UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ARINA DE JESUS AMADOR MONTEIRO SANCHES

UMA ARQUITETURA E IMPLEMENTAÇÃO DO MÓDULO DE PRÉ-PROCESSAMENTO PARA BIBLIOTECA PYMOVE

FORTALEZA

2019

ARINA DE JESUS AMADOR MONTEIRO SANCHES

UMA ARQUITETURA E IMPLEMENTAÇÃO DO MÓDULO DE PRÉ-PROCESSAMENTO PARA BIBLIOTECA PYMOVE

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

Orientador: Prof. Dr. José Antônio Fernandes Macedo

Coorientador: Me. Nicksson Ckayo Arrais de Freitas

FORTALEZA

2019

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)

S19a Sanches, Arina de Jesus Amador Monteiro.
Uma arquitetura e implementação do módulo de pré-processamento para biblioteca PyMove / Arina de Jesus Amador Monteiro Sanches. – 2019.
52 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Ciências, Curso de Computação, Fortaleza, 2019.
Orientação: Prof. Dr. José Antônio Fernandes Macedo.
Coorientação: Prof. Me. Nicksson Ckayo Arrais de Freitas.

1. Trajetória. 2. Análise de trajetória. 3. Pré-processamento de trajetória.. 4. Filtragem de ruído. 5. Segmentação. I. Título.

CDD 005

ARINA DE JESUS AMADOR MONTEIRO SANCHES

UMA ARQUITETURA E IMPLEMENTAÇÃO DO MÓDULO DE PRÉ-PROCESSAMENTO PARA BIBLIOTECA PYMOVE

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

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. José Antônio Fernandes Macedo (Orientador)
Universidade Federal do Ceará (UFC)

Me. Nicksson Ckayo Arrais de Freitas
(Coorientador)
Universidade Federal do Ceará (UFC)

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

Prof[a]. Ma. Lívia Almada Cruz Rafael
Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

Em primeiro lugar, agradeço a Deus esta conquista e por todas as pessoas que ele colocou no meu caminho.

Agradeço aos meus pais, pelo amor, por terem sempre acreditado em mim, nos meus sonhos e por todos os sacrifícios que fizeram por mim. Obrigado por serem meu porto seguro e meus maiores exemplos.

Agradeço a minha irmã Swely, por ter estado sempre ao meu lado, pelo amor, pelo cuidado e por todo apoio.

Ao Prof. Dr. José Antônio Fernandes Macedo por ter aceitado ser meu orientador e por ter-me dado a oportunidade de fazer parte do Insight Data Science Lab.

Agradeço ao Regis Pires Magalhães, Nicksson Arrais, Francisco Carlos Júnior, a Lívia Almada e ao Lucas Peres por terem sido parte do time PyMove. Obrigada pela disponibilidade e toda a ajuda ao longo da elaboração deste trabalho.

Ao meu namorado, Carlos Rógerio agradeço pelos momentos felizes, pelo companheirismo e por ter estado comigo nos momentos difíceis. Obrigada por estar do meu lado, pela paciência, pela força e todo o apoio.

Agradeço a minha amiga Eveline Amado, pela amizade, pelos momentos compartilhados, pela força e por todo o apoio.

Agradeço a minha amiga Vanda Medina, que mesmo estando longe, sempre se fez presente. Obrigado pelos conselhos, pela amizade e pelo carinho.

Ao meu amigo Carlos Felix, pela amizade e pelos conselhos.

Aos meus amigos, Andreza Fernandes, Fernanda Bezerra, Pedro Nunes e Thais do Nascimento, por terem sido companheiros e tornado a jornada pela faculdade mais alegre. Agradeço ainda a Andreza Fernandes por todo o apoio, principalmente durante a elaboração do deste trabalho.

"Toda a manhã, na África, a gazela acorda. Ela sabe que precisa correr mais rápido que o mais rápido dos leões para sobreviver. Toda manhã um leão acorda. Ele sabe que precisa correr mais rápido que a mais lenta das gazelas senão morrerá de fome.

Não importa se você é um leão ou uma gazela. Quando o sol nascer, comece a correr."

(Desconhecido)

RESUMO

O volume de dados de trajetória teve um grande aumento nos últimos anos. Os avanços das tecnologias de coleta de dados de localização, o aumento do número de dispositivos que usam estas tecnologias, e a redução no custos destes dispositivos, contribuíram para o aumento no volume dos dados. Os dados de trajetórias ajudam a desenvolver os trabalhos de diferentes áreas do conhecimento. No entanto, na literatura é possível constatar que não existem muitas ferramentas cujo principal foco seja o pré-processamento de dados de trajetória. O presente trabalho tem como principal objetivo projetar uma biblioteca, extensível e flexível, para auxiliar na análise de dados de trajetória, tendo o seu maior foco nas operações de pré-processamento de dados. A biblioteca se chama PyMove e vem sendo desenvolvida na linguagem Python, já oferece suporte para as seguintes etapas do pré-processamento de dados: filtragem de ruído, detecção de pontos de parada, compressão, map-matching e segmentação. O PyMove tem como vantagem, em relação aos outros trabalhos relacionados, ser uma biblioteca flexível e possuir suporte para uma maior número de atividades de pré-processamento de dados. Entretanto o PyMove possui ainda poucas operações para a etapa de compressão de dados.

Palavras-chave: Trajetória. Análise de trajetória. Pré-processamento de trajetória. Filtragem de ruído. Detecção de pontos de parada. Compressão. Segmentação.

ABSTRACT

Trajectory data volume has increased greatly in recent years. Advances in location data collection technologies, an increase in the number of devices that use these technologies, and a reduction in the cost of these devices have all contributed to increased data volume. Trajectory data helps to develop work from different areas of knowledge. However, in the literature there are not many tools whose main focus is the preprocessing of trajectory data. The present work has as main goal of designing an extensible and flexible library to assist in the analysis of trajectory data, focusing on data preprocessing operations. The library is called PyMove and it has been developed in Python language, it supports the following stages of data preprocessing: noise filtering, stay point detection, compression, map-matching and segmentation. PyMove has the following advantages over other related work: It is a flexible library and supports more data preprocessing activities. However, PyMove has few operations for data compression.

Keywords: Trajectory. Trajectory analysis. Trajectory preprocessing. Noise Filtering, Stay Point Detection. Compression. Segmentation.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

GPS	Sistema de Posicionamento Global
PEP	*Python Enhancement Proposal*
STPM	*Semantic Trajectory Preprocessing Module*
WEKA	*Waikato Environment for Knowledge Analysis*

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos foram gerados grandes volumes de dados de trajetória. Estes dados podem representar movimentos de diferentes objetos, como: pessoas, veículos e animais. Diversos fatores contribuem para o aumento, a proliferação do uso de Sistema de Posicionamento Global (GPS) em diferentes dispositivos móveis, que contribuíram para a redução do custo da coleta (Andrienko *et al.*, 2017), o aumento do uso de postagem com localização em diferentes redes sociais (NOULAS *et al.*, 2012).

A análise dos dados de trajetória pode oferecer poderosos *insights* para diferentes áreas do conhecimento, ajudando a desenvolver os seus trabalhos. Hoje estes dados já são utilizados em aplicações como: sistemas de transporte inteligentes e computação urbana (ZHENG *et al.*, 2014); análise de mobilidade sustentável (Jia Wang; MORIARTY, 2018); aplicações na área da saúde, auxiliando na identificação de tumores em imagens Chandola *et al.* (2009), no rastreamento de fenômenos naturais e identificação de padrões de migração animal (ZHENG; XIE, 2011; ZHENG *et al.*, 2011).

Para que seja possível extrair informações dos dados e realizar análises consistentes dos mesmos, é preciso que um conjunto de atividades sejam executadas. Tais atividades têm diferentes objetivos, como remover dados que possam estar inconsistentes, reduzir o tamanho das trajetórias para diminuir os custos de processamento, adicionar informações aos dados. Além disso, elas constituem a etapa de pré-processamento de dados.

Embora existam diversas aplicações relevantes, é possível constatar que existe uma carência por ferramentas que ofereçam suporte para a etapa de pré-processamento de dados e as demais atividades de mineração de dados.

Este trabalho tem como principal objetivo desenvolver uma arquitetura que permita criar uma biblioteca extensível e flexível para auxiliar no processamento e análise de dados de trajetória. Reconhecendo a importância do pré-processamento de dados, este trabalho se propõe a desenvolver operações para o módulo de pré-processamento de dados do PyMove.

O trabalho possui os seguintes objetivos específicos:

- Realizar uma revisão bibliográfica, a fim de compreender o campo de mineração de dados de trajetória;
- Integrar funções previamente implementadas, por alunos de mestrados e doutorados do *Insight Data Science Lab* ao PyMove;
- Modelar e implementar a arquitetura interna da biblioteca;

- Padronizar e documentar as funções incorporadas ao PyMove;
- Refatorar o código das funções previamente implementadas, para atender as diretrizes da *Python Enhancement Proposal* (PEP)8.
- Implementar um mecanismo para validar os resultados deste trabalho.

As principais contribuições deste trabalho são: a arquitetura desenvolvida para a biblioteca PyMove e a implementação de novas funções para o módulo de pré-processamento.

O presente trabalho está estruturado da seguinte forma: o Capítulo 2, são apresenta os principais conceitos e técnicas envolvidas neste trabalho; o Capítulo 3 mostra os trabalhos relacionados ao presente trabalho; o Capítulo 4 detalha as etapas realizadas na elaboração deste trabalho; o Capítulo 5 apresenta a conclusão do presente trabalho, assim como propostas para os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os conceitos fundamentais utilizados neste trabalho. Primeiro são apresentados os conceitos básicos, em seguida são apresentadas as principais etapas e técnicas utilizadas nos trabalhos de mineração de dados de trajetória. Também são abordados as boas práticas de programação e as bibliotecas utilizadas neste trabalho.

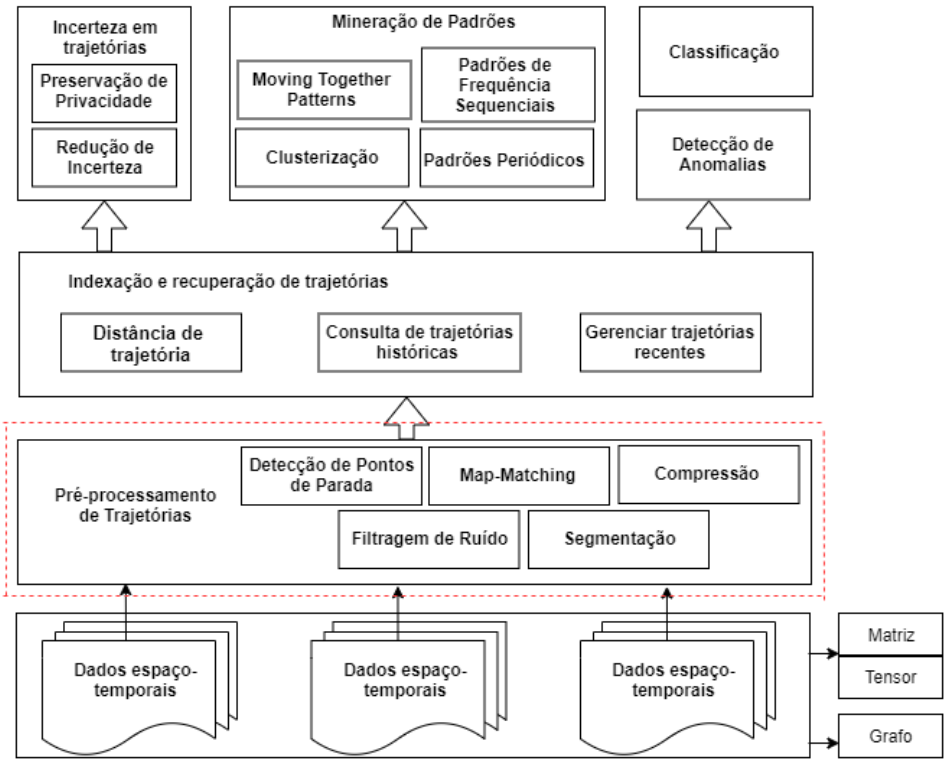
2.1 Trajetória

Uma trajetória é uma sequência de pontos cronologicamente ordenados, que representam o movimento de um objeto em um espaço geográfico. Cada ponto é formado por uma marcação de tempo (*timestamp*) e coordenadas geoespaciais, representadas por p =(x, y, t). Por exemplo: $p_1 \rightarrow p_2 \rightarrow ... \rightarrow p_k \rightarrow ... \rightarrow p_n$, onde n é o número de pontos na trajetória (ZHENG, 2015).

Trajetórias podem ser utilizadas na sua forma original, ou convertidas para outros formatos, como: grafo, matriz e tensor. Diferentes formas de representação, permitem que diferentes abordagens sejam utilizadas para a mineração de dados. A utilização de trajetórias na forma de grafos, facilita a implementação de aplicações que lidam com redes de ruas, por exemplo. Matrizes podem ser utilizadas para ajudar a complementar dados faltantes, ou na detecção de anomalias nos dados. Um tensor, é uma extensão de uma matriz, feito através da adição de uma terceira dimensão, que permite o armazenamento de informações adicionais. Esta forma de representação facilita a detecção de correlação entre objetos. Entretanto, a conversão dos dados para as formas de representação acima apresentadas, não é uma tarefa trivial. Quando se converte dados de trajetória para grafos, é preciso definir o que representa um nó e o que representa uma aresta. Por outro lado, tratando-se de uma matriz, é preciso definir o que é representado por uma linha, uma coluna e uma entrada (ZHENG, 2015).

