Optimisation avec SciPy et idées d'applications pour l'apprentissage automatique

L'optimisation est souvent la dernière frontière, qui doit être conquise pour fournir la valeur réelle, pour une grande variété de processus commerciaux et technologiques. Nous montrons comment effectuer une optimisation avec le package d'analyse scientifique le plus populaire de Python - SciPy et discutons des applications uniques dans l'espace d'apprentissage automatique.

Tirthajyoti Sarkar

1 juin 2019 · 12 min de lecture

introduction

Vous vous souvenez peut-être d'un simple problème de calcul des années de lycée - trouver la quantité minimale de matériel nécessaire pour construire une boîte étant donné une restriction sur son volume.

4. An open-top box with a square bottom and rectangular sides is to have a volume of 256 cubic inches. Find the dimensions that require the *minimum* amount of material.

$$S = x^2 + 4xy$$
$$V = x^2y = 256 \rightarrow S = x^2 + 4x\left(\frac{256}{x^2}\right)$$

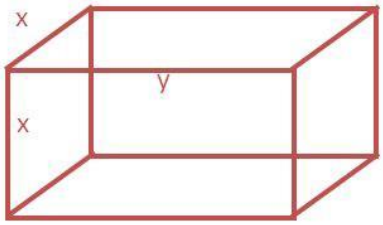
$$S = x^2 + \frac{1024}{x}$$

$$S' = 2x - \frac{1024}{x^2}$$

$$0 = 2x - \frac{1024}{x^2}$$

$$x = 8 \rightarrow y = 4$$

8 x 8 x 4



$$S'' = 2 + \frac{2048}{x^3} > 0$$

therefore a min

Assez simple?

Il est utile de réfléchir un peu à ce problème et de reconnaître que le même principe appliqué ici est largement utilisé dans les problèmes commerciaux et sociaux complexes et à grande échelle.

Examinez attentivement le problème ci-dessus. La contrainte est un volume fixe. **Considérez cela comme un livrable commercial** (**c'est-à-dire un** engagement envers le client).

Mais le but du problème est de trouver le *matériel minimum nécessaire* (en termes de surface). **Considérez cela comme lié à la marge bénéficiaire du producteur** (moins il faut de matériel, moins il y a de coût de production pour le même prix de vente, et donc une marge bénéficiaire plus élevée).

L'optimisation mathématique est au cœur des solutions aux problèmes majeurs de l'entreprise en ingénierie, finance, santé, affaires socio-économiques. Presque tous les problèmes commerciaux se résument à la minimisation d'un certain type de coût des ressources ou à la maximisation d'un type de profit compte tenu d'autres contraintes.

Un processus d'optimisation est également l'âme de la recherche opérationnelle, qui est intimement liée à l'analyse commerciale moderne basée sur les données . De cette manière, il est également étroitement lié au pipeline de la science des données, utilisé dans pratiquement toutes les entreprises aujourd'hui.

Bien que l'on ait beaucoup écrit sur les aspects de la gestion des données et de la modélisation prédictive d'un projet de science des données, la dernière frontière consiste souvent à résoudre un problème d'optimisation à l'aide de modèles basés sur les données qui peuvent **améliorer les résultats de l'entreprise en réduisant les coûts ou en améliorant la productivité** .

Pourquoi un problème d'analyse commerciale exige toutes vos compétences en science des données
Comment la pratique moderne de l'analyse basée sur les données, pour un problème commercial réel, est toujours un mélange complexe de…
medium.com

Outre la motivation purement commerciale, le sujet de l'optimisation mérite d'être étudié à lui seul, car il est au cœur de tous **les** algorithmes d' **apprentissage automatique** (ML) commençant à une **simple régression linéaire** jusqu'aux **réseaux de neurones profonds** . Comprendre les divers algorithmes, les limites et la formulation des problèmes d'optimisation peut produire des informations précieuses pour résoudre efficacement les problèmes de ML.

Ce qui se trouve en dessous? L'optimisation au cœur du Machine Learning
Nous montrons les principaux cadres d'optimisation derrière les techniques d'apprentissage automatique / de modélisation statistique les plus populaires.
versdatascience.com

Par conséquent, il est impératif pour un scientifique des données d'apprendre les outils et les cadres de base pour résoudre les problèmes d'optimisation et avoir un impact réel.

Python et SciPy pour l'optimisation

Python est devenu la lingua franca de facto de l'analyse, de la science des données et de l'apprentissage automatique . Par conséquent, il est logique de discuter des packages et des cadres d'optimisation au sein de l'écosystème Python.

Comment Python est-il devenu le langage de choix pour la science des données?
Pendant de nombreuses années, R a été le choix évident pour ceux qui se lancent dans la science des données. Ces dernières années, quelque chose a changé…
www.netguru.com

