08. Swarm Intelligence

PSO - Particle Swarm Intelligence

IC CPereira 22-23

!

Swarm Intelligence

- Definição de “Swarm”
  - Conjunto estruturado de indivíduos (ou agentes) que interagem entre si;
    - Os Indivíduos pertencentes ao *swarm* interagem para atingirem um **objectivo comum**, de forma mais eficiente do que agindo individualmente;
    - Os indivíduos são relativamente simples, contudo o **comportamento colectivo pode ser complexo** – analogia com colónias de formigas, enxames, bandos, cardumes, …

Particle Swarm Optimization

- PSO - Particle Swarm Optimization
  - Trata-se de um algoritmo de pesquisa baseado na modelação do comportamento social de aves em bandos.
    - A partir da tentativa da descoberta de padrões que governam a habilidade das aves de voarem em sincronismo e em formação, evolui-se para um algoritmo de optimização simples e eficiente.
    - O comportamento de cada partícula é influenciado pelas outras partículas do *swarm*
    - Os indivíduos aprendem com os outros, **evoluindo para um comportamento similar aos melhores indivíduos do *swarm***

3

Particle Swarm Optimization

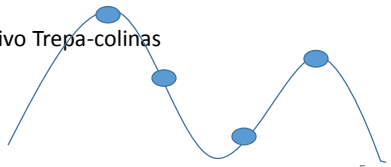
- Rede Social – O Princípio de Vizinhança
  - A estrutura social no PSO é determinada pela formação de regiões de vizinhança;
    - Os indivíduos pertencentes à mesma vizinhança comunicam entre si.
  - As topologias mais conhecidas são:
    - Topologia em estrela
    - Topologia em anel
    - Topologia em roda

4

Particle Swarm Optimization

- Principais componentes:
  - Um enxame de partículas
  - Cada partícula representa uma solução do problema.
  - Os parâmetros (posição da partícula) no espaço de pesquisa representam a solução.
    - Os parâmetros da partícula representam os parâmetros a serem otimizados.
    - As partículas “movem-se” no espaço de pesquisa (**exploração local**).
    - Existe comunicação entre partículas (exploração **global**).

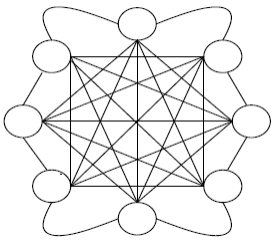
      - Um algoritmo de exploração local? Melhoramento Iterativo Trepa-colinas
      - Um algoritmo de exploração Global? Alg. Genético
      - PSO - Tradeoff Local vs Global



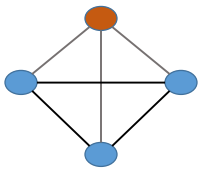
5

Topologias

- …
  - Interação social baseada no concerto de vizinhança
    - Topologia em Estrela



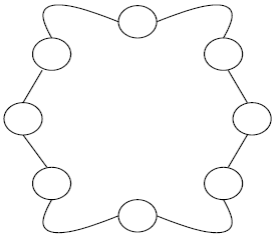
    - Todos os indivíduos comunicam entre si!
      - Global



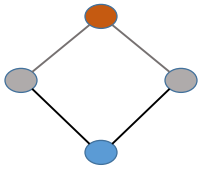
6

Topologias

- …
  - Topologia em Anel



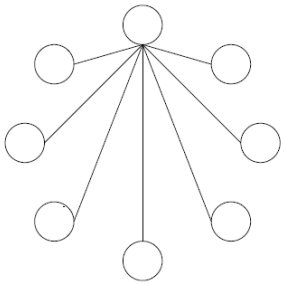
- Os indivíduos comunicam com os da sua vizinhança - é necessária a definição de um critério de vizinhança.



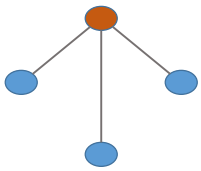
7

Topologias

- …
  - Topologia em roda



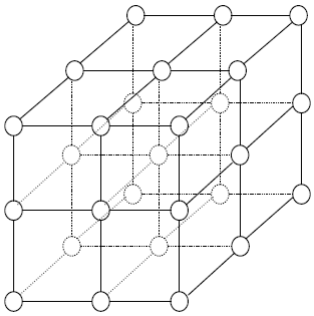
- Apenas um indivíduo comunica com todos os outros.



