# 1年生実習 第3週

B5 研究室

2024 年 7 月 3 日

## 1　SVM(サポートベクターマシン)

本日使用するモデルは，SVM(サポートベクターマシン) です．SVM は，分類問題において高い性能を発揮することが知られています．

それでは，SVM の基本的な原理について見てみましょう．分類対象のデータとして，図 1 のような 2 クラスの 2 次元データを考えます．

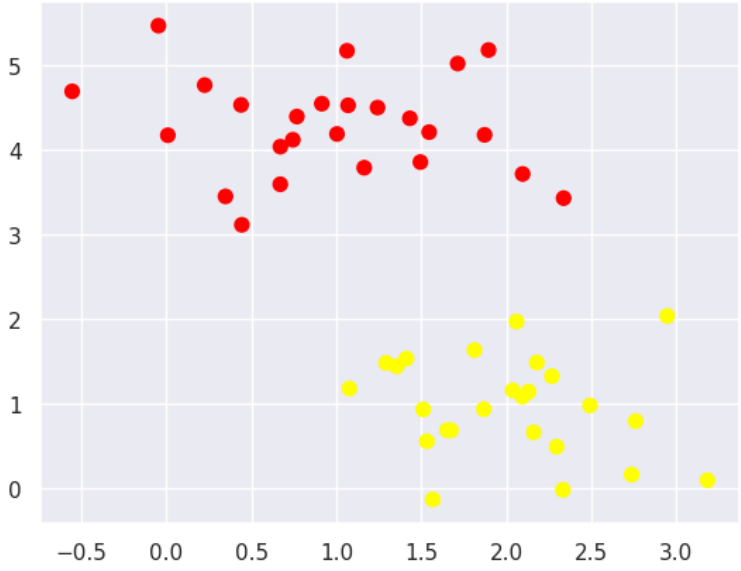


図 1: SVM のデータ

赤色と黄色の 2 クラスのデータを分類するために，2 つのクラスを分離する直線を見つけることが目標です．このとき，どのようにして直線を引くことができるでしょうか? 次のページに進む前に**定規で試しに線を引いてみてください．**

クラスを分割する線の例として，図 2 のような線が考えられます．

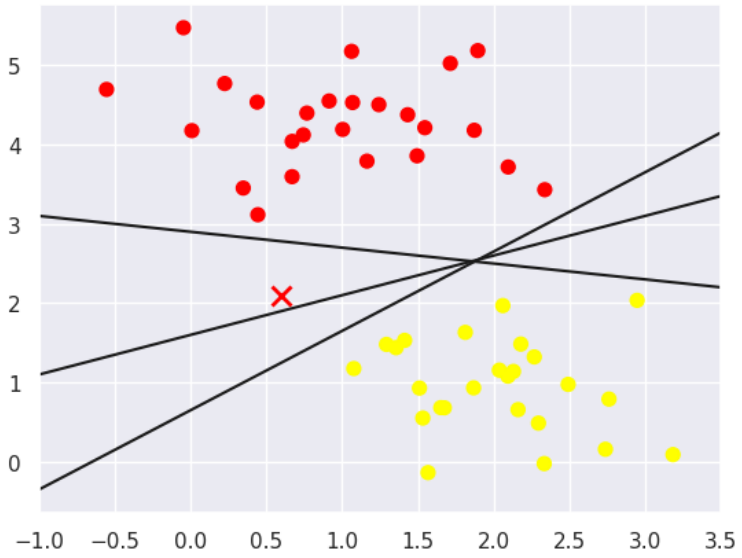


図 2: SVM の直線

これらの線は，赤色と黄色の線をうまく分割できています．しかしながら，どの線を境界とするかによって，図 2 の赤い × で示したデータのクラスが変わってしまいます．「クラス間に線を引く」という考え方は直感的で簡単ですが，実際の問題を解くためにはまだ不十分であることがわかります．

そこで登場するのが，マージン最大化という考え方です．マージンとは，クラス間の最も近いデータ点と境界線の距離のことです．クラス間の最も近いデータ点のことをサポートベクターと呼びます．図 2 の線におけるマージンを可視化した図を図 3 に示します．

