



# Sumário

- 1 Descrição
- 2 Algoritmo FPM Sensível ao Contexto
- 3 Implementação CEP
- 4 Orientações para Utilizar da Ferramenta

# Descrição

Ferramenta implementada através da combinação entre a abordagem de Mineração de Padrões Frequentes (FPM) e Processamento de Eventos Complexos (CEP):

- Realiza o aprendizado incremental dos períodos do dia em que o indivíduo habitualmente socializa com base em informações contextuais;
- Detecta comportamentos sociais anormais e variações nas rotinas sociais;
- Utiliza a lógica *fuzzy* para modelar o conhecimento do especialista;
- Fornece uma API para facilitar a implementação das estratégias de detecção de padrões de sociabilidade e comportamentos anormais.



# Algoritmo FPM Sensível ao Contexto

- Segmentar o dia (24 horas) em slots de tamanhos iguais:

$$n = \frac{24}{w} \Rightarrow$$

S1	S2	S3	S4	...	Sn

- Fase de contagem: Atualizar a contagem no respectivo índice na estrutura (matriz  $C_s$ ) responsável por armazenar o resumo dos eventos.

S1	S2	S3	S4	...	Sn
2	1	0	1	3	1

# Algoritmo FPM Sensível ao Contexto

- Fase de descoberta do padrão de sociabilidade:
  - Identifica os slots candidatos:

$$slot\_th = num\_obs * \vartheta * \frac{1}{\frac{24}{w}} \quad (1)$$

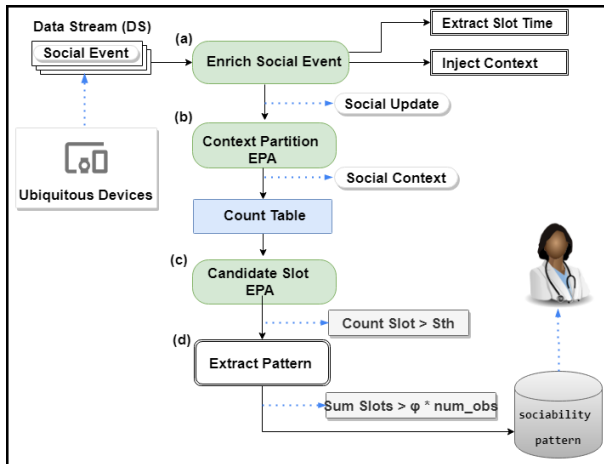
- Identifica os conjuntos de slots candidatos que representam um padrão de sociabilidade:

$$\sum_i^{i+n} C_s[i] > \varphi * num\_obs \quad (2)$$

# Algoritmo FPM Sensível ao Contexto

- Aprendendo as Variações Contextuais: utilizamos a estratégia de atributos de contexto, na qual diversas escalas podem ser usadas para representá-las.
  - Dias da semana: segunda, terça, quarta e sábado.
  - Dias úteis e fins de semana.
  - Dias chuvosos.
  - Dentre outros.
- Cada atributo de contexto foi usado como uma dimensão de segmentação de dados para identificar a variabilidade de comportamento.

# Implementações CEP: Detecção de Padrões de Sociabilidade



# Context Partition EPA

Segmenta o fluxo *SocialUpdate* com base nos CAs. Portanto, um evento derivado chamado *ContextEvent*, que possui o *slot* e o rótulo do contexto, é emitido para cada CA do evento;

---

## Contexto segmentado por categoria.

---

- 1: **CREATE CONTEXT** *CategoryContext*
  - 2: **GROUP** *ctxWeek = Week AS WEEK*,
  - 3: **GROUP** *ctxWeek = Weekend AS WEEKEND*,
  - 4: **GROUP** *ctxDay = Friday AS FRIDAY*,
  - 5: **GROUP** *ctxDay = Saturday AS SATURDAY*
  - 6: **GROUP** *ctxDay = Sunday AS SUNDAY*
  - 7: **FROM** *SocialUpdate*
- 

---

## Particionamento do fluxo

---

- 1: **CONTEXT** *CategoryContext*
  - 2: **INSERT INTO** *ContextEvent*
  - 3: **SELECT** *slot, context.label*
  - 4: **FROM** *SocialUpdate*
-



# Candidate Slot EPA

Verifica quais *slots* alcançaram um número adequado de eventos para se tornarem candidatos a formar um intervalo de sociabilidade

---

Seleção dos slots candidatos.