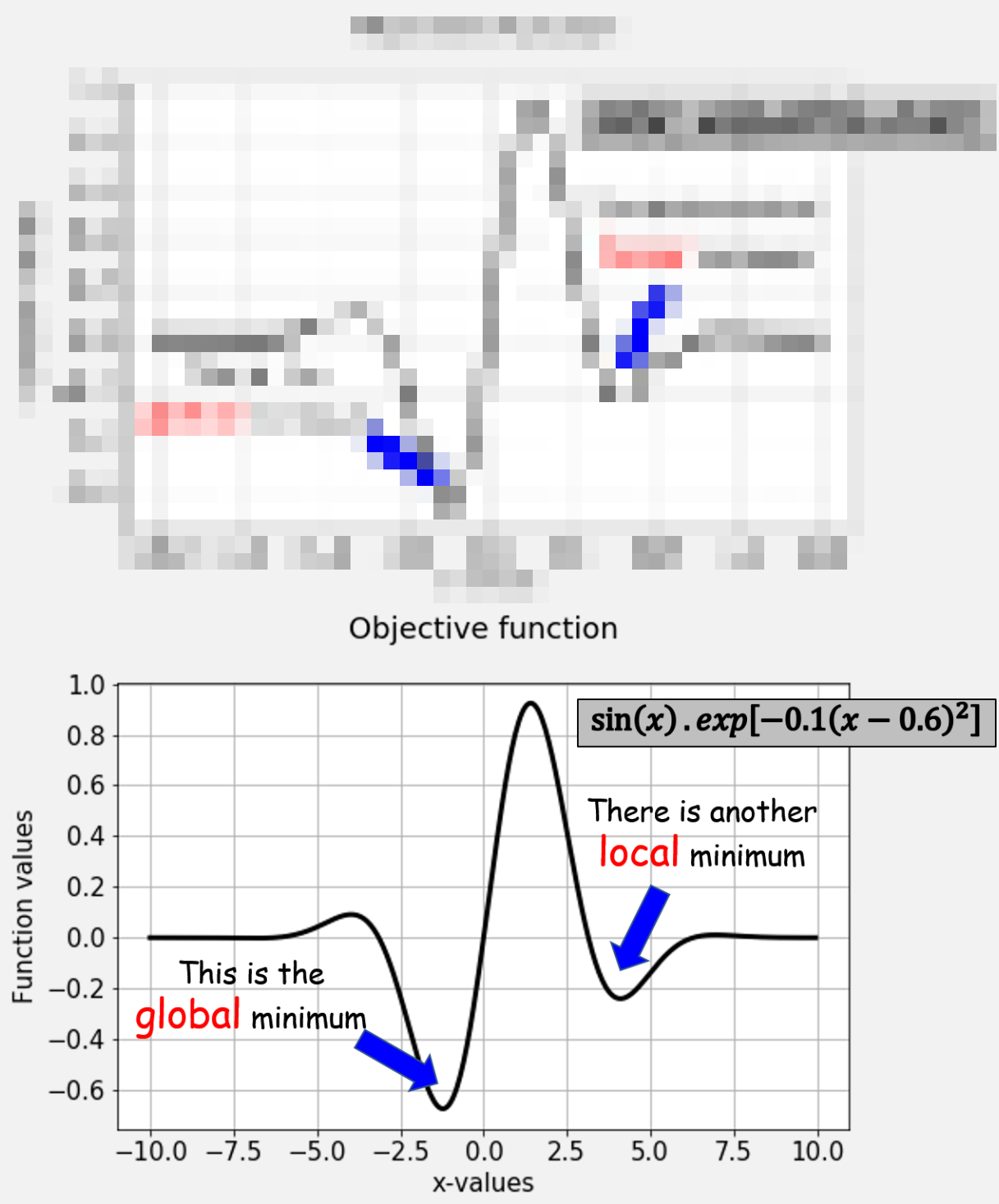
Dans mes articles précédents, j'ai couvert **la programmation linéaire** et d'autres **méthodes d'optimisation discrète** utilisant Python et introduit des packages puissants tels que PuLP et CVXPY .

Dans cet article, je couvrirai les algorithmes d'optimisation disponibles dans l'écosystème SciPy . SciPy est le package Python le plus utilisé pour l'analyse scientifique et mathématique et il n'est pas étonnant qu'il se vante de routines d'optimisation puissantes mais faciles à utiliser pour résoudre des problèmes complexes.

Des exemples de code pertinents peuvent être trouvés dans le référentiel GitHub de l' auteur .

Démarrer simple - optimisation scalaire univariée

Nous commençons par un exemple de minimisation de fonction scalaire simple (d'une variable). Supposons que nous voulions minimiser la fonction suivante, qui est tracée entre x = -10 et x = 10. La fonction ressemble à ce qui suit. Dans le domaine des fonctions, il a un minimum global et un minimum local.



Le code définissant la fonction est,

```
def scalar1 (x):
    retourne np.sin (x) * np.exp (-0,1 * (x-0,6) ** 2)
```

Le code pour déterminer le minimum global est extrêmement simple avec SciPy. Nous pouvons utiliser la **minimize_scalar**fonction dans ce cas.

```
à partir de scipy import optimisez le
résultat = Optimize.minimize_scalar (scalar1)
```

C'est tout. Croyez-le ou non, l'optimisation est terminée! Nous pouvons imprimer l'objet résultant pour obtenir des informations plus utiles.

```
imprimer (résultat)>> fun: -0,6743051024666711
   nfev: 15
   nit: 10
   succès: vrai
   x: -1,2214484245210282
```

La valeur à laquelle le minimum est atteint est stockée dans la `result['x']`variable.

```
print ("Le minimum se produit à:", résultat ['x'])
>> Le minimum se produit à: -1,2214484245210282
```

Les quantités de repos donnent des informations sur le nombre d'évaluation de la fonction, les itérations, l'état de la solution (succès ou non) et la valeur de la fonction à la solution finale.

Et si les variables sont bornées?

Le code ci-dessus a accompli ce que l'on appelle une optimisation sans contrainte / illimitée, c'est-à-dire qu'aucune restriction d'aucune sorte n'a été imposée au problème. Cependant, la plupart des problèmes d'optimisation

pratiques impliquent des contraintes complexes. Un exemple simple de cela est lié à la variable indépendante (x).

Comme on peut voir que cette fonction est caractérisée par deux minima, le résultat serait différent si on ne considérait que les valeurs positives de x . Le code à rechercher avec lié n'est que légèrement différent de celui ci-dessus.

```
result = Optimize.minimize_scalar (scalar1, bounds = (0,10),
method = 'Bounded')
```

Donc, nous devons transmettre l' **bounds**argument avec un tuple approprié contenant les limites minimum et maximum et utiliser l' **method='Bounded'**argument.

```
print ("Lorsqu'il est délimité entre 0 et 10, le minimum se
produit à:", résultat ['x'])>> Lorsqu'il est borné entre 0 et 10,
le minimum se produit à: 4.101466164987216
```

Présentation d'autres contraintes fonctionnelles

Nous aurions pu avoir d'autres contraintes compliquées dans le problème. Supposons que nous souhaitons que les conditions suivantes soient remplies avec l'objectif de trouver le minimum global.

$$\text{minimize: } sin(x).exp\left[-0.1\,(x-0.6)^2\right]$$

s.t.

$$0.5 < log\left(x^2+2\right) < 1.5$$

$$sin(x) + 0.3x^2 = 1$$

Notez que l'un d'eux est l' **inégalité** et l'autre est la contrainte d' **égalité** .

