[5] <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.pivot.html>

Portanto, os experimentos envolveram a modelagem de 5 conjuntos de dados derivados dos arquivos gerados na seção anterior. O Quadro 5 apresenta todos os conjuntos modelados. Os conjuntos de dados foram separados como mencionado na seção 2.5, distribuindo 60% dos dados para o subconjunto de treino, 20% para o subconjunto de validação e 20% para o subconjunto de teste de forma estratificada[6], então aproximadamente 600 mil amostras ficaram com o conjunto de treino e para os conjuntos de validação e teste ficaram aproximadamente 200 mil amostras para cada. As próximas duas sub-sessões explicam como os modelos de aprendizado de máquina e as arquiteturas de redes neurais foram construídas.

Quadro 5 – Modelagens dos conjuntos de dados

Arquivos utilizados	Nome do Conjunto	Número de Atributos
sensores.csv	S	9
sensores.csv dia_discretizado.csv	S_DD	18
sensores.csv dia_discretizado.csv dia_da_semana.csv	S_DD_DDS	27
sensores.csv semana_discretizada.csv	S_SD	18
sensores.csv semana_discretizada_ciclica_sen.csv semana_discretizada_ciclica_cos.csv	S_SDC	27

Fonte: Elaborado pelo autor.

Para fins de simplificação, o termo *algoritmos de aprendizado de máquina* refere-se aos algoritmos NB, DT, RF e XGBoost. Já o termo *Arquitetura de redes neurais recorrentes* refere-se aos algoritmos SimpleRNN, LSTM e BLSTM.

4.3.1 Construção dos modelos de Aprendizado de Máquina

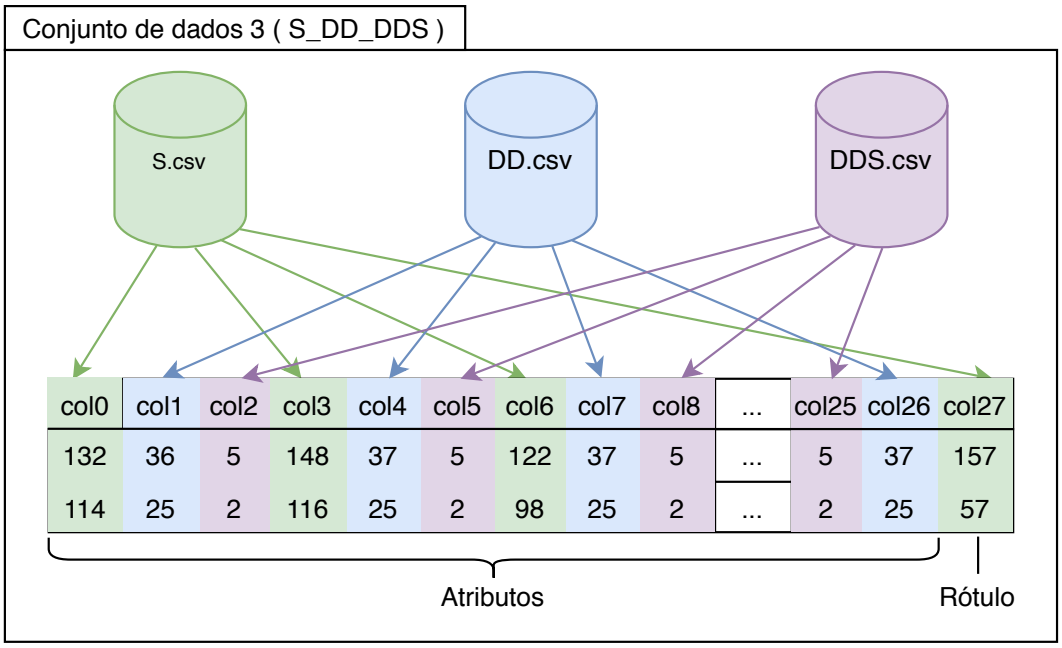
Para construção dos modelos de Aprendizado de máquina, os atributos contidos nos arquivos foram reestruturados para que se adaptassem a entrada dos algoritmos. A Figura 15 ilustra um exemplo geral de como os conjuntos de dados foram reestruturados. Neste exemplo é construído o conjunto de dados S_DD_DDS que utiliza os atributos de três arquivos: *sensores.csv*, *dia_discretizado.csv* e *dia_da_semana.csv*. Então seus valores são agrupados e inseridos em um *DataFrame* do *pandas* na ordem da trajetória.

Todas as trajetórias estão armazenadas na mesma ordem e formato em todos os arquivos *csv* mudando apenas o atributo armazenado, ou seja, a primeira captura da primeira

[6] Significa que as proporções de ocorrência de cada classe serão preservadas sobre cada subconjunto (BUITINCK *et al.*, 2013)

trajetória, refere-se ao primeiro sensor da primeira trajetória no arquivo *sensores.csv*, como também ao primeiro tempo discretizado da primeira trajetória do arquivo *dia_discretizado.csv* e assim sucessivamente. Portanto a modelagem dos conjuntos de dados é feita inserindo os valores da captura de cada arquivo selecionado seguindo a ordem da trajetória. O exemplo da Figura 15 aplica-se para todas as modelagens.

Figura 15 – Exemplo de modelagem de conjunto de dados para os algoritmos de Aprendizado de Máquina.



Fonte: Elaborado pelo autor.

