機械学習による分析

広島大学 AI・データイノベーション教育研究センター
村上 祐子

目標

デジタル化された画像データを機械学習で分類することができる。

この授業で紹介すること

- サポートベクトルマシンを用いた画像データの分類
- 性能評価の方法

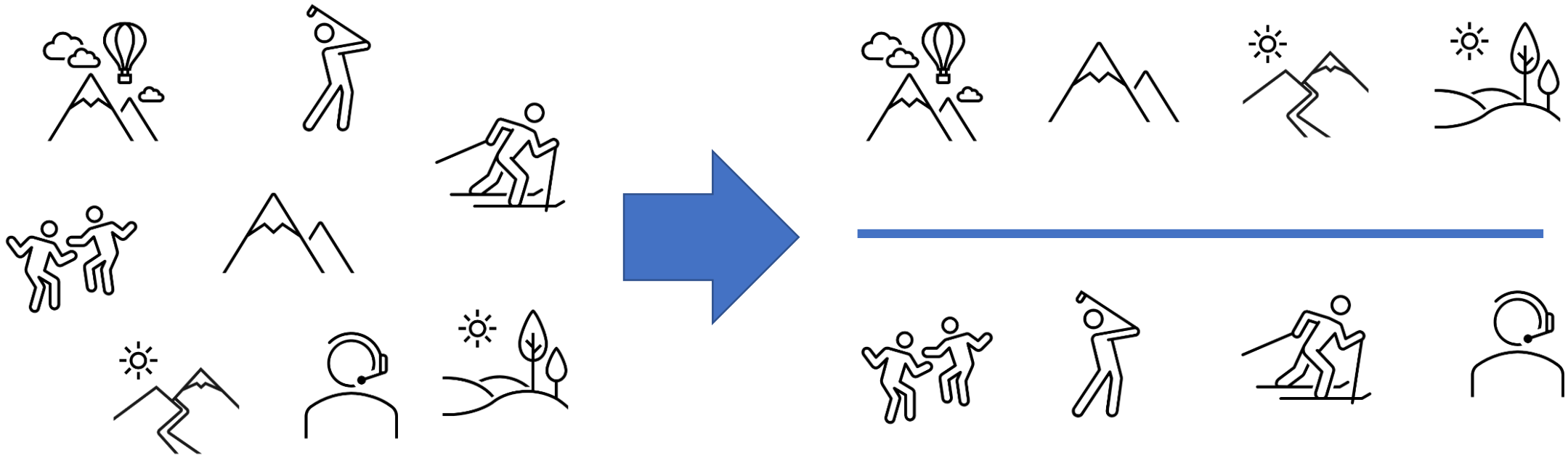
キーワード

分類器、サポートベクトルマシン、正解率、適合率、再現率、F値

「画像データのデジタル化」で作成したデータを使います。
この講義ではGoogle Colab(Colaboratory)を使ったプログラミングを実施します。

こんなことはありませんか？