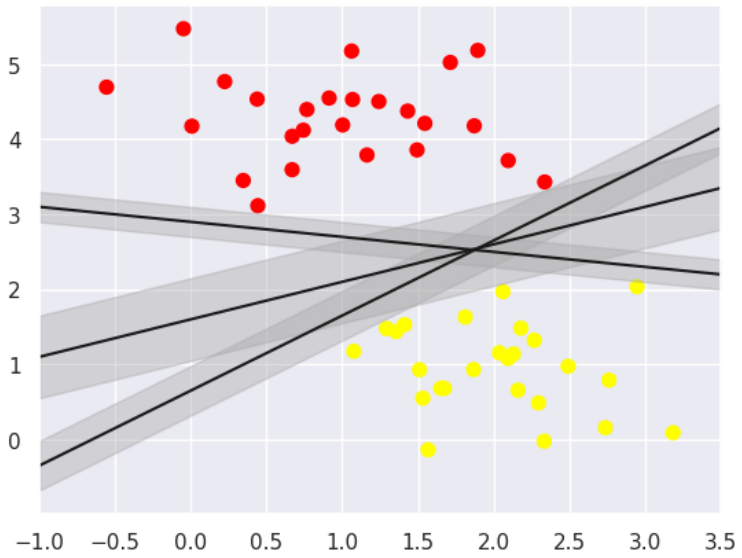


図 3: SVM のマージン

図 3 の例では，中央の線が最も大きなマージンを持っています．しかしながら，この線は最大のマージンを持っているわけではありません．SVM では，このようなマージンを最大化する直線を見つけることができます．SVM によって見つけられた直線は，図 4 のようになります．

