Step by Step: When and How To Use Stochastic Optimization

By Siobhan K Cronin

Goals

- Introduce Stochastic Process
- Introduce Optimization
- Step up SVM tuning scenario
- Explore 3 stochastic methods for SVM tuning

Why does stochastic even mean?

"A mathematical object defined as a collection of random variables."

- Wikipedia (Stochastic Optimization)

“In many stochastic processes, the movement to the next state or position depends on only the current state, and is independent from prior states or values the process has taken. Whereas in deterministic process if the initial point is known, the next step or result is predictable.”

- Ashish P Naik

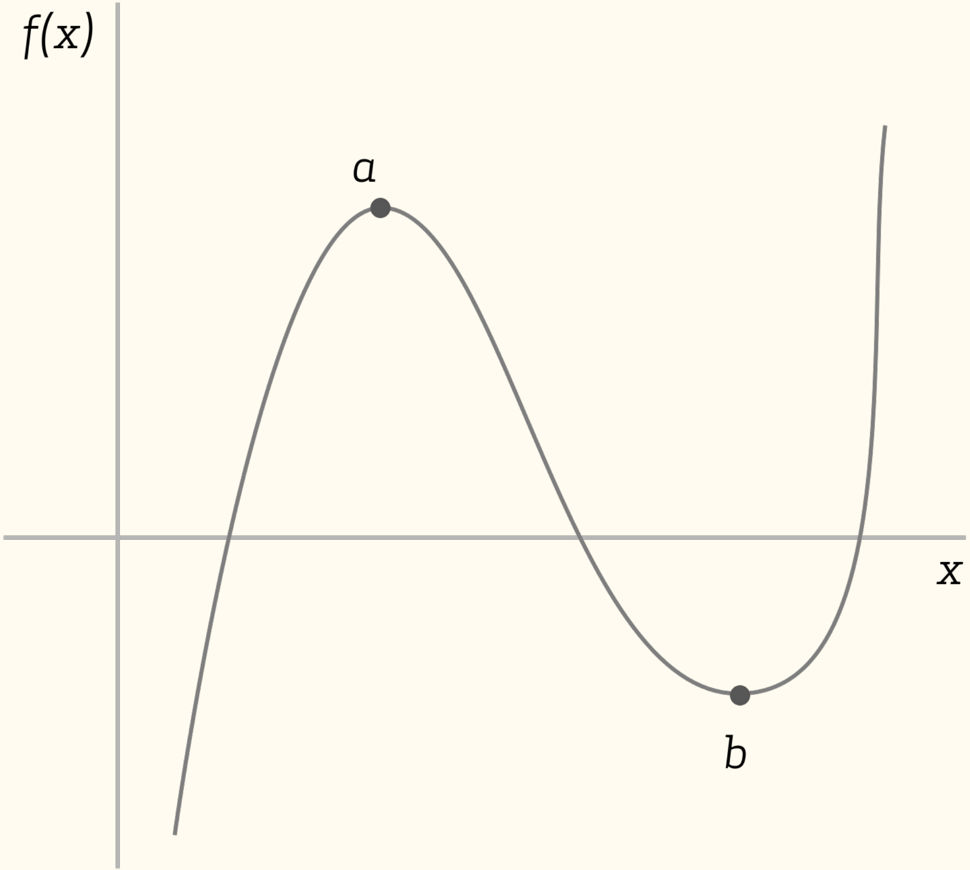
Can you give an example?


Deterministic
Stochastic

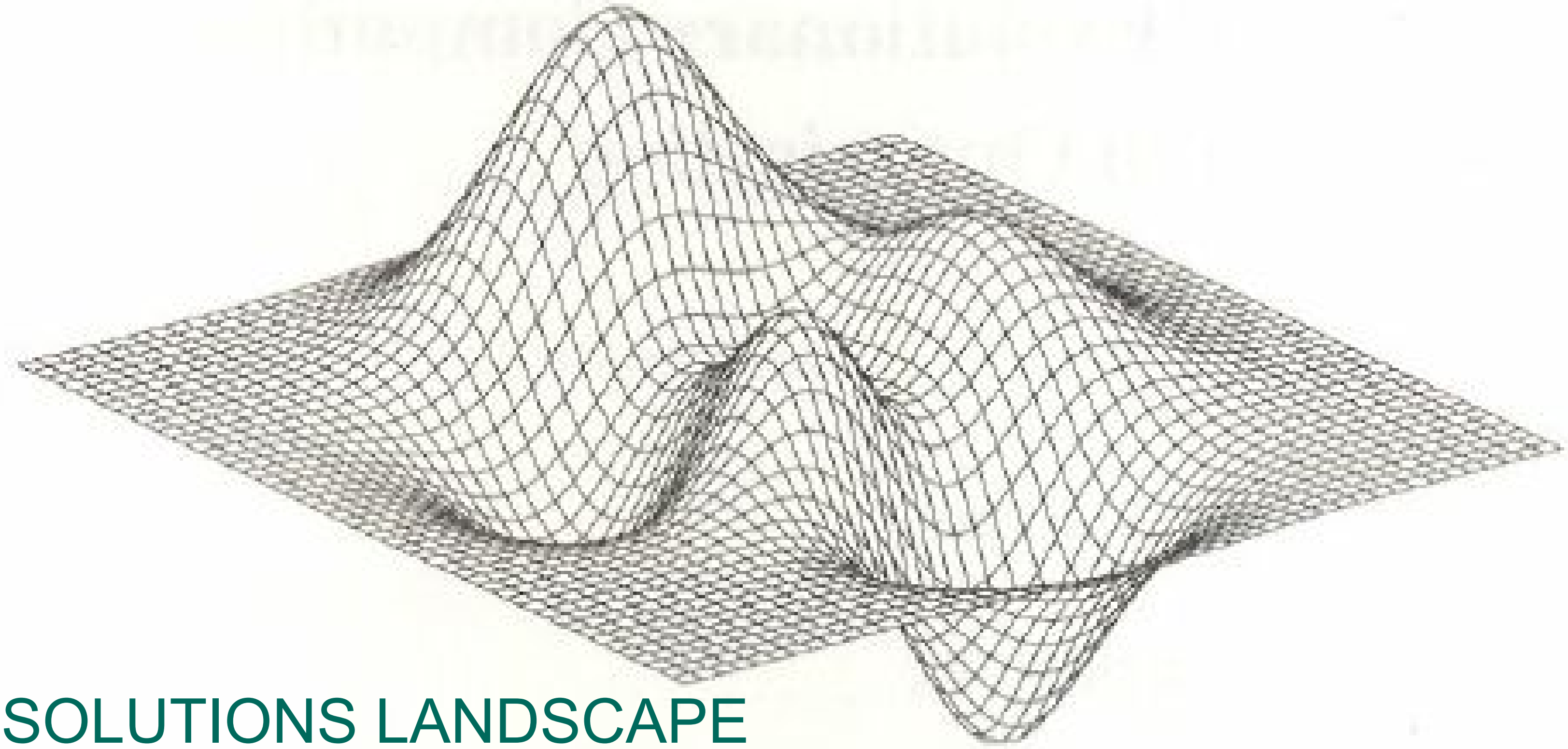
How do we choose what method to use?