8

Topologias

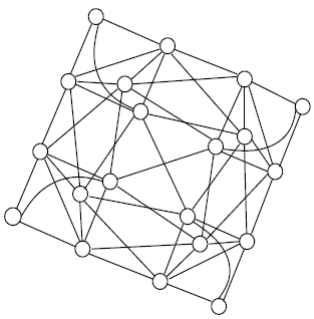
- …
  - Outras topologias
    - Von Neumann

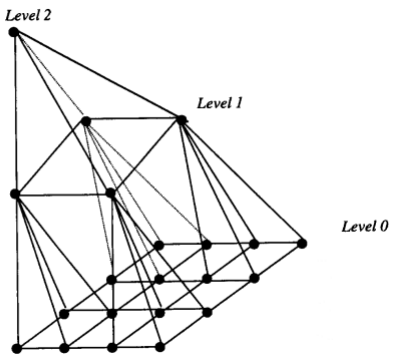


9

Topologias

- …
  - Pirâmide
    - Generalização da estrutura em anel

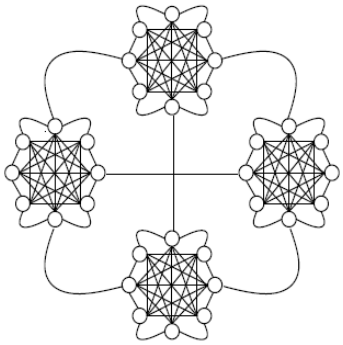




10

Topologias

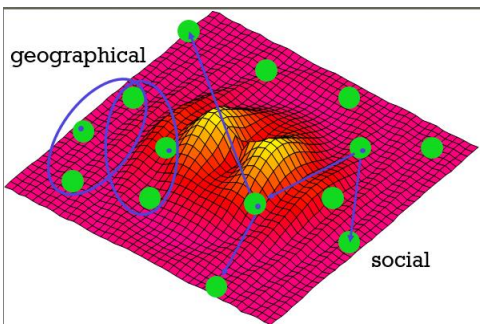
- …
  - Clusters



11

Particle Swarm Optimization

- …critérios de comunicação
  - Social vs localização geográfica



12

PSO – Algoritmo

- Algoritmos
  - Princípio de Funcionamento
    - Cada partícula *(i=1,..,n)* consiste numa possível solução de um problema de otimização;
    - A posição de uma partícula varia de acordo com "a sua experiência individual" e a experiência das partículas "vizinhas";
    - A posição de uma partícula *x(t)* é adaptada adicionando uma velocidade *v(t)* :

$$x_i(t) = x_i(t-1) + v_i(t)$$

13

Algoritmo

- …
  - Principais variantes – associadas ao conceito de vizinhança
    - **Individual Best**
      - As partículas comportam-se sem componente social
      - Nova posição depende de inércia e "personal best"
      - Semelhante a "hill-climbers" independentes!
    - **Global Best**
      - topologia em estrela
      - Nova posição depende de inércia, "*personal best" e "global best"*
    - **Local Best**
      - topologia em anel
      - Nova posição depende de inércia, "*personal best" e "local best"*

14

Algoritmo

- …
  - Individual Best

1. Inicializar o *swarm* de partículas $P(t)$ para $t=0$, de tal forma que a posição $P_i \in P(t)$, de cada partícula $x_i(t)$ é aleatória, dentro do espaço multidimensional de procura.

2. Avaliar a performance de cada partícula na sua posição actual $F(x_i(t))$

3. Comparar a performance de cada indivíduo com a sua melhor performance:

   Se $F(x_i(t)) < pbest_i$ Então:

   (a) $pbest_i = F(x_i(t))$

   (b) $xbest_i = x_i(t)$

15

Algoritmo

- …

4. Actualiza a velocidade de cada partícula:

   $$v_i(t) = v_i(t-1) + \rho(xbest_i - x_i(t))$$

   onde ρ representa um valor aleatório positivo. O limite superior desta constante é especificado pelo utilizador.

5. Move cada partícula para a sua nova posição:

   (a) $x_i(t) = x_i(t-1) + v_i(t)$

   (b) $t = t + 1$

6. Volta ao passo 2 e repete até convergência

16

Algoritmo

- …
  - Global Best

1. Inicializar o *swarm* de partículas $P(t)$ para $t=0$, de tal forma que a posição $P_i \in P(t)$, de cada partícula $x_i(t)$ é aleatória, dentro do espaço multidimensional de procura.

