

量子コンピュータ時代のプログラミングセミナー

～ブラックボックス最適化による機械学習の特徴量選択・コスト削減・精度向上～



本日の予定

第一部

- 本セミナーのゴール
- 会社紹介
- Fixstars Amplify の紹介
- 組合せ最適化事例
- ワークショップ事前準備

第二部

- 組合せ最適化の基本
- 数の分割ハンズオン

第三部

- ブラックボックス最適化とは
- FMQAの概要とフロー
- 機械学習の特徴量選択ハンズオン
 - 問題の説明
 - FM
 - FMQA
- まとめ

質問は随時 Zoom の Q&A へお願いします

会社紹介

(株) Fixstars Amplify の紹介

- **組合せ最適化**のための量子コンピューティングクラウドプラットフォーム「Fixstars Amplify」の提供
- 2021年に設立（株式会社フィックスターズからスピンアウト）
 - 代表取締役社長CEO：松田 佳希（博士）
- 親会社（株）フィックスターズ
 - 東証プライム市場上場
 - 約300名の従業員のうち、約9割がソフトウェアエンジニア
 - ソフトウェア高速化プロフェッショナル集団



量子・量子インスパイアード技術

1. 量子コンピュータ

(量子ゲート方式)

- 古典汎用コンピュータの上位互換。量子ゲートを操作。エラー訂正機能の無いNISQ型実機がクラウド利用可能
- QAOAにより**組合せ最適化問題 (QUBO)** を取り扱うことが可能
- 演算規模：～数100ビット

1.
量子
コンピュータ

IBM
Google
Rigetti
IonQ
...

2.
量子
アニーリング

D-Wave

3.
その他の
イジングマシン

Fixstars Amplify
TOSHIBA, Fujitsu
NEC, HITACHI,
...

3. その他のイジングマシン

(量子インスパイアード技術)

- 二次の多変数多項式で表される目的関数の**組合せ最適化問題 (QUBO)** 専用マシン
- 統計物理学におけるイジング模型に由来。様々な実装により実現。
- 演算規模：
260,000+ビット (*Amplify AE*)

2. 量子アニーリング (量子焼きなまし法式のイジングマシン)

- イジングマシンの一種。量子イジング模型を物理的に搭載したプロセッサで実現。
量子効果を物理的に調整し、自然計算により低エネルギー状態が出力
- **組合せ最適化問題 (QUBO)** を扱う専用マシン
- 演算規模：～数1,000ビット

最適化問題の分類

数理最適化問題

- 連続最適化問題
 - 決定変数が連続値（実数など）
- 決定変数が離散値（整数など）
 - 整数計画問題（決定変数が整数）
 - **0-1整数計画問題（決定変数が二値）**

QUBO目的関数（0-1整数二次計画問題）

$$f(\mathbf{q}) = \sum_{i < j} Q_{ij} q_i q_j + \sum_i Q_{ii} q_i$$

f : 目的関数

q : 決定変数

Q : 係数

量子アニーリング・イジングマシン

Quadratic 二次形

Unconstrained 制約条件なし

Binary 0-1整数（二値）

Optimization 計画（最適化）

$f(\mathbf{q})$ を最小化するような $\mathbf{q}$ を求める

クラウドサービス : **Fixstars Amplify**

Fixstars Amplify とは

- いつでも 開発環境 と 実行環境 がセット
すぐにアプリ開発と実行が出来る
- 誰でも ハードウェアや専門的な知識が不要
無料で開発がスタート可能
多くの解説、サンプルコード
- 高速に 26万ビットクラスの大規模問題の
高速処理と高速実行が可能
- あらゆる 一般に公開されている全てのイジング
マシンを利用可能



The screenshot shows the Fixstars Amplify website. At the top, there's a navigation bar with links for Japanese, お問い合わせ, and a menu. Below the navigation bar, a green banner announces the release of Amplify SDK version 1.0. The main heading is '量子コンピューティングプラットフォーム' (Quantum Computing Platform). Below this, a code block shows '\$ pip install amplify'. A blue button says '無料でアクセストークンを入手' (Get access token for free). Below the button, a link says 'ドキュメントを見る →' (View documents →). The page is divided into four sections, each with an icon and a title: 1. 'シンプルで効率的なアプリ開発' (Simple and efficient app development) with a cube icon, describing automation of complex processes. 2. 'PoCから実問題まで' (From PoC to real problems) with a target icon, describing the ability to handle large-scale problems. 3. '様々なマシン・ソルバーに対応' (Support for various machines/solvers) with a server rack icon, describing support for various quantum annealing machines and solvers. 4. 'すぐに開発をスタート' (Start development immediately) with a power button icon, describing the ease of starting development with pre-set environments.

Japanese • お問い合わせ •

デモ&チュートリアル 製品紹介 リソース セミナー お客様事例 会社紹介 スケジュール最適化

NEWS Amplify SDKバージョン1.0をリリースしました →

利用可能なすべての量子アニーリング・イジングマシン、数理最適化ソルバー
ゲート式量子コンピュータに対応した

量子コンピューティング プラットフォーム

```
$ pip install amplify
```

無料でアクセストークンを入手

ドキュメントを見る →

**シンプルで効率的なアプリ開発**

複雑で専門性の高いプロセスを自動化し、効率的にイジングマシンを使うための学習コストを圧倒的に低くします。

**PoCから実問題まで**

大規模問題の入力と高速実行が可能で、PoCや実問題を視野に入れたアプリケーション開発が行えます。

**様々なマシン・ソルバーに対応**

利用可能なすべての量子アニーリング・イジングマシンや数理最適化ソルバー、ゲート式量子コンピュータの組合せ最適化を解くアルゴリズムなど幅広くサポートしています。

**すぐに開発をスタート**

開発環境と実行環境がセットで提供されるため、すぐに始めることができます。

Fixstars Amplify の対応マシンの一例

アニーリングエンジン



量子アニーリング・イジングマシン

Fixstars Amplify AE

標準マシン

GPUの優れた並列計算能力を最大限に活用し、複雑な組合せ最適化問題を高速・高精度に解く革新的なアニーリングエンジンです。



外部ソルバー連携

標準マシン

Fixstars Amplifyからご利用申し込み可能なマシンです。

量子アニーリング・イジングマシン

D-WAVE
The Quantum Computing Company

標準マシン

D-Wave Systems
2000Q / Advantage

量子アニーリング・イジングマシン

TOSHIBA

標準マシン

東芝
デジタルソリューションズ
SQBM+

量子アニーリング・イジングマシン

FUJITSU

標準マシン

富士通
デジタルアニーラ

BYOLマシン

自身の保有するライセンスを用いて Fixstars Amplify を利用出来ます。

量子アニーリング・イジングマシン

HITACHI

日立製作所
CMOSアニーリングマシン

数値最適化ソルバー

GUROBI
OPTIMIZATION

Gurobi
Gurobi Optimizer

ゲート式量子コンピュータ

IBM

IBM
IBM Quantum

量子回路シミュレータ

Qulacs
Qulacs

標準マシン は、

- ベンダ各社と個別マシン利用契約なし、
- 評価・検証用ベーシックプランなら無料、

で利用可能！ ←「いつでも」、「誰でも」

今後も幅広い対応マシンの追加が続々と行われる予定です！ ←「あらゆる」

活用領域とユースケース（PoC・実稼働）

- **生産計画**

- 多品種少量生産、保全計画、設備投資、在庫

- **従業員割り当て**

- 食品、輸送、製造

- **エネマネ**

- エネルギーミックス、装置の運転制御

- **経路**

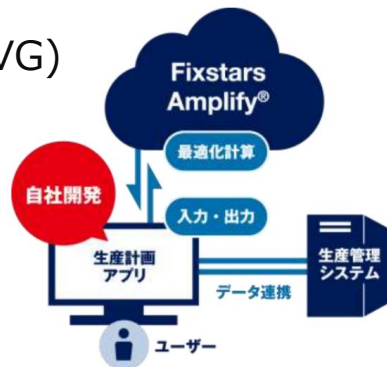
- 配送、船舶、無人搬送車 (AVG)

- **メディア**

- 最適広告配信

- **研究開発、設計**

- 材料設計
- 物理シミュレーション
- **ブラックボックス最適化**



Copyright© Fixstars Group

Amplify インタビュー

検索



活用領域とユースケース（PoC・実稼働）

Amplify インタビュー

検索

- 生産計画

- 多品種少量生産
 - 保全計画
 - 設備投資
 - 在庫

- 従業員管理

- 食品

- エネルギー

- エネルギーミックス、機器の運転制御

- 経路

- 配送

- メディア

- 最適化

- 研究開発、設計

- 材料設計

- 物理シミュレーション

- ブラックボックス最適化

1,000を超える企業、研究所、大学

1億を超える実行回数 (*Amplify AE*)