Optimization

- **Objective Function:** the target output we want to optimize



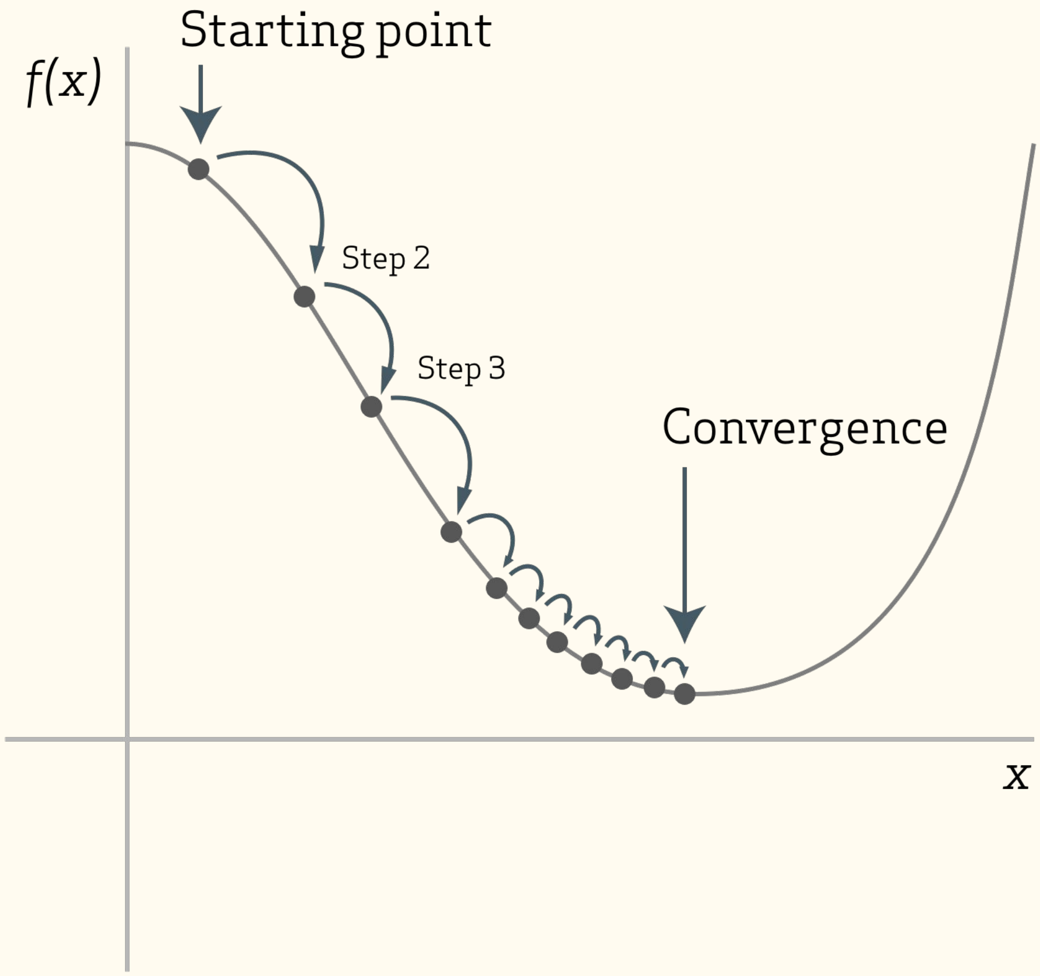
f(x)
a
x
b



SOLUTIONS LANDSCAPE

Optimization

- **Objective Function:** the target output we want to optimize
- **Optima**: highest point (maxima) or lowest point (minima)

