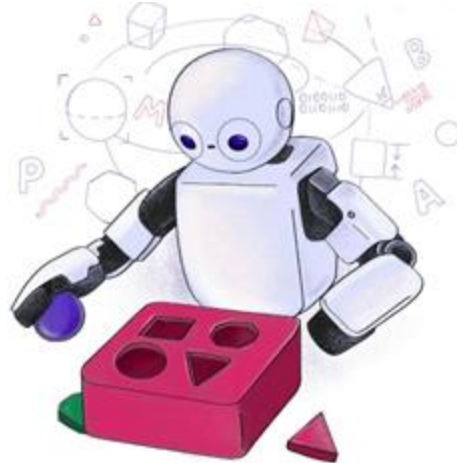


# TP558 - Tópicos avançados em Machine Learning: *Particle Swarm Optimization* (PSO)

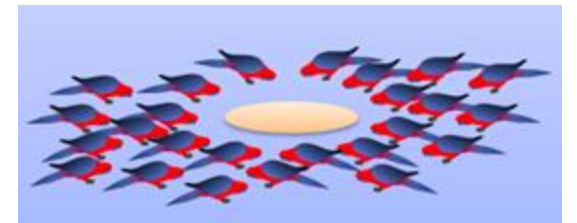


**Inatel**

Paulo Otavio Luczensky de Souza  
paulo.souza@mtel.inatel.br

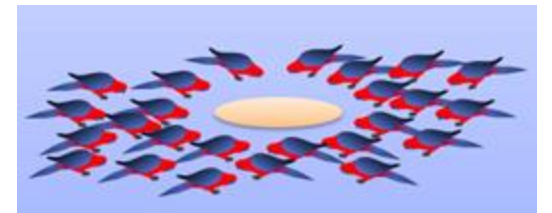
# Introdução

- Proposto pelos cientistas Eberhart e Kennedy (1995)
- Paradigma Evolucionista da IA (Algoritmos Genéticos, Vida Artificial)
- **Inspiração:** o comportamento social de animais (ex.: bandos de pássaros, cardume de peixes).
- Técnica de otimização estocástica que usa do conceito de vida artificial para realizar buscas em largos espaços.



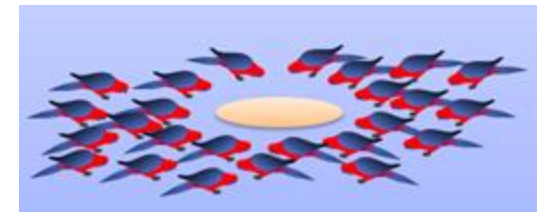
# Introdução

- Usado em problemas complexos de Otimização, como:
  - Engenharia e Indústria:
    - Otimização de trajetórias: rotas de robôs, drones ou veículos autônomos
  - Ciência de Dados e IA:
    - Treinamento de redes neurais: ajuste de pesos e hiperparâmetros



# Fundamentação teórica

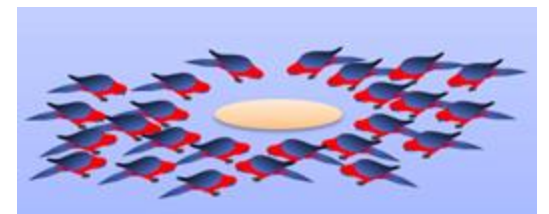
- Mas, por que os sociobiológicos utilizaram deste conceito de vida artificial?
  - Eles acreditavam que um grupo pode se beneficiar de todos os membros. Por exemplo, se um pássaro voa e procura comida de forma aleatória, todos os pássaros do bando podem compartilhar as descobertas e ajudar o bando a obter a melhor caçada.
  - Assim, a melhor solução encontrada pelo bando é a melhor solução no espaço.
- O PSO é uma **solução heurística**, ou seja, busca uma solução **boa o suficiente** para um problema complexo, usando regras práticas ou aproximações, em vez de garantir a solução ótima exata.
- **OBS:** Geralmente, a solução encontrada pelo PSO é bastante próxima do ótimo global.



# Fundamentação teórica

Princípios do comportamento social proposto por Millonas:

1. **Proximidade:** capacidade de realizar cálculos simples de espaço e tempo.
2. **Qualidade:** detectar mudanças de qualidade no ambiente e responder a elas.
3. **Resposta Diversa:** não se limitar a um caminho estreito para buscar recursos.
4. **Estabilidade:** não mudar o comportamento a cada alteração do ambiente.
5. **Adaptabilidade:** mudar o comportamento quando a mudança for relevante.