Mettre des contraintes comme des fonctions dans un dictionnaire

SciPy permet de gérer des contraintes arbitraires grâce à la méthode plus généralisée `optimize.minimize`. Les contraintes doivent être écrites dans un dictionnaire Python en suivant une syntaxe particulière. La contrainte d'inégalité doit être décomposée en inégalités individuelles sous la forme $f(x) < 0$. Le code suivant illustre l'idée.

Choisir une méthode adaptée

Après cela, nous pouvons exécuter l'optimisation en choisissant une méthode appropriée qui prend en charge les contraintes (**toutes les méthodes de la** `minimize`**fonction ne prennent pas en charge les contraintes et les limites**). Ici, nous avons choisi la méthode SLSQP qui représente **la programmation quadratique séquentielle des moindres carrés** .

Première estimation et premier essai

De plus, pour l'utiliser, `minimize`nous devons transmettre une **première estimation** sous forme d' `x0`argument. Supposons que nous passions `x0=0`pour un essai.

```
résultat = optimiser.minimize (scalaire1, x0 = 0, méthode =
'SLSQP',
contraintes = cons, options = {'maxiter': 1000})
```

Échec!

Si nous imprimons le résultat, nous voyons quelque chose de différent du simple résultat d'optimisation sans contrainte.

```
fun: 0.7631695862891654
jac: array ([0.59193639])
message: 'Limite d'itération dépassée'
nfev: 1254
nit: 101
njev: 101
status: 9
success: False
x: array ([0.8773752])
```

Le paramètre d'optimisation **success: False** indique qu'il n'a pas réussi à atteindre le minimum global. Le message est **«Limite d'itération dépassée** », c'est-à-dire qu'il a essayé 101 itérations mais n'a pas pu atteindre le minimum.

Mais pourquoi?

La réponse réside dans la théorie profonde de l'optimisation mathématique (et de l'algorithme associé) mais il suffit de dire que la supposition initiale a joué un grand rôle. En général, un problème d'optimisation non convexe n'a aucune garantie mathématique d'être résolu avec succès et la nature de **notre problème ici est non convexe** . Pour en savoir plus sur la convexité d'un problème d'optimisation, regardez cette vidéo,

Comment pouvons-nous améliorer l'optimisation (recherche)?

Dans les cas généraux, nous ne pouvons pas faire grand-chose. Cependant, dans cet exemple de jouet, nous avons déjà le tracé de la fonction et pouvons observer la solution optimale. Par conséquent, nous pouvons simplement donner une meilleure estimation initiale à l'algorithme. Nous donnons **x0=-2.**

```
résultat = optimiser.minimiser (scalaire1, x0 = -2, méthode =
'SLSQP',
contraintes = cons, options = {'maxiter': 100})
```

Maintenant, le résultat est favorable!

```
fun: -0.2859494456768613
jac: array ([- 0.4675066])
message: "L'optimisation s'est terminée avec succès."
nfev: 811
nit: 71
njev: 67
status: 0
success: True
x: array ([- 2.37569791])
```

Qu'en est-il du nombre d'itérations?

Et si nous restreignions le nombre d'itérations effectuées par l'algorithme? À des fins de démonstration uniquement, nous limitons sévèrement le nombre d'itérations à 3.

```
résultat = optimiser.minimize (scalaire1, x0 = -20, méthode =
'SLSQP',
contraintes = cons, options = {'maxiter': 3})
```

Le résultat n'est, comme prévu, pas favorable.

```
fun: -0,4155114388552631
jac: array ([- 0.46860977])
message: 'Limite d'itération dépassée'
nfev: 12
nit: 4
njev: 4
status: 9
```

```
success: False
x: array ([ -2.10190632 ])
```

Notez que l'optimisation s'est approchée du minimum global, mais ne l'a pas tout à fait atteint - bien sûr, du fait qu'il n'était pas autorisé à itérer un nombre suffisant de fois.

Pourquoi est-ce important d'en tenir compte?

Cela est dû au fait que chaque itération équivaut à **un coût** de **calcul (et parfois pas de calcul mais physique réel)** .

Il s'agit d'un aspect commercial du processus d'optimisation. Dans la vie réelle, il se peut que nous ne puissions pas exécuter l'optimisation pendant une longue période si l'évaluation de la fonction individuelle coûte des ressources importantes.

Ce type de scénario survient lorsque l'optimisation n'est pas effectuée avec une simple évaluation mathématique mais une simulation complexe et chronophage ou une expérimentation coûteuse et laborieuse.

Lorsque chaque évaluation coûte de l'argent ou des ressources, non seulement le choix de l'algorithme mais aussi les détails les plus fins deviennent importants à considérer.

Aller plus complexe - fonction multi-variée

Bien que nous ayons examiné tous les aspects essentiels de la résolution d'un problème d'optimisation standard dans les sections précédentes, l'exemple consistait en une simple fonction analytique à variable unique.