アニーリングマシンのプログラミング 体験

イジングマシンの実行手順

ハンズオンセミナーのメイントピック

1. 数理モデル検討 解きたい課題の「**目的関数**」「**決定変数**」とその「**制約条件**」を検討する
2. 定式化 「**多項式**」で「**目的関数**」と「**決定変数**」を記述 (変換) する
「**決定変数**」に対する「**制約条件**」を Amplify で表現する
3. モデル変換
(論理・物理) 各マシンの仕様や制限に準拠した形式にモデルを変換する
(例: 二次項に制約がある場合は「**グラフマイナー埋め込み**」問題を解く)
4. 入力データの準備 各マシンのSDKやAPI仕様に合わせてQUBO模型 (物理) をデータ化する
5. マシンの実行 マシンを実行して出力の変数値やエネルギー値(コスト値)を解析する
上記の逆の手順を**逆**に解きたい課題の「**決定変数**」を解釈する

Amplify SDK による
サポート

Amplifyの基本的な使用方法 (1)

- まずはインポート

```
# Install Amplify SDK to Google Colab
! pip install -q amplify

#Import all functions and classes
from amplify import *
```

- 使用するマシンを選択

```
# Fixstars Amplify AE
client = AmplifyAECClient ()

# Timeout 1s
client.parameters.time_limit_ms = 1000 #ms

# API token
client.token = "AE/XXXXXXXXXXXXXXXXXXXXX"
```

その他のクライアントを使用する場合は
ドキュメントを参照

<https://amplify.fixstars.com/ja/docs/amplify/v1/clients.html>

Amplifyの基本的な使用方法 (2)

- 目的関数の定式化 (多項式)
 - バイナリ多項式の構築

```
# 決定変数生成器を作成
g = VariableGenerator()

# 長さ 2 の決定変数配列を作成
q = g.array("Binary", 2)
```

バイナリ変数 (“Binary”) だけでなく、イジング変数 (“Ising”) や、
整数変数 (“Integer”)、実数変数 (“Real”) も指定可能

- 式の構築

```
f = 2 * q[0] * q[1] + q[0] - q[1] + 1
print(f)
# 2 q_0 q_1 + q_0 - q_1 + 1
```

3次以上の高次多項式も可能。マシンが対応していない
場合は Amplify SDK が内部的に次数下げを行う

Amplifyの基本的な使用方法 (3)

- モデルの作成とマシンの実行

```
model = Model(f)
result = solve(model, client)
```

目的関数と制約条件からモデルを作成し
使用するクライアントと共に求解

- 結果の取得

```
print(f"objective = {result.best.objective}")
# objective = 0.0

print(f"q = {q.evaluate(result.best.values)}")
# q = [0. 1.]
```

最良解を `best` で指定
目的関数の値を `objective` にて
変数の値を `values` で得る

Amplify SDK によるプログラミング例

```
from amplify import *

# 決定変数を生成
g = VariableGenerator()
q = g.array("Binary", 2)

#目的関数を構築
f = 2 * q[0] * q[1] + q[0] - q[1] + 1

# Amplify モデルを構築
model = Model(f)

#ソルバーの設定
client = AmplifyAECClient()
client.parameters.time_limit_ms = 1000 #ms
client.token = "AE/XXXXXXXXXXXXXXXXXXXXX"

# 求解の実行
result = solve(model, client)

# 結果の表示
print(f"objective = {result.best.objective}")
print(f"q = {q.evaluate(result.best.values)}")
```

1. 定式化

- 決定変数：スカラーあるいは配列型
- 目的関数：決定変数による数式処理
- 制約条件：制約条件の構築及び管理

2. ソルバークライアントの選択

- ソルバークライアントオブジェクトの構築
- ほぼ全てのパラメータの設定が可能

3. ソルバーを実行

- 論理モデルをハードウェアのスペック等に合わせたモデルに変換
- 適切なモデル変換・定式化手法を選択

4. 解の取得

- マシンの出力解を逆変換し決定変数の形式で出力

Fixstars Amplify ご利用プラン

料金のご紹介

<https://amplify.fixstars.com/ja/pricing>

	ベーシック 評価・検証用の無料プラン	スタンダード 実用レベルの計算環境	プレミアム 高性能な計算環境	Sプレミアム 最高性能の計算環境
	使い始める	見積りを依頼	見積りを依頼	見積りを依頼
利用料金	無料	月額10万円（1名） （税込11万円） 月額20万円（最大5名） （税込22万円）	月額20万円（1名） （税込22万円） 月額60万円（最大5名） （税込66万円）	月額30万円（1名） （税込33万円） 月額90万円（最大5名） （税込99万円）
計算環境	スモール	ミディアム	ラージ	スーパーラージ
D-Waveマシンの無料実行	3分/月	3分/月	3分/月	3分/月
SQBM+オプション	無料	月額30万円（1名） （税込33万円） 月額90万円（最大5名） （税込99万円）	月額30万円（1名） （税込33万円） 月額90万円（最大5名） （税込99万円）	月額30万円（1名） （税込33万円） 月額90万円（最大5名） （税込99万円）
サポート	ベーシック	スタンダード	プレミアム	プレミアム
評価・検証フェーズでの利用				
実運用フェーズでの利用				

開発支援サービス(個別見積り)

コンサル・システム開発等
数百万円～数千万円



月額利用料
百万円～

定式化や実装を
手厚く
支援します！

セミナー・トレーニングのご紹介

<https://amplify.fixstars.com/ja/news/seminar>

お客様の実際の課題解決をご支援するために、**無料セミナー**や**有償トレーニング**を提供しています。

無料セミナー・ワークショップ

ビジネス向け、エンジニア向けに分けて開催しています！

ビジネス向け

製造業向け量子コンピュータ時代のDXセミナー 見える化、予測・分析、その先の最適化へ

組合せ最適化問題や量子アニーリング・イジングマシンの概要をご紹介したのち、製造業における組合せ最適化を活用したDX推進の一例として、生産計画最適化や生産ラインのシフト最適化などの事例とデモをご紹介します。「Fixstars Amplify」を通じて量子アニーリング・イジングマシンを活用することで、どのようなビジネス上の効果が期待できるのかを感じていただきたいと思います。

エンジニア向け

製造業向け量子コンピュータ時代のDXセミナー 最適化の中身を覗いてみよう

製造業における組合せ最適化を活用したDX推進の一例として、生産計画最適化、勤務シフト最適化などの事例を用いて、問題設定の考え方、目的関数や制約条件の定式化、実装のポイントなどを実際のコードを見ながら解説します。また、サンプルコードを用いて、ご自身の環境で実際に量子アニーリング・イジングマシンを動かす体験をしていただけます。

企業向けプライベートトレーニング

お客様が抱える実際の課題やデータを使った**カスタムメイド**のトレーニングです！

全4回のレクチャーとお客様に実施いただく「課題」を含む約1.5か月のコースです。コースの前半では、量子アニーリング・イジングマシン専用の開発／実行環境であるFixstars Amplifyを用いてPython言語による組合せ最適化アプリケーション開発方法を学びます。後半では、お客様が抱える実際の課題やデータを使ったトレーニングを実施します。量子アニーリング・イジングマシンを使って実課題の解決に取り組んでみたい方に最適なコースです。

第1回
3時間

…
1週間

第2回
3時間

課題
2週間

第3回
1.5時間

…
2週間

第4回
1.5時間

ワークショップ

事前準備（事前メールの内容）

ワークショップの事前準備 (1)

- 【事前メールに記載】ご自身のPC (ブラウザ上) で Python プログラミングを行います。Google Colaboratory を使うので、事前にログイン出来ることを確認をお願いします (要 Google アカウント)

Google Colab 検索
<https://colab.research.google.com/>

- 【事前メールに記載】 Fixstars Amplify ホームページより
ユーザ登録の上、無料トークンの取得をお願いします
(1分で終わります)

Fixstars Amplify 検索
<https://amplify.fixstars.com/>



質問は随時 Zoom の Q&A へお願いします



ワークショップの事前準備 (2)

【事前メールに記載】

- 取得されたトークンを用いて、トークンチェック用サンプルコードが動くか確認をお願いします。

<https://colab.research.google.com/drive/1-Jh2nlhWO970O96WgBwCSOQmP8BVa6-T> (※URLはZoomのチャット欄を参照)

- サンプルコードは閲覧のみ可能な状態です。「ファイル」→「ドライブにコピーを保存」の上、ご自身のトークンを入力してください。その後、Shift + Enterで実行下さい。

```
! pip install amplify
```

```
token = "AE/*****" # ご自身のトークンを入力
```

- ご自身のトークン番号は、Amplifyウェブページ → よりご確認ください。
- 実行後、以下の結果が出力されればOKです。

```
result: [q_0, q_1] = [1. 1.] (f = 0.0)
```



「数の分割問題」のハンズオン

通常の組合せ最適化
(ブラックボックス最適化への導入)



数の分割問題

- 与えられた n 個の整数 $a_0, \dots, a_{n-1}$ を二つの集合に分ける。集合内の数の和が、もう一方の集合内の数の和と等しくなるようにできるか？
- NP完全問題**：とても難しい問題として知られている → 全通り試すしか方法は無い