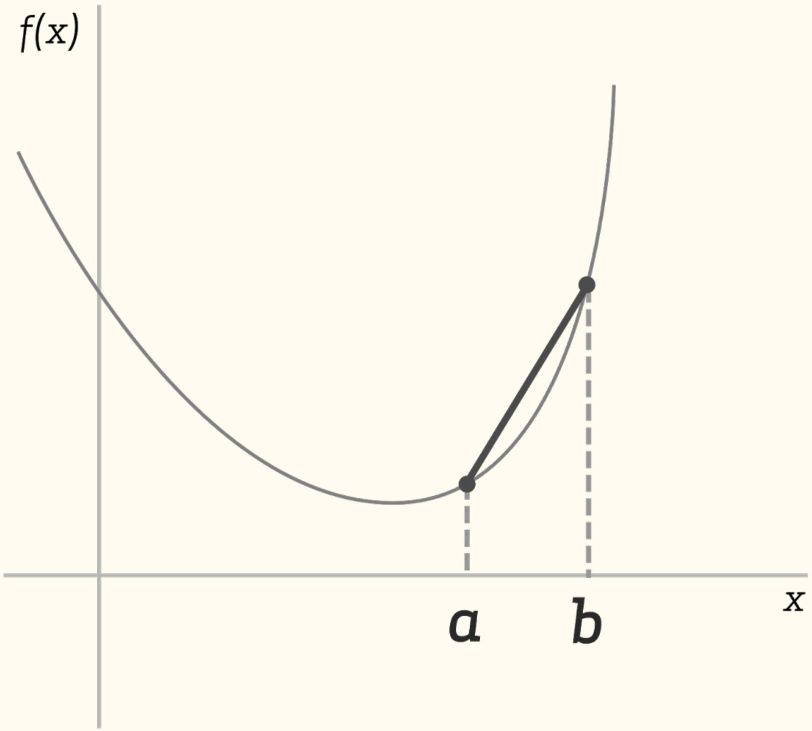


Starting point
$f(x)$
Step 2
Step 3
Convergence
x

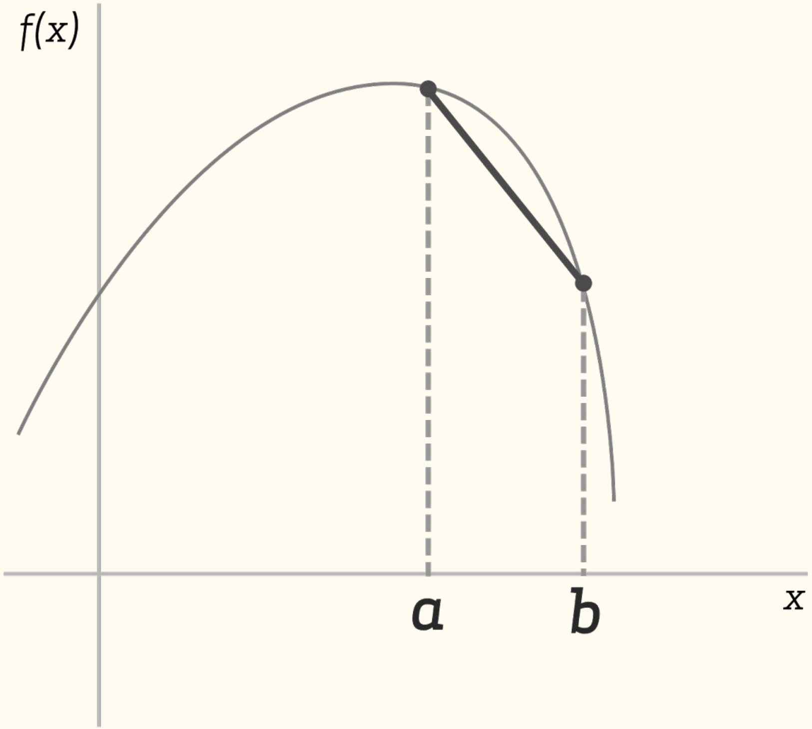
Optimization

- **Objective Function:** the target output we want to optimize
- **Optima**: highest point (maxima) or lowest point (minima)
- **Convex Functions**: functions where the local optima = global optima. Examples are the quadratic or exponential function.

Convex

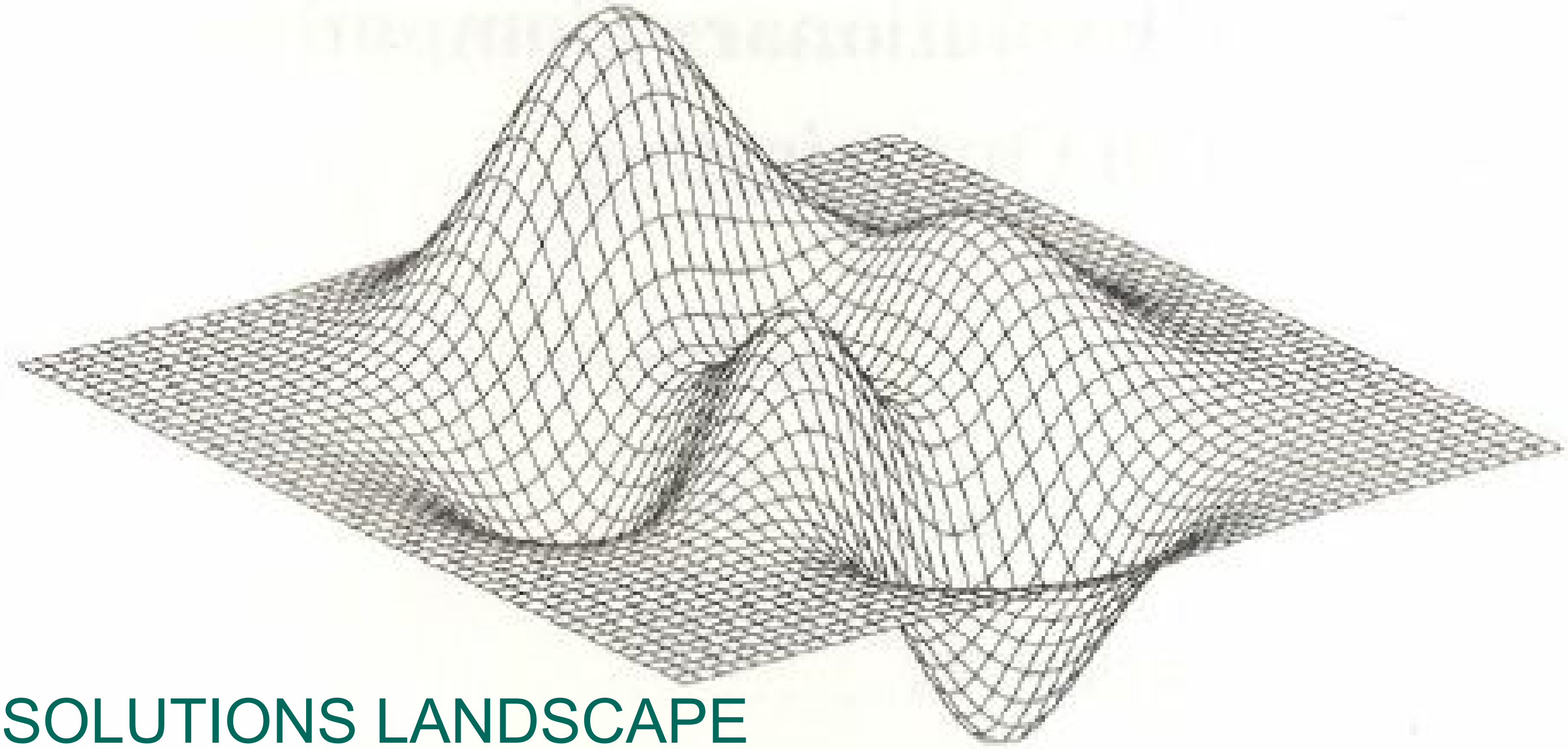


Concave



Optimization

- **Objective Function:** the target output we want to optimize
- **Optima**: highest point (maxima) or lowest point (minima)
- **Convex Functions**: functions where the local optima = global optima. Examples are the quadratic or exponential function.
- **Solutions Landscape**: mapping of all input parameters and the solutions they generate.