Note que no exemplo do conjunto de dados S_DD_DDS só estão sendo usadas 28 colunas, mas se somarmos a quantidade de atributos em cada arquivo *csv* o valor resultante é 30. Isso acontece porque os últimos valores dos arquivos *dia_discretizado.csv* e *dia_da_semana.csv* referem-se ao rótulo e por esse motivo não estão sendo utilizados.

Foram selecionados quatro algoritmos de Aprendizado de Máquina, dentre eles o NB e o DT da biblioteca *Scikit-Learn*; duas implementações do RF da biblioteca *H2O*[7]; e duas implementações do XGBoost, uma da biblioteca *xgboost*[8] e outra da biblioteca *H2O*.

Para cada conjunto de dados foram construídos diversos modelos preditivos para a maioria desses algoritmos, para que fosse possível obter uma varição dos hiper-parâmetros, pois os resultados podem ser potencialmente melhorados escolhendo os hiper-parâmetros corretos.

O *Distributed Random Forest* (DRF) é a implementção do algortimo RF na biblioteca

[7] *H2O* é uma plataforma de código aberto para Aprendizado de Máquina (H2O.AI, 2020)
[8] <https://xgboost.readthedocs.io/en/latest/>

H2O, existem duas formas de utilizá-la, uma delas é chamando a função do algoritmo diretamente, através da *Application Programming Interface* (API); e a outra forma é utilizando a função *AutoML*[9] também disponível na API. Então, criamos um modelo DRF fazendo a chamada direta e um outro modelo utilizando a função *AutoML*. Neste cenário, o *AutoML* gerou dois modelos, um DRF e um *eXtremely Randomized Trees* (XRT). O XRT é uma variação do modelo DRF que utiliza o parâmetro *histogram_type* como *Random*, isso faz com que o algoritmo gere aleatoriamente limiares para cada atributo e não para um conjunto de atributos, e os melhores limiares serão utilizados para a regra de divisão, normalmente reduzindo a variância do modelo (H2O.AI, 2020). O *AutoML* e considerou o modelo DRF como seu *leader*, o que significa que ele foi o modelo que teve a melhor performance.

Através do *AutoML* também foram gerados quatro modelos XGBoost para cada um dos conjuntos de dados, variando diversos hiper-parâmetros. O Quadro 6 mostra como os hiper-parâmetros variaram utilizando o conjunto de dados *S_SD*. Neste caso, o modelo *leader* foi o *modelo 1*. O objetivo da utilização do *H2O* neste trabalho foi encontrar os melhores hiper-parâmetros de forma mais automatizada.

Quadro 6 – Variação dos hiper-parâmetros para os modelos XGBoost gerados pelo *AutoML*

			Modelos			
		Padrão	Modelo 1	Modelo 2	Modelo 3	Modelo 4
Hiper-parâmetros	distribution	AUTO	multinomial	multinomial	multinomial	multinomial
	categorical_encoding	AUTO	OneHotInternal	OneHotInternal	OneHotInternal	OneHotInternal
	ntrees	50	10000	10000	10000	10000
	max_depth	6	15	5	20	10
	min_rows	1	15	3	10	5
	score_tree_interval	0	5	5	5	5
	tree_method	AUTO	hist	hist	exact	hist
	booster	gbtree	dart	gbtree	gbtree	gbtree
	reg_lambda	1	100	1	1	1
	reg_alpha	0	0.5	0	0	0
	dmatrix_type	AUTO	dense	dense	dense	dense
	backend	AUTO	GPU	GPU	CPU	GPU

Fonte: Elaborado pelo autor.

Deve ser enfatizado que para todos os algoritmos DRF e *XGBoosting*, foi utilizado o hiper-parâmetro *stopping_rounds* com valor igual a 20, esse hiper-parâmetro é usado para interromper o treinamento do modelo caso não haja melhora de performance para o número de iterações especificado (H2O.AI, 2020). O Quadro 7 apresenta todos os algoritmos utilizados e os seus respectivos hiper-parâmetros que foram variados. Note que para as funções *AutoML*

[9] O AutoML ou *Automatic Machine Learning* é uma ferramenta do H2O que pode ser usado para automatizar o fluxo de trabalho de aprendizado de máquina incluindo treinamento e ajuste de muitos modelos automaticamente (H2O.AI, 2017)

do *H2O* não existem variações de hiper-parâmetros de forma manual, isso acontece por que o *AutoML* tenta encontrar os melhores parâmetros de forma completamente automática.

Quadro 7 – Algoritmos selecionados

Algoritmo	**Biblioteca**	**Hiper-parâmetros ajustados**
Naive Bayes	Scikit-Learn	Alpha
Decision Tree	Scikit-Learn	criterion e max_depth
Random Forest (DRF)	H2O	max_depth
Random Forest (DRF)	AutoML - H2O	Histogram_type
XGBoosting	xgboost	max_depth e learning_rate
XGBoosting	AutoML - H2O	Disponível na Tabela 6

Fonte: Elaborado pelo autor.

Até onde se sabe, não existem muitos trabalhos que lidam com problemas que possuem esse tipo de dado, por este motivo, é interessante a seleção de múltiplos algoritmos e abordagens de dados, para que seja possível fazer uma comparação detalhada de como os modelos se comportam perante este tipo de dado, dessa forma é possível obter informações valiosas, como por exemplo, se realmente vale a pena gastar mais tempo em processamento de algoritmos mais complexos dado que um algoritmo mais simples obteve resultados similares.