数の分割問題（具体例と解法の方針）

具体例

$\{2, 10, 3, 8, 5, 7, 9, 5, 3, 2\}$ の10個の数の完璧な分割は見つけられるか？

答え

- 存在する
 - $\{2, 3, 5, 7, 10\}$ と $\{2, 3, 5, 8, 9\}$
 - どちらも和は 27
- 分割方法は 23 通り存在する（対称を除く）

どうやって解くか？

- ひとつの『数』がどちらの集合に分割されるか全通り試す $\rightarrow 2^{10} = 1024$ 通り
- 効率のよい厳密な方法は知られていない・・・（もし発見されたら大騒ぎ）

数の分割問題（定式化）

最適化問題：数の分割において最も惜しい組合せは何か？

- 目的関数

{集合1の和} - {集合2の和} の絶対値を最小化

- 決定変数

数 a_i がどちらの集合に属するかをイジング変数 s_i で表す

- $a_i = \{ 2, 10, 3, 8, 5, 7, 9, 5, 3, 2 \}$

- $s_i = \{-1, 1, -1, 1, -1, -1, 1, 1, 1, 1\}$ (例)

数理モデル

- 目的関数

$$f = \left| \sum_{i=0}^{N-1} s_i a_i \right| \quad (s_i \in \{-1, +1\})$$

$\sum s_i a_i$ は、自然と
{『1』の集合の和} - {『-1』の集合の和}
となる！

数の分割問題（バイナリへの式変形）

- 0-1整数二次計画問題への変換
 - Quadratic Unconstrained Binary Optimization (QUBO) 式

$$f = \left| \sum_{i=0}^{N-1} s_i a_i \right| \quad (s_i \in \{-1, +1\})$$

$\sum s_i a_i$ は、自然と
『1』の集合の和 - 『-1』の集合の和
となる！

$$\rightarrow \left(\sum_{i=0}^{N-1} s_i a_i \right)^2 \quad (s_i \in \{-1, +1\})$$

絶対値を二次式で表す

$$\rightarrow \left(\sum_{i=0}^{N-1} (2q_i - 1) a_i \right)^2 \quad (q_i \in \{0, +1\})$$

± 1 をバイナリ変数で表す

数の分割問題（定式化の具体例）

問題

- $a_i = \{2, 10, 3, 8, 5, 7, 9, 5, 3, 2\}$ の10個の数の完璧な分割は見つけられるか？

決定変数

- $q_i = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9\} (q_i \in \{0, 1\})$ で集合0又は集合1、どちらに所属するかを表す

目的関数

$$f = \left(\sum_{i=1}^N (2q_i - 1)a_i \right)^2$$

目的関数を展開

$$f = \left(2(2q_0 - 1) + 10(2q_1 - 1) + 3(2q_2 - 1) + 8(2q_3 - 1) + 5(2q_4 - 1) \right. \\ \left. + 7(2q_5 - 1) + 9(2q_6 - 1) + 5(2q_7 - 1) + 3(2q_8 - 1) + 2(2q_9 - 1) \right)^2$$

数の分割問題（プログラムコード）

- 問題の定義と決定変数生成器による決定変数の生成

```
a = [2, 10, 3, 8, 5, 7, 9, 5, 3, 2]
q = amplify.VariableGenerator().array("Binary", len(a))
```

- 目的関数、 $f = (\sum_{i=1}^N (2q_i - 1)a_i)^2$ 、の定式化（①②③は同等）

1. 定式化

① `f = ((2 * q - 1) * a).sum() ** 2`

② `f = 0`
`for i in range(len(a)):`
`f += (2 * q[i] - 1) * a[i]`
`f **= 2`

③ `f = amplify.sum((2 * q - 1) * a) ** 2`

色々な書き方が出来る

2. 実行

```
result = amplify.solve(f, client)
```

得られた目的関数の値0

各集合の合計値27

3. 結果

```
q = [1, 1, 1, 0, 1, 1, 0, 0, 0, 0], f = 0.0, w = 27
```

各数字に対して、集合0か、集合1か

オンラインデモ & チュートリアル

Amplify デモ

検索

<https://amplify.fixstars.com/ja/demo>



デモアプリケーション

ビクロスパズルの求解

プログラミング難易度 ★★★★★
複雑な定式化の例として、数字で与えられるヒントを元に数字を塗り、絵を完成させるビクスゲーム、ビクロスを解くアプリを開発します。

デモアプリ

サンプルコード



チュートリアル応用編

ブラックボックス最適化 (1)

プログラミング難易度 ★★★★★
複雑で未知な目的関数にも適用可能な、機械学習と組み合わせ最適化を組み合わせたブラックボックス最適化手法を紹介し、Amplifyを用いて実装します。

サンプルコード



チュートリアル応用編

ブラックボックス最適化 (2)

プログラミング難易度 ★★★★★
機械学習と量子アニーリング・インジマンソンを活用するブラックボックス最適化の適用例として、緑色の高温超伝導を実現する材料探索を取り扱います。

サンプルコード



チュートリアル応用編

ブラックボックス最適化 (3)

プログラミング難易度 ★★★★★
化学プラントにおける生産量を最大化するための運転条件最適化を行います。最適化には、機械学習モデルに基づくブラックボックス最適化と流体シミュレーションを用い、真の最適化を最大化するように異質の探索を行います。

サンプルコード



チュートリアル応用編

ブラックボックス最適化 (4)

プログラミング難易度 ★★★★★
流体機器設計に不可欠な異質の最適化問題を取り上げます。最適化には、組み合わせ最適化及び機械学習に基づくブラックボックス最適化と流体シミュレーションを用い、真の最適化を最大化するように異質の探索を行います。

サンプルコード



デモアプリケーション

容量制約つき運搬経路問題 (CVRP)

プログラミング難易度 ★★★★★
運送業における効率的な配送計画の策定やごみ収集や道路清掃における巡回順序の最適化等での応用が期待される容量制約つき運搬経路問題 (CVRP) を取り扱います。

デモアプリ

サンプルコード



チュートリアル応用編

ブラックボックス最適化 (5)

プログラミング難易度 ★★★★★
ブラックボックス最適化により、商業施設における交通量が激増する都市における交通渋滞を低減するような信号機の最適化を実現します。また、その様な信号機制御を実施した際の都市の交通量をシミュレーションします。

サンプルコード



チュートリアル応用編

定式化による交通信号機の最適化

プログラミング難易度 ★★★★★
都市における交通量を最小化するために、第一歩として交通状況に応じ、組合せ最適化を用いてリアルタイムに信号機の最適制御を実施します。また、その様な信号機制御を実施した際の都市の交通量をシミュレーションします。

サンプルコード



チュートリアル応用編

10. 整数長ジョブスケジューリング問題

プログラミング難易度 ★★★★★
あらかじめ決まった数のジョブとマシンがあり、それぞれのジョブにかかる時間が分かっているものとします。それぞれのジョブをいくつかのマシンに割り当てます。ジョブスケジューリング問題では、最も早く全ジョブが完了するような割り当て方を求めます。

サンプルコード



デモアプリケーション

画像のノイズ除去

プログラミング難易度 ★★★★★
画像のノイズ除去を行うアプリケーションを開発します。

サンプルコード



チュートリアル応用編

会議室割当問題

プログラミング難易度 ★★★★★
制約条件を用いて定式化するアプリケーションの例として会議室割当問題のアプリケーションを開発します。

サンプルコード



チュートリアル応用編

タクシーマッチング問題

プログラミング難易度 ★★★★★
目的関数と制約条件を用いて定式化するアプリケーションの例としてタクシーマッチング問題のアプリケーションを開発します。

サンプルコード



デモアプリケーション

グラフ彩色問題

Fixstars Amplifyによる、グラフ彩色問題の定式化を体験します。

デモアプリ

サンプルコード



デモアプリケーション

巡回セールスマン問題

Fixstars Amplifyによる、巡回セールスマン問題の定式化を体験します。

デモアプリ

サンプルコード



デモアプリケーション

数独

Fixstars Amplifyによる、数独の定式化を体験します。

デモアプリ

サンプルコード



デモアプリケーション

ライドシェア

集合型ライドシェアの最適化アプリケーションを体験します。

デモアプリ

サンプルコード



デモアプリケーション

タスク割当問題

店舗とタスクに従業員を割り当てる組合せ最適化問題のアプリケーションを体験します。

デモアプリ

サンプルコード



デモアプリケーション

ポートフォリオ最適化

リスクとリターンを考慮した株式ポートフォリオの最適化アプリケーションを体験します。

デモアプリ

サンプルコード

組合せ最適化と ブラックボックス最適化

組合せ最適化とブラックボックス最適化

通常の場合最適化

- 課題に対し、2次定式化 (QUBO) を実施

- 数の分割

$$f = [\Sigma(2q_i - 1)a_i]^2$$

- 生産計画最適化

$$f = \Sigma \Sigma (\Sigma a_i p_i q_{m,i,t} + \Sigma b_i s_i q_{m,i,t} q_{m,i-1,f}) \dots$$