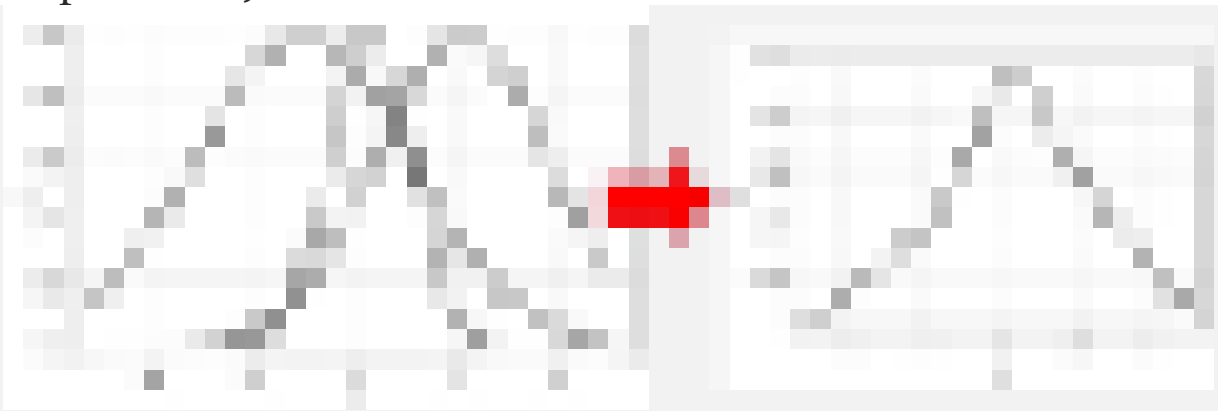
Mais cela ne doit pas être le cas!

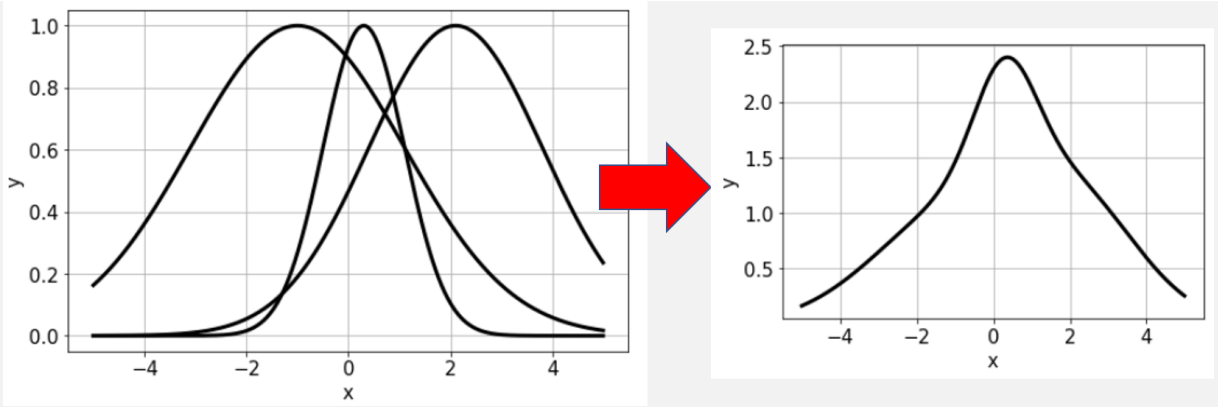
Les méthodes SciPy fonctionnent avec n'importe quelle fonction Python - pas nécessairement une fonction mathématique unidimensionnelle de forme fermée.

Montrons un exemple avec une fonction multi-valeurs.

Maximisation d'un mélange gaussien

Souvent, dans un processus chimique ou de fabrication, plusieurs sous-processus stochastiques sont combinés pour donner naissance à un **mélange gaussien** . Il peut être souhaitable de maximiser la sortie de processus résultante finale en **choisissant les points de fonctionnement optimaux dans les sous-processus individuels** (dans certaines limites de processus).





L'astuce consiste à **utiliser un vecteur comme entrée de la fonction objectif** et à s'assurer que la fonction objectif renvoie toujours une seule valeur scalaire. De plus, comme le problème d'optimisation concerne ici la maximisation de la fonction objectif, nous devons **changer le signe et renvoyer le négatif** de la somme des fonctions gaussiennes comme résultat de la fonction objectif.

Le même `result['x']`stocke le réglage optimal des processus individuels en tant que vecteur. C'est la seule différence entre l'optimisation d'une fonction à valeur unique et d'une fonction multivariée est que nous récupérons un vecteur au lieu d'un scalaire.

```
x: tableau ([- 1.00017852, 0.29992313, 2.10102748])
```

Entrées limitées

Inutile de dire que **nous pouvons changer les limites ici pour refléter les contraintes pratiques** . Par exemple, si les paramètres du sous-processus ne peuvent occuper qu'une certaine plage de valeurs (certaines doivent être positives, d'autres doivent être négatives, etc.), la solution sera légèrement différente - il se peut que ce ne soit pas l'optimum global.

Ici, la solution est la suivante. Cela oblige à pousser le troisième paramètre de sous-processus à la valeur maximale possible (zéro) tout en ajustant les deux autres de manière appropriée.

```
x: tableau ([- 1.00000644e + 00, 3.00115191e-01, -8.03574200e-17])
```

Les contraintes pour l'optimisation à plusieurs variables sont traitées de la même manière que pour le cas à une seule variable.

Documentation et exemples plus détaillés

SLSQP n'est pas le seul algorithme de l'écosystème SciPy capable de gérer des tâches d'optimisation complexes. Pour une documentation plus détaillée et leur utilisation, consultez les liens suivants,

- Optimisation et recherche de racine (scipy.optimize)
- Optimisation SciPy (TutorialsPoint)
- Routines d'optimisation pratiques