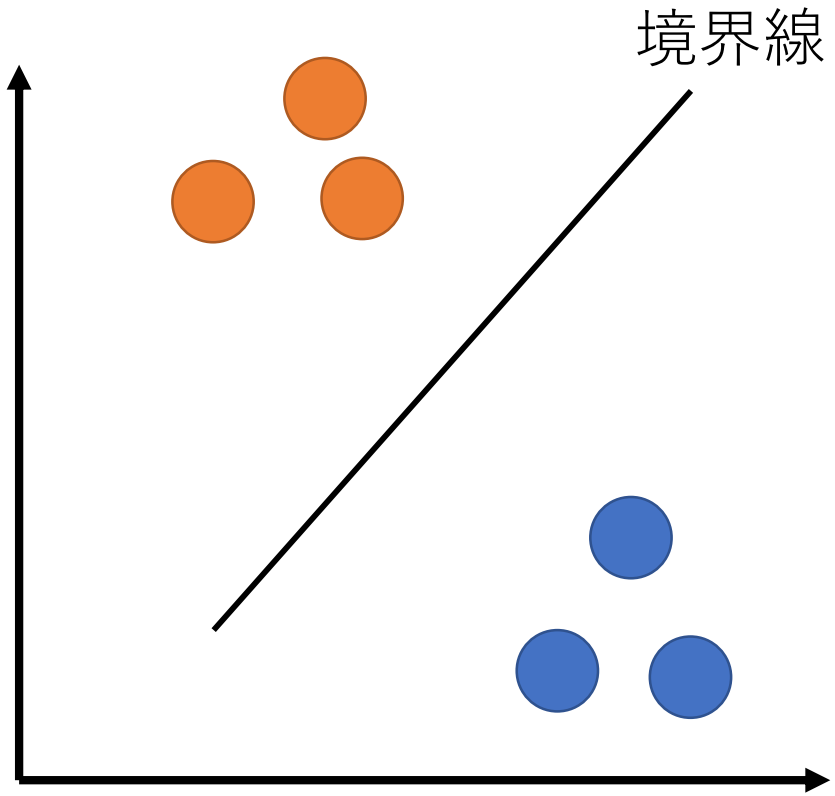
100枚程度の写真から山が主に映っている写真と人が主に映っている写真を分ける過程を自動化したいと思います。プログラムではどのようにこの作業を行っているのでしょうか？



機械学習における分類

分類とはデータを共通の性質に基づいて種類に分けること

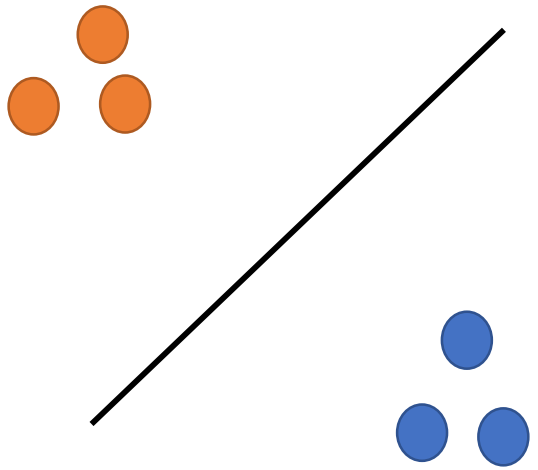
- 機械学習の分類では
  ① 種類に分けるための境界線を決める
  ② 決めた境界線にしたがってデータを分ける
  という操作が行われている
- ①、②を行うアルゴリズムを**分類器**という