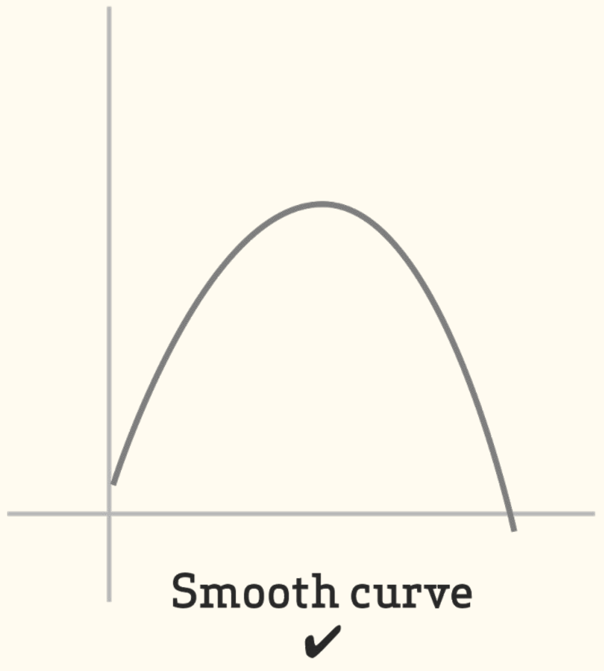


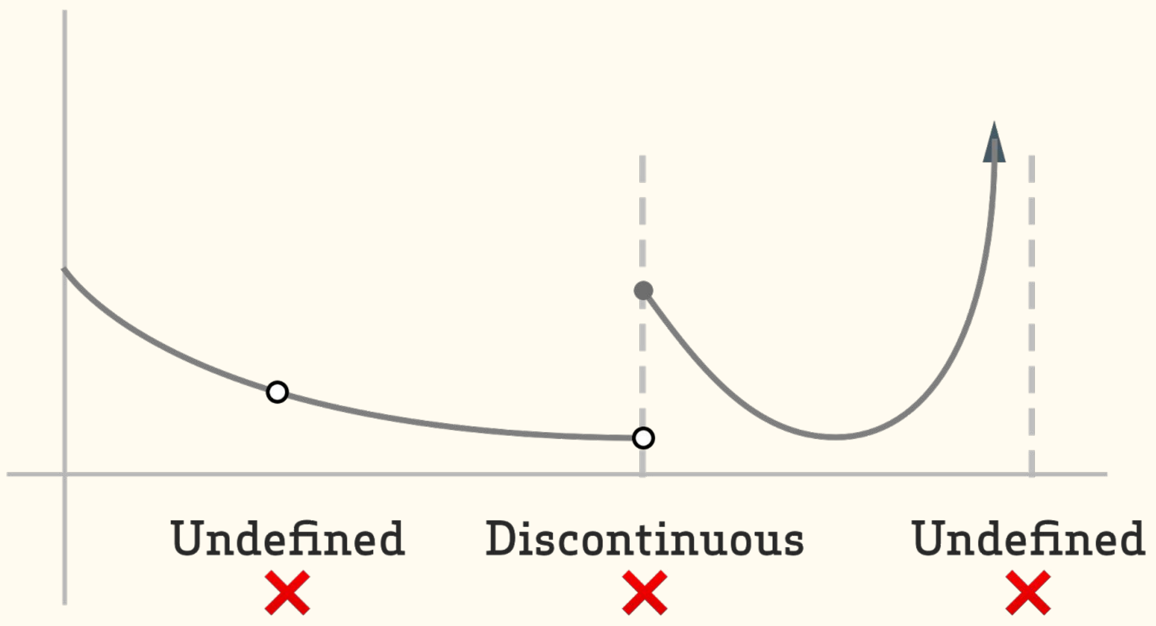
SOLUTIONS LANDSCAPE

Can’t we just use calculus to find the maxes and mins?

Short answer:

Yes. But sometimes we can’t differentiate, or would be satisfied with a solution that doesn’t require computing gradients.



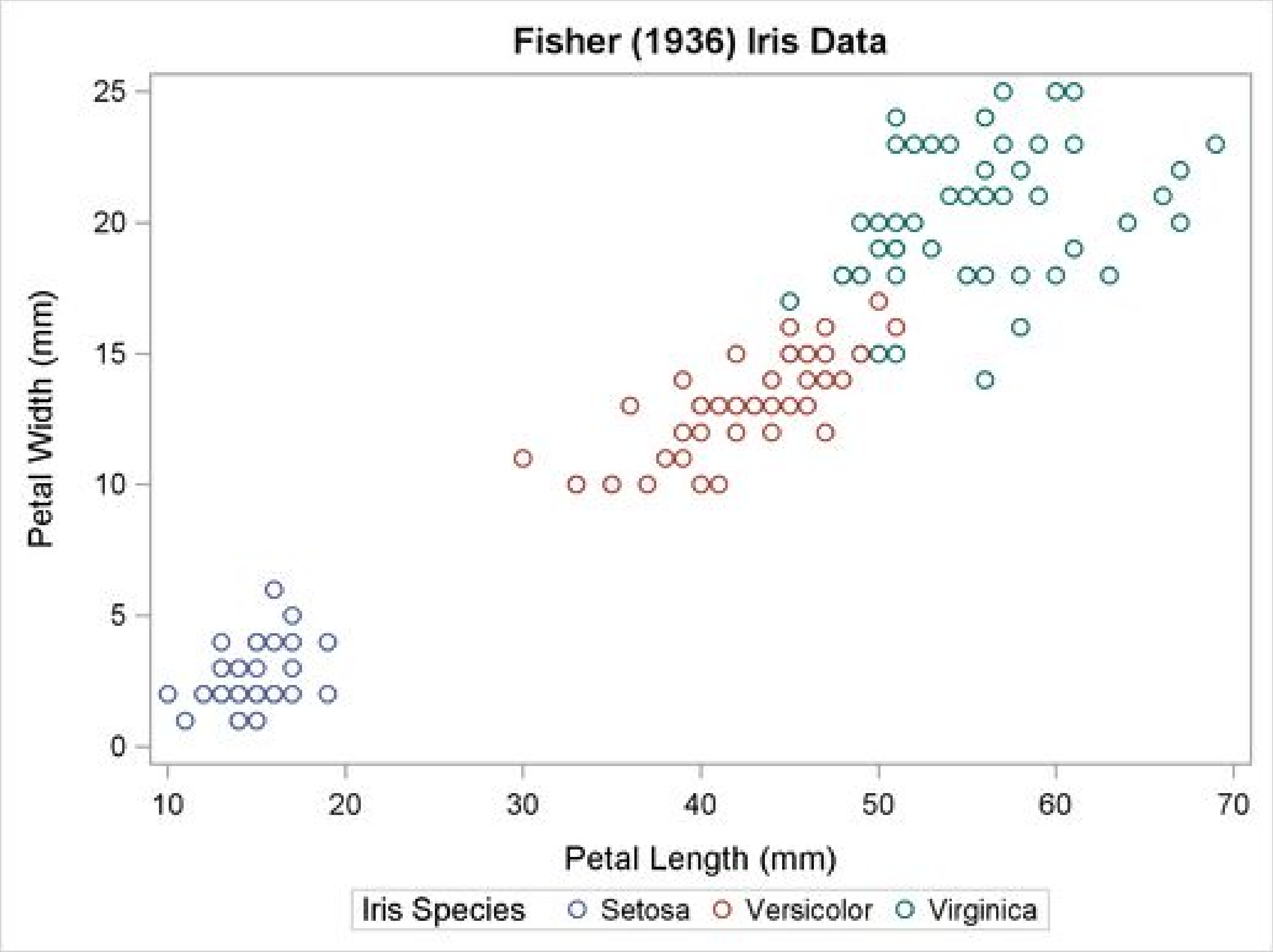


Will it differentiate?

OK, so what else can we do?