2. Avaliar a performance de cada partícula na sua posição actual $F(x_i(t))$

3. Comparar a performance de cada indivíduo com a sua melhor performance:

   Se $F(x_i(t)) < pbest_i$ Então:

   (a) $pbest_i = F(x_i(t))$

   (b) $xbest_i = x_i(t)$

17

Algoritmo

- …

4. Comparar a performance de cada indivíduo com melhor performance global:

   Se $F(x_i(t)) < gbest$ Então:

   (a) $gbest = F(x_i(t))$

   (b) $xgbest = x_i(t)$

5. Actualiza a velocidade de cada partícula:

   $v_i(t) = v_i(t-1) + \rho_1(xbest_i - x_i(t)) + \rho_2(xgbest_i - x_i(t))$

   onde ρ_1 e ρ_2 representam valores aleatórios positivos, respectivamente designados de componente cognitiva e componente social.

6. Move cada partícula para a sua nova posição:

   (a) $x_i(t) = x_i(t-1) + v_i(t)$

   (b) $t = t + 1$

7. Volta ao passo 2 e repete até convergência

18

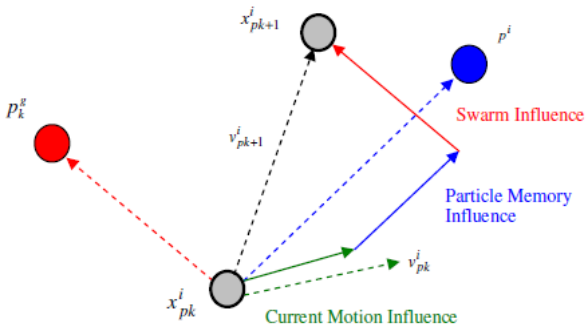
Algoritmo

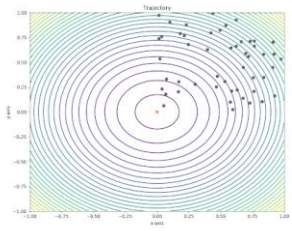
- …
  - Local Best
    - Reflecte o conceito de vizinhança.
    - As partículas são influenciadas pela sua própria experiência e pelas partículas dentro do seu raio de vizinhança.
    - Usa o melhor local – lbest, no lugar d melhor global –gbest.
      - Nos passos 4 e 5, alterar “gbest” para “lbest”

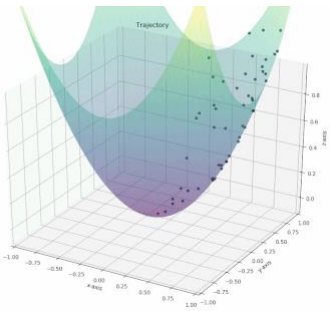
19

Algoritmo

- …







20

Parâmetros

- Parâmetros que influenciam o comportamento do algoritmo:
  - Dimensão do problema;
  - Número de indivíduos;
  - Limite superior das constantes cognitiva e social;
  - Limite superior da Velocidade
  - Tamanho da vizinhança
  - Peso de inércia

21

Parâmetros

- …
  - Velocidade Máxima
    - Habitualmente define-se uma velocidade máxima para todas as dimensões do problema, prevenindo-se movimentos muito bruscos dentro do espaço de procura.
      - não coloca limite na posição da partícula
      - Geralmente depende da gama de valores do problema. Por exemplo se -10<x<10, a velocidade é proporcional a 10

22

Parâmetros

- …
  - Foi demonstrado que não será necessário um limite ao valor de velocidade se:

$$v_i(t) = \gamma\left(v_i(t-1) + \rho_1\left(xbest_i - x_i(t)\right) + \rho_2\left(xgbest_i - x_i(t)\right)\right)$$

onde

$$\gamma = 1 - \frac{1}{\rho} + \frac{\sqrt{|\rho^2 - 4\rho|}}{2}$$

com

$$\rho = \rho_1 + \rho_2 > 4$$

23

Parâmetros

- …
  - Tamanho da Vizinhança
    - No limite, consideramos qualquer partícula do swarm parte da vizinhança e usamos "gbest".
      - Neste caso o algoritmo é mais sensível a mínimos locais, pois todos os indivíduos são "conduzidos" para a mesma solução
    - Quanto menor o tamanho da vizinhança, menor a sensibilidade a mínimos locais, mas a convergência será mais lenta.