データの境界線の種類

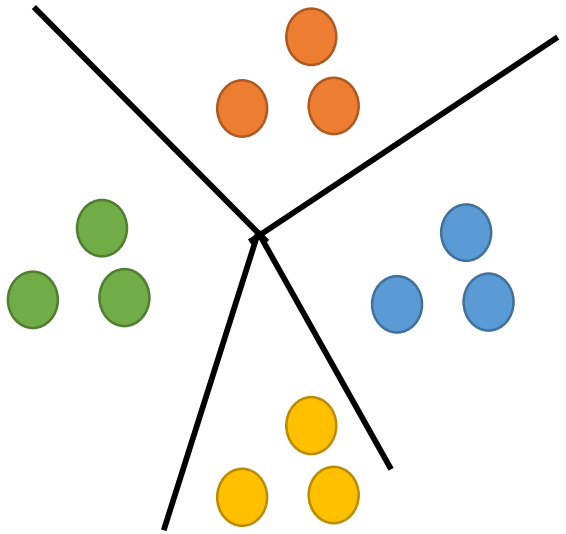
線形分類器

データを一本の直線で分類**できる**場合に使用するアルゴリズム



非線形分類器

データを一本の直線で分類**できない**場合に使用するアルゴリズム



データの境界線の種類

線形分類器

データを一本の直線で分類**できる**場合に使用するアルゴリズム

- 単純パーセプトロン
- 線形サポートベクトルマシン
- ロジスティック回帰

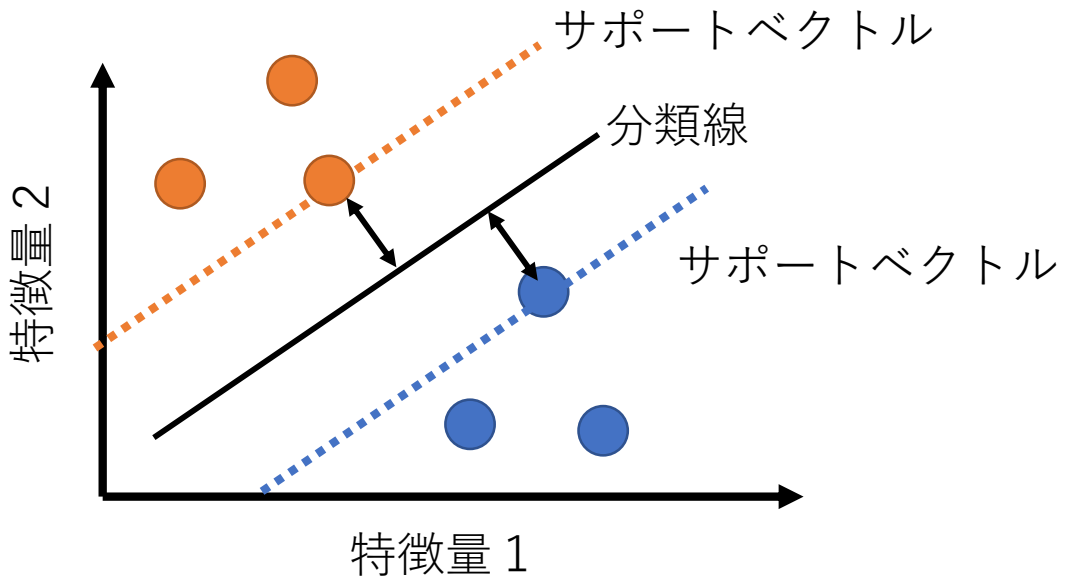
非線形分類器

データを一本の直線で分類**できない**場合に使用するアルゴリズム

- k-近傍法
- 決定木（分類木）
- ランダムフォレスト
- サポートベクトルマシン
- ニューラルネットワーク

サポートベクトルマシン

1. クラスを分類する境界線の候補を決める。
2. 候補の境界線から各クラスに属する最も近いデータを見つける。このデータを含む直線をサポートベクトルと呼ぶ。
3. 各クラスのサポートベクトルからの距離が最も遠い境界線を選択する。



例題①

サポートベクトルマシンを使ってリバーシの勝ち負けを判定する
分類器を作ります。次ページのプログラムを実行してみましょう。

「画像データのデジタル化」の最後の問題を実行した後に実施してください。

例題①

```
import numpy as np
from sklearn import model_selection
x = np.array(df_all.drop('winner', axis=1))
y = np.array(df_all['winner'])
test_size = 0.9
data_train, data_test, label_train, label_test ¥
= model_selection.train_test_split(x, y, test_size=test_size)
from sklearn import svm
clf = svm.SVC()
clf.fit(data_train, label_train)
```

例題②

分類器でリバーシの勝敗を分類できるか、１回分の勝負で判定してみましょう。次のページのプログラムはリバーシ勝負一回分のデータを作成しています。(黒が10個で白が勝つ。)

勝ち負けの判定は合っているでしょうか？

```
df_test = make_table_reversi(10)
print(clf.predict(df_test))
```