- TSP問題 (お遍路巡り最短経路探索)

$$f = \Sigma \Sigma \Sigma d_{i,j} q_{n,i} q_{n+1,j} \dots$$

- QUBO式に対し、直接アニーリングを行う
(量子アニーリング・イジングマシン)

ブラックボックス最適化

- 直接の定式化が困難な問題

- 例：低損失な流体デバイス形状の同定 (シミュレーション・実験)

- 例：空間計測誤差を最小とする複数の計測装置の最適配置

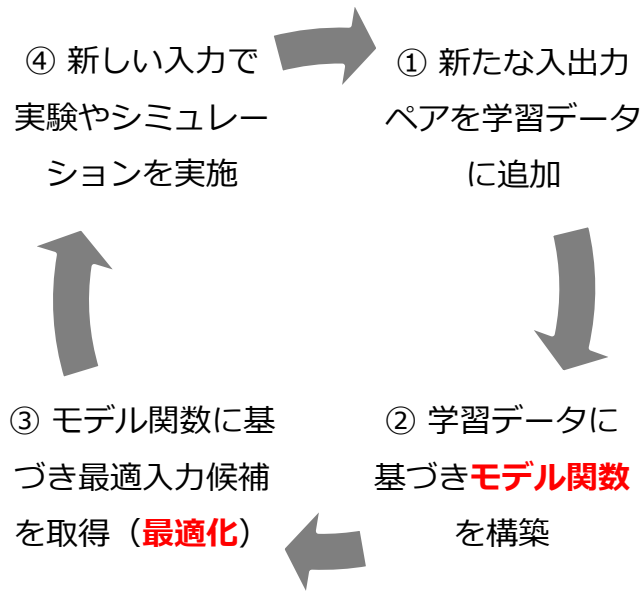
- 例：ターゲットのシミュレーション結果に近い出力を実現するシミュレーション条件

- 最適化の実施

- 繰り返しの実験シミュレーションによる試行錯誤

ブラックボックス最適化のフロー

・ 逐次最適化：最適化サイクルの実施



東大津田先生率いる研究グループ

FMQA

Kitai, et al., *Phys. Rev. Res.* (2020)

・ **モデル関数** → FM

・ **最適化** → QA

*Quadratic-optimization **Annealing**

FMQAの特徴：



- 高次元の最適化問題に強い！
⇔ 次元の呪い
- 制約条件にも強い！

モデル関数としての Factorization Machine (FM)

- モデル関数 $g(x)$ に機械学習モデルの一種である Factorization Machine (FM) を用いると、次のように変数 x に対する2次式での記述ができる。

$$\begin{aligned} g(x|\mathbf{w}, \mathbf{v}) &= w_0 + \langle \mathbf{w}, \mathbf{x} \rangle + \sum_{i=1}^n \sum_{j=i+1}^n \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j \\ &= w_0 + \sum_{i=1}^n w_i x_i + \frac{1}{2} \sum_{f=1}^k \left(\left(\sum_{i=1}^n v_{if} x_i \right)^2 - \sum_{i=1}^n v_{if}^2 x_i^2 \right) \end{aligned} \quad \rightarrow \text{QUBO式}$$

- k はハイパーパラメータ、 $\mathbf{w}$ 及び $\mathbf{v}$ は FM 学習後に取得される FM パラメータ。
- FM パラメータ数は k に依存。 $k = n$ のときは QUBO の相互作用項と同じ自由度がある一方、 k を小さくすることでパラメータ数を減らし過学習を抑制する効果

- このようなブラックボックス最適化手法を **FMQA** と呼ぶ場合がある。

[K. Kitai, et al., Phys. Rev. Res. [\(2020\)](#).
T. Inoue, et al., Opt. Express [\(2022\)](#).]



ブラックボックス最適化 活用例

材料分野に限らず、幅広い分野へ適用可能

FMQA: 活用例 (Amplify サンプルプログラム)

Amplify デモ

検索



チュートリアル応用編

ブラックボックス最適化 (2)

プログラミング難易度 ★★★★★

機械学習と量子アニーリング・イジングマシンを活用するブラックボックス最適化の適用例として、疑似的な高温超電導を実現する材料探索を取り扱います。

サンプルコード

材料最適化

FMQA

×

物理モデル



チュートリアル応用編

ブラックボックス最適化 (3)

プログラミング難易度 ★★★★★

化学プラントにおける生産量を最大化するための運転条件最適化を行います。最適化には、機械学習モデルに基づくブラックボックス最適化と化学反応に関する物理シミュレーションを用います。

サンプルコード

化学プラント 運転条件最適化

FMQA

×

化学シミュレーション



チュートリアル応用編

ブラックボックス最適化 (4)

プログラミング難易度 ★★★★★

流体機器設計に不可欠な翼型の最適化問題を取り上げます。最適化には、組み合わせ最適化及び機械学習に基づくブラックボックス最適化と流体シミュレーションを用い、翼の揚抗比を最大化するように翼型の探索を行います。

サンプルコード

翼形状最適化

FMQA

×

流体シミュレーション



チュートリアル応用編

ブラックボックス最適化 (5)

プログラミング難易度 ★★★★★

ブラックボックス最適化により、商業施設による交通集中が発生し得る都市における、交通渋滞を低減するような信号機群の最適制御を実施します。最適化の実施及び実証には、マルチ・エージェント・シミュレーションによる交通シミュレーションを用います。

サンプルコード

信号制御最適化

FMQA

×

マルチ・エージェント
・シミュレーション



チュートリアル応用編

ブラックボックス最適化 (6)

プログラミング難易度 ★★★★★

ブラックボックス最適化により、攪拌性能に影響を与える設計パラメータに対して、混合効率が最大化されるような攪拌機の最適設計を実施します。最適化の実施および評価には、濃度分布に基づく簡易的な攪拌シミュレーションを用います。

サンプルコード

機器設計最適化

FMQA

×

攪拌シミュレーション

FMQA: 活用例 (Amplify ユーザー)

• 活用領域

- 化学、創薬、食品、自動車、電機、通信、重工、エネルギー、ヘルスケア...

非線形現象の逆
問題

機械学習：
コスト↓精度↑

設計開発におけ
る部品選定

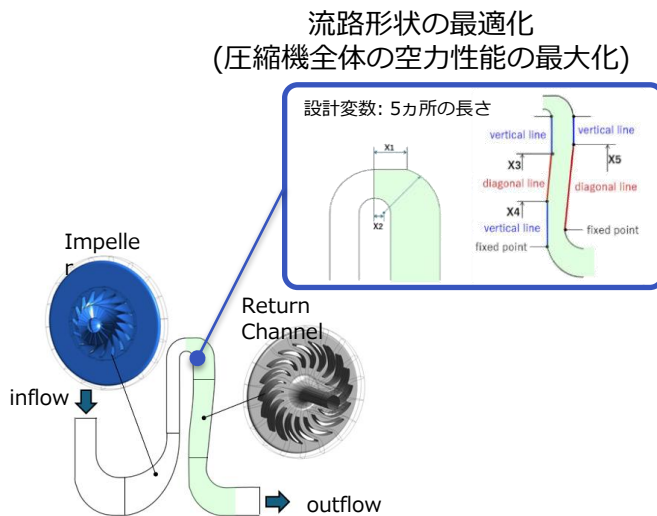
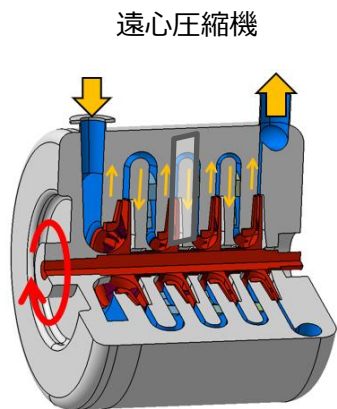
材料配合最適化

多目的最適化

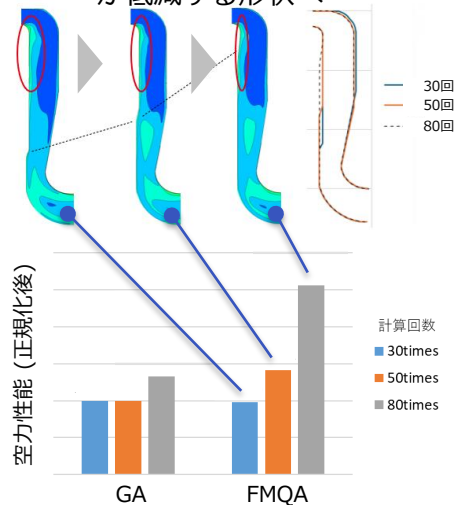
物理モデルの
簡略化

事例：ターボ機械の形状最適化（川崎重工業様）

- 従来より遺伝的アルゴリズム (GA) により形状最適化を行うことが多かった。規模が大きくなると求解までに時間がかかり、開発期間が長期化するといった課題があった
- FMQA により、従来手法と比べ、**同じ計算回数でもより優れた解が得られることを確認。**