Zheng (2015) propõe um paradigma que serve como um guia para realizar trabalhos de mineração de dados de trajetória. A Figura 1 apresenta uma versão adaptada deste paradigma. Na versão original são consideradas outras atividades de mineração, que não são explanadas neste trabalho, por não fazerem parte do escopo deste trabalho. Nesta figura encontra-se destacada a caixa que representa o pré-processamento de dados, pois nela se concentra o foco deste trabalho. Ao longo da seguinte seção, são exploradas algumas das atividades representadas na Figura 1.

Figura 1 – Um paradigma para mineração de trajetória.



Fonte: adaptado de (ZHENG, 2015)

2.1.1 Pré-processamento

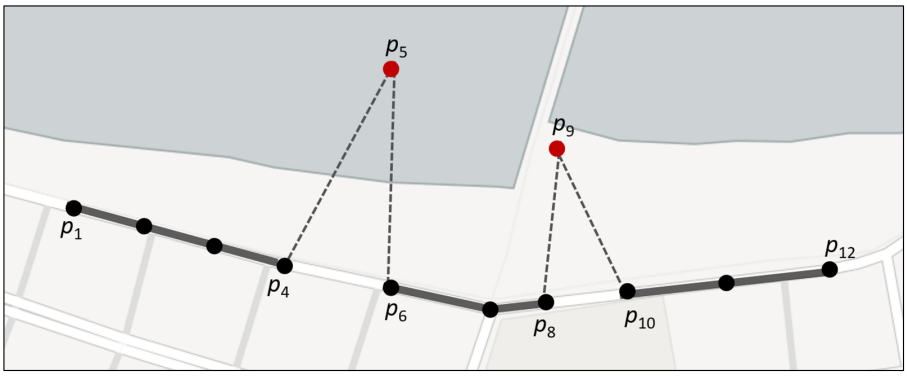
Para Zheng (2015) o pré-processamento de dados é definido como uma etapa de extrema importância na mineração dos dados. Nela são conduzidas atividades, visando melhorar a qualidade do dados para obtenção de melhores resultados em processamentos e análises futuras que podem ser realizadas. Zheng (2015) define quatro técnicas básicas para esta etapa: filtragem de ruído, detecção de pontos de parada, compressão e segmentação de trajetória.

2.1.1.1 Filtragem de Ruído

Alguns pontos da trajetória podem estar inconsistentes devido a problemas nos dispositivos de coleta de dados, na comunicação, ou outros fatores externos, tais pontos são denominados ruídos. Os pontos p_5 e p_9 na Figura 2 são, neste contexto, dados com ruído. Em casos em que o erro é pequeno e o ponto não esteja muito distante da rota da trajetória, o problema pode ser resolvido usando algoritmos que fazem o alinhamento dos pontos com a rede de ruas. Porém, em situações nas quais o erro é grande, é preciso utilizar diferentes técnicas para filtrar esses pontos da trajetória, antes de começar as etapas posteriores. As principais técnicas

de filtragem podem ser divididas em 3 grandes categorias: *Mean (or Median) Filter*, *Kalman and Particle Filters* e *Heuristics-Based Outlier Detection* (ZHENG, 2015).

Figura 2 – Trajetória com pontos lidos incorretamente.



Fonte: (Gomes *et al.*, 2018)

Mean (or Median) Filter. Nestas técnicas a média ou a mediana são usadas para aproximar o real valor de um ponto com ruído na trajetória. Usando o próprio ponto e os $n-1$ pontos da trajetória que o precedem cronologicamente, sendo n um valor escolhido pelo usuário. Em casos cuja taxa de amostragem é muito pequena, ou quando existem vários pontos com ruídos em sequência, estas técnicas não são as melhores escolhas, uma vez que o erro entre a real posição do ponto e o valor estimado pode ser muito grande. A média e a mediana obtêm melhores resultados quando aplicadas em pontos de ruído isolados e em trajetórias com representações densas (ZHENG, 2015).

Kalman and Particle Filters. Os filtros de Kalman e partículas estimam uma trajetória a partir de medições de grandezas realizadas durante um período de tempo. As estimativas produzidas por estes filtros obedecem as leis da física. Apesar dos dois algoritmos conseguirem fazer estimativas, que se aproximam do real valor da trajetória, os seus resultados são dependentes da medição de uma posição inicial. A eficiência dos dois algoritmos é prejudicada em situações em que o primeiro ponto da trajetória possui ruído.

Heuristics-Based Outlier Detection. Nesta abordagem, pontos com ruído são removidos da trajetória com o auxílio de algoritmos de detecção de *outliers*. Podem ser usados algoritmos baseados em distância. Nestes, um ponto é considerado um *outlier* se o número de vizinhos que ele possui a uma distância d é menor que um valor p preestabelecido. Os valores de d e p são baseados em heurísticas. Após a detecção, os pontos são eliminados. Pontos com ruído podem ser considerados *outliers* numa trajetória, uma vez que eles possuem características que se diferenciam do restantes dos pontos comuns (não ruidosos) e aparecem em menor quantidade

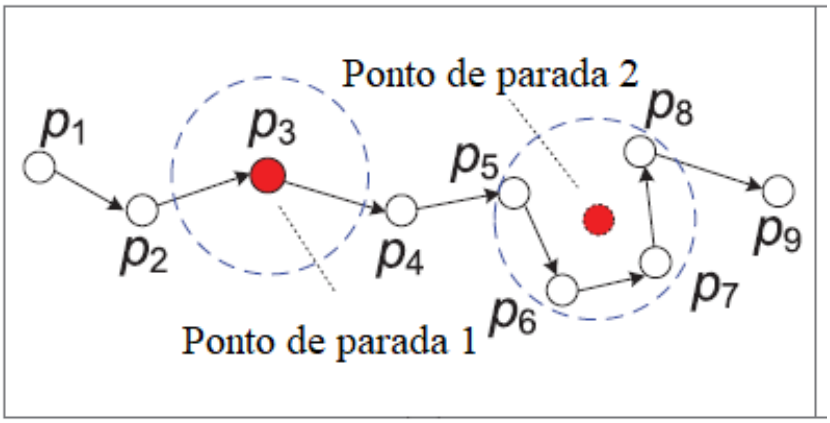
(ZHENG, 2015).

2.1.1.2 Detecção de pontos de parada

Os pontos da trajetória possuem diferentes significados semânticos. Pontos onde os objetos ficaram parados ou se movimentando ao redor, por um período de tempo mais frequente, são chamados pontos de parada. Estes pontos podem representar: shopping, atrações turísticas, postos de gasolina, restaurante, entre outros. Usando os pontos de parada podemos representar uma trajetória como uma sequência de pontos com significado: $casa \xrightarrow{\Delta t1} shopping \xrightarrow{\Delta t2}, .., \xrightarrow{\Delta tn} restaurante$, onde Δti representa o intervalo de tempo entre os pontos (ZHENG, 2015).

A Figura 3 apresenta diferentes tipos de pontos de parada. O tipo de ponto representado pelo Ponto de parada 1 (Figura 3) é mais raro e acontece quando o ponto de parada é representado por um único ponto. O Ponto de parada 2 (Figura 3) ilustra um ponto de parada em que o objeto se moveu ao redor. No processo de identificação de pontos de parada, é preciso levar em consideração que os dispositivos de coleta de dados de localização podem gerar diferentes leituras, mesmo o objeto estando parado na mesma localização.

Figura 3 – Trajetória com os pontos de parada destacados em vermelhos.



Fonte: (ZHENG, 2015)

De acordo com o objetivo da aplicação, pode ser necessário eliminar os pontos de parada da trajetória. Em aplicativos como os de recomendação de táxis ou de destinos de viagens, os pontos de parada ajudariam na sua implementação. Porém tratando-se de uma aplicação para calcular o tempo de viagem de um percurso, teriam de ser eliminados, pois a este aplicativo, não interessa o tempo que o usuário permaneceu estacionário (ZHENG, 2015). Li *et al.* (2008) propuseram um algoritmo para detectar pontos de parada baseado em distância e intervalo de tempo.

2.1.1.3 Compressão de trajetórias

Dispositivos podem gerar trajetórias com taxa de amostragem maior do que a necessária para a resolução do problema da aplicação. A taxa de amostragem influencia o custo de computação e armazenamento dos dados. Utilizando as técnicas de compressão de trajetórias é possível reduzir o tamanho de uma trajetória, sem diminuir a sua utilidade (ZHENG, 2015). As duas principais estratégias para compressão de trajetória são: compressão offline e compressão online.

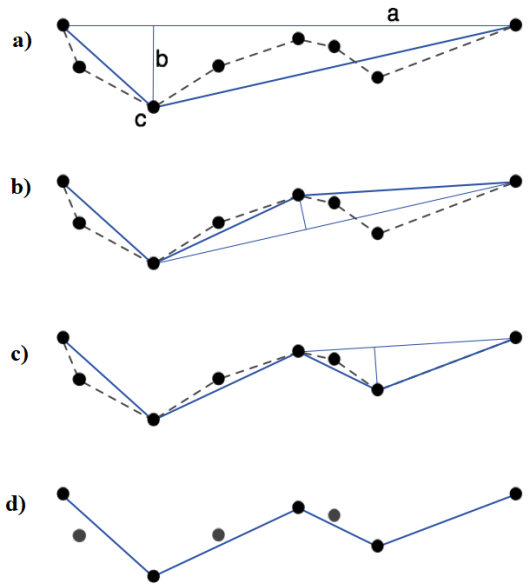
Compressão online: nesta estratégia as trajetórias são comprimidas simultaneamente com a sua geração. O desafio é decidir se um novo ponto coletado, deve ou não ser adicionado à trajetória. Os métodos de compressão podem ser divididos em duas categorias: algoritmos *window-based* e algoritmos baseados em *features* como: velocidade e direção (ZHENG, 2015).

Compressão offline: usando esta estratégia as trajetórias são comprimidas apenas após terem sido completamente geradas. O algoritmo de Douglas-Peucker é uma das técnicas que pode ser utilizada para este tipo de compressão. Ele inicia traçando um segmento de reta ligando o primeiro e o último ponto da trajetória (Figura 4a). O algoritmo, então calcula a distância entre os pontos e o segmento de linha (Figura 4b). Se a distância entre o ponto mais afastado, em relação à linha, for maior que um valor de erro e preestabelecido, então este ponto é utilizado para subdividir a trajetória original em duas partes (Figura 4c). O algoritmo é aplicado recursivamente nas subdivisões, até que o erro seja menor do que e, gerando assim uma aproximação para as subdivisões. A Figura 4d apresenta o resultado da execução do algoritmo, onde foi gerada uma trajetória comprimida, formada por um subconjunto de pontos da trajetória original (HERSHBERGER; SNOEYINK, 2000; ZHENG, 2015).

2.1.1.4 Segmentação de trajetórias

Nesta etapa, a trajetória é subdividida em segmentos, de tal forma que os dados pertencentes ao mesmo segmento tenham comportamento homogêneo (YOON; SHAHABI, 2008). A segmentação ajuda a descobrir padrões sobre os dados, diminui a complexidade computacional e é um passo obrigatório para processos de análise, como a clusterização e a classificação (ZHENG, 2015). As trajetórias podem ser segmentadas considerando três critérios: intervalo de tempo, geometria e semântica. Quando a trajetória é segmentada baseada no tempo, dois pontos consecutivos são separados em duas sub trajetórias, se o intervalo de tempo entre

Figura 4 – Execução do algoritmo de Douglas-Peucker.



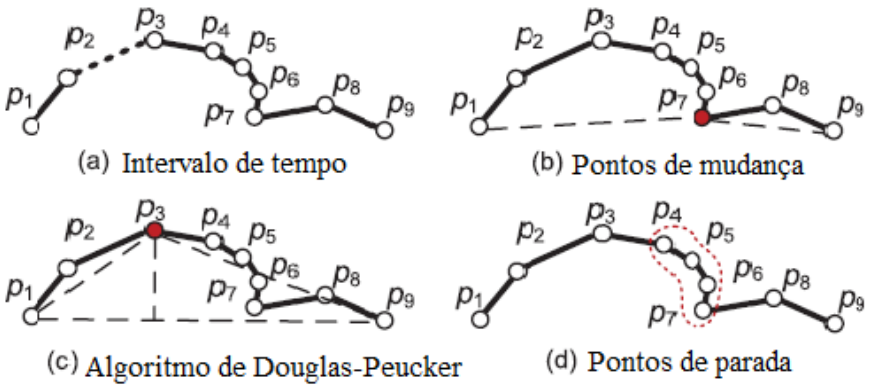
Fonte: adaptado de (GRAELLS-GARRIDO; SAEZ-TRUMPER, 2016)

eles for maior que um valor preestabelecido. Como acontece na Figura 5(a), em que a trajetória é dividida nos segmentos $p_1 \rightarrow p_2$ e $p_3 \rightarrow, ..., \rightarrow p_9$. Existem diversas abordagens que podem ser seguidas para segmentar trajetórias levando a geometria em consideração. Uma delas é baseada em pontos de mudança, neste método a segmentação é feita identificando os pontos onde a trajetória muda de direção, considerando um limiar mínimo. Na Figura 5(b), o ponto de virada é o p_7, podendo ser utilizado para dividir a trajetória em duas. Como alternativa, é possível utilizar os pontos chaves que mantém a forma da trajetória para segmentá-la, os pontos podem ser descobertos utilizando o algoritmo de Douglas-Peucker, conforme mostra a Figura 5(c). A Figura 5(d) mostra uma forma de aplicação da segmentação baseada no valor semântico. Nesta técnica, a divisão é feita utilizando os pontos de parada que, dependendo do objetivo da aplicação que irá utilizar os dados, podem ou não ser mantidos após a divisão (ZHENG, 2015).