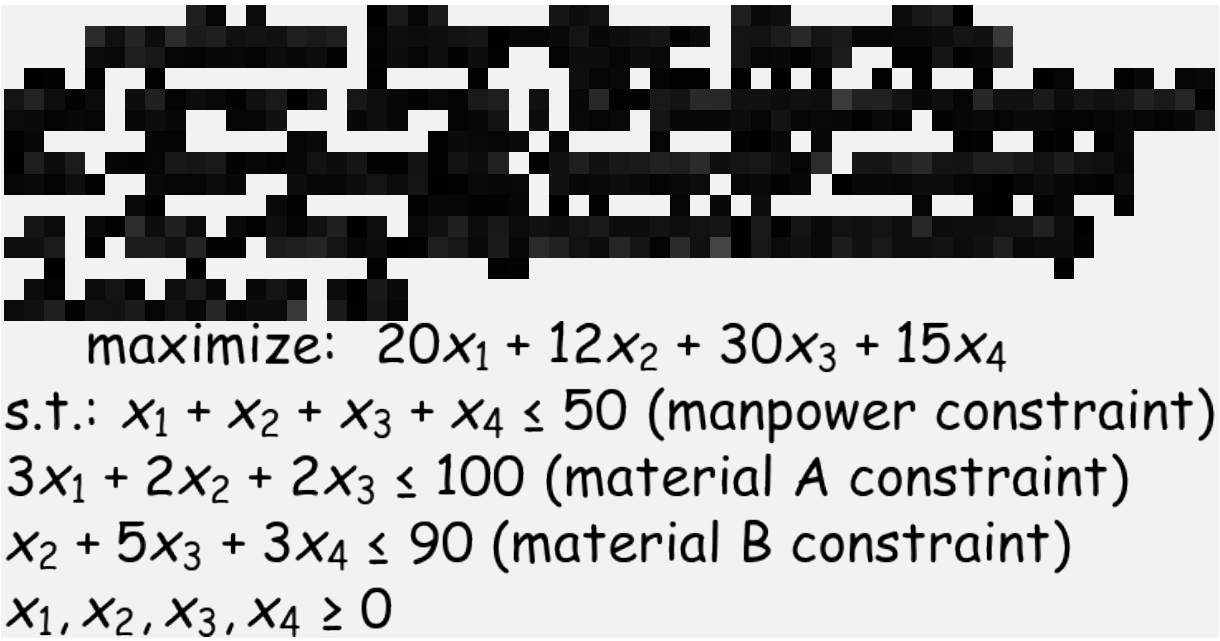
Programmation linéaire avec Scipy

Des problèmes de programmation linéaire (LP) simples et directs peuvent également être résolus par Scipy. Avant 2014, il n'avait pas de solveur LP intégré, mais il a changé depuis.

Prenons un problème pratique de production en usine (emprunté à cet exemple et légèrement modifié)

Une usine fabrique quatre produits différents et que la quantité produite quotidiennement du premier produit est *x1* , la quantité produite du deuxième produit est *x2* , et ainsi de suite. L'objectif est de déterminer la quantité de production journalière maximisant le profit pour chaque produit, avec les contraintes suivantes,

- Le bénéfice par unité de produit est de 20, 12, 30 et 15 pour le premier, deuxième, troisième et quatrième produit, respectivement.
- En raison de contraintes de main-d'œuvre, le nombre total d'unités produites par jour ne peut excéder cinquante (50).
- Pour chaque unité du premier produit, trois unités de la matière première A sont consommées. Chaque unité du deuxième produit nécessite deux unités de matière première A et une unité de matière première B.Chaque unité du troisième produit a besoin de deux unités de A et de cinq unités de B.Enfin, chaque unité du quatrième produit nécessite trois unités de B.
- En raison des contraintes de transport et de stockage, l'usine peut consommer jusqu'à cent unités de matière première A et quatre-vingt dix unités de B par jour.



La configuration de ce problème est facile dans Scipy.

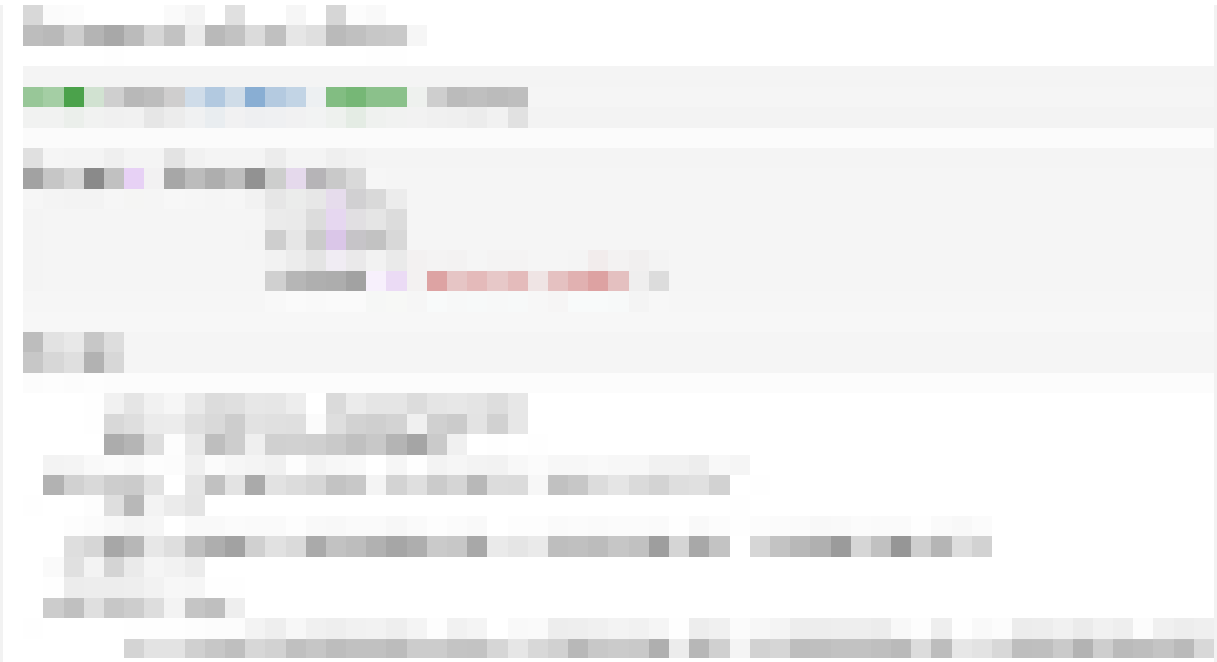
Negative coefficients for the objective function because it is a maximization

```
obj = [-20, -12, -30, -15]
```