最適化が進むごとに損失発生領域
が低減する形状へ



事例：車両設計最適化（マツダ様）

Amplify マツダ

検索

- 車体設計の複数車種同時最適化問題。実数変数200以上の大規模問題。
- **車体の軽量化と共通部品数の最大化の実現（衝突性能・製造制約などの制約を満たした上で）**
- 国内外研究グループ^{*1,2}により様々な手法が試されてきており、**1～3万回程度の試行**により、ある程度良い解が見つけられることは確認されていた

- ▼
- FMQA により、**1,000回（3%）程度の試行**で、従来手法と同等以上の解を見つけることに成功！

*1 [進化計算コンペティション2017開催報告](#)

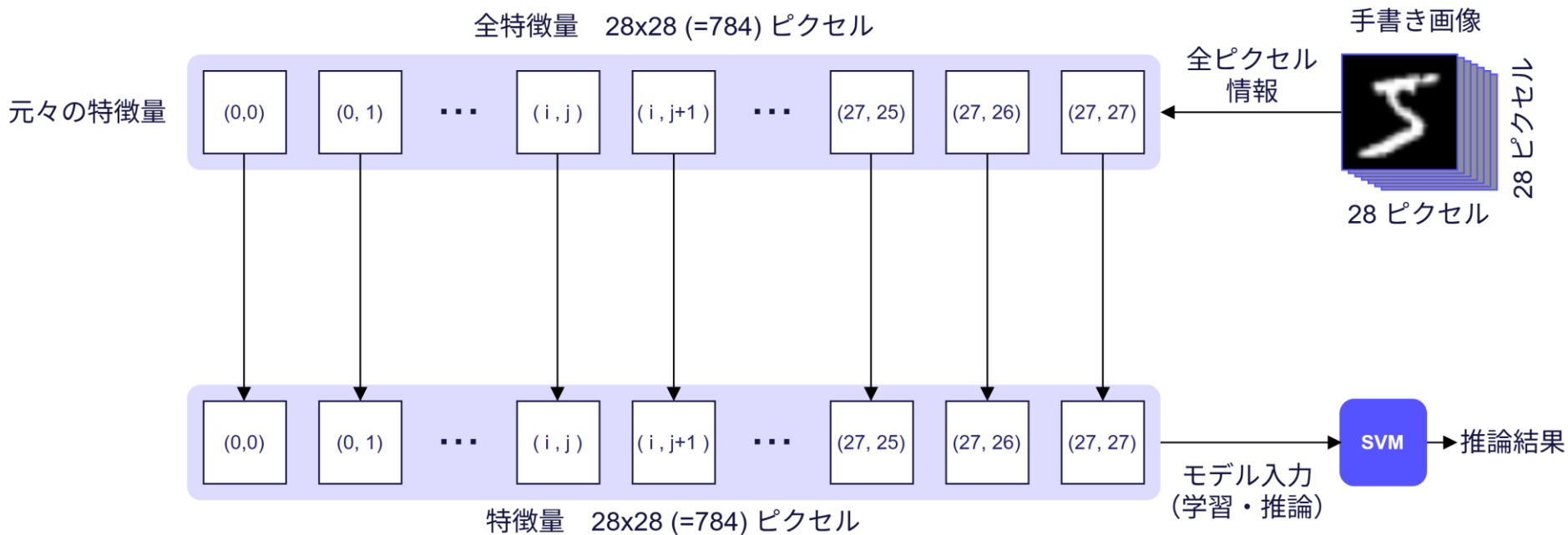
*2 [Multi-objective Bayesian optimization over high-dimensional search spaces](#)