2.1.1.5 Map-Matching

Como mencionado na seção de filtragem de dados, os pontos gerados de uma trajetória podem não corresponder precisamente à real localização de onde o objeto em movimento passou (Figura 6(a)). Esta falta de precisão não é desejável para alguns tipos de aplicações, entre elas: as que trabalham com análise de fluxo de trânsito ou serviços de navegação (ZHENG, 2015). Para resolver esses e outros problemas, são aplicados algoritmos de *map-matching*, que têm como objetivo alinhar os pontos da trajetória com a rede de ruas Figura 6(b) (LOU *et al.*, 2009). Alguns fatores que aumentam a complexidade de sua implementação são: a taxa de

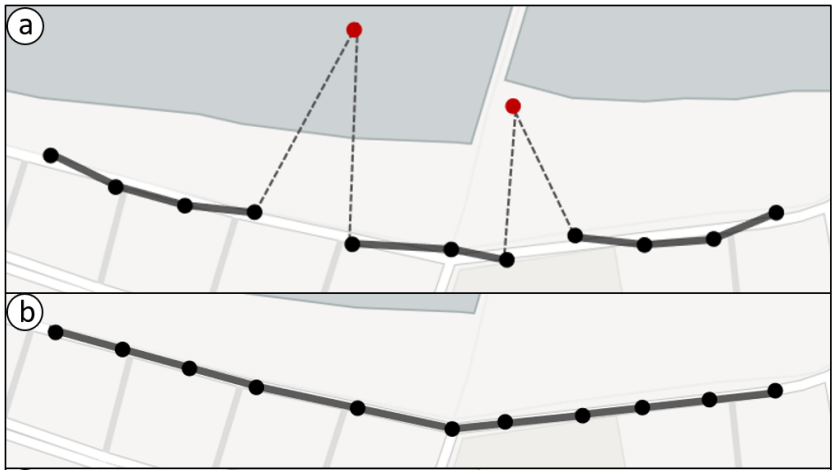
Figura 5 – Segmentações baseadas em diferentes critérios.



Fonte: (ZHENG, 2015)

amostragem, a acurácia da leitura dos pontos (LOU *et al.*, 2009), características da rede de ruas, como cruzamentos ou estradas paralelas (ZHENG, 2015).

Figura 6 – Uma trajetória, antes e depois da aplicação do *map-matching*.



Fonte: adaptado de (Gomes *et al.*, 2018)

Zheng (2015), Lou *et al.* (2009) classificam os algoritmos de *map-matching* em duas categorias, dependendo da taxa de amostragem de pontos utilizado pelo método: local/incremental e global. No método local ou incremental, são mapeados uma posição ou um conjunto de posições vizinhas por vez ao mapa, enquanto os métodos globais mapeiam a trajetória completa. Algoritmos globais são mais precisos, porém requerem um maior poder computacional. Os algoritmos locais são mais rápidos, mas a sua acurácia diminui quando a taxa de amostragem da trajetória é pequena.

2.1.2 Indexação e recuperação de trajetórias

No cenário atual, onde existe um grande volume de dados e aplicações que requerem processamento muito rápido dos dados, são necessários algoritmos que consigam rapidamente recuperar trajetórias que satisfaçam os critérios definidos pelas aplicações. A criação de índices, que permitem acessar diretamente as trajetória ou partes delas, também é uma parte importante do processo.

2.1.3 Mineração de padrões

Diferentes padrões de mobilidade podem ser identificados numa trajetória individual ou em um conjunto de trajetórias. Zheng (2015) faz a divisão dos padrões de mobilidade de trajetória em quatro categorias: *Moving Together Patterns*, Clusterização de trajetórias, Padrões de Frequências Sequenciais e Padrões Periódicos.

Na categoria *Moving Together Patterns* são identificados grupos que se movem juntos, por um intervalo pré-determinado de tempo. Critérios são empregados para que diferentes tipos de sub-grupos dessa categoria possam ser identificados, como: a densidade do grupo, a existência de um formato definido pelo usuário, o período de duração do padrão e/ou se ocorreu por um intervalo contínuo de tempo. Aplicações que realizam análise de padrões migratórios de espécies, de fluxo de trânsito e predição de futuros movimentos de objeto, são algumas das aplicações que se beneficiam com a identificação desse tipo de padrão.

Outra categoria é a de clusterização. Ela se preocupa em agrupar trajetórias que possuem características similares, com o objetivo de facilitar o processo de análise de padrões e características partilhadas por diferentes objetos em movimento. Em aplicações como recomendação de trajetória, predição de localização, compressão de dados, é útil descobrir grupos de objetos que partilhem a mesma sequência de localização, num período similar de tempo. Os pontos da sequência não necessariamente precisam ser pontos consecutivos na trajetória original. Este tipo de padrão se encaixa na categoria *Padrões de Frequências Sequenciais*. A última categoria é a de *Padrões Periódicos*, que inclui os padrões que se repetem periodicamente no movimento de um objeto. A análise deste tipo de padrão auxilia na previsão de comportamentos futuros do objeto e na compressão de dados (ZHENG, 2015).

2.1.4 Incerteza em trajetórias

Por mais precisa que seja a leitura de um dispositivo de leitura de dados, sempre existirá um intervalo de tempo entre uma leitura e outra. Intervalos entre as leituras também podem ser propositalmente introduzidos para reduzir o custo de comunicação ou o consumo de bateria dos dispositivos. Por esses e outros fatores, existem pontos onde a localização do objeto em movimento é incerta ou até mesmo desconhecida. Quando o intervalo entre as leituras é muito grande, resultando em uma trajetória com baixa taxa de amostragem chamamos a trajetória resultante de trajetória incerta. Para algumas aplicações, como recomendação de viagens ou controle de tráfico, é preciso reduzir a incerta das trajetórias, para poder obter resultados consistentes. Em contrapartida, em alguns casos a incerteza é propositalmente introduzida para proteger a privacidade do usuário. Pois, através dos dados de trajetória é possível inferir diversas informações sobre o usuário, como: onde mora, lugares que costuma frequentar, o tempo médio de seus pontos de parada. Estes tipos de informações podem ser utilizadas de forma maliciosa (ZHENG, 2015).

2.1.5 Classificação

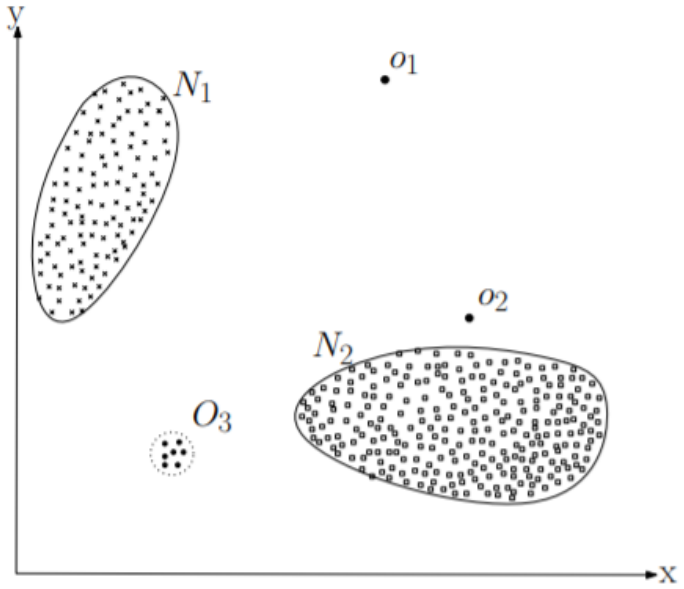
Através da etapa de classificação, é feito um enriquecimento semântico das trajetórias, ou segmentos de trajetórias, que beneficia diferentes tipos de aplicações. Isto se torna possível devido às técnicas de aprendizagem supervisionada. Trajetórias podem se enquadrar em diferentes categorias, usando critérios como: meio de transporte, atividades humanas e movimento. A segmentação é um pré-requisito básico para a classificação. Na maioria dos algoritmos existentes, o primeiro passo é segmentar a trajetória para, posteriormente extrair as *features* e classificar cada segmento utilizando um modelo de classificação (ZHENG, 2015).

2.1.6 Detecção de anomalias

Podem existir subconjuntos dos dados cuja distribuição, se afasta muito da distribuição normal do restantes dos dados. Esses subconjuntos são chamados anomalias ou *outliers*. Eles podem ser segmentos de trajetória, uma única trajetória ou até mesmo um conjunto de trajetórias. Na Figura 7, N_1 e N_2 são consideradas as regiões normais/padrão, pois nelas está a maior concentração de dados. Os pontos o_1, o_2 e o conjunto de pontos em O_3 são considerados anomalias, uma vez que se distanciam significativamente de N_1 e N_2s (CHANDOLA *et al.*,

2009).

Figura 7 – Os pontos os o_1, o_2 e o conjunto de pontos em O_3 são considerados *outliers* neste conjunto de dados.



Fonte: (CHANDOLA *et al.*, 2009)

A etapa de detecção de anomalia, preocupa-se em encontrar trajetórias ou segmentos que possuem comportamento anormal em relação ao conjunto de dados como um todo. Suas aplicações envolvem muitas áreas. Na saúde, por exemplo, ela auxilia na identificação de tumores em análises. A detecção de anomalia, também pode auxiliar em atividades como: detecção de fraudes, identificação de falhas em sistemas críticos, serviços de vigilância militar (CHANDOLA *et al.*, 2009).

2.2 Boas práticas de programação

As boas práticas de programação são convenções e diretrizes aceitas e geradas pela comunidade de programação, que procuram melhorar a legibilidade de código fonte e contribuir com a geração de um produto final simples, eficiente e elegante.

2.2.1 PEP 8

As PEP [1] são documentos que define convenções de codificação para o código Python. O objetivo das diretrizes da PEP é guiar o programador na elaboração de um código legível e consistente com outros códigos da comunidade. A PEP 8 é uma versão da glsPEP.

[1] https://www.python.org/dev/peps/pep-0008/

Este guia de boas práticas está em constante evolução, uma vez que novas convenções são identificadas e outras são depreciadas em razão de mudanças da própria linguagem Python.

2.3 Padrões de projeto

Padrões de projeto são soluções reutilizáveis, para um problema que se repete com frequência num contexto particular. Os padrões de projeto ajudam a incorporar boas práticas de programação ao código, uma vez que eles apresentam soluções, que já foram validadas e evoluídas diversas vezes ao longo do tempo (GAMMA *et al.*, 1995).

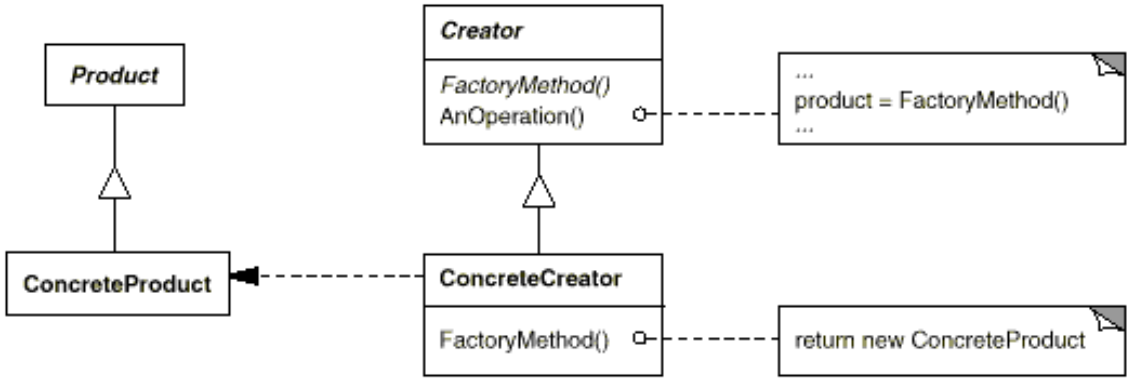
2.3.1 Factory Method

Gamma *et al.* (1995), definiram o *Factory Method* como sendo um padrão de projeto que delega as subclasses a responsabilidade de decidir qual classe instanciar, onde apenas a interface para a criação de objetos é definida a priori.