Inequality matrices

```
# LHS matrix of inequality equations
lhs = [[1, 1, 1, 1],
 [3, 2, 2, 0],
 [0, 1, 5, 3]]
```

Et puis résolvez-le avec une ligne de code,



Setup and solve in Scipy

```
from scipy.optimize import linprog
```

```
lp_opt = linprog(c=obj,
                 A_ub=lhs,
                 b_ub=rhs,
                 method = 'interior-point')
```

```
lp_opt
```

```
     con: array([], dtype=float64)
     fun: -1033.3333113034805
 message: 'Optimization terminated successfully.'
     nit: 4
   slack: array([1.06529068e-06, 2.14344466e-06, 1.86209118e-06])
  status: 0
 success: True
       x: array([2.66666661e+01, 1.84050438e-08, 9.99999980e+00, 1.33333330e+01])
```

Donc, la solution dit que,

- L'usine devrait produire 26,66 unités de *x1* , 10 unités de *x3* et 13,33 unités de *x4* chaque jour. Le nombre extrêmement petit correspondant à *x2* indique essentiellement qu'aucune quantité de *x2* ne doit être produite.

- Le profit maximal pouvant être obtenu est de 1033,33 $ en vertu de cet arrangement.

Il convient de noter que la solution indique un choix fractionnaire, ce qui peut ne pas être faisable dans une situation pratique. C'est la limitation du solveur Scipy qu'il ne peut pas résoudre les problèmes de programmation dits entiers. D'autres packages Python comme PuLP pourraient être une option pour de tels problèmes. Voir mon article ici.

Programmation linéaire et optimisation discrète avec Python en utilisant PuLP
La programmation linéaire et entière sont des techniques clés pour l'optimisation discrète et elles apparaissent partout dans la modernité…
versdatascience.com

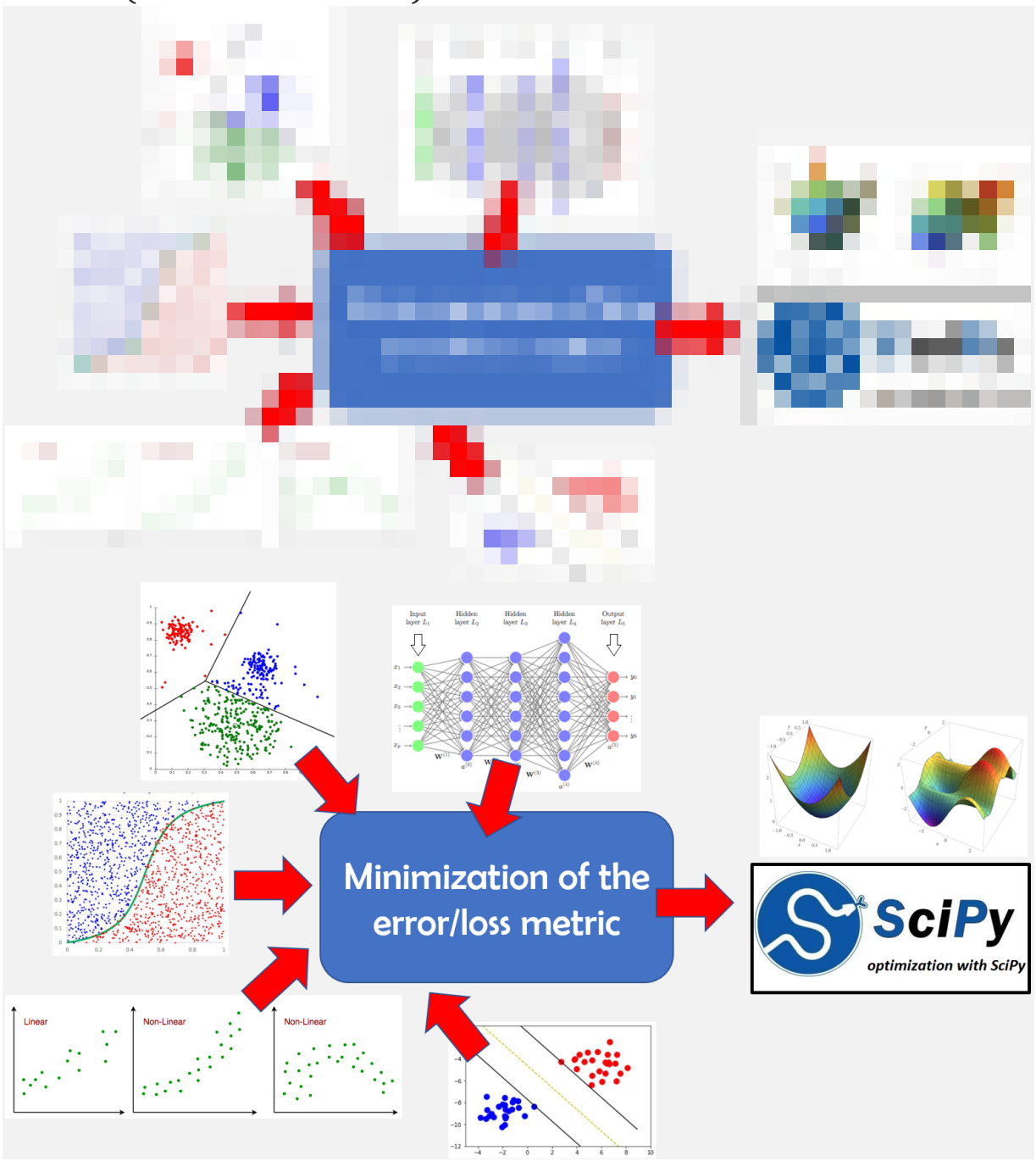
Étendre le processus au domaine de l'apprentissage automatique

Pour être honnête, il n'y a pas de limite au niveau de complexité, vous pouvez pousser cette approche tant que vous pouvez définir une fonction objectif appropriée qui génère une valeur scalaire et des limites et des contraintes appropriées correspondant au scénario de problème réel.