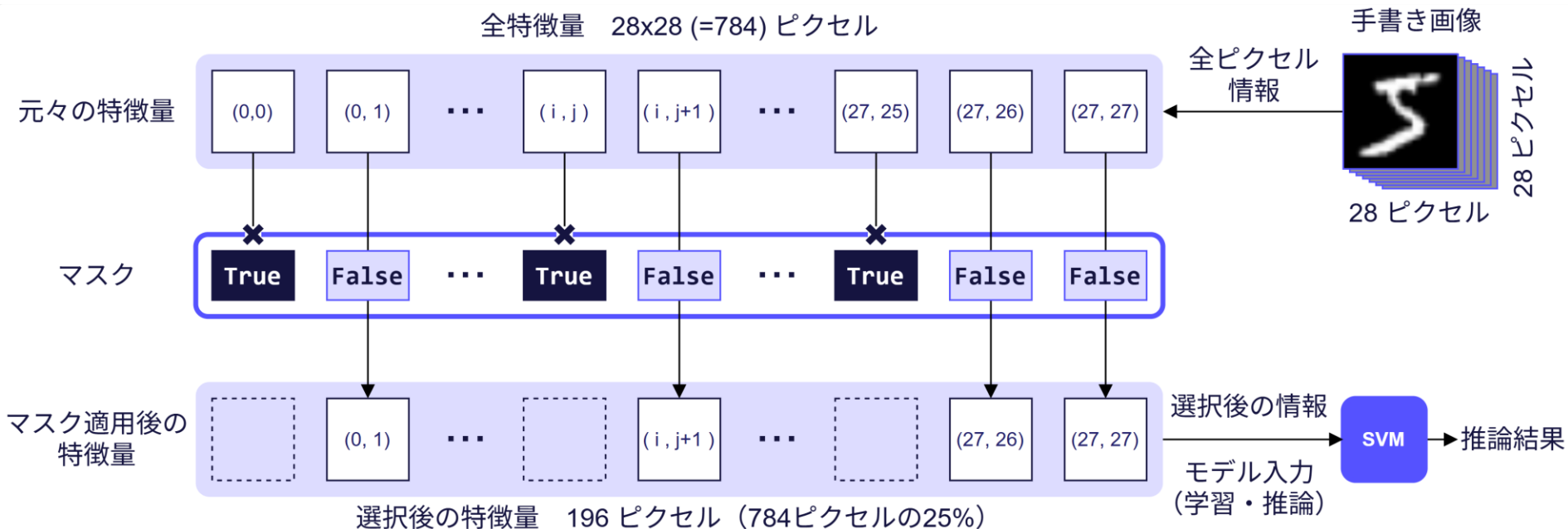
ブラックボックス最適化

1. 機械学習 (FM) ハンズオン
2. FMQA ハンズオン

SVMによるMNIST画像認識モデル



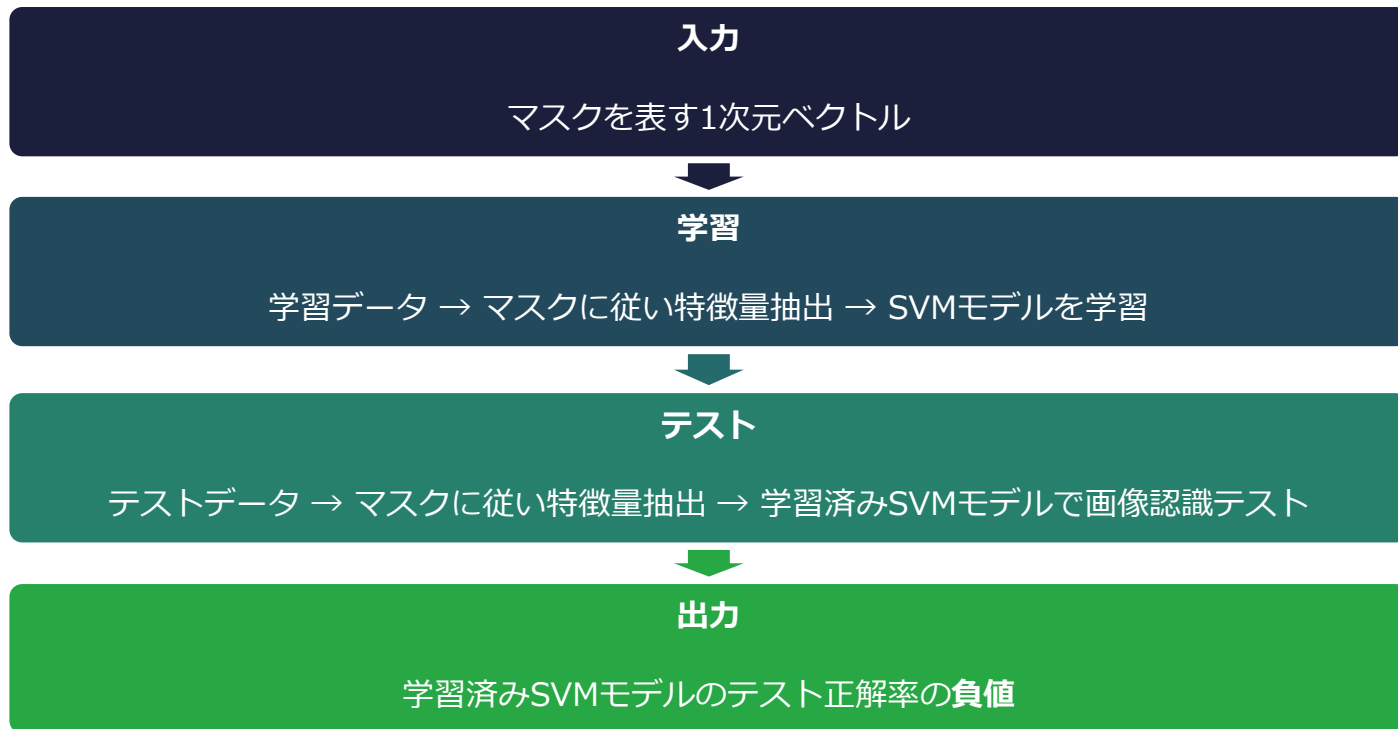
SVM画像認識モデルの最適特徴量選択



最適なマスクを探索し、軽く、正解率が高いSVMを目指したい！

画像認識モデルの特徴量選択・最適化フロー

- ブラックボックス目的関数



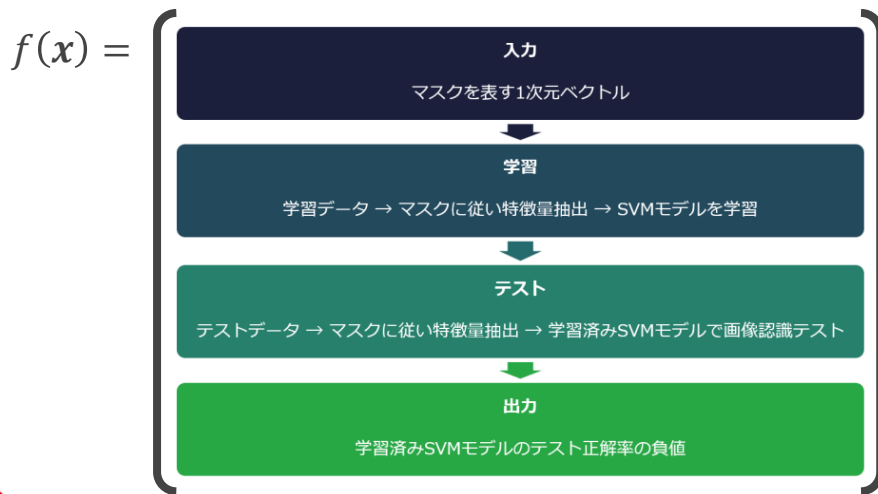
ブラックボックス最適化のデモプログラム：機械学習（FM）

ブラックボックス関数を予測するFMモデル

『マスク』を入力とし、『マスクに従って削減された特徴量に基づく SVM モデルの精度』を出力する

問題設定

マスクを x とすると、ブラックボックス関数 $f(x)$ は、



最適化サイクル

1. 学習データからモデル関数 $g(x)$ を構築 (FM) 
2. モデル関数 $g(x)$ が最小となる点 $\hat{x}$ を推定 (QA)
3. 評価結果 $(\hat{x}, f(\hat{x}))$ を学習データに追加

↑のサイクルにより、最適化点近傍におけるFMの予測精度が向上し、組合せ最適化により、より良い $\hat{x}$ の推定が期待される

FMデモプログラム 1/6 (SVMクラスの実装)

```
PIX_SIZE = 28 # 画像1辺のピクセルサイズ
```

```
class MnistSvm:
```

```
    def __init__(
        self, mask_ratio: float, train_size=1000, test_size=1000, seed=0
    ) -> None:
        """与えられたマスク率に応じてクラスを初期化"""
        self._data, self._label = datasets.fetch_openml("mnist_784", version=1, return_X_y=True, parser="auto")
        ...

        self._num_features = len(self._data.columns) # 特徴量の数
        self._num_masked_features = int(self._num_features * mask_ratio) # マスクする特徴量の数

    def generate_random_mask(self) -> np.ndarray[bool, Any]:
        """マスク率を満たすマスクをランダムに生成"""
        mask = [True] * self._num_masked_features + [False] * (self._num_features - self._num_masked_features)
        self._rng.shuffle(mask)
        return np.array(mask)

    def train_eval(self, mask: np.ndarray[bool, Any]) -> float:
        """マスク適用後のデータに基づき機械学習を行い、学習後のモデル評価を実施、モデルの正答率を返却"""
        self._mask = mask
        train_data, test_data, train_label, test_label = self._fetch_masked_data()
        self._train(train_data, train_label)
        pred = self._eval(test_data, test_label)
        return pred
```

- `__init__`
 - 引数のマスク率に応じてクラスを初期化
 - `_num_features`: 特徴量の総数 (28x28=784個)
 - `_num_masked_features`: マスクする特徴量の数
- `generate_random_mask`
 - マスク率を満たすマスクをランダムに生成する関数
- `train_eval`
 - 与えられたマスクに基づく特徴量で学習を行い、得られたモデルを評価し、モデルの正答率を返却する関数。

FM デモプログラム 2/6 (ブラックボックス関数の定義)

```
# 問題サイズ (=最適化対象のマスクの長さ)
problem_size = PIX_SIZE * PIX_SIZE

def make_blackbox_func(mnist_svm: MnistSvm) -> Callable[[np.ndarray], float]:

    def blackbox(x: np.ndarray) -> float:
        assert x.shape == (problem_size,) # x は要素数 28*28 の一次元配列
        return -mnist_svm.train_eval(
            x
        ) # ブラックボックス関数として、モデルの正解率の負値を返却

    return blackbox
```

関数の利用例 :

```
▶ # マスク率 75% を考慮する MnistSvm クラス
mnist_svm = MnistSvm(mask_ratio=0.75)

blackbox_func = make_blackbox_func(mnist_svm=mnist_svm)

# [True, ..., True, False, ... False] であるようなマスク率 75%のマスク
test_mask = np.array([True] * 588 + [False] * 196)

# 上記のマスクを与えた場合のSVMモデルの正解率 (の負値)
print(f"{blackbox_func(test_mask) = }")
```

- `make_blackbox_func`: ブラックボックス関数を作成する関数
- `blackbox_func`: `MnistSvm.train_eval` から返却される正答率を(-1)倍して返却
→つまり、ブラックボックス関数からの返却値は、学習済みSVMモデルのテストデータに対する正答率の負値
- マスク率75%を考慮する場合のブラックボックス関数
 - 特徴量588個をマスク、それ以外 (196個)をSVMで考慮

FM デモプログラム 3/6 (FMの定義)

```
class TorchFM(nn.Module):
    def __init__(self, d: int, k: int):
        """モデルを構築する

        Args:
            d (int): 入力ベクトルのサイズ
            k (int): パラメータ k
        """
        super().__init__()
        self.d = d
        self.v = nn.Parameter(torch.randn((d, k)))
        self.w = nn.Parameter(torch.randn((d,)))
        self.w0 = nn.Parameter(torch.randn(()))

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        """入力 x を受け取って y の推定値を出力する

        Args:
            x (torch.Tensor): (データ数 × d) の 2 次元 tensor

        Returns:
            torch.Tensor: y の推定値 の 1次元 tensor (サイズはデータ数)
        """
        out_linear = torch.matmul(x, self.w) + self.w0
        out_1 = torch.matmul(x, self.v).pow(2).sum(1)
        out_2 = torch.matmul(x.pow(2), self.v.pow(2)).sum(1)
        out_quadratic = 0.5 * (out_1 - out_2)

        out = out_linear + out_quadratic
        return out
```

- FM モデルを PyTorch で定義する

$$f(\mathbf{x}|\mathbf{w}, \mathbf{v}) = \underbrace{w_0 + \sum_{i=1}^d w_i x_i}_{\text{out_linear}} + \underbrace{\frac{1}{2} \left[\sum_{f=1}^k \underbrace{\left(\sum_{i=1}^d v_{if} x_i \right)^2}_{\text{out_1}} - \sum_{f=1}^k \underbrace{\sum_{i=1}^d v_{if}^2 x_i^2}_{\text{out_2}} \right]}_{\text{out_quadratic}}$$

FM デモプログラム 4/6 (FMの学習)

```
def train(x: np.ndarray, y: np.ndarray, model: TorchFM, plot_learning_curve=False):
    # イテレーション数
    epochs = 2000
    # モデルの最適化関数
    optimizer = torch.optim.Adam([model.v, model.w, model.w0], lr=0.3)
    # 学習率スケジューラ
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=200, gamma=0.9)
    # 損失関数
    loss_func = nn.MSELoss()
    # データセットの用意
    x_tensor, y_tensor = (
        torch.from_numpy(x).float(),
        torch.from_numpy(y).float(),
    )
    dataset = TensorDataset(x_tensor, y_tensor)
    train_set, valid_set = random_split(dataset, [0.8, 0.2])
    batch_size = 8
    train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True)
    valid_loader = DataLoader(valid_set, batch_size=batch_size, shuffle=True)

    # 学習の実行
    min_loss = 1e18 # 損失関数の最小値を保存
    max_corrcoef = -1e18 # 相関係数の最大値を保存
    losses: list[float] = [0.0] * epochs
    corrcoefs: list[float] = [0.0] * epochs
    best_state = model.state_dict() # 最も良いモデルのパラメータを保存するための変数
    for i in range(epochs):
        # 学習フェイズ
        for x_train, y_train in train_loader:
            optimizer.zero_grad()
            pred_y = model(x_train)
            loss = loss_func(pred_y, y_train)
            loss.backward()
```