A Figura 8 apresenta as classes participantes do padrão e as ligações existentes entre elas, sendo as classes:

- **Product**: que define a interface a ser seguida para a criação de objetos.
- **ConcreteProduct**: implementa a interface definida por Product.
- **Creator**: é uma classe abstrata que define o método *factory*, utilizado pelas subclasses para criar um novo produto.
- **ConcreteCreator**: que retorna uma instância do ConcreteProduct.

Figura 8 – Estrutura do padrão de projeto *Factory Method*.



Fonte: (GAMMA *et al.*, 1995)

2.4 Pandas

O Pandas pandas é uma biblioteca de código aberta, implementada na linguagem de programação Python e publicada sob a linceça BSD. Esta biblioteca foi projetada para preencher a lacuna que existia no Python, por um ambiente para análise de dados de alta performance e produtividade. Ela oferece um conjunto de estruturas de dados e ferramentas, que para ao usuário realizar os diferentes processos e etapas necessárias na análise de dados.

2.5 Dask

O Dask é uma biblioteca *open source* para análise de dados e computação paralela desenvolvida na linguagem Python (ROCKLIN, 2015). Esta biblioteca foi projetada para ser flexível, rápida, escalável e responsiva. As estruturas paralelas do Dask, imitam a sintaxe das coleções do Pandas, do Numpy (OLIPHANT, 2010) e de iteradores do Python. Por exemplo, o Dask Array e o Dask Bag, imitam a sintaxe do Pandas e do Numpy, respectivamente. O Dask Bag imita iteradores, Tools[2] e o PySpark[3].

[2] https://toolz.readthedocs.io/en/latest/api.html
[3] https://spark.apache.org/docs/latest/api/python/index.html

3 TRABALHOS RELACIONADOS

O presente trabalho se propõe a desenvolver uma biblioteca para auxiliar no processamento e análise de dados de trajetória, tendo o seu maior foco, no pré-processamento de dados. Os trabalhados apresentados neste capítulo possuem uma proposta similar.

O primeiro trabalho abordado é o de Pappalardo *et al.* (2019), que propõe uma ferramenta para análise de dados de mobilidade humana. É também apresentado, o trabalho de Ruan *et al.* (2018) que é uma solução online, para auxiliar na etapa de pré-processamento de dados. Por fim é descrito, o trabalho de Bogorny *et al.* (2011), que tem seu foco no enriquecimento semântico das trajetórias.

3.1 Scikit-mobility

O Scikit-mobility é uma biblioteca Python *open source* que oferece suporte para análise de dados de mobilidade humana. A biblioteca permite ao usuário representar trajetórias e fluxos de movimentação, realizar atividades de pré-processamento de dados, analisar os riscos de privacidade associados a análise de um conjunto de dados, gerar trajetórias e fluxos de mobilidade sintéticos, extrair métricas e padrões dos dados (PAPPALARDO *et al.*, 2019).

A biblioteca estende o *DataFrame* do Pandas, descrito na Seção 2.4, para criar duas estruturas de dados básicas, sendo elas:

- *TrajDataFrame*: estrutura projetada para representar um conjunto de trajetórias. Cada linha do TrajDataFrame representa um ponto da trajetória. Obrigatoriamente, os dados devem conter os seguintes atributos: latitude, longitude e datetime. O usuário pode adicionar outros atributos de acordo com a sua necessidade.
- *FlowDataFrame*: representa os fluxos de movimentação de objetos entre duas localizações. O usuário não possui limitações para a adição de atributos, porém três atributo obrigatórias devem constar nos dados, sendo elas: origin, destination e flow. Onde origin e destination definem o início e o final do fluxo respectivamente. O flow define o número de objetos que percorrem o fluxo.

Atualmente a biblioteca oferece suporte para as seguintes etapas do pré-processamento de dados, descritas na Secção 2.1.1: filtragem de ruídos, detecção de pontos de parada e compressão. O Scikit-mobility disponibiliza as seguintes funcionalidades:

- Filtragem de ruídos, na função oferecida pela biblioteca, um ponto é filtrado da trajetória,

se a velocidade entre o ponto e o ponto que o antecede, for maior que o especificado pelo usuário;

- Detecção de pontos de parada, com base no tempo que um usuário permanece a uma distância *p* de um ponto da trajetória. Opcionalmente o atributo leaving_datetime, pode ser adicionado ao TrajDataFrame, para indicar o momento em que objeto saiu do ponto de parada;
- Compressão de dados: usada para fundir pontos que estejam separados por uma distância menor que um valor p, definido pelo usuário.

Pappalardo *et al.* (2019) apresentam duas funcionalidades que o scikit-mobility pretende disponibilizar para o futuro. A primeira é expandir a ferramenta com a finalidade de oferecer um maior suporte a análise de dados de mobilidade. Eles propõem a adição de algoritmos predição e de *map-matching*, descrito na Seção 2.1.1.5. A segunda funcionalidade é aumentar a escalabilidade da biblioteca, através da do uso de bibliotecas que oferecem maior eficiência computacional, como o Dask, descrito na Seção 2.5.

3.2 CloudTP: A Cloud-Based Flexible Trajectory Preprocessing Framework

O CloudTP é uma ferramenta online, para pré-processamento de dados de trajetória. Ela gera trajetórias organizadas, a partir de dados brutos de GPS carregados pelos usuários. A ferramenta provê uma interface que permite ao usuário remover: os dados processados, estatísticas sobre o conjunto de dados e visualizações para auxiliar em trabalhos futuros (RUAN *et al.*, 2018).

A Figura 9 mostra uma visão em alto nível da estrutura e do funcionamento da ferramenta. Ela utiliza o armazenamento em nuvem e o Spark[1] para melhorar o seu desempenho, uma vez que trabalha com grandes volumes de dados. O CloudTP oferece suporte para as seguintes etapas do pré-processamento: filtragem de ruído, segmentação de trajetórias, *map matching* e criação de índices.

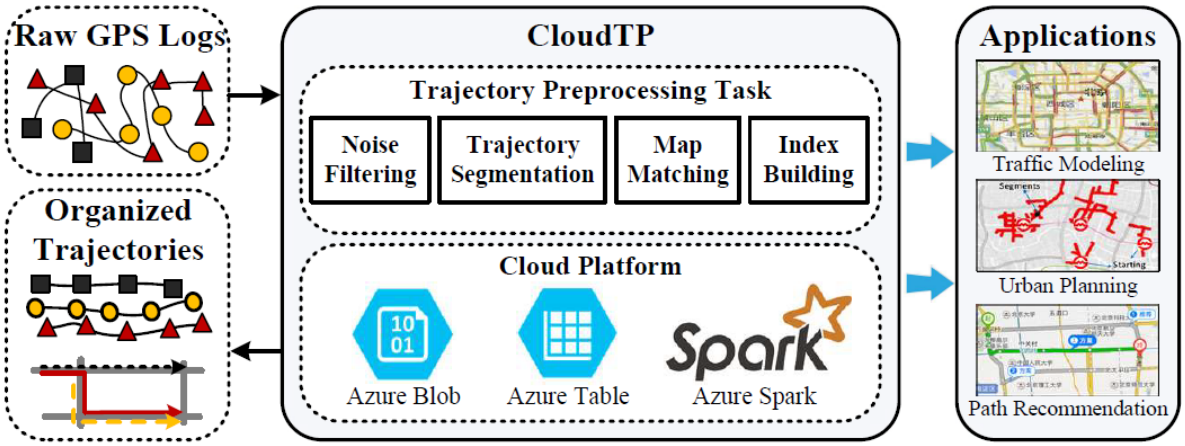
Esta é uma ferramenta considerada flexível por dois motivos: todos os parâmetros relacionados com o modo de transporte são configuráveis e usuários podem customizar os algoritmos disponibilizados pelo sistema, adaptando-os às suas necessidades.

A ferramenta prevê que dois tipos de usuários irão utilizar o sistema:

- O usuário comum, que apenas carrega os dados na ferramenta e utiliza as operações

[1] https://spark.apache.org/

Figura 9 – Visão alto nível da estrutura e funcionamento do CloudTP.



Fonte: (RUAN *et al.*, 2018)

predefinidas da ferramenta. Ele consegue obter trajetórias organizadas, estatísticas e diferentes visualizações;

- O usuário avançado pode customizar as operações oferecidas pelo sistema, adequando-as a sua necessidade;

3.3 WEKA-STPM

Witten *et al.* (2009) apresentam a WEKA[2], uma coleção de algoritmos de aprendizado de máquina para mineração de dados. Ela também provê suporte para o pré-processamento de dados e visualização dos mesmos. Esta ferramenta é *open source* e desenvolvida na linguagem JAVA[3]. Bogorny *et al.* (2011) projetaram e desenvolveram o WEKA-STPM, que é uma extensão do WEKA.

A Figura 10 ilustra uma visão global da arquitetura WEKA-STPM. No nível inferior, aparecem os repositórios de dados, entre eles: os dados brutos, os dados de trajetória enriquecidas pontos de parada, dados espaciais, padrões. *Semantic Trajectory Preprocessing* engloba 4 módulos, sendo eles:

- *Trajectory Cleaning*: oferece funcionalidades que permitem ao usuário realizar filtragem de ruídos e compressão dos dados. A ferramenta oferece um conjunto de critérios, através dos quais a trajetória pode ser filtrada, como: o limite de velocidade entre os pontos, número mínimo de ponto por trajetórias. Para a segmentação, algumas das possibilidades são: remover aleatoriamente uma percentagem dos dados, ou a cada dois pontos, remover

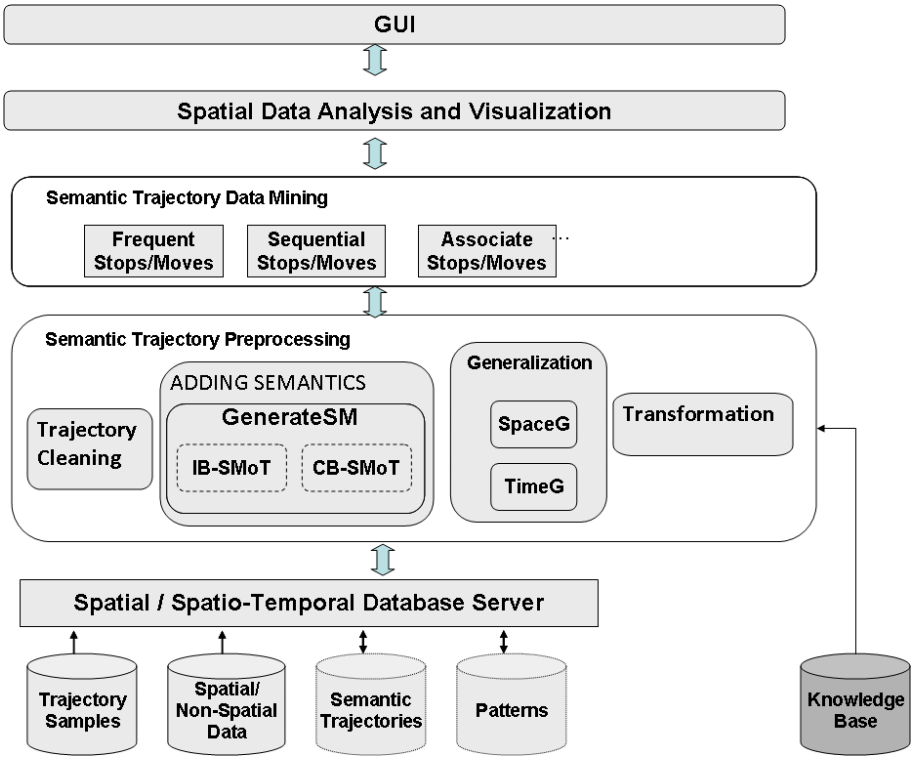
[2] https://www.cs.waikato.ac.nz/ml/WEKA/
[3] https://www.oracle.com/technetwork/java/javase/documentation/index.html

um.

- *Adding Semantics*: disponibiliza funções que permitem a detecção de pontos de parada e movimento nas trajetória, que permitem realizar o enriquecimento semântico. O usuário pode inclusive, adicionar novas funções para detectar pontos de parada a este módulo. .
- *Generalization*: realiza a agregação dos dados da trajetória em um nível mais alto de abstração. O usuário pode especificar diferentes níveis de granularidade. Por exemplo: se o usuário decidisse considerar, intervalos de tempo de uma hora. um evento que acontece às 18:05 e um outro que acontece às 18:50, seriam considerados como estando no mesmo período de tempo. Esta passo, auxilia na detecção de padrões nos dados em etapas futuros da mineração de dados.
- *Transformation*: converte os dados gerados pelo módulo de *Generalization* para o formato requerido pelo algoritmo ou ferramenta de mineração de dados escolhido pelo usuário.

Terminada a etapa de pré-processamento, podem ser aplicados diferentes algoritmos de mineração e diferentes padrões podem ser identificados através dessas analises. A WEKA-STPM, possui interface gráfica, que permite visualizar os dados no mapa.