Minimisation des erreurs dans l'apprentissage automatique

Le nœud de presque tous les algorithmes d'apprentissage automatique (ML) est de définir une fonction d'erreur appropriée (ou métrique de perte), d'itérer sur les données et de trouver les paramètres optimaux du paramètre du modèle ML qui minimise l'erreur totale. Souvent, l'erreur est une mesure

d'une sorte de distance entre la prédiction du modèle et la vérité terrain (données données).

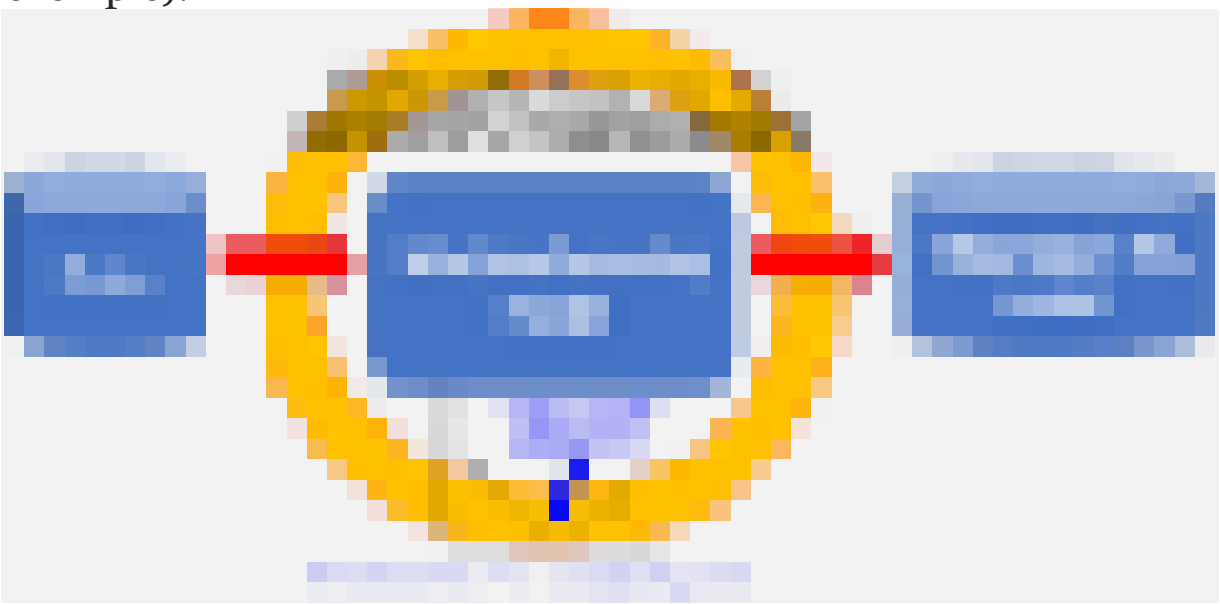


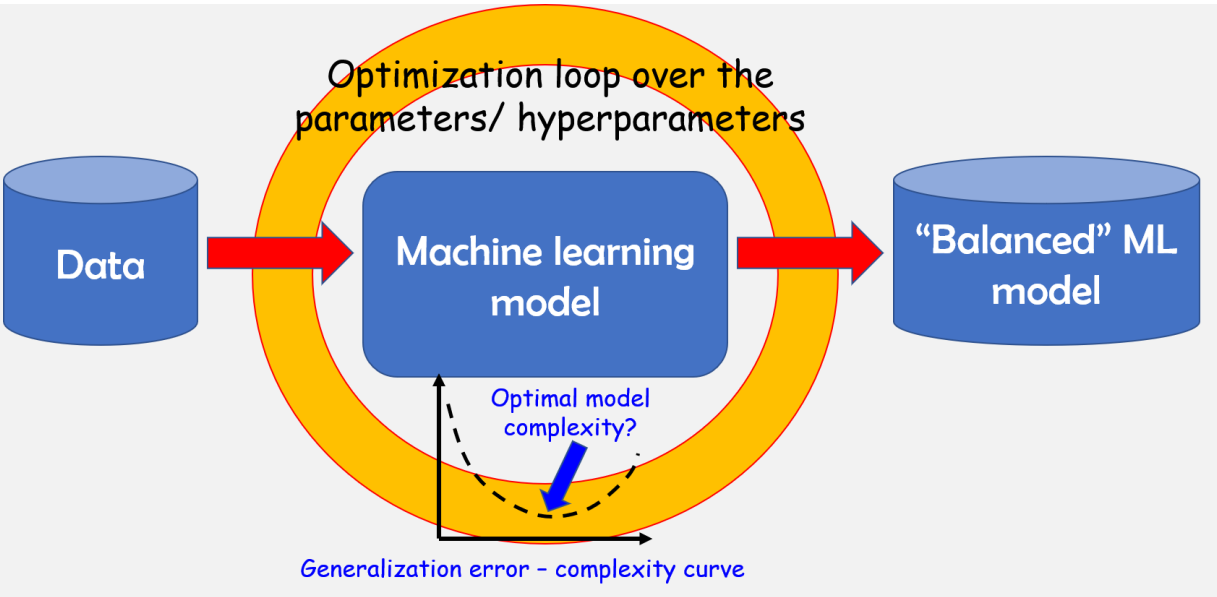
Par conséquent, il est parfaitement possible d'utiliser les routines d'optimisation SciPy pour résoudre un problème de ML.

Cela vous donne un aperçu approfondi du fonctionnement réel de l'algorithme car vous devez construire vous-même la métrique de perte et ne pas dépendre d'une fonction prête à l'emploi et prête à l'emploi.