4.3.2 Construção das arquiteturas de redes neurais recorrentes

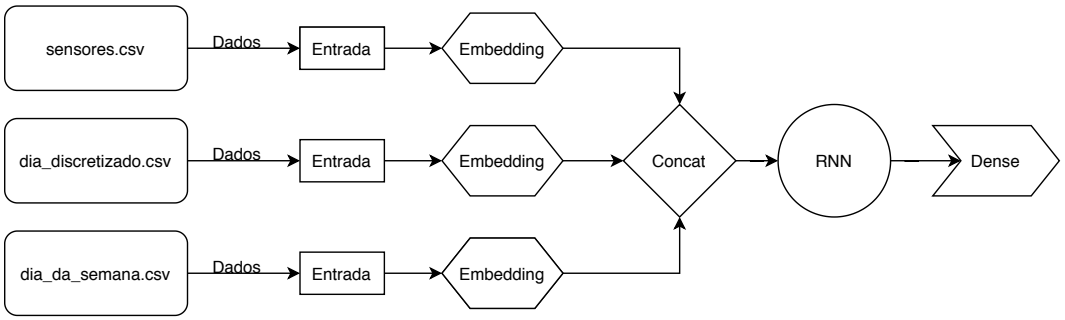
As arquiteturas de redes neurais recorrentes utilizadas foram a SimpleRNN, a LSTM e a BLSTM, elas foram construídas utilizando os dados separadamente, como a Figura 16 ilustra. Cada atributo é dado como entrada separadamente e processado em uma camada *embedding*, depois os valores obtidos são concatenados na camada *concat* e levados à camada recorrente, onde os complexos padrões das trajetórias serão aprendidas. Por fim, a camada *dense*, processa o resultado da rede através da função *softmax*[10], retornando os valores que serão preditos.

De forma similar à construção dos modelos no passo anterior, aqui os modelos foram gerados variando apenas o parâmetro *recurrent_dropout*[11] na camada RNN. Para as arquiteturas de redes neurais foi utilizado o *earlyStopping* como *callback* tendo o valor de *paciência* igual a 20. Este recurso é usado de forma similar ao *stopping_rounds*, também usado para interromper o treinamento do modelo caso sua performance não melhore no decorrer das iterações.

[10] A função *softmax* converte o vetor de entrada em um vetor de probabilidades categóricas, normalmente utilizadas em redes classificadoras pois o seu resultado pode ser interpretado como uma distribuição probabilística (CHOLLET *et al.*, 2015).

[11] Utilizado para combater o Overfitting, o *Dropout* é responsável por zerar aleatoriamente as unidades de entrada de uma camada, a fim de quebrar eventuais correlações nos dados de treinamento aos quais a camada é exposta (CHOLLET, 2017).

Figura 16 – Arquitetura de Rede Neural que tem como entrada o conjunto de dados *S_DD_DDS*



Fonte: Elaborado pelo autor.

Na camada *Embedding*, o tamanho da saída varia de acordo com o tamanho da entrada. Foi utilizado um valor empírico que melhor se ajustou nos testes realizados. Os valores usados para cada atributo estão dispostos na Quadro 8.

Quadro 8 – Valores de entradas e saídas na camada *embedding* para cada atributo

	Atributos			
	S	DD	DDS	SD
Dimensão de Entrada	190	48	7	336
Dimensão de Saída	30	14	5	42

Fonte: Elaborado pelo autor.

Note que para o conjunto de dados *S* a dimensão de entrada está descrita como 190, porém seus valores variam de 1 a 272 com valores não utilizados, pois, como descrito anteriormente, apenas 190 dos 272 sensores estão sendo usados, então a função *LabelEncoder*[12] do *Scikit-Learn* foi utilizada para deixar os valores que representam os *ids* em um intervalo contínuo de inteiros, tornando a dimensão de entrada igual à quantidade de classes.

Os dados referentes ao tempo discretizado cíclico não foram utilizados nas arquitetura de redes neurais, pois a entrada de uma rede neural, como citado anteriormente, normalmente precisa ser codificado como *one-hot* ou *word embedding*, então quando quando o tempo discretizado cíclico é transformado para uma dessas duas representações o significado de ciclo ou é perdido para a representação *one-hot* ou se torna ambíguo para a representação *word embedding*, tornando-os significativamente similares a outras combinação já utilizadas.

[12] Codifica a entrada com valores de 0 até a quantidade de classes -1 (BUITINCK *et al.*, 2013).

4.4 Execução e Análise dos Modelos Preditivos

Para que o desempenho dos modelos sejam analisados devemos levar em consideração alguns aspectos da performance, como o tempo gasto na etapa de treinamento e os valores obtidos através das métricas definidas na seção 2.7.

Neste trabalho foram gerados 185 modelos preditivos diferentes, variando nos algoritmos, conjuntos de dados e hiper-parâmetros. A seguir, na Tabela 1, é possível ver os resultados dos modelos que obtiveram as melhores acurácias. Pela análise desses resultados, podemos constatar que os modelos XGBoost obtiveram os melhores resultados quando não se trata de aprendizado profundo, e a rede LSTM obteve o melhor resultado geral dentre as três arquiteturas de Aprendizado Profundo. Acredita-se que o modelo BLSTM não consegue superar o modelo LSTM em nenhuma dessas abordagens, pois utilizar a ordem inversa das trajetórias, provavelmente, não adiciona significado nenhum ao modelo preditivo neste cenário.