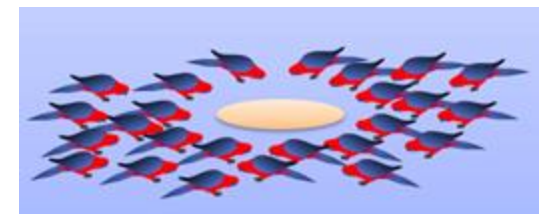


# Fundamentação teórica

Ano	Autor(es)	Modelo	Observações
1986/1987	Craig Reynolds	Boids(Bird-oid)	<ul style="list-style-type: none"><li>• Modelo de <b>simulação comportamental</b> (vida artificial).</li><li>• Objetivo: simulação realista de <b>movimento coletivo</b>.</li></ul>
1987	Heppner	Cornfield Model	<ul style="list-style-type: none"><li>• Modelo de <b>busca/otimização</b>.</li><li>• Introdução do “poleiro”.</li><li>• Objetivo: atingir o alvo.</li></ul>
1998	Shi & Eberhart	PSO com <i>Inertial Weight</i>	<ul style="list-style-type: none"><li>• Introduz o <i>inertial weight</i>(w), a fim de “equilibrar” a exploração.</li></ul>
2002	Clerc & Kennedy	Cornfield Vector	<ul style="list-style-type: none"><li>• Mantém a ideia do “poleiro”.</li><li>• Focado em convergência estável e no controle da velocidade.</li></ul>

# Fundamentação teórica - **Cornfield Model**

- **Cornfield Model** desenvolvido por Heppner em 1987.
  - Modelo de **busca/otimização**.
  - Introdução do “poleiro”.
  - Objetivo: atingir o alvo.
- Lista de variáveis a serem utilizadas:
  - **x, y**: posição atual da partícula (nos eixos).
  - **Vx, Vy**: velocidade da partícula (nos eixos).
  - **pbestx, pbesty**: melhor posição individual já encontrada pela partícula.
  - **gbestx, gbesty**: melhor posição global encontrada pelo enxame.
  - **a**: fator de ajuste para a influência de **pbest**.
  - **b**: fator de ajuste para a influência de **gbest**.
  - **rand**: número aleatório no intervalo [0,1], usado para fornecer estocasticidade.
  - **x0, y0**: posição do objetivo.



# Fundamentação teórica - Cornfield Model

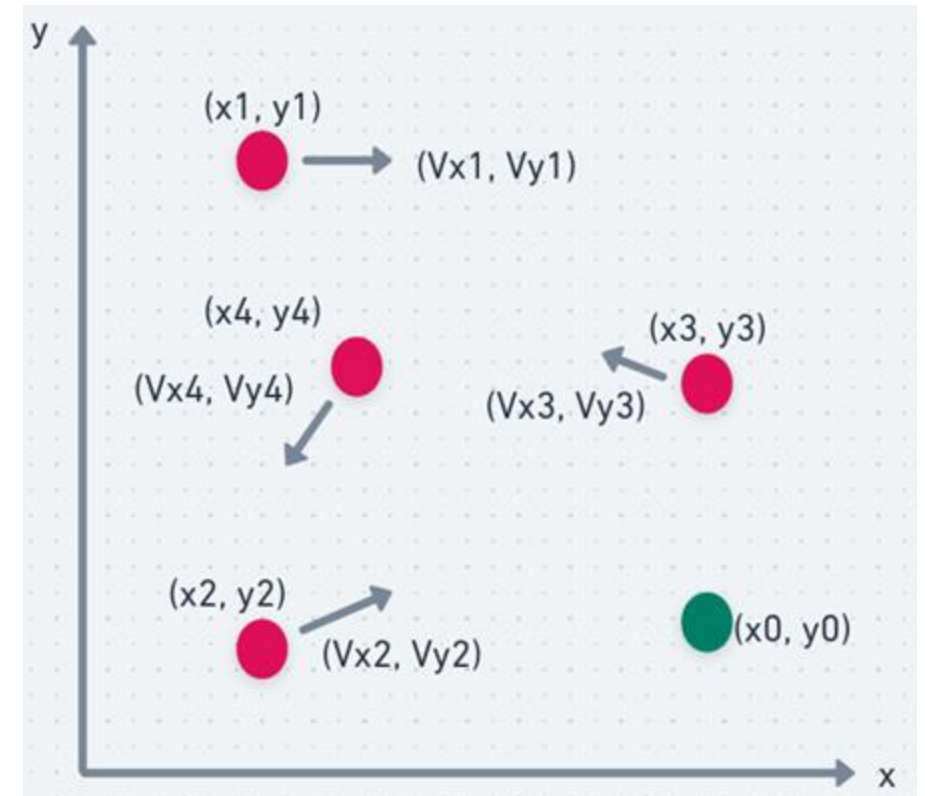
- A posição do objetivo é  $(x_0, y_0)$ .
- Cada indivíduo tem posição  $(x, y)$  e velocidade  $(V_x, V_y)$ .
- A distância até o objetivo mede o desempenho: quanto mais longe, pior o desempenho.
- Cada indivíduo tem **memória** e guarda sua melhor posição encontrada (**pbest**).
- Um fator  $\alpha$  ajusta a velocidade.
- Um número aleatório  $rand \in [0,1]$  é usado para atualizar a velocidade.