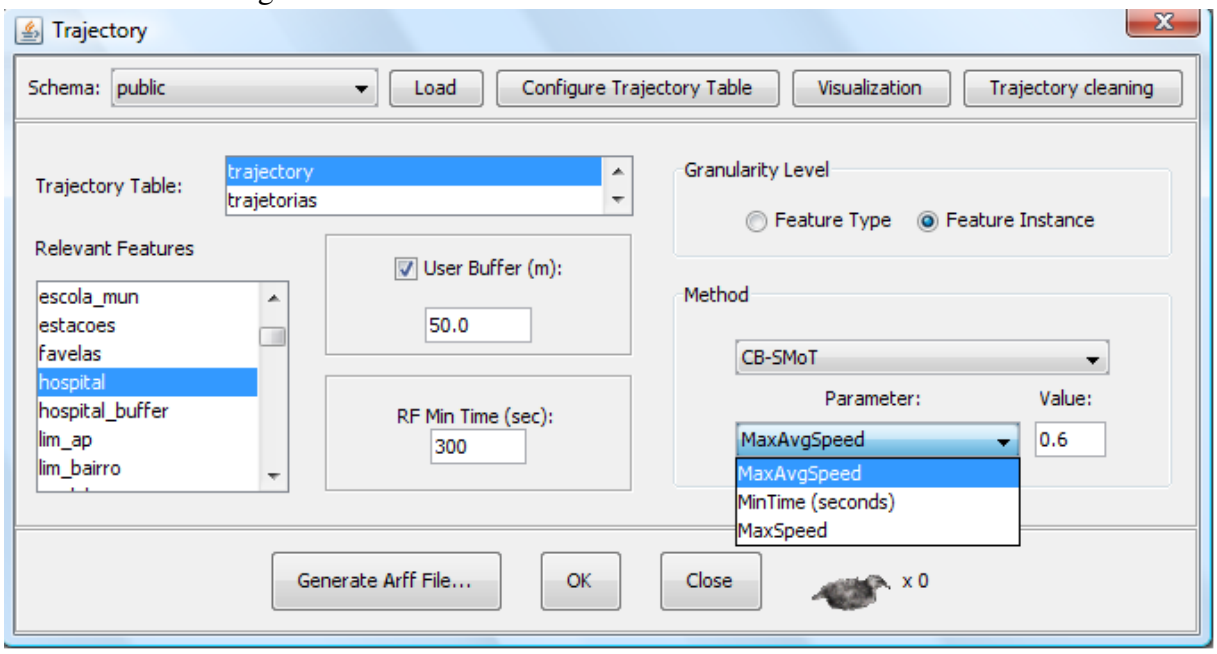
Figura 10 – Visão alto nível da arquitetura do WEKA -STPM.



Fonte: (BOGORNY *et al.*, 2011)

A Figura 11 representa a tela principal do WEKA -STPM, que é uma extensão da tela do WEKA.

Figura 11 – A tela inicial do WEKA -STPM.



Fonte: (BOGORNY *et al.*, 2011)

3.4 Quadro comparativo

Dentre os trabalhos apresentados neste capítulo, o Scikit-mobility é o único que oferece suporte para trabalhar para técnicas realizar avaliação de riscos de privacidade. O trabalho de Pappalardo *et al.* (2019) é o que mais se assemelha ao PyMove, entretanto o PyMove foi projetado para ser flexível e extensível, o Scikit-mobility não. O CloudTP, é a única das ferramentas, que atualmente oferece processamento paralelo e é escalável. Estas propriedades ainda estão em desenvolvimento no PyMove e não são oferecidas pelas outras duas bibliotecas. Porém, o CloudTP não está disponível para download e não oferece suporte para compressão de dados e detecção de pontos de parada. A Weka-STPM tem como principal foco a o enriquecimento semântico das trajetórias. Dentre as ferramentas, é a que mais se foca nos trabalhos com detecção de pontos de parada.

O Quadro1 apresenta as principais características de cada trabalho relacionado apresentado, assim como as deste trabalho.

Quadro 1 – Comparativo das características dos trabalhos.

Trabalho	**Plataforma**	**Suporte para pré-processamento**	**Extensibilidade**	**Paralelismo e escalabilidade**	**Disponibilidade**
Scikit-mobility	Biblioteca Python	Filtragem de ruído Compressão Detecção de pontos de parada	Não se aplica	Não é escalável e não oferece suporte para processamento paralelo	Disponível
CloudTP	Ferramenta Java online	Filtragem de ruído Segmentação Map-matching	Permite que o usuário adicione novas operações	Oferece suporte para processamento paralelo e é escalável	Não se aplica
Weka-STPM	Protótipo Java	Filtragem de ruído Compressão Detecção de pontos de parada	Permite que o usuário adicione novas operações	Não é escalável e não oferece suporte para processamento paralelo	Disponível
Este trabalho	Biblioteca Python	Filtragem de ruído Compressão Segmentação Map-matching Detecção de pontos de parada	Permite adição de novas estruturas de dados internas	Em desenvolvimento	Disponível

Fonte: elaborado pelo autor.

4 MODELAGEM, DESENVOLVIMENTO E VALIDAÇÃO DA BIBLIOTECA PYMOVE

Neste capítulo são apresentadas as etapas do desenvolvimento deste trabalho, essas etapas são ilustradas na Figura 12. Tais capítulo está dividido em três seções: modelagem, desenvolvimento e estudo de caso.

Antes do início das demais etapas do desenvolvimento do PyMove, foi realizado um estudo na literatura para entender o problema de mineração dos dados de trajetória, compreender as principais técnicas utilizadas e suas aplicações. Além disso, foi conduzido um estudo sobre as bibliotecas disponíveis na literatura para auxiliar nos trabalhos com trajetória, a fim de compreender as funcionalidades que elas oferecem e o que poderia ser melhorado.

O próximo passo foi a elaboração de uma modelagem que permitisse criar uma biblioteca extensível e escalável para auxiliar no processamento de dados de trajetória. Nesta etapa, são definidos a estrutura de dados interna e os módulos básicos para a biblioteca. Detalhes da modelagem são descritos na Seção 4.1 deste capítulo.

Após a modelagem, foram iniciados os trabalhos de implementação e refatoração dos códigos pertencentes ao PyMove. A primeira atividade realizada nesta etapa, foi a distribuição das funções que tinham sido previamente implementadas, nos módulos definidos para o PyMove. Terminada a distribuição, foi implementado o módulo *core*, seguido pela implementação e adição de novas funcionalidades ao módulo de filtragem de ruido e segmentação. Concluídas as implementações foram realizadas a revisão, refatoração e documentação dos códigos pertencentes a PyMove. Foram também elaborados tutoriais, com o objetivo de auxiliar no uso da biblioteca. Todas as etapas descritas neste paragrafo, são explanadas em detalhes na Seção 4.2.

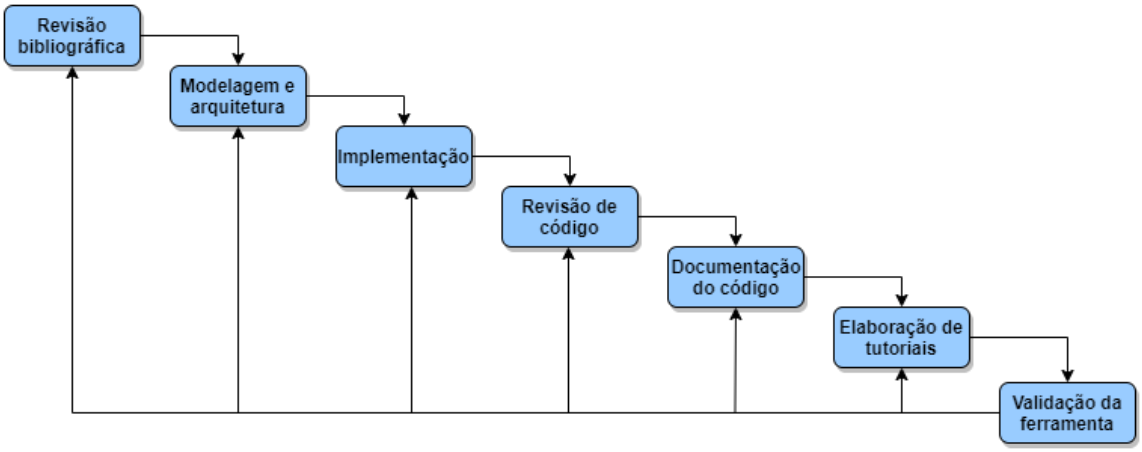
Na Seção 4.3, é apresentado um estudo de caso, elaborado com o objetivo de validar a biblioteca proposta.

4.1 Modelagem da biblioteca

Com base no estudo das bibliotecas existentes, foi dado início aos trabalhos de modelagem da biblioteca. O paradigma proposto por Zheng (2015), apresentado na Figura 1, foi utilizado como base, para definir os módulos constituintes do PyMove e assim propor uma arquitetura para a biblioteca. A modelagem definida, pode ser utilizada como base para estruturar outras bibliotecas que se propõem a trabalhar com trajetórias.

Para dar suporte visual ao processo, foram criados diferentes diagramas. Um dos

Figura 12 – Etapas da elaboração do trabalho.



Fonte: elaborado pela autora

diagramas gerados é apresentado na Figura 13. Nela são ilustradas os módulos constituintes do sistema, sendo eles: *utils*, *preprocessing*, *models*, *uncertainty* e *visualization* e o *core*. Os pacotes *uncertainty* e *models* ainda estão em desenvolvimento. O pacote de *uncertainty* conterá operações para lidar com incerteza em trajetórias, assunto foi explanado na Seção 2.1.4. O pacote *models*, contém os seguintes sub-módulos:

- ***anomaly-detection***: disponibiliza operações para detecção de anomalias em trajetória (Seção 2.1.6).
- ***classification***: fornece operações para auxiliar na etapa de classificação (Seção 2.1.5).
- ***pattern-mining***: auxilia na etapa de detecção de padrão (Seção 2.1.3).

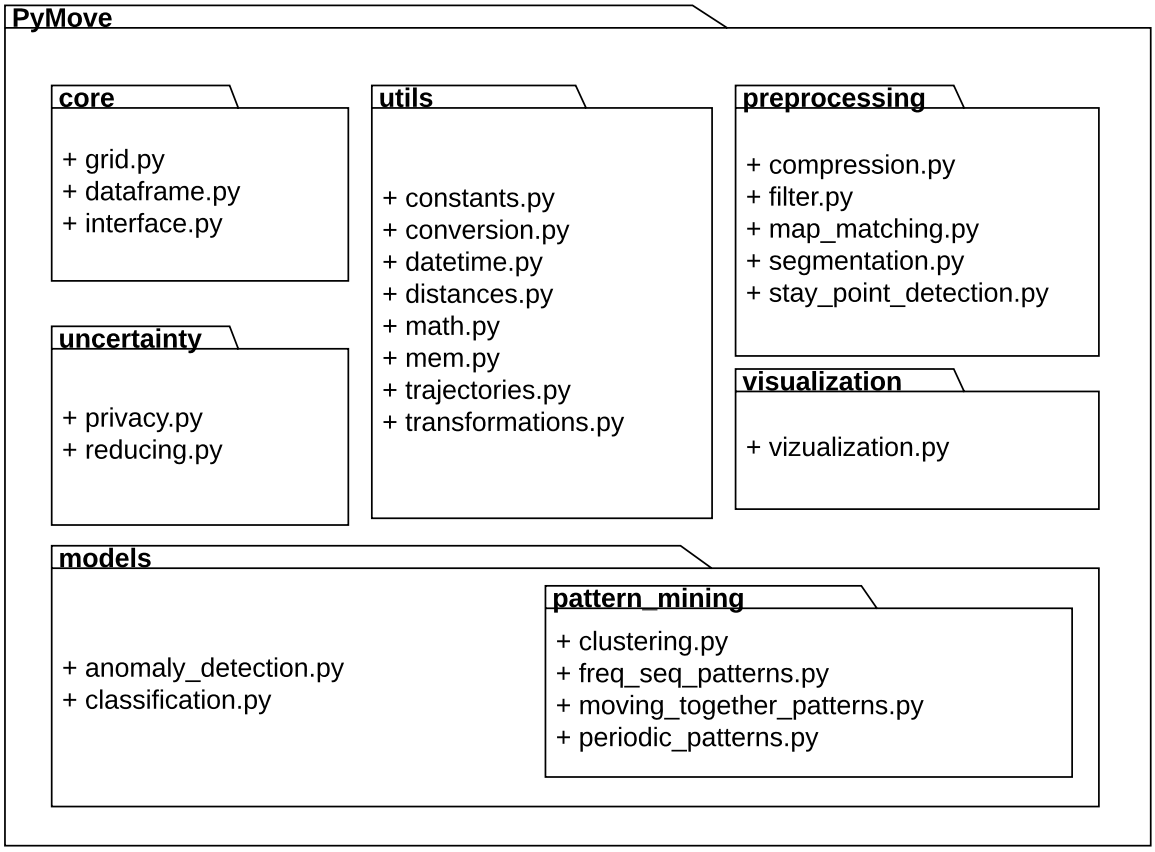
A seguir os pacotes: *Utils*, *Preprocessing* e *Core* serão explanados em maior detalhe.

4.1.1 Utils

Este pacote encapsula diferentes módulos utilitários, que auxiliam a implementação de operações em outros módulos e oferecem diferentes funcionalidades ao usuário. Ele encapsula os seguintes submódulos:

- ***Constants***: define todas as constantes utilizadas no sistema.
- ***Conversion***: provê acesso a operações para conversão de datas, distancias e listas.
- ***Distances***: provê acesso a funções para calcular métricas de distância.
- ***Datetime***: encapsula as operações que lidam com data e hora.
- ***Math***: define operações matemáticas.
- ***Mem***: define operações relacionadas com consumo de memória.