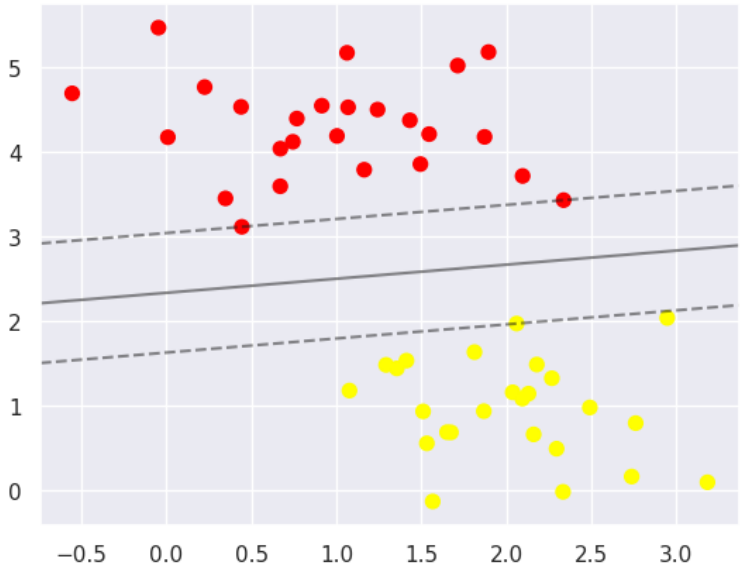


図 4: SVM のサポートベクターとマージン

図 4 の直線は，サポートベクターによって定義されるマージンを最大化する直線です．

では，データ数が変わった場合はどうなるでしょうか? 図 5 に，データ数が変化した場合の SVM の分割例を示します．

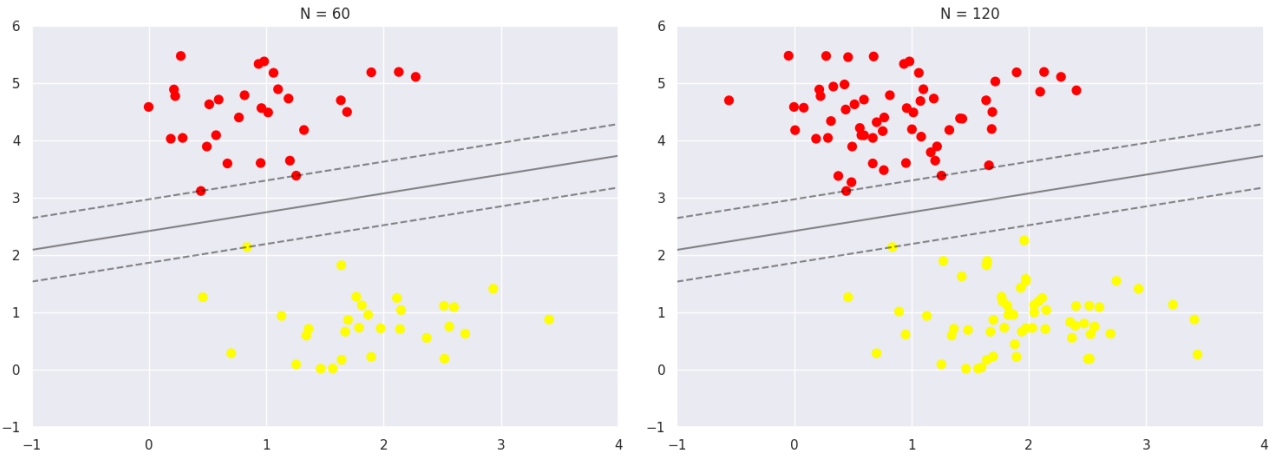


図 5: データ数が変化した場合の決定境界 (右：60 点，左：120 点)

SVM は境界線から遠いデータに影響されず，サポートベクターのみを用いて境界線を引いていることがわかります．このような，境界から離れたデータに対する影響が少ない性質が，SVM の特徴の一つです．

ここまで見てきたデータは，決定境界が直線の場合のデータです．直線で分割できるデータのことを，線形[1]分離可能なデータといいます．では，境界線が直線ではない場合はどうなるでしょうか? 図 6 に，直線で分割できないデータを示します．

[1] “線形” という用語は，様々な場面で登場しますが，基本的には，直線で表せる関係を意味します．

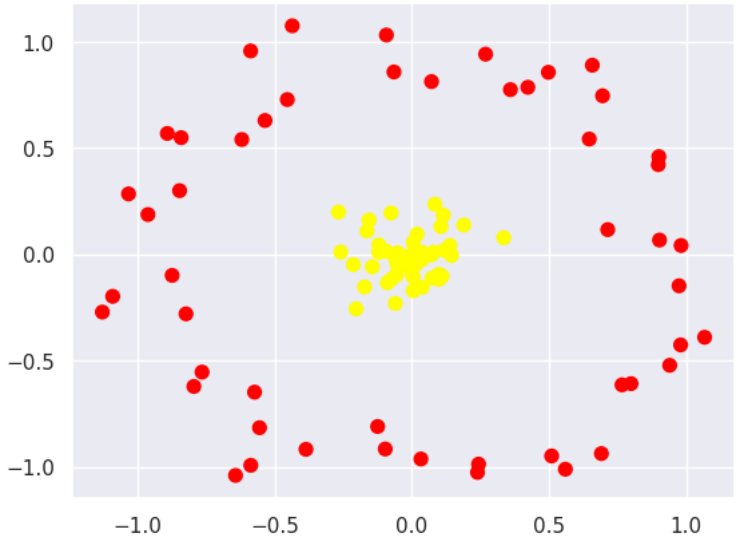


図 6: 直線で分割できないデータ

図 6 のデータを分割するために，図 6 に高さ方向の次元を追加した 3 次元空間を考えてみましょう．このとき，図 6 のデータを 3 次元空間にプロットするための式として，次のような式が考えられます．

$$r = e^{-(x^2+y^2)} \tag{1}$$

式 (1) は，2 次元データ $(x, y)$ を 3 次元データ $(x, y, r)$ に変換する式です．この関数は，原点が最も高い値を持ち，原点から離れるほど値が小さくなるような関数です．(図 7) このような，距離に基づいて値が決まる関数のことを，放射基底関数 (Radial Basis Function: RBF) といいます．実際の RBF 関数は，グラフの変化をコントロールするためのパラメータとして，$\gamma$ という値を持ちます．