Fisher’s Iris flower data set

- 50 samples
- 3 species
- 4 features: length and width of sepals and petals


Fisher (1936) Iris Data
Petal Width (mm)
0
5
10
15
20
25
Petal Length (mm)
10
20
30
40
50
60
70
Iris Species
Setosa
Versicolor
Virginica

Support Vector Machine

- Supervised learning (we have labeled training data)
- Classification and Regression
- Effective in high dimension spaces

Support Vector Machine (how it works)

GOAL: Maximize the margin while minimizing the classification.

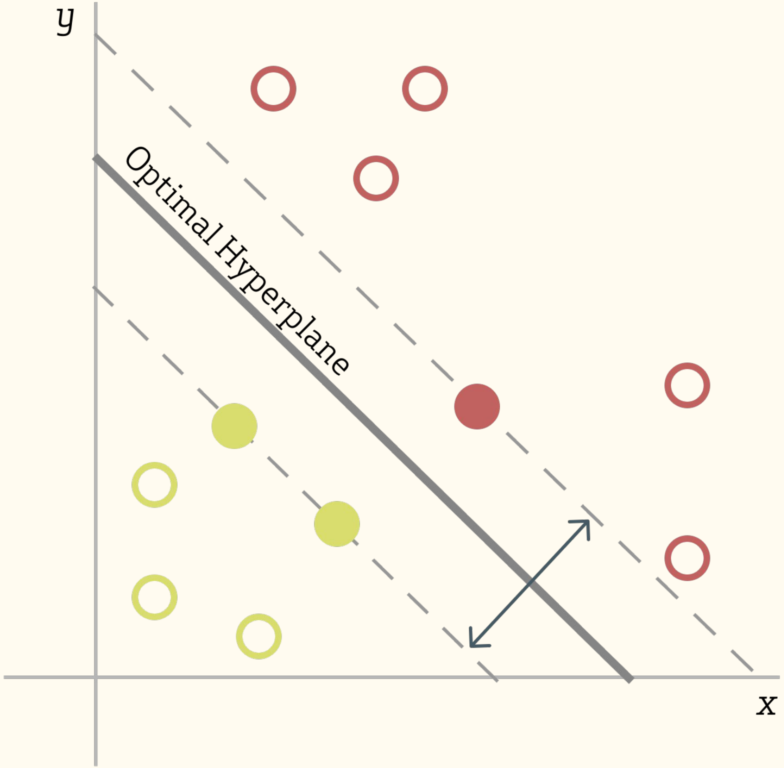
Data:

$$(\vec{x_1}, y_1), \ldots, (\vec{x_n}, y_n)$$

For each i, either:

$$\vec{w} \cdot \vec{x} - b \geq 1, y_i = 1$$

$$\vec{w} \cdot \vec{x} - b \leq -1, y_i = -1$$



code

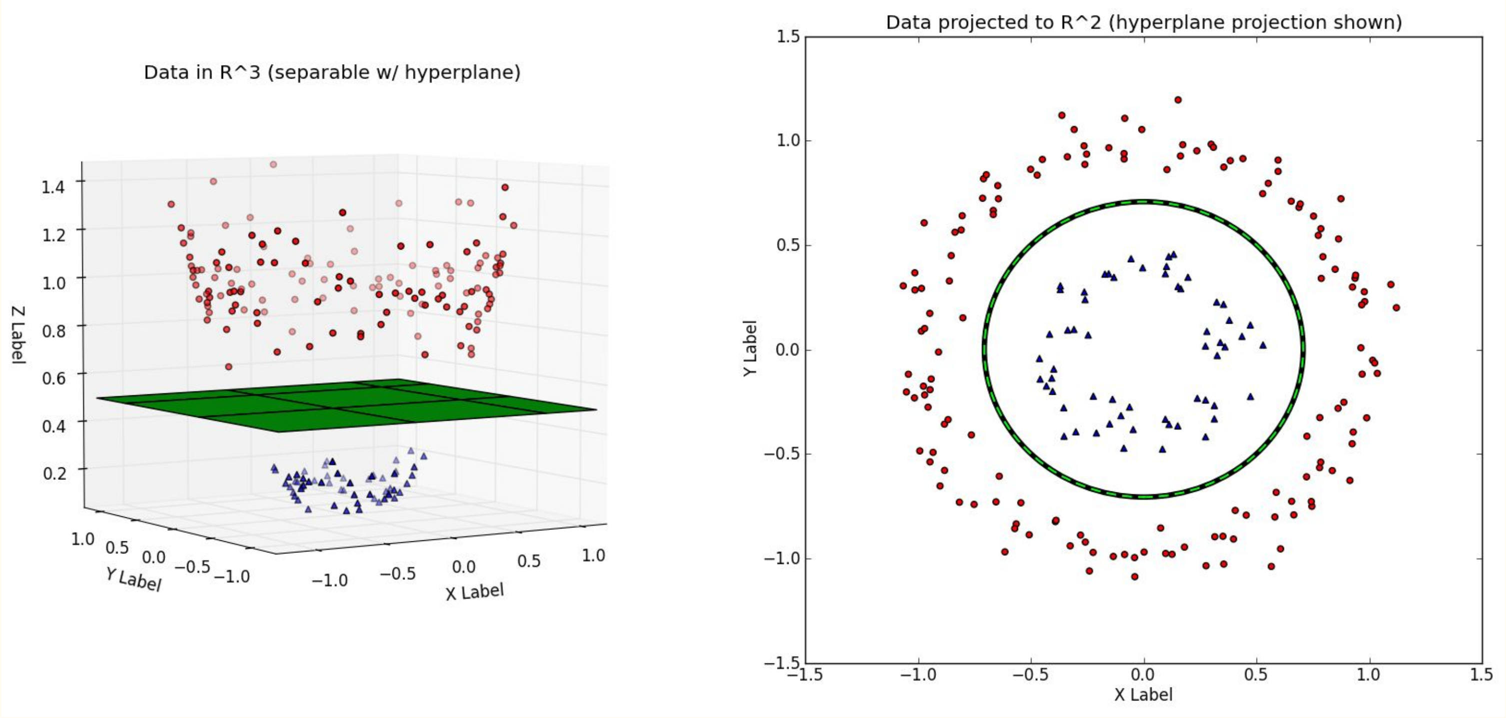
Support Vector Machine (code)

```python
from sklearn import svm, datasets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    iris.data,iris.target, test_size=0.4, random_state=0)

clf = svm.SVC(kernel='linear', C=1).fit(X_train, y_train)
clf.score(X_test, y_test)