---

- 1: **SELECT** \* **FROM** *CountTable AS ct*
  - 2: **WHERE** *ct.label = labelContext* **AND**
  - 3: *ct.countSlot*  $\geq$  *ct.numObs* \* *teta* \*  $(1/(nSlot))$
  - 4: **ORDER BY** *slot*
-

# Extract Pattern

Identifica quais conjuntos de *slots* candidatos formam um intervalo de tempo no qual o indivíduo habitualmente socializa.

---

Extração do padrão social.

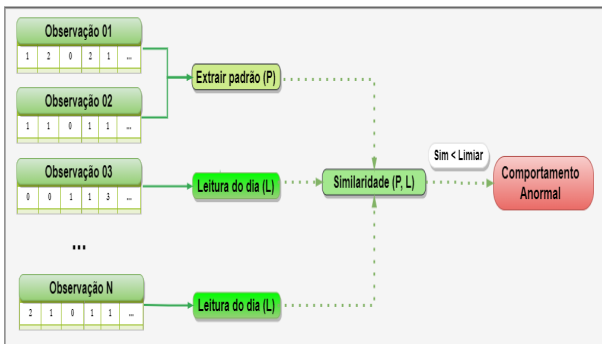
---

<p>1: <b>Inputs:</b></p> <p>2: Candidate Slots <math>CS_s[]</math></p> <p>3: Phi <math>\varphi</math></p> <p>4: Number of Observations <math>nobs</math></p> <p>5: <b>Output:</b></p> <p>6: A set of sociability patterns.</p> <p>7: <math>\phi \leftarrow \varphi</math></p> <p>8: <math>n \leftarrow nobs</math></p>	<p>9: <b>for all</b> slot <math>\in CS_s</math> <b>do</b></p> <p>10:     <b>while</b> slot isAdjacent(slot.next) <b>do</b></p> <p>11:         <math>adjacentSlots \leftarrow \text{merge}(\text{slot}, \text{slot.next})</math></p> <p>12:         <math>slot \leftarrow slot.next</math></p> <p>13:     <b>end while</b></p> <p>14:     <b>if</b> sum(adjacentSlots) &gt; <math>n * \phi</math> <b>then</b></p> <p>15:         <math>intervals[] \leftarrow adjacentSlots</math></p> <p>16:     <b>end if</b></p> <p>17:     <b>end for</b></p> <p>18: <b>return</b> <math>intervals[]</math></p>
--	--

---

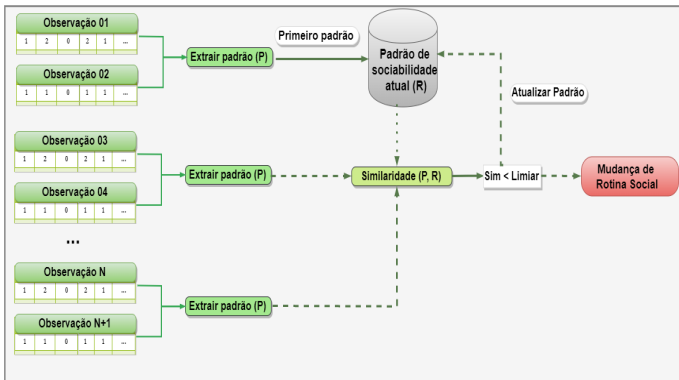
# Detecção de Comportamentos Sociais Anormais

Utiliza a abordagem de processamento de janelas de dados em combinação com uma métrica de similaridade, que permite verificar a mudança de padrão de um instante de tempo  $t_1$  para  $t_2$ . Compara um padrão de sociabilidade e uma observação.



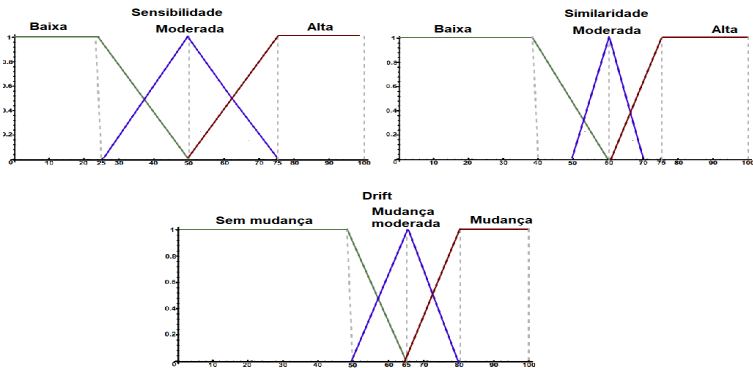
# Detecção de Mudança de Rotina Social

Avalia a similaridade entre dois padrões de sociabilidade. Ao detectar mudança de rotina, a ferramenta: (i) atualiza o padrão de sociabilidade atual; (ii) emite um evento para notificar as partes interessadas.