Figura 13 – Diagrama de pacotes da biblioteca PyMove.



Fonte: elaborado pela autora

- ***Trajectories***: encapsula funções utilitárias menos específicas, que não se encaixam nos outros módulos.
- ***Transformations***: implementa funções de conversão de *features*.

4.1.2 Preprocessing

Este pacote dá suporte para a realização do pré-processamento de dados, descrito na Seção 4.2.3. A divisão dos seus módulos foi feita de acordo com as etapa descritas na mesma Seção:

- ***Compression***: implementa operações de compressão de dados Seção 2.1.1.3).
- ***Filter***: oferece diferentes tipos de filtros que podem ser aplicados a trajetórias. Também pode ser utilizado para auxiliar em trabalhos na filtragem de ruídos (Seção 2).
- ***Map-matching***: encapsula um conjunto de funções que tem como objetivo, corrigir problemas que podem surgir após a utilização de algoritmos de map-matching (Seção 2.1.1.5).

- ***Segmentation***: provê acesso a operações de segmentação de dados de trajetórias (Seção 2.1.1.4).
- ***Stay-point-detection***: encapsula operações para detecção de pontos de parada (Seção 2.1.1.2).

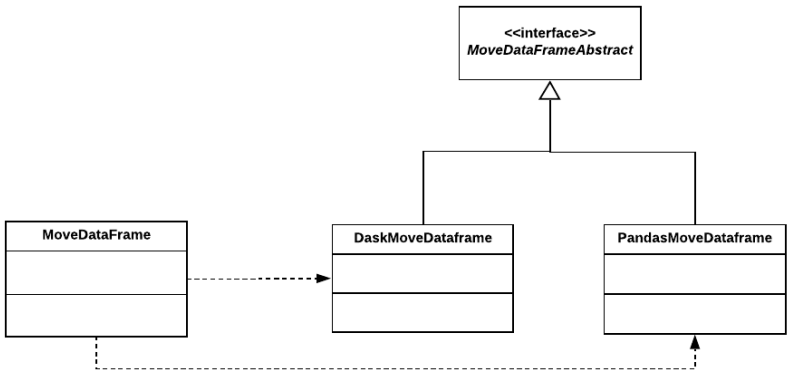
4.1.3 Core

Após a conclusão da divisão dos demais módulos, foi definido o *core*, que é a unidade fundamental do PyMove. Ele fornece funcionalidades básicas, como: a estrutura de representação dos dados, funções para manipular essa estrutura, funcionalidades para adição de *features* aos dados, como: distância entre pontos, velocidade de deslocamento e algumas formas de visualizar os dados. Todas as funcionalidades disponíveis nos outros módulos são implementadas usando o *core*. Ele foi projetado utilizando o padrão de projeto *Factory Method*, referido na Seção 2.3.1, de forma a permitir, que diferentes estruturas de dados especiais, possam ser acopladas ao *core* e utilizadas para implementar as funcionalidades que ele disponibiliza. Essas estruturas de dados especiais, podem ser criadas utilizando bibliotecas, como: o Pandas, descrito na Seção 2.4, o Dask, explanado na Seção 2.5, ou alguma outra biblioteca de manipulação de dados.

O *core* é formado por: MoveDataFrameAbstract, MoveDataFrame, PandasMoveDataFrame, DaskMoveDataFrame e Grid. A Figura 14 ilustra as relações existentes entre as classes: MoveDataFrameAbstract, MoveDataFrame, PandasMoveDataFrame, DaskMoveDataFrame. Fazendo um paralelo com os componentes do padrão de projeto escolhido, a classe MoveDataFrameAbstract, seria o *Product*. Ela define o conjunto de funções oferecidas pelo *core*. As assinaturas das operações na interface são definidas com uma sintaxe similar a do Pandas, por ser uma biblioteca muito utilizada pela as pessoas que trabalham com análise de dados. Todas as classes que o usuário deseje acoplar ao core do PyMove, como estrutura de dados especial, devem estender esta interface e implementar as funções nela definidas.

As classes PandasMoveDataFrame e DaskMoveDataFrame implementam as estruturas de dados especiais discutidas acima. Elas desempenham o papel do *ConcreteProduct* e são implementadas utilizando o Pandas e o Dask, respectivamente. Todas as classes, que implementam uma estrutura de dados do PyMove, devem realizar uma validação, no momento em que o usuário tenta carregar um conjunto de dados para a estrutura. Nessa validação, é verificado se o conjunto de dados possui informações referentes à: latitude, longitude e datetime. Caso possua, os dados são carregados para a estrutura, caso contrário, uma mensagem de erro

Figura 14 – Diagrama de classes das estruturas de dados da proposta.



Fonte: elaborado pela autora

deve ser apresentada ao usuário.

A classe MoveDataFrame, desempenha um papel semelhante ao do *Product*. É a classe conhecida pelo usuário, é através dele que o usuário, carrega os seus dados para a estrutura do PyMove. No momento da instanciação, o usuário deve passar uma *flag* ao construtor do MoveDataFrame, indicando qual biblioteca deve ser utilizada em baixo nível para implementar as funções do PyMove.

A forma como o *core* foi modelado, permite que o PyMove seja uma biblioteca extensível e flexível. O PyMove é extensível, na medida que permite que novas estruturas possam ser acopladas ao sistema e utilizas para implementar as funcionalidades que ele se propõe a implementar. A biblioteca é também flexível, pois permite que o usuário mude a estrutura de baixo nível de acordo com as suas necessidades. Por exemplo, quando o DaskDataFrame estiver concluído, o usuário poderá utilizar dados que não cabem em memória e usar o DaskMoveDataFrame, que estende as funcionalidades do Dask e, por isso, oferece suporte para esse tipo de dados.

A classe *Grid*, oferece a estrutura necessária para a criação de uma *grid*, a partir de uma área definida pelo usuário. Esta classe oferece ainda, operações para manipular e visualizar a *grid*.

4.1.4 Visualização

A visualização exploratória é muitas vezes, um importante passo na análise de dados. Ela auxilia na identificação e padrões e na compreensão dos resultados de atividades de mineração de dados (Gomes *et al.*, 2018; ANDRIENKO; ANDRIENKO, 2013). Pensando nisso, o PyMove disponibiliza um pacote para visualizações. Este pacote oferece diferentes formas de visualização de dados de trajetória, ao usuário, utilizando o Folium[1] e o Matplotlib[2].

4.2 Desenvolvimento

Terminada a modelagem do sistema, foi integrado ao PyMove, um conjunto funções que oferecem operações trabalhar com trajetórias, operações para realizar filtragem de ruídos, segmentação compressão de dados, detecção de pontos de parada e operações para resolver problemas, como trajetórias com marcas de tempo desordenadas, que podem ser introduzidos nas trajetórias após o map-matching. Estas funções tinham sido previamente implementadas por mestrandos e doutorandos do laboratório *Insight Data Science Lab*. Foi necessário uma cuidadosa análise, para definir em qual dos módulos do PyMove, cada função melhor se encaixava. Em um primeiro momento as funções não foram alteradas, Isto aconteceu apenas depois da implementação do core, para que elas melhor se adaptassem à biblioteca.

4.2.1 Implementação do core e algumas funções do módulo de pré-processamento

Nesta etapa foi implementado o *core*, seguindo a modelagem elaborada na Seção 4.1. Por restrições de tempo, o DaskMoveDataFrame ainda está em desenvolvimento, por isso ainda não é possível executar todas as funcionalidades oferecidas pelo PyMove utilizando-o. Hoje é possível criar instanciar um DaskMoveDataFrame através do MoveDataFrame, mas ele possui apenas algumas funcionalidades básicas definidas pelo MoveDataFrameAbstract e não está integrado com os outros módulos da biblioteca. O foco deste trabalho foi o PandasMoveDataFrame. Todas as funções dos módulos de *utils* e *Preprocessing* podem ser executadas utilizando o PandasMoveDataFrame. Esta estrutura utiliza o DataFrame[3] do Pandas para manipular os dados. O DataFrame do PandasMoveDataFrame, possui colunas com nomes e tipos específicos, definidos na própria classe, sendo essas colunas: lat, lon, datetime e traj_id, referentes às informações de

[1] https://python-visualization.github.io/folium/
[2] https://matplotlib.org/contents.html
[3] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html

latitude, longitude, *datetime* e id da trajetória.

A Figura 15 ilustra a criação de um PandasMoveDataFrame, através do MoveDataFrame, passando um DataFrame com os dados a serem carregados. O parâmetro data, recebe os dados utilizados para criar a estrutura, os parâmetros latitude, longitude e *datetime* servem para indicar as colunas no DataFrame lido, que possuem as informações de latitude, longitude e *datetime*, respectivamente. O MoveDataFrame pode receber ainda o parâmetro *type*, que indica qual estrutura de dados de baixo nível deve ser utilizada, nos casos como o da Figura 15, em que esse parâmetro não é passado, o PandasMoveDataFrame é utilizado, por ser o *default*.

Figura 15 – Instanciação do PandasMoveDataFrame.

```
move_df = MoveDataFrame(data=df, latitude="latitude", longitude="longuitude", datetime="time")
```

```
move_df.head()
```

	datetime	lat	lon	alt	label	user
0	2008-10-23 02:53:04	39.984702	116.318417	492.0	0	0
1	2008-10-23 02:53:10	39.984683	116.318450	492.0	0	0
2	2008-10-23 02:53:15	39.984686	116.318417	492.0	0	0
3	2008-10-23 02:53:20	39.984688	116.318385	492.0	0	0
4	2008-10-23 02:53:25	39.984655	116.318263	492.0	0	0

Fonte: elaborado pela autora

Após a conclusão da implementação dos componentes do *core*, foram feitas extensões ao módulo *filters* e ao módulo *segmentation*. Foram adicionadas as seguintes funções ao módulo *filters*: *outliers*, *clean_gps_jumps_by_distance*, *clean_gps_nearby_speed_max_radius*, *clean_trajectories_short_and_few_points*, *clean_trajectories_short_and_few_point*. Ao módulo de *segmentation* foram adicionadas as seguintes funções: *by_speed*,*by_time*, *by_max_dist*. Essas funções são apresentadas em detalhes no Apêndice A, neste apêndice também são apresentadas as funcionalidades de todas as funções pertencentes ao pacote *preprocessing*.

4.2.2 Revisão e refatoração do código

Uma vez terminada as implementações, foram realizadas a revisão e refatoração de todo o código da biblioteca para adequá-lá às boas práticas de programação da linguagem Python.

Foram escolhidas as seguintes regras da PEP8, para serem aplicadas na biblioteca:

- Utilizar 4 espaços por nível de indentação.
- Utilizar espaços, em vez de *tabs* no código.

- Todas as linhas do código, devem ter no máximo 79 caracteres.
- Em casos em que é preciso fazer um quebra de linha e exista um operador matemático, a quebra de linha deve ser feita antes do operador.
- Devem existir duas linhas em branco, antes da definição de uma classe ou uma função.
- Importações de diferentes bibliotecas, devem estar em linhas diferentes e no início do arquivo;
- Os nomes da funções devem começar com uma letra minúscula e as palavras devem estar separadas por "_". Os nomes de variáveis devem seguir a mesma convenção.

Posteriormente foi realizada a análise estática do código, para encontrar e corrigir possíveis problemas ou más práticas no desenvolvimento de código, que não tinham sido percebidos nas análises anteriores. Esta análise aconteceu em dois momentos. Num primeiro momento, a IDE PyCharm[4], foi utilizada para realizar tal tarefa, em paralelo com a refatoração acima mencionada. Em um segundo momento, foi utilizada a ferramenta PyLint[5] para realizar a análise.

4.2.3 Documentação

Terminada a revisão e refatoração do código, deu-se início a documentação das funções. Para escolher qual padrão utilizar para os comentários, foram analisadas os padrões utilizados por bibliotecas já conhecidas, como o Pandas e o Scikit-learn. A Figura 16 mostra um trecho de código da função *by_bbox* do módulo de *Filters*. Neste trecho é apresentada a documentação desta função. As demais funções seguem o mesmo formato. A documentação das funções tem como objetivo auxiliar, tanto o usuário no uso da biblioteca, quanto na manutenção e extensão futuras do sistema.