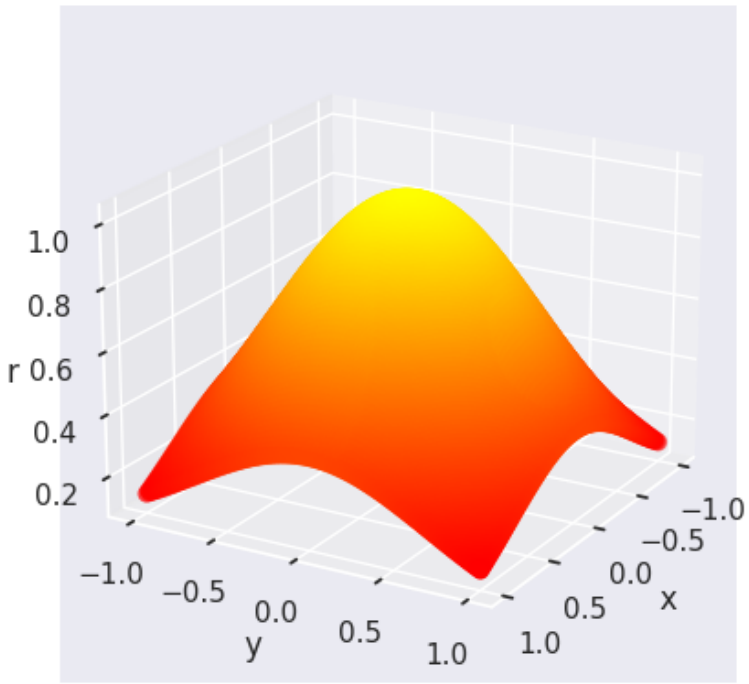


図 7: RBF 関数

では，RBF 関数を用いて，図 6 のデータを 3 次元空間にプロットしてみましょう．図 8 に，RBF 関数を用いて 3 次元空間にプロットしたデータを示します．

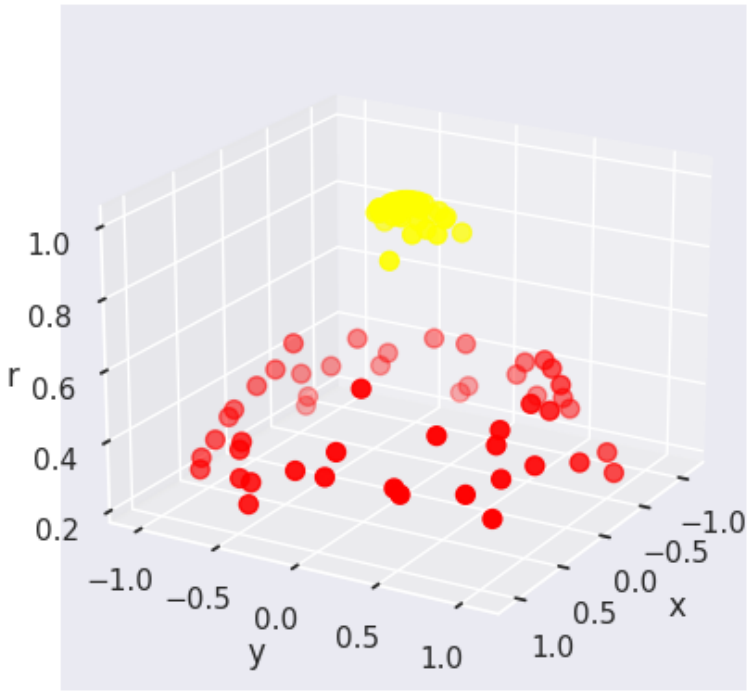


図 8: RBF 関数を用いて 3 次元空間にプロットしたデータ

図 8 を見れば，だいたい $r = 0.7$ となるあたりの平面で分割すればうまくいきそうだとわかります．しかしながら，式 1 はたまたまうまくいっただけであり，(当たり前ですが) すべての場合でうまくいくわけではありません．今回の例では 3 次元空間にデータをプロットしましたが，実際のデータはさらに高次元の空間にプロットされる場合がほとんどです．高次元空間において，データを分割するための平面を見つけるのは非常に難しいです．詳細は割愛しますが，SVM では，カーネルトリックという素晴らしい手法によりこの問題を解決しています．カーネルトリックを使用することで，SVM は高次元空間での計算を効率的に行い，非線形なデータでも効果的に分離することができます．SVM はカーネルトリックという手法の恩恵を受けている，ということだけ覚えておいてください．