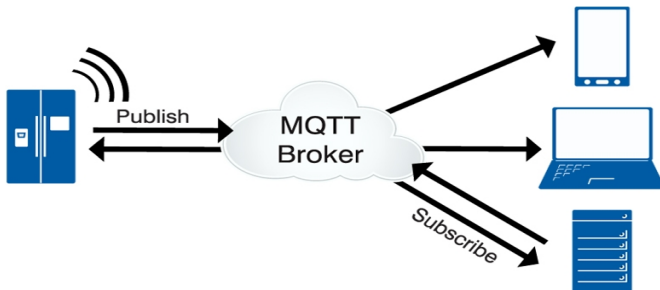
# Modelagem do Conhecimento do Especialista com Lógica Fuzzy

A biblioteca *jFuzzyLogic* foi utilizada para emitir eventos de mudanças de comportamentos com julgamento de grau de crença. Especifica Conjuntos de fuzzificação (Sensibilidade e Similaridade) e defuzzificação (Drift).



# Message Queue Telemetry Transport (MQTT)

O protocolo de comunicação MQTT permite a ferramenta publicar os padrões de sociabilidade e as notificações de mudanças de comportamentos em um *broker*. Aplicações clientes se inscrevem neste tópico para receber atualizações.



# Códigos-fontes

- Ferramenta:  
<https://github.com/Ivan-Rodrigues/SocialMHealth.;>
- Gerador de fluxo de dados: <https://github.com/Ivan-Rodrigues/SocialStreamGenerator>

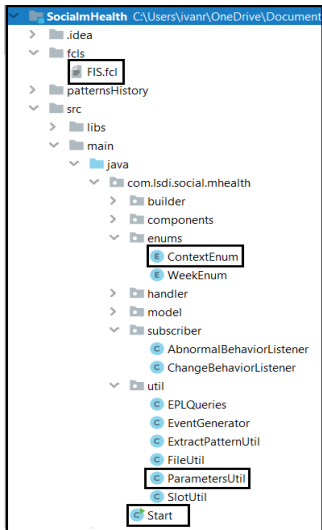
# Orientações para Utilizar da Ferramenta

- Abrir o projeto java em uma IDE (por exemplo, IntelliJ e Eclipse);
- Implementar os conjuntos fuzzy e as respectivas proposições através da Linguagem de Controle Fuzzy (FCL);
- Definir as estratégias de detecção através da API disponibilizada;
- Executar o script responsável por gerar o fluxo de dados.





# Estrutura do Projeto



- **Start:** classe que contém a implementação da API de detecção de padrões de sociabilidade e comportamentos anormais.
- **FIS.fcl:** implementação dos conjuntos fuzzy e proposições lógicas;
- **ContextEnum:** atributos de contexto considerados pela rede de processamento;
- **ParametersUtil:** classe que configura os parâmetros do algoritmo (quantidade de slots,  $\vartheta$ , e  $\varphi$ )

# Sistema de Inferência Fuzzy (FIS.fcl)

O usuário deve utilizar a Linguagem de Controle Fuzzy (FCL) para especificar os conjuntos de fuzzyficação (Similaridade e Sensibilidade) e defuzificação (Drift). Também deve especificar as proposições lógicas.

# Exemplo de Implementação do Conjunto Sensibilidade

---

1: FUZZIFY *sensibilidade*

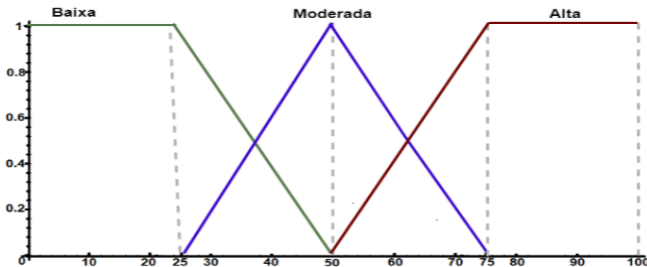
2: TERM *baixa* := (0, 1) (25, 1) (50, 0) ;

3: TERM *moderada* := (25, 0) (50, 1) (75, 0);