4.2.4 Elaboração de tutoriais

Após a conclusão da documentação, foi elaborado um conjunto de tutoriais utilizando o *Jupyter Notebook*[6] com o objetivo de apresentar ao usuário as diferentes funcionalidades oferecidas pela biblioteca, servindo os tutoriais como guias para novos usuários. Para cada um dos módulos do sistema foi projetado um tutorial. Estes podem ser acessados no repositório do

[4] https://www.jetbrains.com/pycharm/
[5] https://pylint.readthedocs.io/en/latest/
[6] https://jupyter.org/documentation

Figura 16 – Documentação da função by_bbox do módulo de *Filters*.

```
def by_bbox(move_data, bbox, filter_out=False, inplace=False):
    """Filters points of the trajectories according to especified bounding box.

    Parameters
    ----------
    move_data : dataframe
       The input trajectories data
    bbox : tuple
        Tuple of 4 elements, containg the minimum and maximum values of latitude and longitude of the bounding box.
    filter_out : boolean, optional(false by default)
        If set to false the function will return the trajectories points within the bounding box,
        and the points outsideotherwise
    dic_labels : dict, optional(the classe"s dic_labels by default)
        Dictionary mapping the user"s dataframe labels to the pattern of the PyRoad"s lib
    inplace : boolean, optional(false by default)
        if set to true the original dataframe will be altered to contain the result of the filtering,
        otherwise a copy will be returned.

    Returns
    -------
    move_data : dataframe
        Returns dataframe with trajectories points filtered by bounding box.
```

Fonte: elaborado pela autora

PyMove[7] no GitHub.

O conjunto de dados utilizado para elaborar os tutoriais foi um subconjunto do conjunto de dados *Geolife GPS trajectory dataset* (ZHENG *et al.*, 2011). Este conjunto de dados foi escolhido por ser um conjunto conhecido e que já foi utilizado em diversos trabalhos da literatura. Os dados utilizados para montar este conjunto de dadosforam coletados durante um período de tempo superior a quatro anos (de Agosto de 2012, até Abril de 2017), por 178 usuários no projeto Geolife, da *Microsoft Research Asia*. Para realizar a coleta dos dados, foram utilizados diferentes dispositivos como: GPS *loggers* e celulares com GPS. O conjunto de dados possui um total de 17.621 trajetória com duração total de 48.203 horas e distância total de 1.251.654 quilômetros. 91% das trajetórias possui representações densas, leituras a 5-10 metros por exemplo. Neste conjunto de dados, uma trajetória é representada por um conjunto de pontos, onde cada um desses pontos possui um *timestamp*, uma latitude, uma longitude e uma altitude. Neste trabalho foram utilizados 1.206.506 pontos referentes a quatro usuários.

4.3 Estudo de caso

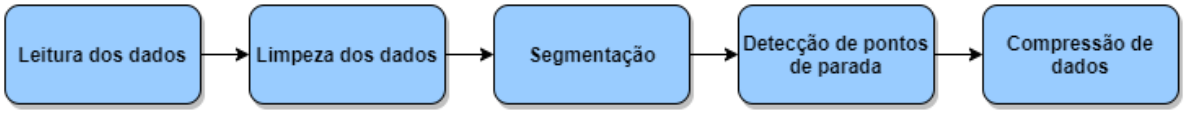
O estudo de caso apresentado nesta seção foi elaborado com o objetivo demonstrar como o PyMove pode auxiliar nas atividades de pré-processamento de dados, além de validar

[7] https://github.com/InsightLab/PyMove/tree/developer

as operações disponibilizadas pela biblioteca e a sua estrutura de dados. O conjunto de dados *Geolife GPS trajectory dataset* apresentado na Seção 4.2.4 foi utilizado para elaborar este estudo de caso. Ao final deste estudo de caso, é realizada a compreensão dos dados. Como foi descrito na Seção 2.1.1.3, muitas aplicações se beneficiam ou têm a compressão como um pré-requisito para a sua execução.

A Figura 17 ilustra as cinco etapas realizadas para a elaboração deste estudo. O primeiro passo é a criação do PandasMoveDataFrame com o conjunto de dados escolhido. A criação do PandasMoveDataFrame é ilustrada na Figura 15.

Figura 17 – Atividades empregadas para elaborar o estudo de caso.



Fonte: elaborado pela autora

4.3.1 Limpeza dos dados

O objetivo desta etapa é eliminar pontos que podem estar inconsistentes, duplicados ou contenham dados inválidos, a fim de contribuir para um melhor resultado nas etapas posteriores.Na Seção 2.1.1.1 é explicado que, diversos fatores podem influenciar a leitura dos dados. Por isso é realizadA a filtragem de ruído. Através de da visualização dos dados, é possível constatar que existem pontos possivelmente inconsistentes nas trajetórias. A abordagem *Heuristics-Based Outlier Detection*, apresentada na Seção 2.1.1.1 é utilizada para realizar a filtragem. Na Figura 18(a) é possível observar o ponto destacado no interior do círculo vermelho. Este ponto se afasta da distribuição do outros pontos da trajetória e, por isso, pode ser considerado um *outlier*. Para eliminar este e outros *outliers*, que possam existir no conjunto de dados, é aplicada a função *clean_gps_jumps_by_distance*, que identifica e remove os *outliers* da trajetória. A Figura 18(b) apresenta a trajetória, após a aplicação após a filtragem de ruído. 675 pontos foram eliminados pela função *clean_gps_jumps_by_distance*.

Ainda nesta etapa, foram removidos os pontos duplicados da trajetória, utilizado a função *clean_duplicates* do módulo *filtering*. Levou-se em consideração as *features*, id e *datetime*. Foi também utilizada a função *clean_consecutive_duplicates*, para eliminar pontos consecutivos com a mesma latitude e a mesma longitude. Por fim, a função *clean_trajectories_with_few_points* é utilizada para remover as trajetórias que possuem menos de 100 pontos e a função *clean_nan_values*

Figura 18 – Filtragem de ruído.

(a) Trajetória com ponto inconsistente. (b) Trajetória após a filtragem de ruído.

Fonte: elaborado pela autora

foi utilizada para eliminar os dados nulos. Ao final desta etapa, o número de pontos no conjunto de dados foi reduzido para 1.153.289.

4.3.2 Segmentação das trajetórias

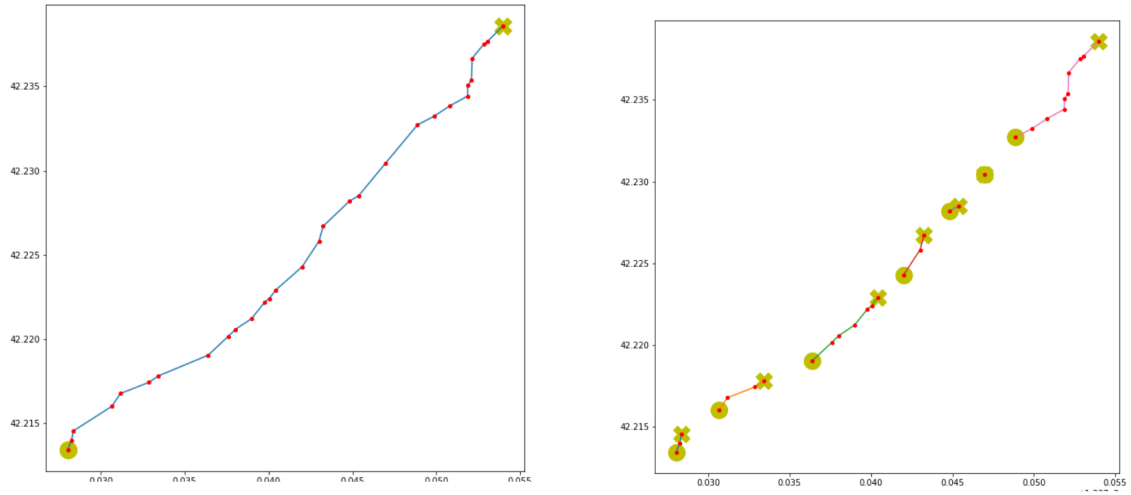
Terminada a filtragem de ruído, foi realizada a segmentação dos dados (Seção 2.1.1.4), utilizando a função *segmentation_by_max_dist* do módulo *segmentation*. A geometria foi o critério escolhido para realizar a segmentação. Foram segmentadas todas as trajetórias que possuíam uma distância superior a 200 metros entre os seus pontos adjacentes. A Figura 19(a) apresenta uma trajetória do conjunto de dados, na Figura 19(b), é apresentada a mesma trajetória após ter sido segmentada. É possível identificar os diferentes segmentos em que a trajetória foi dividida. A segmentação aqui realizada, é utilizada na próxima etapa do estudo, na identificação dos pontos de parada do conjunto de dados.

4.3.3 Compressão das trajetórias

A técnica de compressão utilizada neste estudo de caso, é a compressão *offline*, pois é a estrategia utilizada quando o conjunto de dados já está totalmente gerado. Foi utilizada a função *compress_segment_stop_to_point* do módulo *compression*. Esta função foi implementada com uma estratégia que utiliza os pontos de parada e um limite de tempo para realizar a compressão dos dados. A função *create_or_update_move_stop_by_dist_time*, do módulo *stay_point_detection*, foi utilizada para calcular os pontos de parada utilizados na compressão.

A Figura 20 apresenta os dados após as atividades de limpeza, segmentação e identificação dos pontos de parada do conjunto. A compressão diminuiu significativamente a quantidade dos dados. Os 1153289 pontos que faziam parte do PandasMoveDataFrame foram

Figura 19 – Segmentação de trajetória.

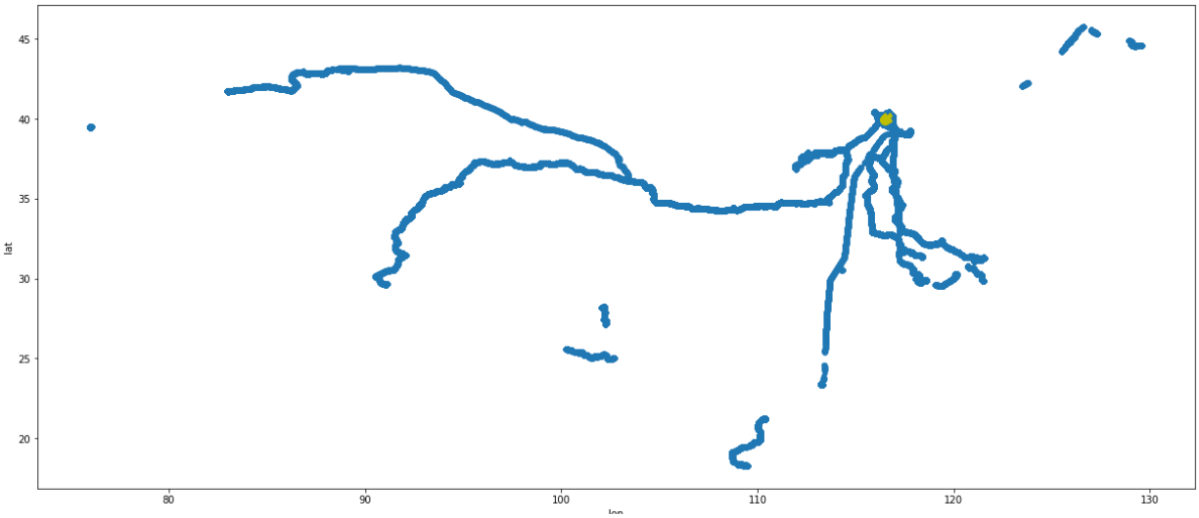


(a) Trajetória original antes da segmentação. (b) Trajetória após a segmentação de dados.

Fonte: elaborado pela autora

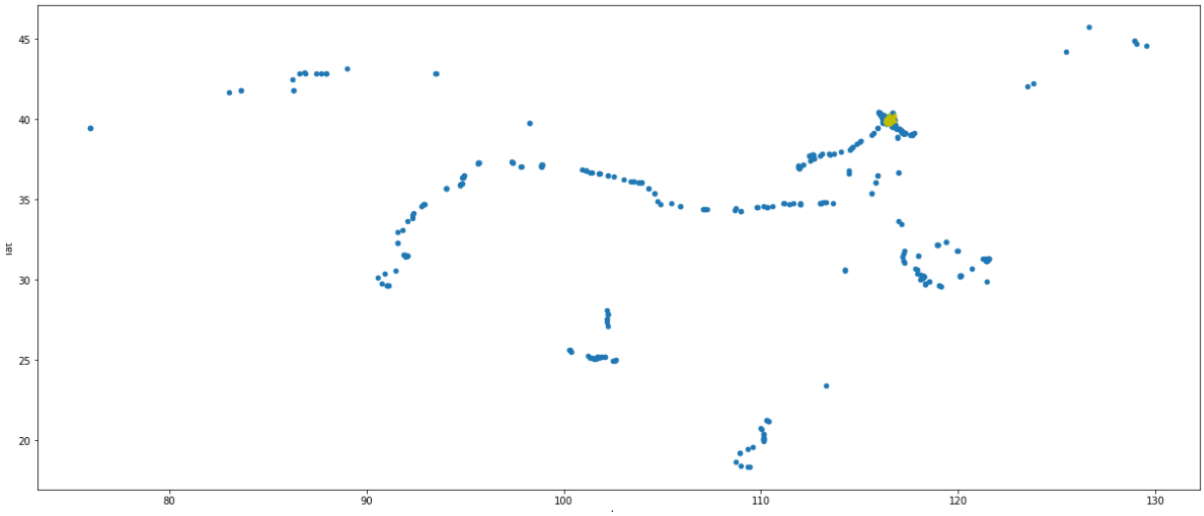
comprimidos para 946. A Figura 21 apresentada os dados comprimidos.

Figura 20 – As trajetórias do conjunto de dados antes da compressão.



Fonte: elaborado pela autora

Figura 21 – As trajetórias do conjunto de dados depois da compressão.