Regras de atualização (exemplo para o eixo x):

- # Atualização de  $V_x$
- if  $x > pbestx$ :
- $vx = vx - \text{random.random()} * \alpha$
- else:
- $vx = vx + \text{random.random()} * \alpha$

Regras de atualização (exemplo para o eixo y):

- # Atualização de  $V_y$
- if  $y > pbesty$ :
- $vy = vy - \text{random.random()} * \alpha$
- else:
- $vy = vy + \text{random.random()} * \alpha$





# Fundamentação teórica - Cornfield Model

- Os indivíduos são capazes de saber e memorizar a melhor localização do grupo (**gbest**).
- Um fator  $b$  ajusta a velocidade.

Regras de atualização (exemplo para o eixo x):

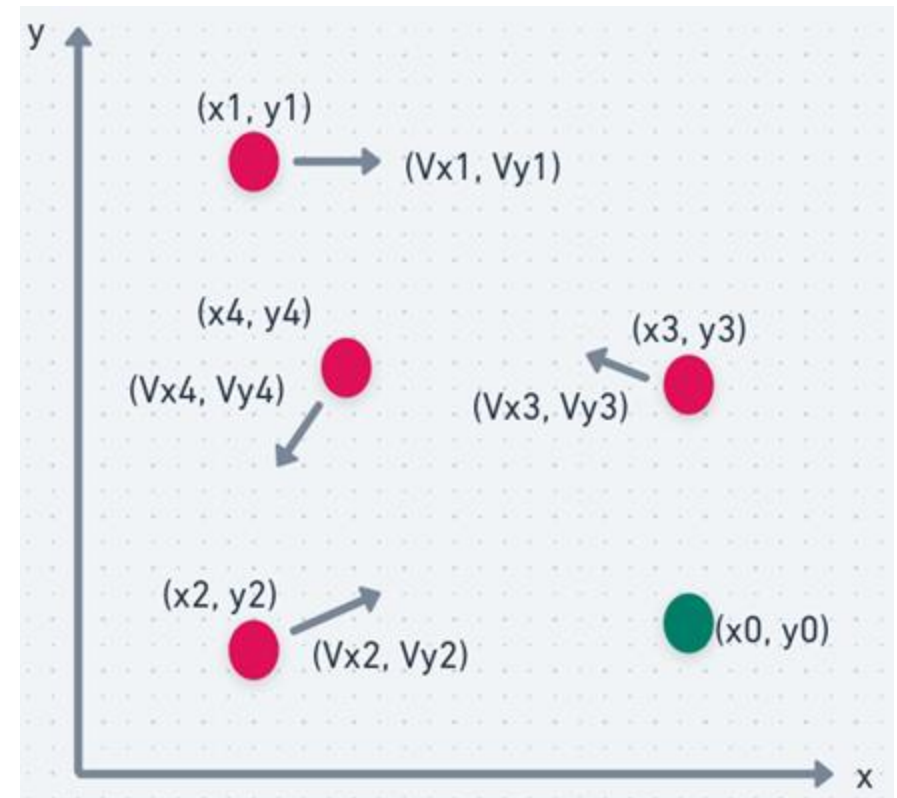
- # Atualização de  $V_x$  em relação ao  $gbest_x$
- if  $x > gbest_x$ :
- $vx = vx - \text{random.random()} * b$
- else:
- $vx = vx + \text{random.random()} * b$

Resultados de simulação mostraram que:

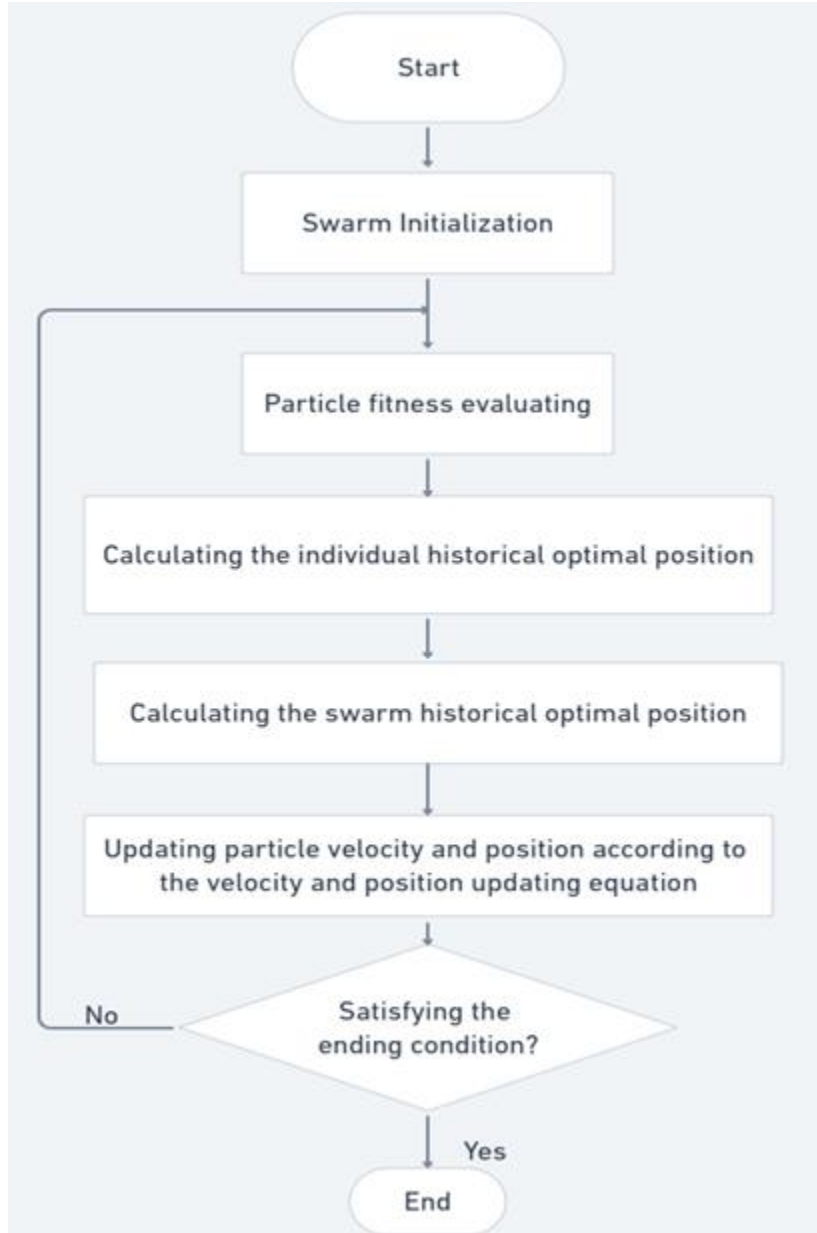
- Se  $a/b$  é **grande**, as partículas convergem rapidamente ao milharal.
- Se  $a/b$  é **pequeno**, as partículas se movem lentamente e de forma instável.

Modelo final de atualização para o eixo x:

- # Atualização de  $V_x$
- $vx = vx + 2 * \text{random.random()} * (pbest_x - x) + 2 * \text{random.random()} * (gbest_x - x)$
- # Atualização de  $x$
- $x = x + vx$



# Arquitetura e funcionamento - Cornfield Model

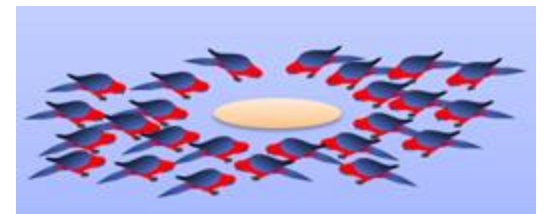


## Etapas do Fluxo (Cornfield Model – PSO)

- **Swarm Initialization**
  - Gerar partículas com posições (hiperparâmetros) e velocidades iniciais aleatórias.
- **Particle Fitness Evaluating**
  - Avaliar cada partícula usando uma função objetivo (ex.: acurácia do modelo).
- **Calculating the Individual Historical Optimal Position (pbest)**
  - Guardar o melhor desempenho já alcançado por cada partícula individualmente.
- **Calculating the Swarm Historical Optimal Position (gbest)**
  - Determinar o melhor desempenho global entre todas as partículas.
- **Updating Particle Velocity and Position**
  - Ajustar a velocidade e posição de cada partícula usando pbest, gbest e equações de atualização.
- **Satisfying the Ending Condition?**
  - Verificar critério de parada (número de iterações, convergência, etc.).
- **End**
  - Retornar os melhores hiperparâmetros encontrados (solução final).

# Fundamentação teórica - PSO com *Inertial Weight*

- PSO com *Inertial Weight* desenvolvido por Shi e Eberhart em 1998.
  - Introduz o *inertial weight*( $\omega$ ), a fim de “equilibrar” a exploração.
  - Controle da influência da velocidade anterior da partícula.
- Lista de variáveis a serem utilizadas:
  - $x_{i,t}^d$  : posição da partícula  $i$  na dimensão  $d$  no instante de tempo  $t$ .
  - $v_{i,t+1}^d$  : velocidade da partícula  $i$  na dimensão  $d$  no tempo  $t$ .
  - $p_{i,t}^d$  : melhor posição individual encontrada pela partícula  $i$  (pbest).
  - $p_{g,t}^d$  : melhor posição global do enxame (gbest).
  - $\omega$ : fator de inércia (controla exploração).
  - **c1**: coeficiente cognitivo (influência do pbest).
  - **c2**: coeficiente social (influência do gbest).
  - **rand**: número aleatório em  $[0,1]$ , adiciona estocasticidade.