Optimisation des hyperparamètres en ML

Le réglage des paramètres et des hyperparamètres des modèles ML est souvent une tâche fastidieuse et sujette aux erreurs. Bien qu'il existe des méthodes de recherche de grille disponibles pour rechercher la meilleure combinaison paramétrique, un certain degré d'automatisation peut être facilement introduit en **exécutant une boucle d'optimisation sur l'espace des paramètres** . La fonction objective, dans ce cas, doit être une métrique de la qualité de la prédiction du modèle ML (erreur quadratique moyenne, mesure de complexité ou score F1 par exemple).





Utilisation de l'apprentissage automatique comme évaluateur de fonction

Dans de nombreuses situations, vous ne pouvez pas avoir une fonction analytique de forme fermée agréable à utiliser comme objectif d'un problème d'optimisation.

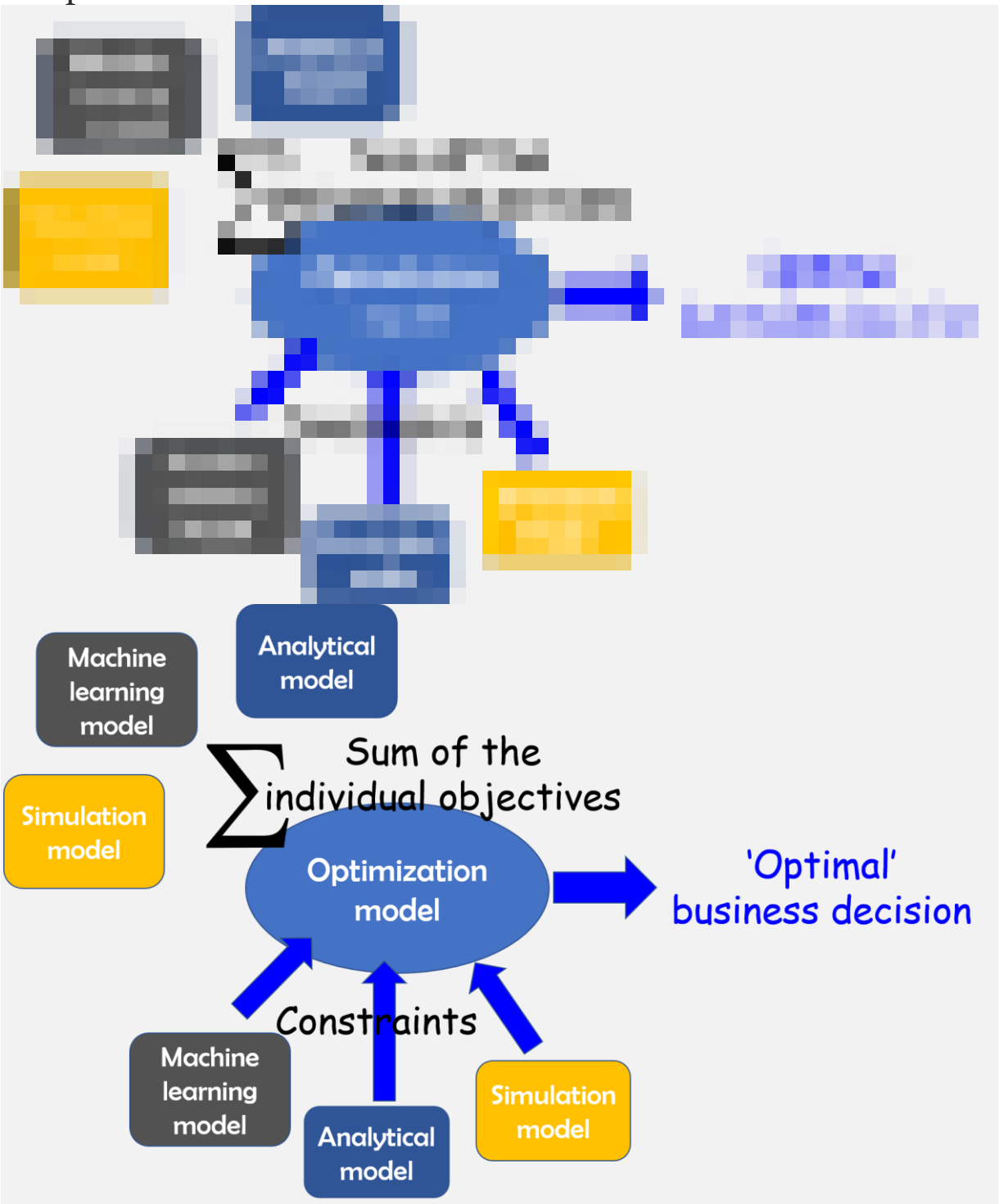
Mais qui se soucie d'être gentil quand nous avons un apprentissage en profondeur?

Imaginez la puissance d' **un modèle d'optimisation qui est alimenté (tant pour sa fonction objective que pour les contraintes) par une multitude de modèles** - de nature fondamentale différente mais standardisés par rapport au format de sortie afin qu'ils puissent agir à l'unisson.

Vous êtes libre de choisir une fonction analytique, un réseau d'apprentissage en profondeur (peut-être comme modèle de

régression), ou même un modèle de simulation complexe, et de les jeter tous ensemble dans le gouffre de l'optimisation.

Les possibilités sont infinies!



Si vous avez des questions ou des idées à partager, veuillez contacter l'auteur à **tirthajyoti [AT] gmail.com** . Vous pouvez également consulter les **référentiels GitHub de** l'auteur pour d'autres extraits de code amusants dans Python, R ou MATLAB et des ressources d'apprentissage automatique. Si vous êtes, comme moi, passionné par l'apprentissage automatique / la science des données, n'hésitez pas à m'ajouter sur LinkedIn ou à me suivre sur Twitter.

Tirthajyoti Sarkar - Ingénieur principal principal - Semiconducteurs, IA, apprentissage automatique - ON…

Georgia Institute of Technology Master of Science - MS, Analytics Ce programme MS donne des cours théoriques et pratiques…

www.linkedin.com