Fonte: elaborado pela autora

5 CONCLUSÃO

Neste trabalho foi projetada a arquitetura da biblioteca PyMove e realizada a expansão do módulo de pré-processamento. Foram também documentadas as funções pertencentes a biblioteca e desenvolvidos tutoriais para auxiliar o seu uso. Para validar os resultados deste trabalho, foi elaborado um estudo de caso, onde foram realizadas as etapas de: filtragem de ruídos, segmentação de trajetórias, detecção de pontos de parada e compressão dos dados.

Após a conclusão do estudo de caso, é possível constatar que o PyMove facilita a realização de atividades, que seriam mais custosas e requereriam maior esforço, se fossem implementadas sem o auxilio de uma biblioteca especializada. Outro fato a ser notado, é que o estudo de caso demonstra que o PyMove possibilita a realização das seguintes atividades do pré-processamento de dados: filtragem de ruídos, segmentação, detecção de pontos de parada e compressão de dados.

Um dos principais diferenciais do PyMove, advém do fato da biblioteca ter sido projetada com uma arquitetura que permite que ela seja flexível e extensível. Possibilita, assim, que a biblioteca se adapte frente às novas demandas dos usuários. Por ser uma biblioteca extensível, é possível que o PyMove se torne uma biblioteca escalável, após a conclusão do DaskDataFrame. Estas são vantagens oferecidas pelo PyMove em relação ao Scikit-mobility, que apesar de ser uma biblioteca similar ao do PyMove, não oferece essas características. Dentre os trabalhos apresentados na Seção 3, o PyMove é o que oferece suporte para um maior número de atividades de pré-processamento. Todavia, apesar do PyMove oferecer suporte para as etapas de compressão e *map-matching*, esses módulos contam com poucas operações, o que limita as possibilidades do usuário para realizar tais atividades. A biblioteca ainda não oferece suporte para mineração de padrões, classificação e técnicas para manipulação de incerteza dos dados.

Considerando os pontos acima explanados, se propõe os seguintes trabalhos futuros:

- Implementação do módulo *uncertainty* para oferecer suporte para as atividades de redução de incerteza e preservação de privacidade. Adicção de operações aos módulos de classificação, mineração de padrões e detecção de anomalias do pacote *models*. Possibilitando assim que a biblioteca ofereça maior suporte para as atividades de mineração de dados de trajetórias.
- Dar continuidade ao desenvolvimento do DaskDataMoveFrame, que utiliza a biblioteca Dask, para transformar o PyMove em uma biblioteca escalável e que consiga trabalhar com dados que não cabem em memória.

- Criação de mais operações para o módulo de pré-processamento, com maior foco nos módulos de compressão e *map-matching* que hoje contam com poucas operações. Permintido assim que o usuário realize todas as operções de pré-processamento de dados de trajetórias utilizando o PyMove e com uma vasta gama de operações para cada uma das atividades.
- Expansão da biblioteca, de forma a permitir que diferentes formas de representação, como grafos e tensores, possam ser utilizados para representar os dados de trajetória.

REFERÊNCIAS

Andrienko, G.; Andrienko, N.; Chen, W.; Maciejewski, R.; Zhao, Y. Visual analytics of mobility and transportation: State of the art and further research directions. **IEEE Transactions on Intelligent Transportation Systems**, v. 18, n. 8, p. 2232–2249, Aug 2017.

ANDRIENKO, N.; ANDRIENKO, G. Visual analytics of movement: An overview of methods, tools and procedures. **Information Visualization**, v. 12, n. 1, p. 3–24, 2013. Disponível em: <https://doi.org/10.1177/1473871612457601>.

BOGORNY, V.; AVANCINI, H.; PAULA, B. C. de; KUPLICH, C. R.; ALVARES, L. O. Weka-stpm: a software architecture and prototype for semantic trajectory data mining and visualization. **Trans. GIS**, v. 15, p. 227–248, 2011.

CHANDOLA, V.; BANERJEE, A.; KUMAR, V. Anomaly detection: A survey. **ACM Comput. Surv.**, v. 41, p. 15:1–15:58, 2009.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object-oriented Software**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN 0-201-63361-2.

Gomes, G. A. M.; Santos, E.; Vidal, C. A. **VISUALIZAÇÃO INTERATIVA DE DINÂMICAS DE TRÁFEGO ATRAVÉS DE DADOS DE TRAJETÓRIAS**. Tese (Doutorado) — PD thesis, Universidade Federal do Ceará, 2018.

GRAELLS-GARRIDO, E.; SAEZ-TRUMPER, D. A day of your days: Estimating individual daily journeys using mobile data to understand urban flow. **ACM International Conference Proceeding Series**, v. 24-25-May-2016, p. 1–7, 2016.

HERSHBERGER, J.; SNOEYINK, J. Speeding up the douglas-peucker line-simplification algorithm. **5th Intl Symp on Spatial Data Handling**, 11 2000.

Jia Wang, S.; MORIARTY, P. Big data for urban sustainability: A human-centered perspective. **Big Data for Urban Sustainability: A Human-Centered Perspective**, n. January, p. 1–160, 2018.

LI, Q.; ZHENG, Y.; XIE, X.; CHEN, Y.; LIU, W.; MA, W. Y. Mining user similarity based on location history. **GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems**, n. January, p. 298–307, 2008.

LOU, Y.; ZHANG, C.; ZHENG, Y.; XIE, X.; WANG, W.; HUANG, Y. Map-matching for low-sampling-rate GPS trajectories. **GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems**, n. January, p. 352–361, 2009.

NOULAS, A.; SCELLATO, S.; LAMBIOTTE, R.; PONTIL, M.; MASCOLO, C. A tale of many cities: Universal patterns in human urban mobility. **PLoS ONE**, v. 7, n. 5, 2012. ISSN 19326203.

OLIPHANT, T. E. Guide to NumPy. **Methods**, v. 1, p. 378, 2010. Disponível em: <http://scholar.google.com/scholar?hl=en{&}btnG=Search{&}q=intitle:Guide+to+N>.

PAPPALARDO, L.; SIMINI, F.; BARLACCHI, G.; PELLUNGRINI, R. **scikit-mobility: a Python library for the analysis, generation and risk assessment of mobility data**. 2019.

ROCKLIN, M. Dask: Parallel Computation with Blocked algorithms and Task Scheduling. **Proceedings of the 14th Python in Science Conference**, n. SCIPY, p. 126–132, 2015.

RUAN, S.; LI, R.; BAO, J.; HE, T.; ZHENG, Y. Cloudtp: A cloud-based flexible trajectory preprocessing Framework. **Proceedings - IEEE 34th International Conference on Data Engineering, ICDE 2018**, p. 1601–1604, 2018.

WITTEN, I.; HALL, M.; FRANK, E.; HOLMES, G.; PFAHRINGER, B.; REUTEMANN, P. The weka data mining software: An update. **SIGKDD Explorations**, v. 11, p. 10–18, 11 2009.

YOON, H.; SHAHABI, C. Robust time-referenced segmentation of moving object trajectories. **Proceedings - IEEE International Conference on Data Mining, ICDM**, n. December 2008, p. 1121–1126, 2008. ISSN 15504786.

ZHENG, Y. Trajectory data mining: An overview. **ACM Transaction on Intelligent Systems and Technology**, September 2015. Disponível em: <https://www.microsoft.com/en-us/research/publication/trajectory-data-mining-an-overview/>.

ZHENG, Y.; CAPRA, L.; WOLFSON, O.; YANG, H. Urban computing: Concepts, methodologies, and applications. **ACM Transaction on Intelligent Systems and Technology**, October 2014. Disponível em: <https://www.microsoft.com/en-us/research/publication/urban-computing-concepts-methodologies-and-applications/>.

ZHENG, Y.; FU, H.; XIE, X.; MA, W.-Y.; LI, Q. GeoLife User Guide 1.2. **Microsoft Reasearch Asia**, v. 2, n. April 2007, p. 31–34, 2011. Disponível em: <https://www.microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset-user-guide/>.

ZHENG, Y.; XIE, X. Learning travel recommendations from user-generated gps traces. **ACM Transaction on Intelligent Systems and Technology**, January 2011. Disponível em: <https://www.microsoft.com/en-us/research/publication/learning-travel-recommendations-from-user-generated-gps-traces/>.

ZHENG, Y.; ZHANG, L.; MA, Z.; XIE, X.; MA, W.-Y. Recommending friends and locations based on individual location history. **ACM Transactions on the Web**, ACM TWEB, February 2011. Disponível em: <https://www.microsoft.com/en-us/research/publication/recommending-friends-and-locations-based-on-individual-location-history/>.

APÊNDICE A – FUNÇÕES PERTENCENTES AO PACOTE *PREPROCESSING* DA BIBLIOTECA PYMOVE

Neste apêndice são apresentadas as funcionalidades oferecidas pelo pacote *preprocessing* da biblioteca PyMove. Cada um dos quadros abaixo apresenta as funcionalidades de um dos módulos pertencentes do pacote *preprocessing*. Na primeira linha dos quadros é indicado o nome do módulo a que pertencem as funções. A primeira coluna indica o nome da função e a segunda coluna explica para que serve a função.

Quadro 2 – Funções do módulo *filters*.

Filters	
Nome da função	**Funcionalidade**
by_bbox	Filtra trajetórias com base em um bounding box definido pelo usuário.
by_datetime	Filtra trajetórias com base no intervalo de tempo definido pelo usuário.
by_label	Filtra as trajetórias de acordo com um atributo e valor especificado pelo usuário.
by_id	Filtra as trajetórias de acordo com identificador especificado pelo usuário.
by_tid	Filtra trajetórias de acordo com um valor de identificador da trajetória especificado pelo usuário.
outliers	Filtra pontos considerados outliers das trajetórias.
clean_duplicates	Remove linhas duplicadas do conjunto de dados, opcionalmente apenas algumas colunas podem ser consideradas.
clean_consecutive_duplicates	Remove linhas duplicadas consecutivas das trajetórias, opcionalmente apenas algumas colunas podem ser consideradas.
clean_nan_values	Remove linhas com valores faltantes do conjunto de dados.
clean_gps_jumps_by_distance	Removes pontos considerados outliers das trajetórias.
clean_gps_nearby_points_by_distances	Remove pontos da trajetória que possuem uma distância inferior a um valor setado pelo usuário entre eles o ponto anterior a eles na trajetória.
clean_gps_nearby_points_by_speed	Remove pontos da trajetória que possuem uma velocidade inferior a um valor definido pelo usuário entre eles o ponto anterior a eles na trajetória.
clean_gps_speed_max_radius	Remove recursivamente os pontos da trajetória com velocidade superior a um valor especificado pelo usuário.
clean_trajectories_with_few_points	Remove do quadro de dados fornecido, trajetórias com menos pontos do que o especificado pelo usuário.
clean_trajectories_short_and_few_points_	Remove as trajetórias que possuem menos pontos do que um valor especificado pelo usuário e possuem um comprimento menor que um limiar definido também pelo usuário.

Fonte: elaborado pela autora.

Quadro 3 – Funções do módulo *segmentation*.

Segmentation	
Nome da função	**Funcionalidade**
bbox_split	Divide um bounding box passado por parâmetro em N segmentos de tamanhos iguais.
by_dist_time_speed	Divide as trajetórias em segmentos usando os atributos distância, tempo e velocidade como parâmetros para a segmentação.
by_speed	Divide as trajetórias em segmentos com base na velocidade do objeto entre dois pontos consecutivos..
by_time	Divide as trajetórias em segmentos com base no intervalo de tempo existente entre dois pontos consecutivos.
by_max_dist	Segmenta as trajetória com base na distância entre dois pontos consecutivos.

Fonte: elaborado pela autora.

Quadro 4 – Funções do módulo *stay_point_detection*.

Stay_point_detection	
Nome da função	**Funcionalidade**
create_update_datetime_in_format_cyclical	Converte os dados de tempo para um formato cíclico.
create_or_update_move_stop_by_dist_time	Define os pontos de parada das trajetórias utilizando os atributos distância e tempo. Se os pontos de parada já tiverem sido calculados, eles serão atualizados.
create_update_move_and_stop_by_radius	Define os pontos de parada das trajetórias utilizando o atributo distância. Se os pontos de parada já tiverem sido calculados, eles serão atualizados.

Fonte: elaborado pela autora.

Quadro 5 – Funções do módulo *compression*.

Compression	
Nome da função	**Funcionalidade**
compress_segment_stop_to_point	Realiza a compressão offline das trajetórias, utilizando os pontos de parada.

Fonte: elaborado pela autora.

Quadro 6 – Funções do módulo *Map_matching*.

Map_matching	
Nome da função	**Funcionalidade**
check_time_dist	Verifica se os pontos de trajetória estão na ordem correta após o map-matching, considerando os atributos tempo e distância.
fix_time_not_in_ascending_order_id	Corrige a ordem temporal entre os pontos de uma trajetória, após o map-matching.
fix_time_not_in_ascending_order_all	Corrige a ordem temporal entre os pontos de uma trajetória, após o map-matching.
interpolate_add_deltatime_speed_features	Interpolar distâncias (x) para encontrar os tempos (y) das trajetórias.

Fonte: elaborado pela autora.