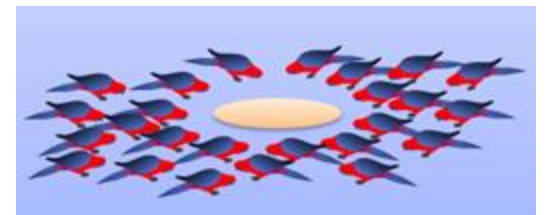
# Fundamentação teórica - PSO com *Inertial Weight*

Regras de atualização:

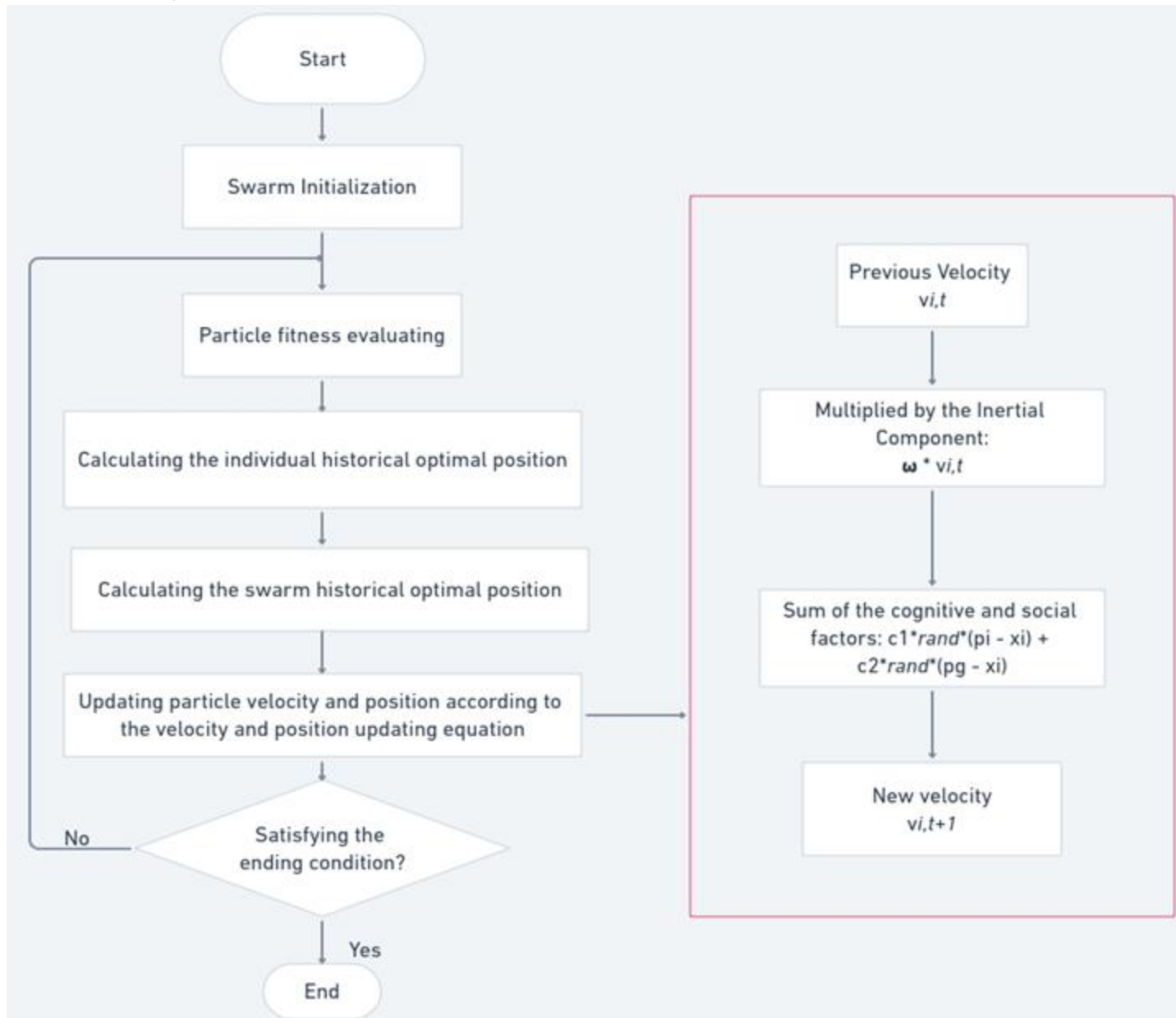
- # Atualização da velocidade
- $v[i][d] = ($ 
  - $w * v[i][d]$
  - $+ c1 * \text{rand}() * (pbest[i][d] - x[i][d])$
  - $+ c2 * \text{rand}() * (gbest[d] - x[i][d])$
- # Atualização da posição
- $x[i][d] = x[i][d] + v[i][d]$

Pontos a serem destacados:

- Valores maiores de  $\omega$  favorecem **exploração global**
- Valores menores de  $\omega$  favorecem **exploração local**
- Estratégia: **redução linear de  $\omega$  ao longo do tempo**
- Faixa sugerida:  $\omega \in [0.9, 1.2]$
- Garantiu **melhora significativa no desempenho** do PSO