話をデータの分類に戻します．図 9 に RBF 関数によって分類された結果を示します．

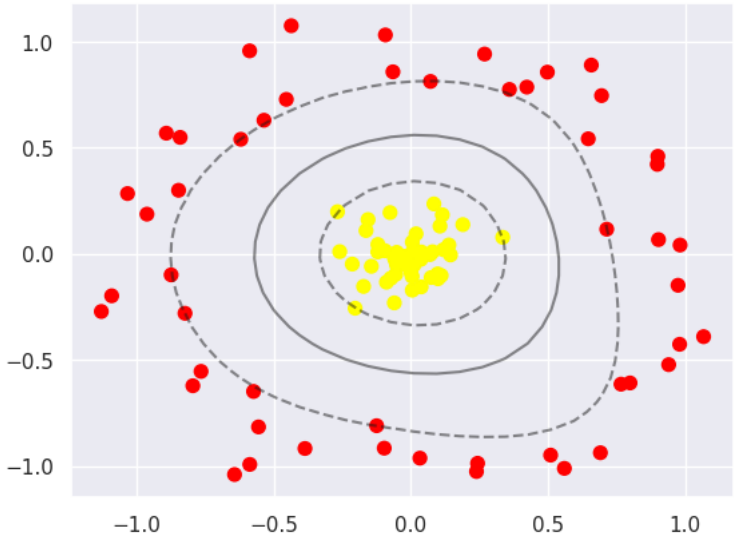


図 9: SVM の分類結果 (RBF 関数)

線形に分割できなかったデータを RBF 関数を用いて，マージンが最大となるようにうまく分割することができ

ました．RBF 関数のような，データを高次元空間に写像する関数をカーネル関数といいます．RBF カーネル以外にも，多項式カーネルやシグモイドカーネルなど，様々なカーネル関数が存在します．カーネル関数を用いて，データを高次元空間に写像する手法のことをカーネル法といいます．カーネル法を使用した SVM は，非線形なデータに対しても高い性能を発揮することが知られているため，頻繁に使用されます．

# 2 機械学習によるデータ分類

今週の実習では，先週の実習で皆さんから集めた手書き文字データを分類するための機械学習モデルを構築します．現状のデータセットの状態は以下の通りです．

- 取得文字: ◯× の 2 文字
- 取得文字数: 3 セット × 10 人の 60 文字
- データ長：最長のデータに合わせて引き伸ばし (M1 の方で処理済)
- 座標軸：X, Y, Z の 3 軸

このデータセットを用いて，分類を行うための機械学習モデルを構築します．

今回の演習で使用するライブラリは，Scikit-learn です．Scikit-learn は，Python で使用できる機械学習ライブラリの一つで，多くの機械学習モデルを提供しています．

## 2.1 データの準備

Python を用いて Google Drive から分類用データをダウンロードします．(ソースコード 1)

ソースコード 1: データのダウンロード

```
# 分類するデータのダウンロード
import gdown

url = "https://drive.google.com/uc?id=1rcAg1R0rYLoeQ2ColX4BvM80u841qx5W"
output = "data.zip"
gdown.download(url, output, quiet=False) # Google Drive から data.zip をダウンロード

# ダウンロードしたデータを解凍
!mkdir data
!unzip -o data.zip -d data
!rm data.zip
```

Jupyter Notebook では，“!”を先頭につけることで，Linux コマンドを実行することができます．上記のコードでは，Python コードで Google Drive からデータをダウンロードしたあと，Linux コマンドを用いて ZIP ファイルを解凍 (不要なファイルを削除) しています．

## 2.2 データの分割

今回使用するデータは実際に計測した数値データに前処理を適用したあとのデータです．したがって，今回は正規化などの前処理は不要です．

分類器による学習のために，データを訓練データとテストデータに分割します．分割する割合は，訓練データ:テストデータ = 8:2 とします．(ソースコード 2)

ソースコード 2: データの分割

```
from sklearn.model_selection import train_test_split # データ分割を行う関数のインポート

# データの分割
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1234)
```

**ライブラリのインポート**

Python では，様々な処理を行うための外部ライブラリが豊富に用意されています．ライブラリをインポートするためには，import 文を用います．

ソースコード 3: ライブラリのインポート

```
import ライブラリ名 # ライブラリのインポート
import ライブラリ名 as 別名 # ライブラリのインポート（別名をつける）
from ライブラリ名 import 関数名 # ライブラリ内の特定の関数のみをインポート
from ライブラリ名 import * # ライブラリ内の全ての関数をインポート（非推奨）
```

random_state は，データを分割する際の乱数のシードを指定する引数です．random_state を指定することで，データの分割を再現可能にすることができます．再現性を確保しておくことは，実験結果の検証の際に非常に重要となります．シードの値は任意で構いませんが，実験結果を再現するためには，同じ値を指定しておくことが一般的です．

## 2.3 SVM による分類

データの分割が完了したら，SVM による分類を行います． Scikit-learn の SVM を使用するためには，SVC クラスをインポートします．(ソースコード 4)