```

```python
>>> clf.predict([[2., 2.]])
array([1])
```

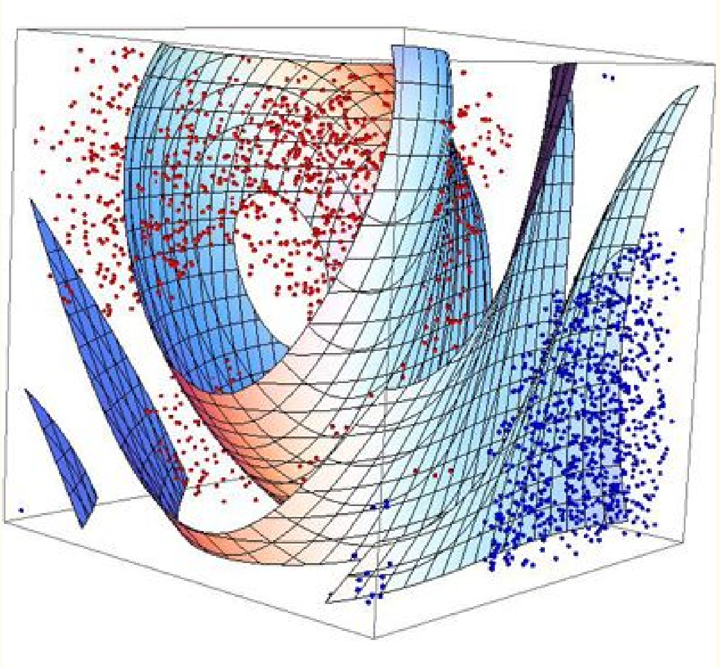
FYI: SVMs can also be used for nonlinear classification.

Support Vector Machine (kernel trick)



REF: How does going to a higher dimension help?

http://bit.ly/2xOOVZ4



What are the hyperparameters?

Support Vector Machine (hyperparameters)

Let’s assume we’re using a RBF kernel, and have the following two hyperparameters to tune:

- C
- Gamma

Support Vector Machine (hyperparameters)

C ("penalty" param)

"The C parameter trades off misclassification of training examples against simplicity of the decision surface."

"A low C makes the decision surface smooth, while a high C aims at classifying all training examples correctly by giving the model freedom to select more samples as support vectors." — scikit-learn documentation

Support Vector Machine (hyperparameters)

Gamma

“Intuitively, the gamma parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’. The gamma parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors.”

— scikit-learn documentation

Great.

How can we tune?

Grid Search CV (how it works)

```
from sklearn import svm, datasets
from sklearn.model_selection import GridSearchCV
parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
svc = svm.SVC()
clf = GridSearchCV(svc, parameters)
clf.fit(X_train, y_train)
```

Should we just use Grid Search unless the search space is intractable?

Yes, that’s a good place to start.

However…

Stochastic Methods Solving Hard Problems

MULTI

- Swarm intelligence systems for transportation engineering: principles and applications
- A comparative study on hyperparameter optimization for recommender system

ACO

- Applying ant colony optimization metaheuristic to solve forest transportation problems with side constraints
- Ant colony optimization algorithms for solving transportation problems

PSO

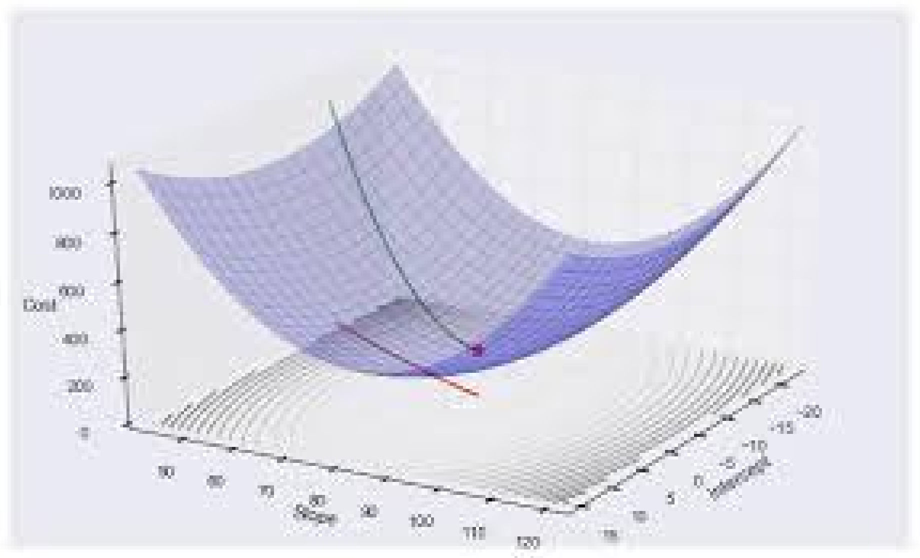
- Towards an optimal support vector machine classifier using a parallel particle swarm optimization strategy
- Particle swarm optimization for parameter determination and feature selection of support vector machines
- Accelerated particle swarm optimization and support vector machine for business optimization and applications

How do they work?

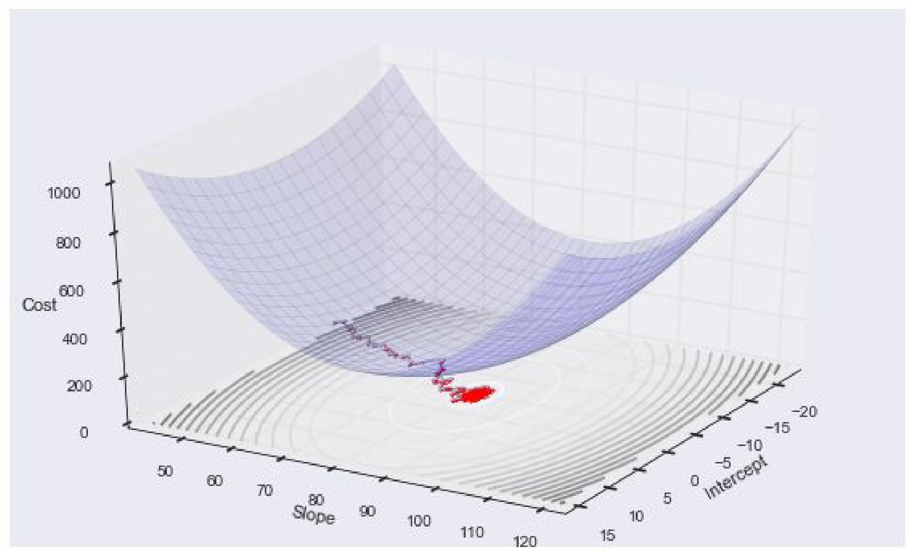
Random shuffling
Random sampling
Random making of “wrong” moves
Random “listening” to other agents

3 stochastic approaches:

1. SGD - Stochastic Gradient Descent
2. SA - Simulated Annealing
3. PSO - Particle Swarm Optimization



"Batch" Gradient Descent



Stochastic Gradient Descent

Stochastic Gradient Descent (how it works)

Select some initial weights $\vec{w}$ and a learning rate η

Iterate until approximate minimum has been reached.