# Arquitetura e funcionamento - PSO com *Inertial Weight*



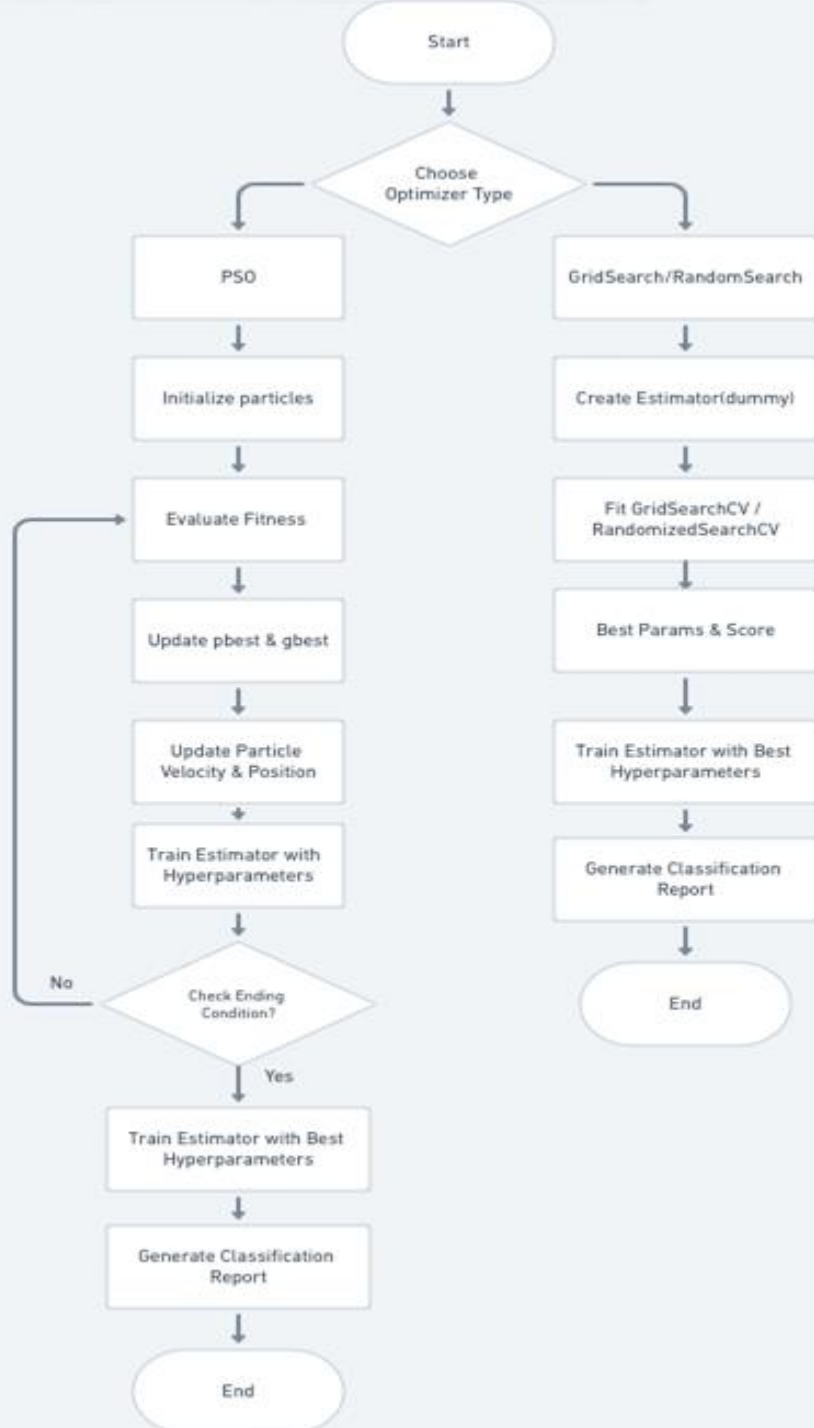
## Etapas com Inertial Weight

- **Previous Velocity ( $v_{i,t}$ )**
  - Cada partícula já possui uma velocidade da iteração anterior.
- **Inertial Component ( $\omega * v_{i,t}$ )**
  - O peso inercial ( $\omega$ ) controla quanto da velocidade anterior será mantido.
  - $\omega$  alto  $\rightarrow$  mais exploração (busca global).
  - $\omega$  baixo  $\rightarrow$  mais exploração local (refinamento).
- **Cognitive e Social Factors**
  - $c1 * rand() * (p_i - x_i) \rightarrow$  tendência de seguir o **melhor próprio histórico** (pbest).
  - $c2 * rand() * (p_g - x_i) \rightarrow$  tendência de seguir o **melhor global** (gbest).
- **New Velocity ( $v_{i,t+1}$ )**
  - A nova velocidade é a soma do termo inercial + componente cognitivo + componente social.
  - A partícula atualiza sua posição usando essa nova velocidade.

# Exemplo de Aplicação

- Utilizou-se o PSO para encontrar os melhores hiperparâmetros de um determinado modelo de ML.
- Comparou-se os seus resultados com 2 técnicas clássicas de hyperparameter tuning: **GridSearchCV** e **RandomizedSearchCV**.