ソースコード 4: SVM による分類

```
from sklearn.svm import SVC # SVM のインポート

# SVM のインスタンスを生成
clf = SVC(kernel='linear', random_state=1234)

# モデルの学習
clf.fit(X_train, y_train)
```

SVC クラスには，いくつかの引数があります．kernel は，カーネル関数を指定する引数です．この例では，線形カーネルを使用しているため，kernel='linear' と指定しています．他にも，多項式カーネル (kernel='poly') や RBF カーネル (kernel='rbf') など，様々なカーネル関数を指定することができます．

また，他の引数として，C や gamma などがあります．これらのパラメータを調整することで，SVM の性能を向上させることができますが，今回はデフォルト値を使用します．

## 2.4 予測と評価

学習が完了したら，テストデータを用いて予測を行い，モデルの性能を評価します．予測を行うためには，predict メソッドを使用します．(ソースコード 5)

ソースコード 5: 予測と評価

```
# テストデータを用いて予測
y_pred = clf.predict(X_test)

# モデルの評価
from sklearn.metrics import accuracy_score, classification_report # 評価指標を計算する関数のインポート
classification_report(y_test, y_pred) # 分類結果の表示
accuracy = accuracy_score(y_test, y_pred) # 正解率の計算
print(f"正解率: {accuracy:.2f}")

# 分類結果の表示
print(classification_report(y_test, y_pred))
```

分類結果を表示するために，`classification_report` 関数を使用しています．classification_report 関数は，適合率 (precision) や再現率 (recall)，F1 スコアなどの評価指標を計算して表示します．これらの指標はどのような意味を持つのでしょうか?

### 2.4.1 混同行列

適合率などの指標の話をする前に，混同行列について説明します．混同行列は，予測結果と実際の結果まとめた表です．混同行列は，表 1 のように表されます．

表 1: 混同行列

| | | 予測値 | |
|---|---|---|---|
| | | 陽性 | 陰性 |
| 実際の値 | 陽性 | TP | FN |
| | 陰性 | FP | TN |

混同行列には，真陽性 (True Positive: TP)，偽陽性 (False Positive: FP)，真陰性 (True Negative: TN)，偽陰性 (False Negative: FN) の 4 つの要素があります．真陽性は，実際に正であり，正と予測されたデータの数を示します．偽陽性は，実際に負であるが，正と予測されたデータの数を示します．真陰性は，実際に負であり，負と予測されたデータの数を示します．偽陰性は，実際に正であるが，負と予測されたデータの数を示します．

### 2.4.2 正解率，適合率，再現率，F1 スコア

正解率，適合率，再現率，F1 スコアの 4 つの指標は，先述した混同行列を用いて計算されます．
各項目の計算式は以下の通りです．

- 正解率 (accuracy)：$\frac{TP+TN}{TP+FP+FN+TN}$
- 適合率 (precision)：$\frac{TP}{TP+FP}$
- 再現率 (recall)：$\frac{TP}{TP+FN}$
- F1 スコア：$\frac{2\times適合率\times再現率}{適合率+再現率}$

正解率とは，全体のデータのうち，正しく分類されたデータの割合を示します．適合率とは，正と予測されたものの中で実際に正であったものの割合を示します．再現率とは，実際に正であったものの中で正と予測されたものの割合を示します．F1 スコアは，適合率と再現率の調和平均です．

モデルを評価するとき，正解率が高いモデルが良いモデルであるとは限りません．正解率だけでモデルの性能を評価するのは危険です．正解率だけの評価では不十分な例として，データの偏りがある場合が挙げられます．筆跡から利き手を予測するモデルを作ることを考えてみます．人口のだいたい 90%以上が右利きであると言われています．したがって，筆跡を根拠としなくても，右利きと予測するだけで正解率が 90%以上になります．しかしながら，このモデルは，左利きの人を全て右利きと予測してしまうため，実用的なモデルとは言えません．このような場合，適合率や再現率などの指標を用いてモデルの性能を評価することが重要です．

では，適合率や再現率はどのような意味を持つのでしょうか? 例として，顧客のプロフィール情報からある商品を購入するかを予測するモデルを考えます．このとき，適合率が表すのは，購入すると予測されたデータのうち，実際にその商品を購入したデータがどれくらいあるかということです．適合率が低いことがわかった場合，購入すると予測されたデータのうち，実際に購入したデータが少ないことを示します．この場合，購入すると予測されたデータのうち，実際に商品を購入しないデータが多いため，無駄な広告費用がかかったり，過剰発注をしてしまうなどの損失が発生することが想定されます．