Tabela 1 – Acurácia dos melhores modelos para os conjuntos de dados e algoritmos

	Conjuntos de Dados				
Algoritmos	S	S_D	S_D_DS	S_D	S_DC
NB	39.15%	37.40%	35.70%	36.44%	34.33%
DT	47.63%	47.72%	47.85%	47.63%	47.60%
DRF	49.20%	47.05%	46.66%	47.02%	46.61%
RF AutoML	46.90%	47.07%	46.35%	47.88%	45.97%
XGBoost	49.14%	49.35%	49.43%	49.37%	49.27%
XGBoost AutoML	49.01%	34.18%	49.00%	26.59%	**49.43%**
SimpleRNN	49.35%	48.05%	49.61%	49.34%	-
LSTM	**49.77%**	**49.98%**	**50.01%**	**50.09%**	-
BLSTM	49.69%	49.98%	49.99%	49.97%	-

Fonte: Elaborado pelo autor.

Entretanto as tabelas 2 e 3, que dispõem os valores das métricas precisão e *f1* de forma balanceada[13], obtidas pelos modelos da Tabela 1, mostram que os modelos BLSTM obtiveram valores superiores aos modelos LSTM, isso acontece pois os modelos BLSTM, provavelmente erram menos predições dos sensores para suas determinadas classes, o que aumentaria o índice de precisão e consequentemente o índice de *F1*, já que o *F1* é a média harmônica entre o *Recall* e o *Precision*. Observando para o cenário do atual trabalho, a precisão é uma boa métrica para avaliação destes modelos, pois ela indica o quão bem os modelos realizam as predições que realmente são de suas classes, portanto utilizar a precisão como métrica de ajuste faria com que

[13] Calcula a métrica utilizando a média ponderada através da quantidade de ocorrências de cada classe (BUITINCK *et al.*, 2013)

os modelos fossem voltados a evitar a predição incorreta para a cada classe.

Tabela 2 – Precisão balanceada obtida para os modelos da Tabela 1

	Conjuntos de Dados				
Algoritmos	S	S_D	S_D_DS	S_D	S_DC
NB	36.98%	35.83%	35.23%	36.44%	34.22%
DT	46.09%	46.07%	46.45%	46.30%	45.91%
DRF	47.40%	43.93%	43.35%	43.87%	46.02%
RF AutoML	44.37%	43.97%	42.71%	43.74%	42.17%
XGBoost	47.56%	47.64%	47.68%	47.68%	47.53%
XGBoost AutoML	47.53%	33.91%	47.24%	32.20%	**47.58%**
SimpleRNN	47.85%	46.34%	47.61%	47.43%	-
LSTM	48.33%	48.44%	48.43%	48.28%	-
BLSTM	**48.49%**	**48.72%**	**48.51%**	**48.53%**	-

Fonte: Elaborado pelo autor.

Tabela 3 – *F1* balanceado obtido para os modelos da Tabela 1

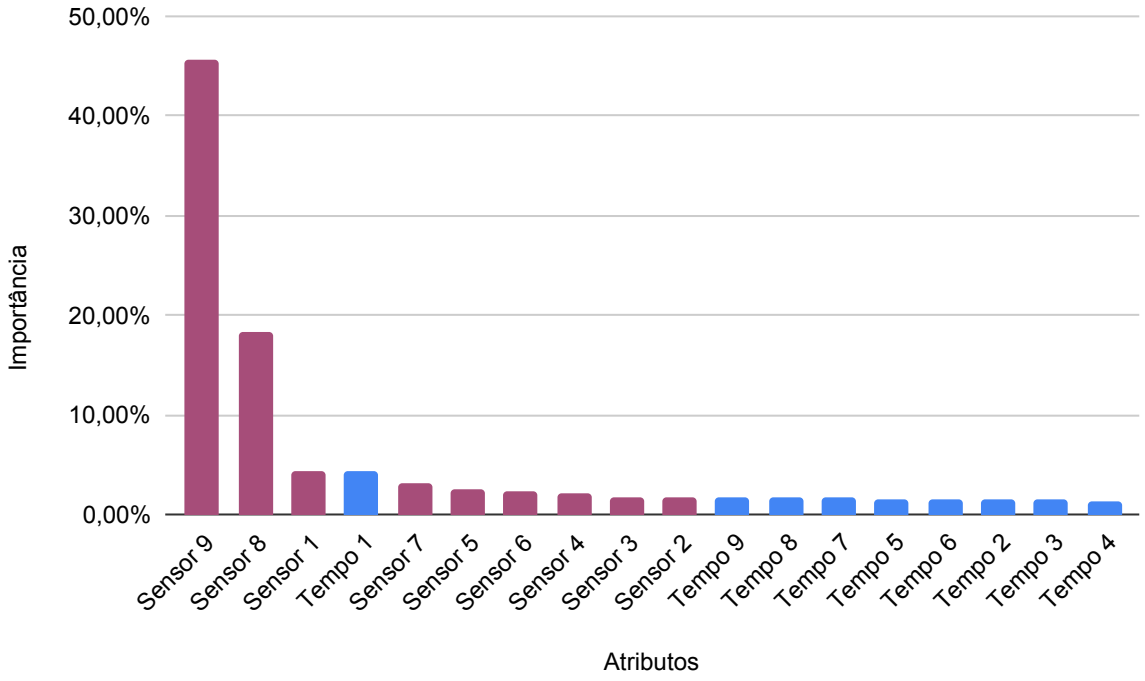
	Conjuntos de Dados				
Algoritmos	S	S_D	S_D_DS	S_D	S_DC
NB	37.23%	35.91%	34.60%	32.49%	34.33%
DT	47.63%	45.99%	46.10%	45.88%	45.85%
DRF	47.62%	43.98%	43.33%	43.95%	46.32%
RF AutoML	45.14%	44.89%	42.86%	44.74%	43.39%
XGBoost	47.51%	47.73%	47.76%	47.73%	47.64%
XGBoost AutoML	47.30%	32.42%	47.05%	24.69%	**47.73%**
SimpleRNN	47.66%	46.21%	47.67%	47.60%	-
LSTM	48.11%	48.36%	48.42%	48.38%	-
BLSTM	**48.30%**	**48.49%**	**48.42%**	**48.51%**	-