解説

```
import numpy as np
from sklearn import model_selection
x = np.array(df_all.drop('winner', axis=1))
y = np.array(df_all['winner'])
```

x:目的変数　　y:説明変数

Index	1番目のマス	2番目のマス	…	36番目のマス	勝敗
0	白	黒	…	白	白
1	白	白	…	黒	黒
2	黒	黒	…	白	白

解説

- 「分類器の作成に使うデータ」と
  「作成した分類器の精度を検証するためのデータ」に分ける
- この割合は状況によって決める

全データのうち「作成した分類器の精度を検証するためのデータ」の割合
- 0.9×60(全データ数)=54
- 分類器の作成に使うデータは60-54=6

test_size = 0.9

```
data_train, data_test, label_train, label_test ¥
= model_selection.train_test_split(x, y, test_size=test_size)
```

解説

```
from sklearn import svm
clf = svm.SVC()
clf.fit(data_train, label_train)
```

- アルゴリズムの選択
- 分類器の作成

解説

勝敗の判定が出力される。

```
df_test = make_table_reversi(10)
print(clf.predict(df_test))
```

出力	勝ち負け
1	白の勝ち
-1	黒の勝ち

実際の勝敗が一致しないこともある。

一致しないことが多いと分類器を安心して使えない。

作成した分類器は性能評価が必要。

機械学習モデルの性能評価

- 分類器の性能評価の指標は複数ある
- 指標を評価するには混同行列を作成しておくと便利

混同行列

- 実際の結果と分類器による予測結果を表にしたもの

		実際の結果	
		正	負
予測結果	正	True Positive (TP)	False Positive (FP)
	負	False Negative (FN)	True Negative (TN)

↑状況によって解釈を変える。

機械学習モデルの性能評価

正解率：Accuracy
全データのうち正しく予測できた割合
「白が勝ち」の分類のものは予測でも「白が勝ち」になり、「白が負け」の分類のものは予測でも「白が負け」となった割合

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

		実際の結果	
		白の勝ち	白の負け
予測結果	白の勝ち	True Positive (TP)	False Positive (FP)
	白の負け	False Negative (FN)	True Negative (TN)

機械学習モデルの性能評価

適合率(精度)：Precision
正（白が勝ち）と予測したデータが実際に正しい割合
「白が勝ち」と予測分類されたもののうち、実際に「白が勝ち」だった割合

$$Precision = \frac{TP}{TP + FP}$$

		実際の結果	
		白の勝ち	白の負け
予測結果	白の勝ち	True Positive (TP)	False Positive (FP)
	白の負け	False Negative (FN)	True Negative (TN)

機械学習モデルの性能評価

再現率(検出率、感度)：Recall
実際に正であるものを正しく抽出できる割合
　実際に「白が勝ち」であるもののうち、「白が勝ち」だと予測できた割合

$$Recall = \frac{TP}{TP + FN}$$

		実際の結果	
		白の勝ち	白の負け
予測結果	白の勝ち	True Positive (TP)	False Positive (FP)
	白の負け	False Negative (FN)	True Negative (TN)

機械学習モデルの性能評価

F値：F-measure
適合率と再現率の調和平均

- 精度は高くても、検出率が低いモデルでないか
- 検出率は高くても、精度が低くなっていないか

を評価する指標

$$Fmeasure = \frac{2 * Recall * Precision}{Recall + Precision}$$

		実際の結果	
		白の勝ち	白の負け
予測結果	白の勝ち	True Positive (TP)	False Positive (FP)
	白の負け	False Negative (FN)	True Negative (TN)

問題

機械学習モデルの性能評価をしてみましょう。

1. 以下のプログラムを実行して、前ページの表を埋めましょう。

```
label_pre = clf.predict(data_test)
from sklearn.metrics import confusion_matrix
tn, fp, fn, tp= confusion_matrix(label_test, label_pre).flatten()
print(tp)
print(fp)
print(fn)
print(tn)
```

2. 正解率、適合率(精度)、再現率(検出率、感度)、F値を計算しましょう。