一方，再現率が表すのは，実際にその商品を購入したデータのうち，購入すると予測されたデータがどれくらいあるかということです．再現率が低い場合，実際に商品を購入したデータのうち，購入すると予測されたデー

タが少ないことを示します．この場合，購入すると予測されたデータが少ないため，在庫が足りなくなってしまうことが想定されます．

これらの例からも分かるように，適合率と再現率はトレードオフの関係にあります．適合率を高くするためには，購入すると予測されたデータを減らす必要があります．一方で，再現率を高くするためには，実際に購入するデータを増やす必要があります．このように，適合率と再現率はトレードオフの関係にあるため，どちらを重視するかによって，モデルの評価が変わってきます．

では，どのようにして適合率と再現率のバランスを取ればよいでしょうか？適合率と再現率のバランスを評価するための指標が F1 スコアです．F1 スコアは，適合率と再現率の調和平均で計算されます．調和平均とは，データの各数値の逆数で平均を取り，その平均の逆数を取ることで求められる平均のことです．算術平均 (=合計 ÷ 個数) ではなく，調和平均を使用する理由として，適合率と再現率のバランスを評価するためです．調和平均は，言い換えれば “作業効率”の平均とも言えます．一方，算術平均は “量”の平均です．作業効率の平均が高いということは，適合率と再現率のどちらも高いスコアであることを示します．

**調和平均の例**

調和平均は，次のように定義されます．

$$\text{調和平均} = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}} \tag{2}$$

数式ばかりでわかりにくいので，具体例で考えてみましょう．徳島市から大阪市までは，大体 150km の距離があります．往路は，時速 50km で走り，復路は時速 75km で走ったとします．このときの平均の速度を考えてみましょう．算術平均を取ると，$\frac{50+75}{2} = 62.5$km/h となります．算術平均の場合，所要時間は $\frac{150}{62.5} = 2.4$ 時間となります．しかし，実際には，徳島市から大阪市までの所要時間は，$\frac{150}{50} + \frac{150}{75} = 3 + 2 = 5$ 時間です．したがって，平均速度は $\frac{300}{5} = 60$km/h となります．
このように，調和平均は効率の平均を計算できます．他の例としては，並列に接続した抵抗器の合成抵抗の計算などがあります．

## 3　課題

本日の課題では，SVM 以外の機械学習モデルを用いて，手書き文字データの分類をしてください．わからない場合や，どの分類器を使用したらよいかわからない場合は，M1 に質問してください．

# A　補足：表のスタイル

ここでは，第 2 週の内容で説明できなかった部分について，補足説明を行います．皆さんは，表を書く際に，罫線の引き方を意識していますでしょうか．たかが罫線，と思うかもしれませんが，適切な罫線を引くことで，表の見やすさが向上します．表 2 と表 3 に良い例と悪い例を示します．

| 実験番号 | 入力 | | 出力 | |
|---|---|---|---|---|
| | 実験条件 | 作業者 | 製品評価 | 能率 [s] |
| 1 | A | a | 優 | 10.5 |
| 2 | A | b | 良 | 12.3 |
| 3 | B | a | 良 | 11.2 |
| 4 | B | b | 良 | 12.0 |

表 2: 悪い表の例

| 実験番号 | 入力 | | 出力 | |
|---|---|---|---|---|
| | 実験条件 | 作業者 | 製品評価 | 能率 [s] |
| 1 | A | a | 優 | 10.5 |
| 2 | A | b | 良 | 12.3 |
| 3 | B | a | 良 | 11.2 |
| 4 | B | b | 良 | 12.0 |

表 3: 良い表の例

表 2 は，長方形で表現されている表であり，とても可読性が低いです．また，罫線が多すぎるので，罫線ばかりに目が行ってしまい，データが見にくくなっています．一方，表 3 は，左右が開放されている表であり，見やすさが向上しています．また，最上段の横罫線は太くすることで，表としての体裁が整っています．

表を作成する際に意識することは，できる限り簡素にすることです．表の項目のジャンルが変わるとき以外は罫線を引かないようにすると，表の見やすさが向上します．

## 参考文献

[1] Jake VanderPlas. Python データサイエンスハンドブック 第 2 版: 菊池彰訳. オライリージャパン, 2021, 545p.