Each step you randomly shuffle your training examples. **# RANDOM #**

Then for i = 1, 2, ..., n:

$$w : w - \eta \nabla Q_i(w)$$

Stochastic Gradient Descent (why stochastic?)

- Cycle through epochs, with a random shuffling every time

Stochastic Gradient Descent (why stochastic?)

- Cycle through epochs, with a random shuffling every time
- (Mini-batch) For N iterations, draw random samples from the training set

code

Stochastic Gradient Descent (code)

```
import numpy as np
from sklearn import linear_model

X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])
Y = np.array([1, 1, 2, 2])
clf = linear_model.SGDClassifier()
clf.fit(X, Y)
print(clf.predict([[-0.8, -1]]))
```

```
>>> clf.coef_
array([[ 9.9...,  9.9...]])
```

```
>>> clf.intercept_
array([-9.9...])
```

Random shuffling
Random sampling

What else do they do?

3 stochastic approaches

1. SGD - Stochastic Gradient Descent
2. SA - Simulated Annealing
3. PSO - Particle Swarm Optimization


GOODWIN
GOODWIN

Simulated Annealing (how it works)

```
S = So

for k = 0 through k_max:
   T = temperature(k/k_max)
   Snew = neighbor(S)
   if P(E(S), E(Snew), T) >= random(0, 1): #RANDOM#
        S = Snew
Output the final state S
```

Simulated Annealing (how it works)

Acceptance probability function:

$$a = e^{\frac{c_{old} - c_{new}}{T}}$$

Simulated Annealing (how it works)

c_old	c_new	T	a
10	8	1	7.39
8	10	1	0.14
8	8	1	1.00
10	8	0.8	12.18
8	10	0.8	0.08
8	8	0.8	1.00
10	8	0.5	54.60
8	10	0.5	0.02
8	8	0.5	1.00
10	8	0.3	785.77
8	10	0.3	0.00
8	8	0.3	1.00
10	8	0.1	485165195.41
8	10	0.1	0.00
8	8	0.1	1.00

Simulated Annealing (how it works)

c_old	c_new	T	a
10	8	1	7.39
8	10	1	0.14
8	8	1	1.00
10	8	0.8	12.18
8	10	0.8	0.08
8	8	0.8	1.00
10	8	0.5	54.60
8	10	0.5	0.02
8	8	0.5	1.00
10	8	0.3	785.77
8	10	0.3	0.00
8	8	0.3	1.00
10	8	0.1	485165195.41
8	10	0.1	0.00
8	8	0.1	1.00

<< Early on we might accept a bad move.

Simulated Annealing (how it works)

c_old	c_new	T	a
10	8	1	7.39
8	10	1	0.14
8	8	1	1.00
10	8	0.8	12.18
8	10	0.8	0.08
8	8	0.8	1.00
10	8	0.5	54.60
8	10	0.5	0.02
8	8	0.5	1.00
10	8	0.3	785.77
8	10	0.3	0.00
8	8	0.3	1.00
10	8	0.1	485165195.41
8	10	0.1	0.00
8	8	0.1	1.00

<< Later on, we stop risking “wrong” moves.

code

Simulated Annealing (code)

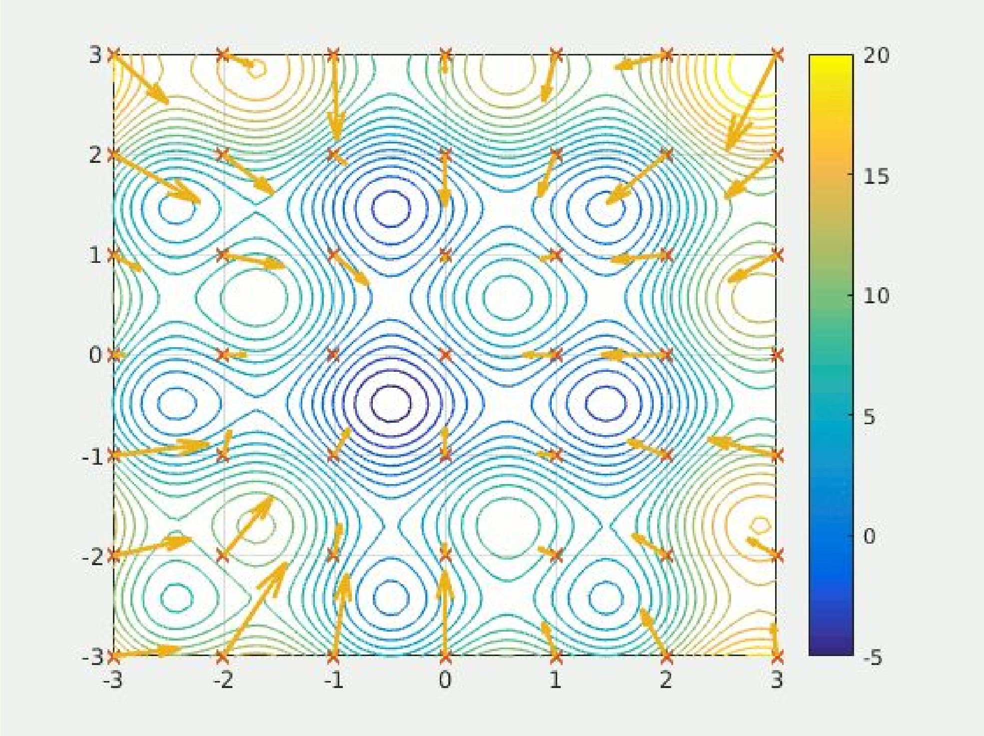
```
from sklearn import svm
from simulated_annealing.optimize import SimulatedAnneal
import numpy as np

svc_params = {'C':np.logspace(-8, 10, 19, base=2),
              'fit_intercept':[True, False]
             }
clf = svm.LinearSVC()
sa = SimulatedAnneal(clf, svc_params, T=10.0, T_min=0.001, alpha=0.75,
                         verbose=True, max_iter=0.25, n_trans=5, max_runtime=300,
                         cv=3, scoring='f1_macro', refit=True)
sa.fit(X_train, y_train)
print(sa.best_score_, sa.best_params_)
```

Random shuffling
Random sampling
Random making of “wrong” moves

3 stochastic approaches

1. SGD - Stochastic Gradient Descent
2. SA - Simulated Annealing
3. PSO - Particle Swarm Optimization

Particle Swarm Optimization (how it works)



Visualization from Wikipedia [Simulated Annealing]

Particle Swarm Optimization (how it works)

Approaches

LOCAL BEST: Will be attracted to their neighbors
GLOBAL BEST: Attracted to the best performing particles