Fonte: Elaborado pelo autor.

Outra característica que podemos extrair desses resultados é que não existe um conjunto de dados melhor, as diferenças dos resultados entre diferentes conjuntos de dados são tão pequenas que qualquer um desses conjuntos podem ser selecionados para a solução do problema. Isso mostra que o atributo que mais importa para a solução deste problema são os *Id*s dos sensores de trânsito, que estão presente em todos os conjuntos de dados. A figura 17 mostra como um modelo XGBoost definiu as importâncias de cada atributo para o conjunto de dados *S_SD*. Note que os últimos *Ids* dos sensores são consideradas os atributos mais importantes, e os atributos que representam o tempo sempre possuem uma importância muito baixa. Esse comportamento é muito semelhante para os outros conjuntos de dados.

Outro aspecto para levar em consideração na avaliação dos modelos é o tempo de treinamento. A Figura 18 mostra o tempo médio em que os modelos de cada algoritmo gastam para realizar este processo. Foi optado pelo tempo médio, pois o tempo de treinamento não variava muito dentre os diferentes conjuntos de dados, apenas para os diferentes algoritmos.

Figura 17 – Importância dos atributos gerados pelo modelo XGBoost da biblioteca *xgboost* e conjunto de dados *S_SD*



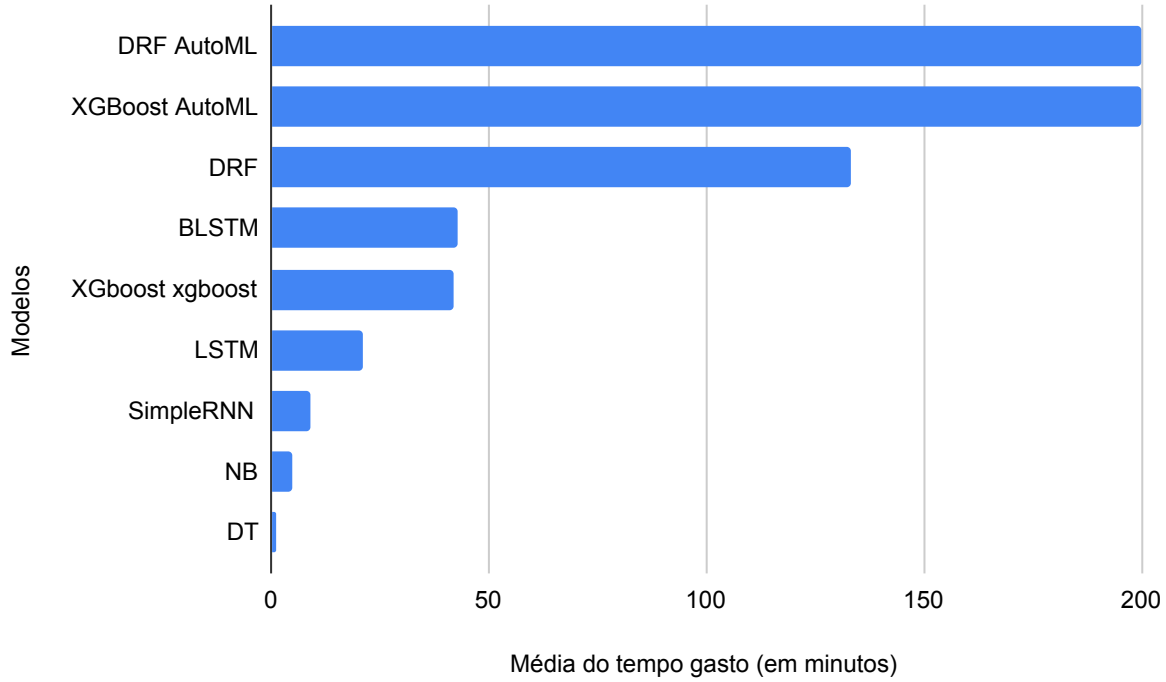
Fonte: Elaborado pelo autor.

Pela análise do Gráfico podermos perceber que os modelos que utilizam a função *AutoML* são as que gastam mais tempo, mas deve ser levado em consideração que a função *AutoML* foi executada apenas uma vez para cada conjunto da dados, afinal sua seleção de parâmetros é feita automaticamente. Porém, algumas inconsistências podem acontecer, pois o *AutoML* treina diversos modelos internamente buscando um que melhor resolva o problema, mas em alguns casos ele não consegue criar um modelo que consiga convergir muito bem, como por exemplo para os modelos *AutoML* do XGBoost para os dados *S_DD* e *S_SD*. Para cada um dos outros algoritmos foram feitos diversos modelos para cada conjunto de dados, variando os seus hiper-parâmetros.