Técnica	Tipo de Busca	Garantia de melhor combinação?	Custo Computacional	Espaço Contínuo	Interpretação
GridSearchCV	Exaustiva	Sim	Alto	Não	Alta
RandomizedSearchCV	Aleatória	Não	Médio	Parcial	Média
PSO	Heurística	Não	Médio-Baixo	Sim	Baixa



## PSO

1. **Initialize Particles** – cria partículas (candidatos de hiperparâmetros).
2. **Evaluate Fitness** – avalia desempenho de cada partícula(ex: acurácia do modelo).
3. **Update pbest & gbest** – guarda o melhor de cada partícula e o melhor global.
4. **Update Velocity & Position** – partículas se movem em direção a soluções melhores.
5. **Train Estimator with Hyperparameters** – treina o modelo com os valores atuais.
6. **Check Ending Condition** – verifica se deve parar (número de iterações/convergência).
7. **Train Final Estimator + Classification Report** – treina com os melhores hiperparâmetros e gera relatório final.

## GridSearch / RandomSearch

1. **Create Estimator** – inicializa o modelo base.
2. **Fit GridSearchCV / RandomizedSearchCV** – faz combinações ou amostragens de hiperparâmetros.
3. **Best Params & Score** – seleciona os melhores hiperparâmetros.
4. **Train Estimator with Best Hyperparameters** – treina o modelo final
5. **Generate Classification Report** – gera relatório de desempenho.

# Exemplo de Aplicação

## Dataset Utilizado

- **Base de dados:** Iris Dataset
- **Características:**
  - 150 amostras
  - 4 atributos (sepal length, sepal width, petal length, petal width)
  - 3 classes: *Setosa*, *Versicolor*, *Virginica*

## Configurações de Treinamento e Avaliação

Configuração	Valor	Observações
Divisão dos dados	70% treino / 30% teste	Proporção recomendada
Random Seed	42	Garante a reprodutibilidade dos resultados
Métricas de avaliação	<ul style="list-style-type: none"><li>• Accuracy</li><li>• Precision</li><li>• Recall</li><li>• F1-Score</li></ul>	Melhor análise do desempenho do modelo
Balanceamento das classes	Balanceado: 50 por classe	Não foi preciso aplicar técnicas de balanceamento



# Aplicação 1: KNN + PSO

## 1º) KNN (K-Nearest Neighbour)

Hiperparâmetro	Valores possíveis	Descrição
k	[1, 2, 3, 4, 5, ..., 15]	Número de vizinhos considerados.
distance_metric	["euclidean", "manhattan", "chebyshev", "minkowski", "l1", "l2"]	Métrica de distância
weighting_method	["uniform", "distance"]	Método de ponderação
algorithm	["brute", "kd_tree", "auto", "ball_tree"]	Algoritmo usado para busca dos vizinhos

## 2º) PSO

Hiperparâmetro	Valor	Descrição
c1	2.05	Coeficiente cognitivo (influência da melhor posição individual da partícula).
c2	2.05	Coeficiente social (influência da melhor posição global do enxame).
num_jobs	-1	Número de núcleos usados em paralelo (-1 = todos disponíveis).
w	0.72984	Peso inercial; controla o equilíbrio entre exploração global e exploração local.

# Resultados da Otimização do KNN

Método	Melhor Score	Melhores Hiperâmetros	Acurácia	Precisão	Observações
PSO	1.00	<ul style="list-style-type: none"><li>• k = 7</li><li>• metric = minkowski</li><li>• weights = uniform</li><li>• algorithm = auto</li></ul>	1.00	1.00	<p>Encontrou o melhor ajuste possível. PSO mostrou maior capacidade de exploração do espaço de busca.</p> <p>OBS: Explora o espaço de busca de forma mais flexível, <b>podendo encontrar soluções fora da grade predefinida.</b></p>
GridSearch	0.96273	<ul style="list-style-type: none"><li>• k = 5</li><li>• metric = chebyshev</li><li>• weights = uniform</li><li>• algorithm = brute</li></ul>	1.00	1.00	<p>Obteve boa performance, mas ficou preso a uma região específica do espaço de hiperparâmetros. Melhor score menor que o PSO.</p> <p>OBS: Garante a melhor solução <b>dentro da grade definida</b>, mas pode ser limitado se a grade não incluir combinações ideais.</p>
RandomSearch	0.96273	<ul style="list-style-type: none"><li>• k = 11</li><li>• metric = minkowski</li><li>• weights = uniform</li><li>• algorithm = kd_tree</li></ul>	1.00	1.00	<p>Similar ao GridSearch. Encontrou solução competitiva, mas sem explorar combinações mais variadas como o PSO.</p>

- **Melhor Score:** média da acurácia obtida na validação cruzada durante a busca de hiperparâmetros.
- **Acurácia:** proporção de previsões corretas no conjunto de teste (taxa global de acertos).
- **Precisão:** proporção de exemplos corretamente classificados como positivos em relação a todos os previstos como positivos (mede a qualidade dos acertos, evitando falsos positivos).