FM 学習は、通常の機械学習と同様に進める。教師データを学習・検証データに分割し、ミニバッチ学習。

- **x, y**: 教師データ
- **model**: FM モデル (TorchFM)
- **epochs**: エポック（繰り返し）の数
- **lr**: (初期) 学習率
- **scheduler**: 学習率スケジューラー

学習後、**最も良いモデル**に対し、次のような評価を実施

```
# 教師データの入力値に基づきモデルを評価
y_pred = model(x_tensor)
print(f"corrcoef: {torch.corrcoef(torch.stack([y_tensor, y_pred]))[0][0]}")
print(f"RMS error: {min_loss:.2f}")
```

FM デモプログラム 5/6（教師データ作成）

```
def init_training_data(n0: int, mnist_svm: MnistSvm, blackbox_func: Callable[[np.ndarray], float]):  
    """n0 組の初期教師データを作成する"""  
    assert n0 < 2**problem_size  
  
    # n0 個のマスクを乱数を用いて作成  
    x = np.array([mnist_svm.generate_random_mask() for _ in range(n0)])  
  
    # マスクの重複が発生していたらランダムに値を変更して回避する  
    x = np.unique(x, axis=0)  
    while x.shape[0] != n0:  
        x = np.vstack((x, mnist_svm.generate_random_mask()))  
        x = np.unique(x, axis=0)  
  
    # blackbox 関数を評価して入力マスクに対応する n0 個の出力を得る  
    y = np.zeros(n0) + 1e10  
    for i in range(n0):  
        start = time.perf_counter()  
        y[i] = blackbox_func(x[i])  
        print(  
            f"Random process {i}: found y={y[i]:.4f}, current best=[np.min(y):.4f], "  
            f"cycle elapsed time=[time.perf_counter()-start:.2f]s"  
        )  
    return x, y
```

教師データを乱数により生成

- **mnist_svm** : **MnistSvm** クラスインスタンス
- **n0** : 初期教師データのサンプル数
- **blackbox** : ブラックボックス関数（実験又はシミュレーション） $f(x)$

FM デモプログラム 6/6 (メイン部分)

```
n0 = 20 # 初期教師データの数
x, y = init_training_data(n0, mnist_svm, blackbox_func)

# 機械学習モデルの作成
model = TorchFM(problem_size, k=20)
train(x, y, model, plot_learning_curve=True)
```

実際にサンプルプログラムを実行してみましょう。

サンプルプログラム :

<https://colab.research.google.com/drive/1jxnlgA79dzwB4Xi9uCQ5MbLKTgamW0t3>

デフォルトの条件から、

- FMのハイパーパラメータ (**k**)
- エポック数 (**epochs**)

を変更した場合、真値と予測値の相関係数及びRMS誤差はどのように変化するでしょうか？

ブラックボックス最適化

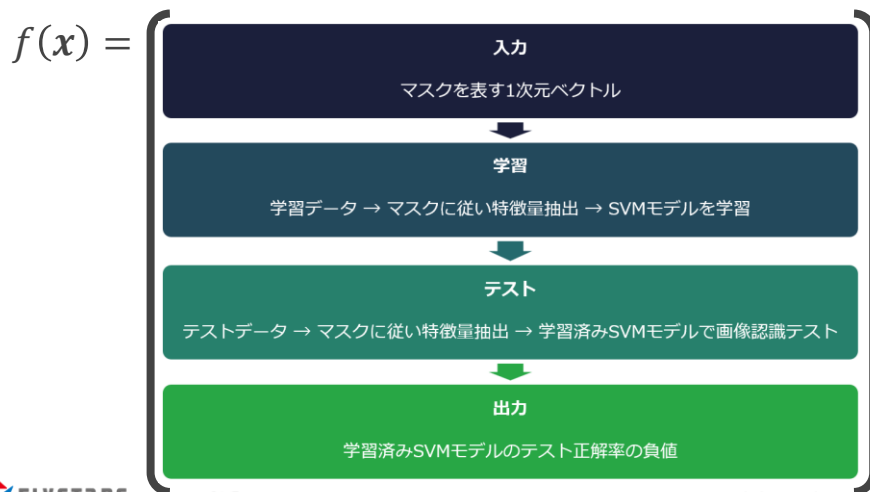
1. 機械学習 (FM) ハンズオン
2. FMQA ハンズオン

ブラックボックス最適化のデモプログラム：FMQA

FMQAによる画像分類モデルのための最適特徴量選択

問題設定

マスクを x とすると、ブラックボックス関数 $f(x)$ は、



最適化サイクル

1. 学習データからモデル関数 $g(x)$ を構築 (FM)
2. モデル関数 $g(x)$ が最小となる点 $\hat{x}$ を推定 (QA)
3. 評価結果 $(\hat{x}, f(\hat{x}))$ を学習データに追加

↑のサイクルにより、最適化点近傍におけるFMの予測精度が向上し、組合せ最適化により、より良い $\hat{x}$ の推定が期待される

FMQA デモプログラム 1/3 (アニーリング部分)

```
# ソルバークライアントを Amplify AE に設定
client = FixstarsClient()
client.parameters.timeout = timedelta(milliseconds=5000)
client.token = token # ローカル環境等で実行する場合はコメントを外して Amplify AEのアクセストークンを

def anneal(
    torch_model: TorchFM,
    constraint: Callable[[PolyArray], Constraint],
) -> np.ndarray:
    """受け取った FM モデルの最小値を与える x を求める"""

    # 長さ d のバイナリ変数の配列を作成
    gen = VariableGenerator()
    x = gen.array("Binary", torch_model.d)

    # TorchFM からパラメータ v, w, w0 を取得
    v, w, w0 = torch_model.get_parameters()

    # 目的関数を作成
    out_linear = w0 + (x * w).sum()
    out_1 = ((x[:, np.newaxis] * v).sum(axis=0) ** 2).sum() # type: ignore
    out_2 = ((x[:, np.newaxis] * v) ** 2).sum()
    objective: Poly = out_linear + (out_1 - out_2) / 2

    # 組合せ最適化モデルを構築
    amplify_model = Model(objective, constraint(x))

    # 最小化を実行 (構築したモデルと、始めに作ったソルバークライアントを引数として渡す)
    result = solve(amplify_model, client)
    if len(result.solutions) == 0:
        raise RuntimeError(f"No solution was found.")

    # モデルを最小化する入力ベクトルを返却
    return x.evaluate(result.best.values).astype(int)
```

学習済みFMモデルに基づき $\hat{x}$ を推定する関数

- 決定変数配列の作成
- 学習済みFMモデルからパラメータ (重み・バイアス) を取得
- パラメータに基づき目的関数のサロゲートモデル $g(x)$ を作成
- サロゲートモデルを制約条件 (後述) と合わせて `amplify.Model` を作成
 - 単に、`objective + constraint(x)` でも良い
- `amplify.solve` によりアニーリングを実行
- このサイクルにおける $\hat{x}$ を返却

FMQA デモプログラム 2/3 (制約条件の設定)

```
def get_constraint(q: PolyArray) -> Constraint:
```

```
    """  
    マスクされる特徴量は mnist_svm.num_masked_features 個という制約条件を  
    返却する関数。制約条件の重みとして 2 を選択。制約条件の重みについては、  
    https://amplify.fixstars.com/ja/docs/amplify/v1/constraint.html#id8 を参照。  
    """
```

```
    return 2 * equal_to(q.sum(), mnist_svm.num_masked_features)
```

- `get_constraint(q)`

最適マスクを決定する決定変数 x に基づく以下の制約条件を返却する関数

制約条件

- $\sum_0^n x_i = \text{mnist_svm.num_masked_features}$

FMQA デモプログラム 3/3 (メイン部分)

```
# N 回のイテレーションを実行
for i in range(n):
    print(f"FMQA cycle {i}")
    # 機械学習モデルの作成
    model = TorchFM(problem_size, k=20)
    corrcoeff = train(x, y, model)
    print(f"-- corrcoeff: {corrcoeff:.2f}")

    # 学習済みモデルの最小値を与える入力ベクトルの値を取得
    x_hat = anneal(model, get_constraint)
    # x_hat が重複する場合、それに近い解を乱数に基づき生成する
    while (x_hat == x).all(axis=1).any():
        idx_0 = rng.choice(np.arange(problem_size))
        idx_1 = rng.choice(np.arange(problem_size))
        x_hat[idx_0], x_hat[idx_1] = x_hat[idx_1], x_hat[idx_0]

    # 推定された入力ベクトルを用いてブラックボックス関数を評価
    y_hat = blackbox_func(x_hat)

    # 評価した値をデータセットに追加
    x = np.vstack((x, x_hat))
    y = np.append(y, y_hat)
    print(f"-- found y = {y_hat:.4f}, current best = {np.min(y):.4f}")
```