4: TERM *alta* := (50, 0) (75, 1) (100, 1);

5: END\_FUZZIFY

---



# Exemplo de Implementação do Conjunto Similaridade

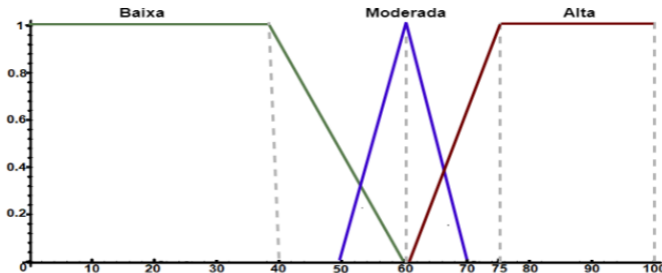
1: FUZZIFY *similaridade*

2: TERM *baixa* := (0, 1) (40, 1) (60, 0) ;

3: TERM *moderada* := (50, 0) (60, 1) (70, 0);

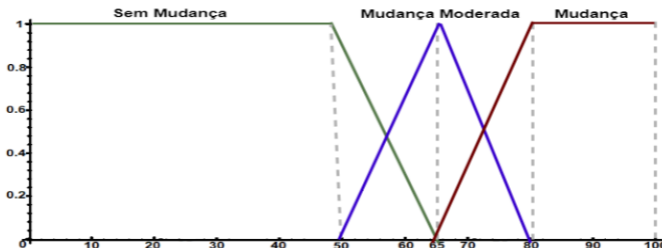
4: TERM *alta* := (60, 0) (75, 1) (100, 1);

5: END\_FUZZIFY



# Exemplo de Implementação do Conjunto Drift

- 
- 1: DEFUZZIFY *drift*
  - 2: TERM *sem\_mudanca* := (0, 1) (50, 1) (65, 0) ;
  - 3: TERM *mudanca\_moderada* := (50, 0) (65, 1) (80, 0) ;
  - 4: TERM *mudanca* := (65, 0) (80, 1) (100,1);
  - 5: METHOD: *COG*;
  - 6: END\_DEFUZZIFY
- 



# Exemplo de Implementação das Proposições Lógicas

- 
- 1: RULEBLOCK No1
  - 2: RULE 1: IF sensibilidade IS baixa AND similaridade IS baixa THEN drift IS mudanca;
  - 3: RULE 2: IF sensibilidade IS baixa AND similaridade IS moderada THEN drift IS sem\_mudanca;
  - 4: RULE 3: IF sensibilidade IS baixa AND similaridade IS alta THEN drift IS sem\_mudanca;
  - 5: RULE 4: IF sensibilidade IS moderada AND similaridade IS baixa THEN drift IS mudanca;
  - 6: RULE 5: IF sensibilidade IS moderada AND similaridade IS moderada THEN drift IS mudaca\_moderada;
  - 7: RULE 6: IF sensibilidade IS moderada AND similaridade IS alta THEN drift IS sem\_mudanca;
  - 8: RULE 7: IF sensibilidade IS alta AND similaridade IS baixa THEN drift IS mudanca;
  - 9: RULE 8: IF sensibilidade IS alta AND similaridade IS moderada THEN drift IS mudanca;
  - 10: RULE 9: IF sensibilidade IS alta AND similaridade IS alta THEN drift IS sem\_mudanca;
  - 11: END\_RULEBLOCK
-

# Definição dos Atributos de Contexto (ContextEnums)

O usuário deve especificar os Atributos de Contexto (CA) na classe ContextEnum. Segue um exemplo de implementação de CAs dos dias da semana (segunda a sexta) e semana (dia útil e final de semana):

```
1 package com.lsdi.social.mhealth.enums;
2
3 public enum ContextEnum {
4     ALL_, WEEK_, WEEKEND_, SUNDAY_, MONDAY_, TUESDAY_, WEDNESDAY_, THURSDAY_, FRIDAY_, SATURDAY_;
5 }
```

# Configurando os Parâmetros da Ferramenta (ParametersUtil)

O usuário deve especificar valores para estes parâmetros estáticos  
(Verificar a seção do algoritmo).

```
1
2 package com.lsdI.social.mhealth.util;
3
4 public class ParametersUtil {
5     //Parâmetros da arquitetura
6     public static double T = 0.5;
7     public static double NUM_SLOTS = 24/T; //48 slots
8     public static double THETA = 0.02;
9     public static double PHI = 0.7;
10 }
```



# Configurando um Broker MQTT

A ferramenta utiliza um Broker MQTT para receber o fluxo de dados e emitir notificações.