Além disso, note que os valores obtidos através das métricas para determinados modelos são bem similares, porém o tempo de treinamento é bem diferente. Como nos casos dos modelos DT e DRF; e nos modelos de Aprendizado Profundo. Dessa forma, são optados pelos modelos que compensem ser treinados por uma relação tempo gasto para treinar e quão bem os modelos performam perante as métricas.

Outra forma de analisar os modelos é através da métrica MDE. A métrica MDE foi utilizada pois o trabalho de selecionar uma classe dentre 180, pode ser muito difícil para um preditor, portanto esta métrica foi criada para que fosse possível saber o quão próximo as

Figura 18 – Comparação entre os tempos de treinamento e predição dos modelos



Fonte: Elaborado pelo autor.

predições do modelo poderia chegar do dado real. Diferentemente da acurácia, da precisão e do *F1*, que não conseguem extrair essa informação, já que o cálculo dos seus resultados são feitos através de representações binárias, ou seja, a partir de acertos e erros.

A Tabela 4 apresenta o MDE referente aos modelos apresentados na Tabela 1. Através desses resultados, podemos constatar que os modelos XGBoost da biblioteca *xgboost* e o DRF foram os que obtiveram os melhores desempenhos neste cenário.

Tabela 4 – *Médias das Distâncias dos erros* em metros dos modelos que obtiveram melhor acurácia

	Conjuntos de Dados				
Algortmos	S	S_D	S_D_DS	S_D	S_DC
NB	3138m	3227m	3327m	3400m	3523m
DT	2349m	2351m	2357m	2364m	2363m
DRF	**2262m**	2659m	2749m	2666m	2738m
RF AutoML	2491m	2584m	2720m	2589m	2771m
XGBoost	2274m	**2274m**	**2275m**	**2276m**	**2278m**
XGBoost AutoML	2299m	4038m	2333m	5459m	2292m
SimpleRNN	9568m	9568m	9580m	9584m	-
LSTM	9576m	9580m	9571m	9583m	-
BLSTM	9584m	9584m	9593m	9555m	-

Fonte: Elaborado pelo autor.

Perceba que os modelos de Aprendizado Profundo tiveram performance muito inferior aos outros modelos, então para fins de teste, foi utilizado o modelo *Dummy Classifier*

(DC)[14] do *Scikit-Learn* para gerar resultados aleatórios baseado nas proporções das classes no subconjunto de dados de treino. Então os resultados obtidos por este classificador foram de 1.32% de acurácia e 9462 metros para o MDE. Isso significa que os modelos de Aprendizado Profundo predizem sensores mais distantes quando erram a predição, levando em consideração que sua performance para o MDE é inferior até mesmo para um modelo que faz suas predições aleatoriamente e que acertou apenas 1.32% das predições. Em Cruz *et al.* (2019) o modelo RNN construído de forma similar ao do presente trabalho obteve 48.07% de acurácia e aproximadamente 6500 metros de MDE para um percentil de 90%, este modelo foi gerado através de apenas uma iteração, portanto, acredita-se que o motivo pelo qual os modelos RNN do atual trabalho tenham MDEs tão altos, seja porque provavelmente possam está muito ajustados aos dados de treino, tendo em vista que cada modelo iterou pelo menos 20 vezes sobre o conjunto de dados.

[14] <https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html>

5 CONSIDERAÇÕES FINAIS

O principal objetivo deste trabalho foi descobrir uma boa técnica para realizar a predição de localização de veículos utilizando dados de sensores de trânsito. Para isso foi selecionado um conjunto de algoritmos desde os mais simples até os mais sofisticados, além do uso de diferentes combinações dos dados gerando de diversos modelos diferentes, para que fosse selecionado o modelo que melhor se ajustou à solução desse tipo de problema. O XGBoost da biblioteca *xgboost* foi o algoritmo que obteve melhor performance pois, apesar de não ter obtido as melhores acurácias, precisões e *F1*s, foi o que obteve o melhor MDE de forma geral, além de que a diferença de valores entre o XGBoost e os outros algoritmos que obtiveram os maiores valores para as outras métricas não é consideravelmente alta. Além disso, o XGBoost obteve uma ótima performance no tempo de treinamento. Portanto o modelo selecionado como melhor foi o que utiliza o algortimo XGBoost da biblioteca *xgboost* e o conjunto de dados *S_DD_DDS* por obter a melhor performance geral.

Porém, temos que enfatizar que o melhor modelo acerta apenas aproximadamente metade das predições o que não é o ideal. Portando trabalhar apenas com este tipo de dado para resolver este tipo de problema, dado suas características, pode ser complexo, tendo em vista que em Cruz *et al.* (2019), as abordagens mais sofisticadas, alcançam uma acurácia de aproximadamente 70% para o melhor modelo e aproximadamente 2000 metros de MDE para um percentil de 90%.

Este trabalho teve como contribuição uma análise comparativa entre diversas técnicas de predição para solução do dado problema, além de que foi possível comparar diversas abordagens de variadas bibliotecas e conjuntos de dados. Outra possível contribuição é mostrar a relevância do uso desse tipo de dado no setor de transporte e mobilidade no cenário de predição de trajetórias.