24

Parâmetros

- …
  - Parâmetro de Inércia
    - A performance pode melhorar se usarmos uma **fracção da velocidade anterior** para cálculo da velocidade atual - constante de inércia:

$$v_i(t) = \varphi v_i(t-1) + \rho_1\left(xbest_i - x_i(t)\right) + \rho_2\left(xgbest_i - x_i(t)\right)$$

      - Valores elevados da constante de inércia implicam uma maior área de pesquisa no espaço de procura.
      - Tipicamente pode-se inicializar este parâmetro com um valor elevado, que deve diminuir ao longo do tempo.

25

Parâmetros

- …
  - Convergência?
    - Diversos estudos comprovam que o PSO não converge para qualquer combinação de parâmetros, devendo-se respeitar a seguinte relação, para o caso da constante de inércia:

$$\frac{1}{2}(\rho_1 + \rho_2) - 1 < \varphi \leq 1$$

26

Variantes

- PSO Binário
  - As implementações do PSO destinam-se maioritariamente a problemas em que o espaço de pesquisa é contínuo.
    - Existem variantes onde as posições das partículas representam valores binários. Nesse caso poderemos actualizar as posições de acordo com:

$$x_{ij}(t+1) = \begin{cases} 0, se\ \ r_i(t) \geq f\left(v_{ij}(t)\right) \\ 1, se\ \ r_i(t) < f\left(v_{ij}(t)\right) \end{cases}$$

$$\text{onde, } f\left(v_{ij}(t)\right) = \frac{1}{1+e^{-v_{ij}(t)}} \text{ e } \ r_i(t) \in [0,1]$$

    - A atualização da velocidade é similar à versão contínua.

27

Variantes

- …
  - Mudança da posição corresponde a trocar o valor de um dos bits do vetor de representação da solução.
  - Interpretação do conceito de velocidade:
    - Número de bits que são trocados em cada iteração.
    - Velocidade =0 : implica que não há trocas
    - Velocidade = número de posições: corresponde à velocidade máxima!
    - E para uma só dimensão?
      - Corresponde à probabilidade de trocar o bit!

28

Variantes

- Mecanismos de Selecção
  - Para alguns problemas, provou-se que será vantajosa a aplicação de mecanismos de selecção semelhantes ao usados nos arguimos de computação evolucionária.
    - O seguinte mecanismo foi sugerido:
      - Antes da actualização das velocidades:
        - Para cada partícula, atribuir um valor (ranking) de acordo a performance desta partícula relativamente a um conjunto aleatório de k partículas;
        - Criar um ranking de partículas de acordo com o valor anteriormente calculado;
        - Seleccionar a metade superior de partículas e copiar as suas posições para as partículas da metade inferior, mantendo contudo os melhores valores de posição;

29

Variantes

- Mecanismos de Reprodução

1. Calcular as velocidades e novas posições das partículas

2. Associar uma constante de reprodução a cada partícula p_b

3. Seleccionar duas partículas para progenitores (P_a *e* P_b) e criar dois descendentes, de acordo com os seguintes valores de posição e velocidade:

$$x_a(t+1) = r\ x_a(t) + (1-r)\ x_b(t)$$

$$x_b(t+1) = r\ x_b(t) + (1-r)\ x_a(t)$$

$$v_a(t+1) = \frac{v_a(t) + v_b(t)}{\| v_a(t) + v_b(t)\|} \| v_a(t)\|$$

$$v_b(t+1) = \frac{v_a(t) + v_b(t)}{\| v_a(t) + v_b(t)\|} \| v_b(t)\|$$

onde, $r \in [0,1]$

4. A melhor posição das partículas seleccionadas para reprodução toma o valor da posição actual.

30

Variantes

- Mecanismos de Mutação
  - Existem várias formas de mutar algumas partículas:
    - Mutar a solução “global best”
    - Mutar as componentes de cada solução:

31

Variantes

- Vizinhanças Dinâmicas
  - Definição das Vizinhanças
    - No algoritmo original, a vizinhança é definida pelo valor dos índices. Outra possibilidade reside na cálculo da vizinhança com base nas posições e correspondentes distâncias entre as partículas:

Uma partícula P_a encontra-se na vizinhança de uma partícula P_b se:

$$\frac{\|x_a - x_b\|}{d_{max}} < \varepsilon$$

onde d_{max} representa a distância máxima entre duas partículas e

$\varepsilon = \frac{3t+0.6t_{max}}{t_{maz}}$,

onde t representa a iteração actual e t_{max} representa o número máximo de iterações

32

Variantes

- Sub-swarms
  - Divisão Cooperativa
  - Divisão Cooperativa Híbrida
  - Predador-Presa
  - Ciclo de vida
  - Attractive and Repulsive PSO

33

Variantes

- …
  - Divisão cooperativa
    - Cada partícula é dividida em K sub-partículas.
    - Cada sub-problema é otimizado separadamente usando um sub-enxame.
      - No limite, cada dimensão e otimizada por um sub-PSO
    - Implicações?
      - E se existir dependência entre as variáveis?
      - Como avaliar a qualidade em cada dimensão?
      - Resolve k problemas de menor dimensão.

34

Variantes

- …
  - Divisão Cooperativa Híbrida
    - Definir sub-enxames (por k dimensões) e um enxame principal.
    - Enxame principal resolve o problema original, completo
    - Após uma iteração do algoritmo cooperativo, substituir uma partícula (selecionada aleatoriamente) do enxame principal com o vetor de contexto
    - Após uma iteração do PSO para o enxame principal, substitua uma partícula selecionada aleatoriamente de cada sub-enxame com o correspondente elemento na melhor posição global do enxame principal.

35

Variantes

- …
  - Predador-Presa
    - Introduz competição para balancear “global exploration – local exploitation”
    - Usa um segundo swarm de “predadores”
      - As “presas” são obrigadas a exploração global para se afastarem das partículas “predadoras”
      - Atualizam-se a sua posição também com base na distância ao predador mais próximo (mais uma componente)
      - Os predadores movem-se com base na distância ao ótimo global.

36

Variantes

- …
  - Ciclo de vida (life-cycle)
    - Permite modificar o comportamento das partículas.
      - Uma partícula pode exibir um dos seguintes três comportamentos: PSO; GA; Hill-climbing estocástico (apenas constante social).
      - Todos partem como partículas PSO.
      - Se o desempenho for inferior a um valor pré-especificado inicia o ciclo seguinte
        - Ideia subjacente: Iniciar com maior exploração global e terminar com exploração local.

37

Variantes

- …
  - ARPSO- Attractive and Repulsive PSO
    - Um swarm passando por duas fases. A transição é determinada pelo conceito de diversidade.
      - Diversidade – mede indiretamente a dispersão do enxame pelo espeço de pesquisa
      - Quando a diversidade ultrapassa um valor limite, inicia fase de atração.
        - Atração - usa algoritmo básico para atualização de velocidade.
        - Repulsão – subtrai componentes cognitivas e sociais à inércia:

$$v_{ij}(t+1) = wv_{ij}(t) - c_1 r_{1j}(t)(y_{ij}(t) - x_{ij}(t)) - c_2 r_{2j}(t)(\hat{y}_{ij}(t) - x_{ij}(t))$$

38

Otimização de Híper-parâmetros

- Problema:
  - UCI Breast Cancer Wisconsin data
    - https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html
- Classificador
  - KNN
    - Híper-parâmetros a otimizar:
      - Number of neighbors (integer)
      - Weight function {'uniform', 'distance'}
      - Algorithm {'ball_tree', 'kd_tree', 'brute'}
      - Leaf size (integer), used with the 'ball_tree' and 'kd_tree' algorithms

        - https://niapy.org/en/stable/tutorials/hyperparameter_optimization.html

```python
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsClassifier

from niapy.problems import Problem
from niapy.task import OptimizationType, Task
from niapy.algorithms.modified import HybridBatAlgorithm

def get_hyperparameters(x):
    """Get hyperparameters for solution `x`."""
    algorithms = ('ball_tree', 'kd_tree', 'brute')
    n_neighbors = int(5 + x[0] * 10)
    weights = 'uniform' if x[1] < 0.5 else 'distance'
    algorithm = algorithms[int(x[2] * 2)]
    leaf_size = int(10 + x[3] * 40)

    params = {
        'n_neighbors': n_neighbors,
        'weights': weights,
        'algorithm': algorithm,
        'leaf_size': leaf_size
    }
    return params

def get_classifier(x):
    """Get classifier from solution `x`."""
    params = get_hyperparameters(x)
    return KNeighborsClassifier(**params)

class KNNHyperparameterOptimization(Problem):
    def __init__(self, X_train, y_train):
        super().__init__(dimension=4, lower=0, upper=1)
        self.X_train = X_train
        self.y_train = y_train

    def _evaluate(self, x):
        model = get_classifier(x)
        scores = cross_val_score(model, self.X_train, self.y_train, cv=2, n_jobs=-1)
        return scores.mean()
```