Parameters

C1 (Cognitive factor): Take into account personal best
C2 (Social factor): Take into account global best
W (inertia weight)

Particle Swarm Optimization (how it works)

n = number of particles
pos = particle's position
p_i = best known position of particle i
g = best known position of entire swarm

SET UP:

```
for each particle i = 1, ..., n:
    initialize particle with uniform random vector
    initialize best known position to its initial position
    if f(p_i) < f(g):
        g = p_i
```

Particle Swarm Optimization (how it works)

PROCESS:

```
while termination not met:
    for each particle i = 1, ..., S:
            # random “listen” to cognitive or social  #RANDOM#
            # update particle’s velocity (either personal or global best) (v_i)
        update particle’s position pos = pos + v_i
        if f(pos) < f(p_i):
            p_i = pos
        if f(pos) < f(g):
            g = pos
```

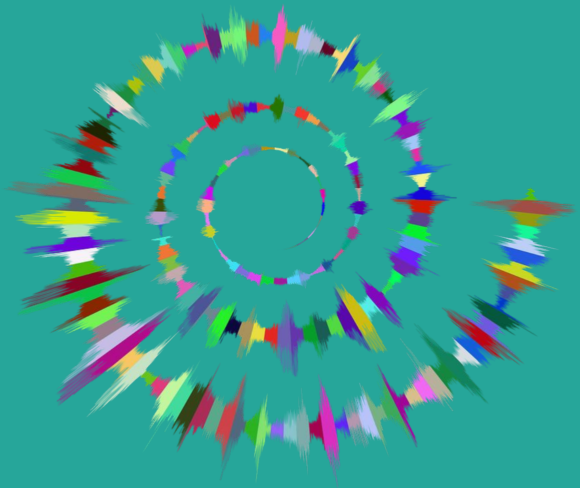
code

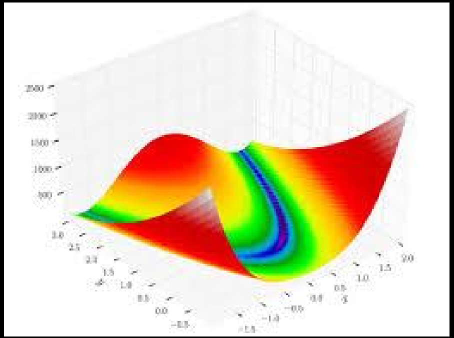
Particle Swarm Optimization (code)

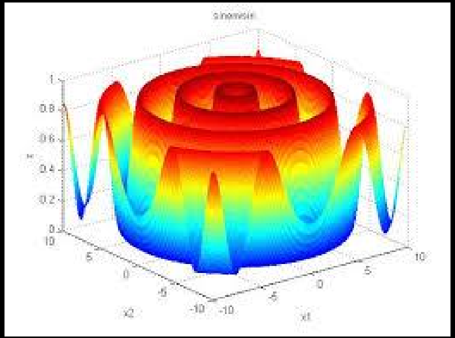
```
import PySwarms as ps
options = {'c1': 0.5, 'c2': 0.3, 'w':0.9}
optimizer = ps.single.GlobalBestPSO(n_particles=10, dimensions=2, options=options)
cost, pos = optimizer.optimize(fx.sphere_func, print_step=100, iters=1000, verbose=3)
```

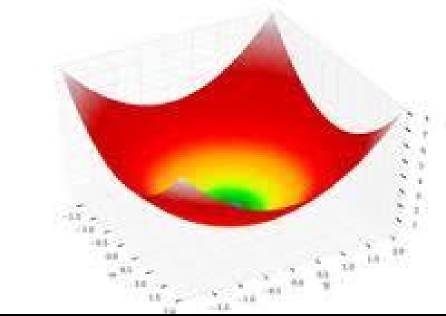
Random shuffling
Random sampling
Random making of “wrong” moves
Random “listening” to other agents

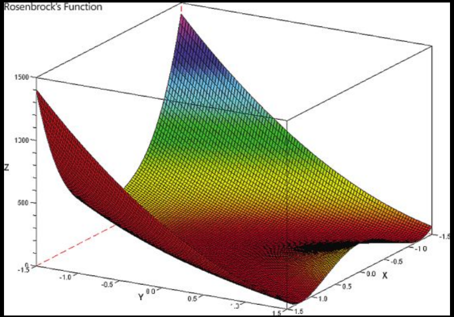
One final vista.

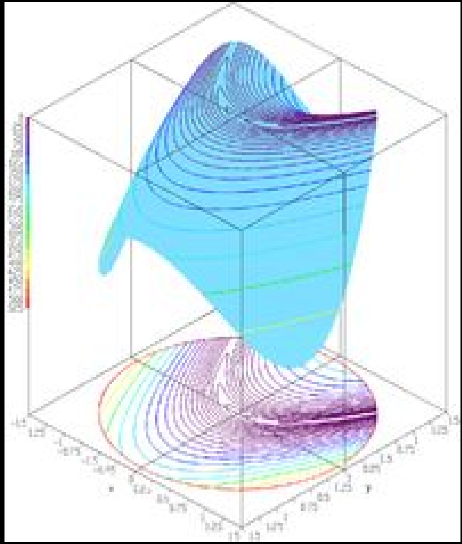


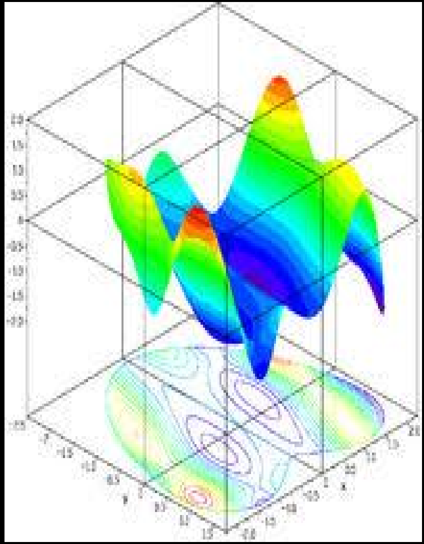












ARTIFICIAL LANDSCAPES (test functions)

Resources

- **Slides on GitHub:** https://github.com/SioKCronin/step_by_step
- **Convex Optimization MOOC**: http://bit.ly/2xrQRtK
- **PySwarms:** https://github.com/ljvmiranda921/pyswarms
- **BayesOpt:** https://github.com/rmcantin/bayesopt
- **HyperOpt:** https://github.com/hyperopt/hyperopt
- **(R) MLR**: http://bit.ly/2z7GhVM
- **SimAnnealing:** https://github.com/perrygeo/simanneal
- **TPOT (Automated ML)**: https://github.com/rhiever/tpot

Thanks and stay in touch!

@SioKCronin

Slides on GitHub

http://bit.ly/step_by_step_metis