Então, como trabalhos futuros, pode-se destacar a utilização de outras combinações de dados para treinar os modelos, como por exemplo, a utilização de apenas um atributo temporal e sequências mais curtas, pois como visto, os atributos temporais e os sensores iniciais não possuem tanta relevância; testar os modelos em conjuntos de dados referentes a outras cidades; utilizar as outras abordagens presentes em Cruz *et al.* (2019) utilizando os algoritmos testados neste trabalho, como o XGBoost; utilizar de arquiteturas de *Redes Neurais* ainda mais sofisticadas usadas para predições de sequências, como a rede *Transformer* (USZKOREIT, 2017).

REFERÊNCIAS

ALHASOUN, F.; ALHAZZANI, M.; ALEISSA, F.; ALNASSER, R.; GONZÁLEZ, M. City scale next place prediction from sparse data through similar strangers. In: **Proceedings of ACM KDD Workshop**. [*S. l.*: *s. n.*], 2017. p. 191–196.

ALPEYDIN, E. **Introduction to Machine Learning**. [*S. l.*]: "The MIT Press", 2010.

BARTLE, R. G.; SHERBERT, D. R. **Introduction to real analysis**. [*S. l.*]: Wiley New York, 2000. v. 2.

BATTY, M.; AXHAUSEN, K. W.; GIANNOTTI, F.; POZDNOUKHOV, A.; BAZZANI, A.; WACHOWICZ, M.; OUZOUNIS, G.; PORTUGALI, Y. Smart cities of the future. **The European Physical Journal Special Topics**, Springer, v. 214, n. 1, p. 481–518, 2012.

BELL, J. **Machine learning**: hands-on for developers and technical professionals. [*S. l.*]: John Wiley & Sons, 2014.

BESCOND, P.-L. **Cyclical features encoding, it's about time!** 2020. Disponível em: https://towardsdatascience.com/cyclical-features-encoding-its-about-time-ce23581845ca. Acesso em: 10 set. 2020.

BEZERRA, E. Introdução à aprendizagem profunda. **Artigo–31º Simpósio Brasileiro de Banco de Dados–SBBD2016–Salvador**, 2016.

BRETZKE, W.-R. Global urbanization: a major challenge for logistics. **Logistics Research**, Springer, v. 6, n. 2-3, p. 57–62, 2013.

BUCHER, D. Vision paper: Using volunteered geographic information to improve mobility prediction. In: ACM. **Proceedings of the 1st ACM SIGSPATIAL Workshop on Prediction of Human Mobility**. [*S. l.*], 2017. p. 2.

BUITINCK, L.; LOUPPE, G.; BLONDEL, M.; PEDREGOSA, F.; MUELLER, A.; GRISEL, O.; NICULAE, V.; PRETTENHOFER, P.; GRAMFORT, A.; GROBLER, J.; LAYTON, R.; VANDERPLAS, J.; JOLY, A.; HOLT, B.; VAROQUAUX, G. API design for machine learning software: experiences from the scikit-learn project. In: **ECML PKDD Workshop: Languages for Data Mining and Machine Learning**. [*S. l.*: *s. n.*], 2013. p. 108–122.

BURKOV, A. **The hundred-page machine learning book**. [*S. l.*]: Andriy Burkov Quebec City, Can., 2019.

CHEN, T.; GUESTRIN, C. Xgboost: A scalable tree boosting system. In: **Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining**. [*S. l.*: *s. n.*], 2016. p. 785–794.

CHOLLET, F. **Deep Learning with Python**. [*S. l.*]: Manning Publications Co., 2017.

CHOLLET, F. *et al.* **Keras**. 2015. Disponível em: https://keras.io. Acesso em: 28 jun. 2020.

CRUZ, L. A.; ZEITOUNI, K.; MACEDO, J. A. F. de. Trajectory prediction from a mass of sparse and missing external sensor data. In: IEEE. **2019 20th IEEE International Conference on Mobile Data Management (MDM)**. [*S. l.*], 2019. p. 310–319.

DIMITROVA, E. S.; LICONA, M. P. V.; MCGEE, J.; LAUBENBACHER, R. Discretization of time series data. **Journal of Computational Biology**, Mary Ann Liebert, Inc. 140 Huguenot Street, 3rd Floor New Rochelle, NY 10801 USA, v. 17, n. 6, p. 853–868, 2010.

GÉRON, A. **Hands-on machine learning with Scikit-Learn and TensorFlow**: concepts, tools, and techniques to build intelligent systems. [*S. l.*]: "O'Reilly Media, Inc.", 2017.

GOOGLE. **What is Colaboratory?** 2020. Disponível em: https://colab.research.google.com/notebooks/intro.ipynb. Acesso em: 28 set. 2020.

GOUTTE, C.; GAUSSIER, E. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In: SPRINGER. **European Conference on Information Retrieval**. [*S. l.*], 2005. p. 345–359.

GRUS, J. **Data Science do zero**: Primeiras regras com o python. [*S. l.*]: Alta Books, 2019.

GRZEÇA, M.; BECKER, K.; GALANTE, R. Drink2vec: Improving the classification of alcohol-related tweets using distributional semantics and external contextual enrichment. **Information Processing & Management**, Elsevier, p. 102369, 2020.