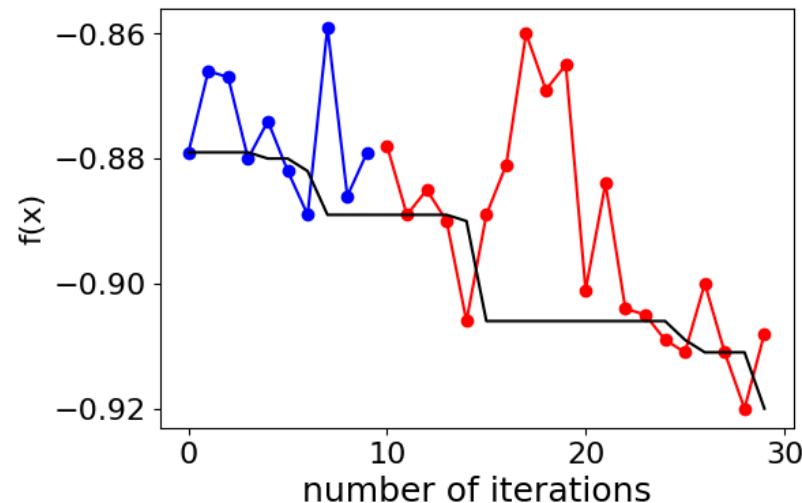
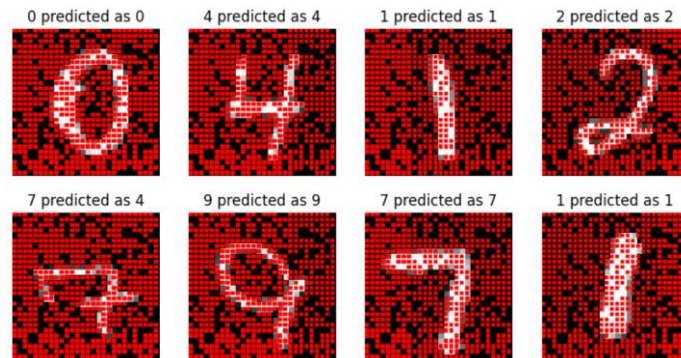
1. `train()` による FM モデルの学習
2. `anneal()` によるアニーリング
 - FM モデル値を最小化するような入力値（マスク）を探索
3. 得られた入力値（マスク）が既存の場合の処理
4. 得られた入力値（マスク）によるブラックボックス関数の評価
 - マスク適用後の学習データを使ってSVMモデル学習
 - マスク適用後のテストデータを使ってSVMモデル評価
 - 正答率の負値を返却
5. 新しい入出力ペアをデータセットに追加

FMQA デモプログラムの実行

サンプルプログラム：

https://colab.research.google.com/drive/1vuMG8aSOpnf0viNa1e0o005GgP_Mwizr

- 全特徴量 (784個) を考慮した場合のモデル評価
 - accuracy_score=0.905
- FMQA の基本条件
 - 初期データ個数：10個
 - FMQAの最適化サイクル数：20回
 - 特徴量：784個→196個（75%減）
- 実行時間
 - およそ5分



（上）最終的に得られたマスクをいくつかの手書き画像に重ねた例と（下）最適化の履歴

FMQA デモプログラムの実務での活用方法

```
def make_blackbox_func(mnist_svm: MnistSvm) -> Callable[[np.ndarray], float]:  
  
    def blackbox(x: np.ndarray) -> float:  
        assert x.shape == (problem_size,) # x は要素数 28*28 の一次元配列  
        return -mnist_svm.train_eval(  
            x  
        ) # ブラックボックス関数として、モデルの正解率の負値を返却  
  
    return blackbox
```



```
def make_blackbox_func(d: int) -> Callable[[np.ndarray], float]:  
    # 必要に応じて最適化対象以外の条件設定や初期設定などをここで実施  
  
    def blackbox(x: np.ndarray) -> float:  
        # x の組合せを元に実験を実施  
        # または、シミュレーションを実施  
        # 実験・シミュレーション結果を後処理し、最小化したい目的関数値を取得  
        objective = hogehoge  
        return objective # type: ignore  
  
    return blackbox
```

基本的に、`blackbox()` を変更する。必要に応じて、現在の教師データの出力などを追加。

- 例① : `blackbox()` 内で特徴量選択ではなく、枝刈りやデータ選択、ハイパラチューニング等を実施し、学習済みモデルの評価結果が最適になる様に手法を調整
- 例② : `blackbox()` 内でシミュレーションを呼び出し、後処理、その戻り値を最小化するように最適化。
- 例③ : `blackbox()` 内で実験を行う。つまり、1回のFMQA で推定された探索候補 $\hat{x}$ を対象に実験し、結果を教師データに追加、次のFMQA 試行を行う。
- $h(x)$ を最大にするような入力 x を推定する場合は、 $1/h(x)$ や $-h(x)$ などを目的関数 $f(x)$ とする。

今後について

ぜひ、デモ・チュートリアルにあるサンプルコードにも挑戦してみてください！

一般的な組合せ最適化問題

目的関数のみで
定式化



チュートリアル編
画像のノイズ除去
プログラミング難易度 ★★★★★
画像のノイズ除去を行うアプリケーションを開発します。

サンプルコード

制約条件のみで
定式化



チュートリアル編
会議室割当問題
プログラミング難易度 ★★★★★
制約条件を用いて定式化するアプリケーションの例として会議室割当問題のアプリケーションを開発します。

サンプルコード

目的関数 + 制約条件



デモアプリケーション
巡回セールスマン問題
プログラミング難易度 ★★★★★
Fixstars Amplifyによる、巡回セールスマン問題の定式化を体験します。

デモアプリ

サンプルコード



デモアプリケーション
容量制約つき運搬経路問題 (CVRP)
プログラミング難易度 ★★★★★
配送先における効率的な配送計画の策定やごり収集や遺棄品回収における巡回経路の最適化等での応用が期待される容量制約つき運搬経路問題 (CVRP) を取り扱います。

デモアプリ

サンプルコード

ブラックボックス最適化問題

概要



チュートリアル編
ブラックボックス最適化 (1)
プログラミング難易度 ★★★★★
複雑で未知な目的関数にも適用可能な、機械学習と組み合わせた最適化を組み合わせたブラックボックス最適化手法を紹介し、Amplifyを用いて実装します。

サンプルコード

材料探索



チュートリアル編
ブラックボックス最適化 (2)
プログラミング難易度 ★★★★★
機械学習と量子アニーリング・イジングマシンを活用するブラックボックス最適化の適用例として、類似的な高温超伝導を実現する材料探索を取り扱います。

サンプルコード

翼型最適化



チュートリアル編
ブラックボックス最適化 (4)
プログラミング難易度 ★★★★★
流体力学設計に不可欠な翼型の最適化問題を取り上げます。最適化には、組み合わせた最適化及び機械学習に基づくブラックボックス最適化と流体シミュレーションを用い、翼の揚力比を最大化するように翼型の探索を行います。

サンプルコード

信号機制御



チュートリアル編
ブラックボックス最適化 (5)
プログラミング難易度 ★★★★★
ブラックボックス最適化により、商業施設による交通集中が発生し得る都市における、交通渋滞を低減するよう信号機制御の最適化を実施します。最適化の実施及び検証には、マルチエージェント・シミュレーションによる交通シミュレーションを用います。

サンプルコード



困った時はドキュメンテーションを！

<https://amplify.fixstars.com/docs/amplify/v1/index.html>

今後のセミナーの予定

今後も定期的に無料セミナーを開催します！

2025/11/20 「ブラックボックス最適化 (機械学習)」

- ・はじめに
- ・会社紹介
- ・Fixstars Amplifyの紹介
- ・ブラックボックス最適化のワークショップ
- ・Wrap Up
 - ・事例のご紹介
 - ・今後の進め方
 - ・Q&A

2025/12/4 (受付中) 「Amplify-BBOpt 技術解説」

- ・はじめに
- ・会社紹介
- ・Fixstars Amplifyのご紹介
- ・Amplify-BBOpt 技術解説
- ・Wrap Up
 - ・事例のご紹介
 - ・今後の進め方
 - ・Q&A

2025/12/18 (受付準備中) 「経路最適化」 (Annealing Engine)

- ・はじめに
- ・会社紹介
- ・Fixstars Amplifyのご紹介
- ・経路最適化のワークショップ
- ・Wrap Up
 - ・事例のご紹介
 - ・今後の進め方
 - ・Q&A

ご質問・ご不明点がありましたら、お問い合わせフォームでご連絡下さい

<https://amplify.fixstars.com/ja/contact>

Q&A