# Aplicação 2: Decision Tree + PSO

## 1º) Decision Tree

Hiperparâmetro	Valores Possíveis	Descrição
criterion	["gini", "entropy", "log_loss"]	Função usada para medir a qualidade da divisão de cada nó.
splitter	["best", "random"]	Estratégia usada para escolher a divisão em cada nó.
min_samples_split	[2, 5, 10]	Número mínimo de amostras necessárias para dividir um nó interno.
max_features	["sqrt", "log2"]	Número máximo de features consideradas ao procurar a melhor divisão em cada nó. Limitar este número ajuda a reduzir overfitting e aumentar a diversidade da árvore.

## 2º) PSO

Hiperparâmetro	Valor	Descrição
c1	2.05	Coeficiente cognitivo (influência da melhor posição individual da partícula).
c2	2.05	Coeficiente social (influência da melhor posição global do enxame).
num_jobs	-1	Número de núcleos usados em paralelo (-1 = todos disponíveis).
w	0.72984	Peso inercial; controla o equilíbrio entre exploração global e exploração local.

# Resultados da Otimização do Decision Tree

Método	Melhor Score	Melhores Hiperparâmetros	Acurácia	Precisão	Observações
PSO	1.0	<ul style="list-style-type: none"><li>• splitter = best</li><li>• criterion = gini</li><li>• min_samples_split = 10</li><li>• max_features=log2</li></ul>	1.00	1.00	<p>Encontrou configuração mais completa, incluindo hiperparâmetros não explorados pelo Grid/Random.</p> <p>OBS: Explora o espaço de busca de forma mais flexível, <b>podendo encontrar soluções fora da grade predefinida.</b></p>
GridSearch	0.9333	<ul style="list-style-type: none"><li>• splitter = best</li><li>• criterion = gini</li><li>• min_samples_split = 10</li><li>• max_features = sqrt</li></ul>	0.97	0.97	<p>Convergiu para uma solução estável, mas com score menor que PSO.</p> <p>OBS: Garante a melhor solução <b>dentro da grade definida</b>, mas pode ser limitado se a grade não incluir combinações ideais.</p>
RandomSearch	0.93333	<ul style="list-style-type: none"><li>• splitter = best</li><li>• criterion = gini</li><li>• min_samples_split = 10</li><li>• max_features = sqrt</li></ul>	0.97	0.97	Resultado idêntico ao GridSearch

- **Melhor Score:** média da acurácia obtida na validação cruzada durante a busca de hiperparâmetros.
- **Acurácia:** proporção de previsões corretas no conjunto de teste (taxa global de acertos).
- **Precisão:** proporção de exemplos corretamente classificados como positivos em relação a todos os previstos como positivos (mede a qualidade dos acertos, evitando falsos positivos).

## Vantagens e Desvantagens: PSO vs Grid/Random Search

Ponto de Comparação	PSO	Grid / Random Search
Complexidade de implementação	Mais difícil de configurar( $c1$ , $c2$ , $w$ , $n^\circ$ de partículas)	Fácil implementação(definir espaço de busca e $n^\circ$ de iterações)
Exploração do espaço	Inteligente e adaptativa	<ul style="list-style-type: none"><li>• Grid Search: exaustivo</li><li>• Randomized: aleatório</li></ul> <p>OBS: Apesar de ser exaustivo, o Grid garante o melhor resultado. Já o Randomized, não possui garantia</p>
Eficiência Computacional	Geralmente mais rápido em espaços grande e contínuos	<ul style="list-style-type: none"><li>• Grid Search: alto custo para muitas combinações</li><li>• Randomized: Mais leve</li></ul>
Determinismo	Estocástico, resultados podem variar entre as soluções	<ul style="list-style-type: none"><li>• Grid Search: Determinístico</li><li>• Randomized: Estocástico</li></ul>
Flexibilidade	Funciona tanto em problemas contínuos quanto discretos	Melhor com parâmetros discretos e limitados

Perguntas?

# Referências

- [1] WANG, Dongshu; TAN, Dapei; LIU, Lei. *Particle swarm optimization algorithm: an overview*. *Soft Computing*, Springer-Verlag Berlin Heidelberg, 2017. DOI: 10.1007/s00500-016-2474-6. Disponível em: [https://kpfu.ru/staff\\_files/F\\_1407356997/overview.pdf](https://kpfu.ru/staff_files/F_1407356997/overview.pdf)
- [2] TAM, Adrian. *A gentle introduction to particle swarm optimization*. Machine Learning Mastery, 12 out. 2021. Disponível em: <https://machinelearningmastery.com/a-gentle-introduction-to-particle-swarm-optimization/>
- [3] JIE, Zheng. *Put Simply: Particle Swarm Optimisation*. Medium, 24 abr. 2023. Disponível em: <https://medium.com/@KrashKart/put-simply-particle-swarm-optimisation-b0e6064c575d>
- [4] BAYRAKTAR, Mert. *PSO-Hyperparameter-Selection* [repositório de código]. GitHub, 2024. Disponível em: <https://github.com/mert-byrktr/PSO-Hyperparameter-Selection>

Obrigado!