GUENICHE, T.; FOURNIER-VIGER, P.; RAMAN, R.; TSENG, V. S. Cpt+: Decreasing the time/space complexity of the compact prediction tree. In: SPRINGER. **Pacific-Asia Conference on Knowledge Discovery and Data Mining**. [*S. l.*], 2015. p. 625–636.

H2O.AI. **H2O AutoML**. 2017. H2O version 3.30.0.1. Disponível em: http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html. Acesso em: 20 set. 2020.

H2O.AI. **Python Interface for H2O**. 2020. Python package version 3.30.0.6. Disponível em: https://github.com/h2oai/h2o-3. Acesso em: 21 set. 2020.

HAN, J.; PEI, J.; KAMBER, M. **Data mining**: concepts and techniques. [*S. l.*]: Elsevier, 2011.

IBGE. **Frota de veículos**. Centro de Documentação e Disseminação de Informações. Fundação Intituto Brasileiro de Geografia e Estatística, 2018. Disponível em: http://biblioteca.ibge.gov.br/visualizacao/livros/liv23907.pdf. Acesso em: 16 out. 2020.

JAMES, G.; WITTEN, D.; HASTIE, T.; TIBSHIRANI, R. **An introduction to statistical learning**. [*S. l.*]: Springer, 2013. v. 112.

KAGGLE. **Start with more than a blinking cursor.** 2019. Disponível em: https://www.kaggle.com/. Acesso em: 28 set. 2020.

KALEKO, D. **Feature Engineering - Handling Cyclical Features**. 2017. Disponível em: http://blog.davidkaleko.com/feature-engineering-cyclical-features.html. Acesso em: 10 set. 2020.

LAIRD, P.; SAUL, R. Discrete sequence prediction and its applications. **Machine learning**, Springer, v. 15, n. 1, p. 43–68, 1994.

MCKINNEY Wes. Data Structures for Statistical Computing in Python. In: WALT Stéfan van der; MILLMAN Jarrod (Ed.). **Proceedings of the 9th Python in Science Conference**. [*S. l.*: *s. n.*], 2010. p. 56 – 61.

MOSKOVITCH, R.; SHAHAR, Y. Classification-driven temporal discretization of multivariate time series. **Data Mining and Knowledge Discovery**, Springer, v. 29, n. 4, p. 871–913, 2015.

RASCHKA, S. Naive bayes and text classification. **Retrieved from sebastianraschka. com/Articles/2014_naive_bayes_1. html**, 2014.

RASCHKA, S.; MIRJALILI, V. **Python machine learning**. [*S. l.*]: Packt Publishing Ltd, 2017.

ROCHA, C. L.; BRILHANTE, I. R.; LETTICH, F.; MACEDO, J. A. F. D.; RAFFAETÀ, A.; ANDRADE, R.; ORLANDO, S. Tpred: a spatio-temporal location predictor framework. In: ACM. **Proceedings of the 20th International Database Engineering & Applications Symposium**. [*S. l.*], 2016. p. 34–42.

ROSSUM, G. V.; DRAKE, F. L. **Python 3 Reference inproceedings**. Scotts Valley, CA: CreateSpace, 2009. ISBN 1441412697.

RUSSELL, S. J.; NORVIG, P. **Artificial intelligence**: a modern approach. [*S. l.*]: Malaysia; Pearson Education Limited,, 2016.

SHAFRANOVICH, Y. **Common format and MIME type for comma-separated values (CSV) files**. [*S. l.*]: RFC 4180, October, 2005.

SOKOLOVA, M.; JAPKOWICZ, N.; SZPAKOWICZ, S. Beyond accuracy, f-score and roc: a family of discriminant measures for performance evaluation. In: SPRINGER. **Australasian joint conference on artificial intelligence**. [*S. l.*], 2006. p. 1015–1021.

SUN, R.; GILES, C. L. Sequence learning: From recognition and prediction to sequential decision making. **IEEE Intelligent Systems**, v. 16, n. 4, p. 67–70, 2001.

TAN, P.-N.; STEINBACH, M.; KUMAR, V. **Introduction to Data Mining**. [*S. l.*]: Pearson Education, 2006.

TEAM, T. pandas development. **pandas-dev/pandas**: Pandas. Zenodo, 2020. Disponível em: https://doi.org/10.5281/zenodo.3509134. Acesso em: 30 set. 2020.

TRASARTI, R.; GUIDOTTI, R.; MONREALE, A.; GIANNOTTI, F. Myway: Location prediction via mobility profiling. **Information Systems**, Elsevier, v. 64, p. 350–367, 2017.

USZKOREIT, J. Transformer: A novel neural network architecture for language understanding. **Google AI Blog**, v. 31, 2017.

YING, J. J.-C.; LEE, W.-C.; WENG, T.-C.; TSENG, V. S. Semantic trajectory mining for location prediction. In: **Proceedings of the 19th ACM SIGSPATIAL international conference on advances in geographic information systems**. [*S. l.*: *s. n.*], 2011. p. 34–43.

ZHAO, X.; TANG, J. Crime in urban areas:: A data mining perspective. **ACM SIGKDD Explorations Newsletter**, ACM, v. 20, n. 1, p. 1–12, 2018.

ZHENG, Y. Trajectory data mining: an overview. **ACM Transactions on Intelligent Systems and Technology (TIST)**, ACM, v. 6, n. 3, p. 29, 2015.