Otimização de Híper-parâmetros

- …

```python
X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=1234)


problem = KNNHyperparameterOptimization(X_train, y_train)

# We will be running maximization for 100 iters on `problem`
task = Task(problem, max_iters=100, optimization_type=OptimizationType.MAXIMIZATION)

algorithm = HybridBatAlgorithm(population_size=10, seed=1234)
best_params, best_accuracy = algorithm.run(task)

print('Best parameters:', get_hyperparameters(best_params))
```

```
   ...: print('Best parameters:', get_hyperparameters(best_params))
Best parameters: {'n_neighbors': 11, 'weights': 'distance', 'algorithm': 'ball_tree', 'leaf_size':
27}
```

Otimização de Híper-parâmetros

- …
  - Comparar o modelo otimizado com o “default model”:

```
In [10]: default_model = KNeighborsClassifier()
    ...: best_model = get_classifier(best_params)
    ...:
    ...: default_model.fit(X_train, y_train)
    ...: best_model.fit(X_train, y_train)
    ...:
    ...: default_score = default_model.score(X_test, y_test)
    ...: best_score = best_model.score(X_test, y_test)
    ...:
    ...: print('Default model accuracy:', default_score)
    ...: print('Best model accuracy:', best_score)
Default model accuracy: 0.9210526315789473
Best model accuracy: 0.9385964912280702
```

41

Seleção de “Features”

- Problema:
  - UCI Breast Cancer Wisconsin data
    - https://niapy.org/en/stable/tutorials/feature_selection.html
- Classificador
  - SVC

```
class SVMFeatureSelection(Problem):
    def __init__(self, X_train, y_train, alpha=0.99):
        super().__init__(dimension=X_train.shape[1], lower=0, upper=1)
        self.X_train = X_train
        self.y_train = y_train
        self.alpha = alpha

    def _evaluate(self, x):
        selected = x > 0.5
        num_selected = selected.sum()
        if num_selected == 0:
            return 1.0
        accuracy = cross_val_score(SVC(), self.X_train[:, selected], self.y_train, cv=2, n_jobs=-1).mean()
        score = 1 - accuracy
        num_features = self.X_train.shape[1]
        return self.alpha * score + (1 - self.alpha) * (num_selected / num_features)
```

```
In [6]: print(feature_names)
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

```
In [7]:
   ...: dataset = load_breast_cancer()
   ...: X = dataset.data
   ...: y = dataset.target
   ...: feature_names = dataset.feature_names
   ...:
   ...: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y,
random_state=1234)
```

42

Seleção de “Features”

- …

```
In [8]: problem = SVMFeatureSelection(X_train, y_train)
   ...: task = Task(problem, max_iters=100)
   ...: algorithm = ParticleSwarmOptimization(population_size=10, seed=1234)
   ...: best_features, best_fitness = algorithm.run(task)
   ...:
   ...: selected_features = best_features > 0.5
   ...: print('Number of selected features:', selected_features.sum())
   ...: print('Selected features:', ', '.join(feature_names[selected_features].tolist()))
Number of selected features: 4
Selected features: mean smoothness, mean concavity, mean symmetry, worst area


In [9]: model_selected = SVC()
   ...: model_all = SVC()
   ...:
   ...: model_selected.fit(X_train[:, selected_features], y_train)
   ...: print('Subset accuracy:', model_selected.score(X_test[:, selected_features], y_test))
   ...:
   ...: model_all.fit(X_train, y_train)
   ...: print('All Features Accuracy:', model_all.score(X_test, y_test))
Subset accuracy: 0.9210526315789473
All Features Accuracy: 0.9122807017543859
```

43

Referências

- Computational Intelligence – An Introduction, Andries Engelbrech, Cap. 16
- https://pypi.org/project/pyswarms/
- https://niapy.org/en/stable/tutorials

44