- O usuário pode usar o Broker eclipse instaciado em `iot.eclipse.org` ou configurar um local em sua máquina.
- Passos da instalação local:
  - Baixar o executável aqui:  
<https://mosquitto.org/download/>
  - Executar o instalador do Broker mosquitto
  - Executar o prompt de comando como administrador e entrar no diretório onde o Broker mosquitto foi instalado ("cd C:\mosquitto")
  - Iniciar o serviço do Broker através do comando: "net start mosquitto"



# Configurando um Broker MQTT


Tutorial completo de instalação do Broker MQTT:

- Linux: <https://medium.com/tht-things-hackers-team/instalar-mqtt-broker-no-linux-debian-ubuntu-f8861da70e>
- Windows: <http://www.bytesofgigabytes.com/mqtt/installing-mqtt-broker-on-windows/>



# API de Programação: StreamReceiver (Start)

O usuário deve configurar o recebimento do fluxo de dados na classe Start. Especificamente, configura-se a url do Broker MQTT e o tópico onde os eventos sociais são publicados.

```
15 ▶  public static void main(String[] args) {  
16  
17     StreamReceiver receiver = new StreamReceiver();  
18     receiver.setBroker("tcp://127.0.0.1:1883");  
19     receiver.setTopic("social");  
20     receiver.receiverStream();  
}
```

# API de Programação: Detecção de Padrões (Start)

O usuário deve criar um objeto denominado *SociabilityPattern*. No construtor é inserido o nome do contexto a ser considerado e o nível de sensibilidade de detecção. Logo após, é inserido o tópico raiz no qual será publicado as notificações no Broker MQTT. Por fim, habilita-se as estratégias de detecção de comportamentos anormais e mudanças de rotina social.

---

```
1: SociabilityPattern sociabilityPattern = new SociabilityPattern
2:     .Builder(ContextEnum.MONDAY_.toString(), sensitivityOfChange: 50.0)
3:     .setRootTopic("com/lldi/sociability")
4:     .setAbnormalBehavior(true)
5:     .setChangeBehavior(true)
6:     .build();
```

---



# Executar a Ferramenta: Start

Ao executar a classe Start a ferramenta estará pronta para processar eventos sociais. As notificações serão publicadas em tópicos no Broker MQTT configurado. As estruturas dos tópicos são:

- tópico raiz/contexto/tipo do evento. Por exemplo:
  - 'com/lldi/sociability/MONDAY/newPattern'
  - 'com/lldi/sociability/FRIDAY/AbnormalBehavior'
  - 'com/lldi/sociability/SUNDAY/ChangeBehavior'



# Estrutura das Notificações Emitidas

A Ferramenta emitirá um JSON que contém informações como a data, contexto, similaridade, e o grau de pertinência a cada intervalo do conjunto fuzzy de saída. abaixo apresentamos um exemplo de notificação de mudança de comportamento social.

```
1: {  
2:     "data": "Qua 1, 2020, 9:17:48 PM",  
3:     "contexto": "Quarta",  
4:     "similaridade": 56.00000000000001,  
5:     "valor de defuzzificação": 74.86688093051647,  
6:     "mudanca": 0.6577920620344315,  
7:     "sem_mudanca": 0.0,  
8:     "mudanca_moderada": 0.3422079379655685,  
9:     "mensagem": "Mudança de rotina social detectada"  
10: }
```

# Script para Gerar o Fluxo de Dados

- Desenvolvemos um script Python capaz de ler os dados de um dataset (conversações) e publicá-los no Broker MQTT configurado.
- Este script também se inscreve em tópicos para receber as notificações da ferramenta.
- É necessário ter instalado um interpretador Python em sua máquina para executá-lo.

# Script para Gerar o Fluxo de Dados

Exemplo de configuração do script:

```
17 broker = '127.0.0.1' #endereço do broker
18 pub_topic = 'social' # tópico para publicar o fluxo
19 sub_topic_abnormal = 'com/lsdi/sociability/MONDAY_/AbnormalBehavior' # sub. notificações Comp. Anormal
20 sub_topic_change = 'com/lsdi/sociability/MONDAY_/ChangeBehavior' # sub. notificações Mudança Rotina
21 sub_topic_pattern = 'com/lsdi/sociability/MONDAY_/newPattern' # sub. notificações Comp. Anormal
```