サポートベクターマシン
Support Vector Machine

1. 基本アイディアとハードマージン
2. 非線形分離とカーネル法
3. カーネルの種類
4. ソフトマージンとクラス分類器の性能評価
5. 分類器とパラメータの選定
   - 交差検証法
   - グリッドサーチ
6. 多クラス分類
   - one vs. one, one vs. all
   - iris, handwritten digits　クラス分類問題
7. その他
8. 付録
   - 式の証明

概要

初めに，2クラスに線形分離可能なデータを対象とする

SVMのアイディアは次である。

- マージン最大化
- サポートベクター

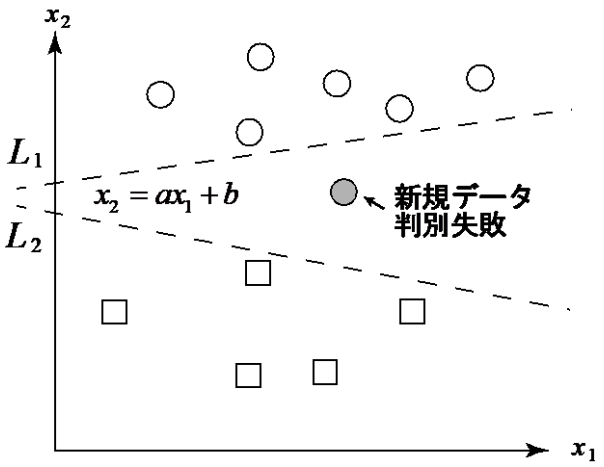
次に，2クラスの非線形分離問題に対しては次のアイディアが活用されている

- カーネルトリック

マージン最大化とサポートベクター

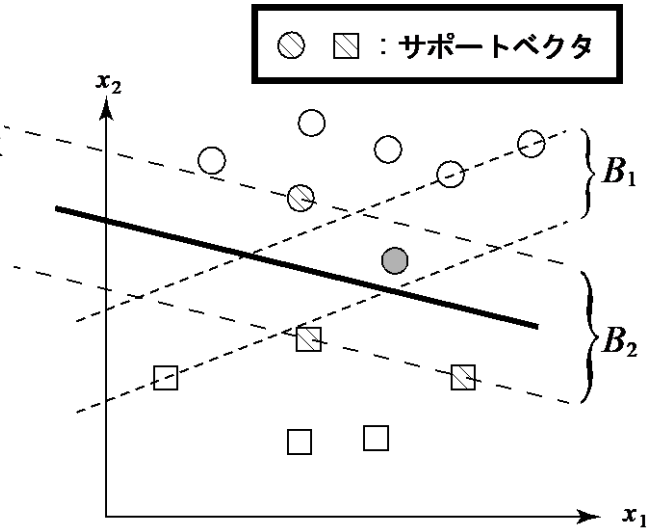
クラス分類

- ある特徴量x1とx2で表される平面に，それぞれの属性が分かっているデータを丸印と四角印に区別してプロットした。
- この二つのクラスを直線で分けて分類することがクラス分類である。
- 図では直線x2 = ax1 + bの候補としてL1とL2の2本を描いているが，実際には無限の候補がある。
- もし，L1を分離する線と決めたとすると，これは，丸印ギリギリに位置するため，新規データ（網掛けの丸印）を得た場合，この新規データは四角のクラスであると誤判別する。
- これを避ける方策としてマージン最大化がある。



マージン最大化

- 右下図に示すように，2つのクラスを分離でき，データに接する平行の境界線の候補を見出す。候補B1,B2の幅は，B2の方が余裕（マージン）が広いので，この真ん中にクラス分類線（実線）を引く。
- この実線ならば，新規データ（網掛けの丸印）を得ても，正しいクラス分類が行われる。
- B2の境界線を支えるデータ（破線の網掛け）は，この方法を支える（サポート）ベクトル（この例では位置ベクトルと考えて差し支えない）であることから，サポートベクターと称する。

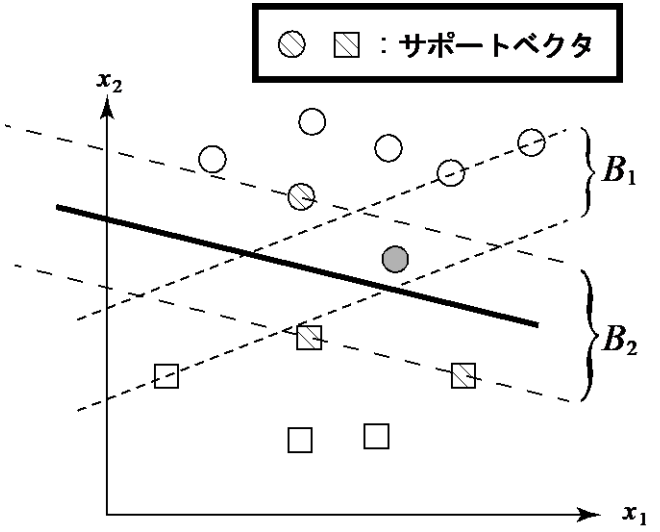


この例のように，超平面（後述）で完全分離できることをハードマージンと呼ぶ。これに対して，完全分離できない事例をソフトーマージン（後述）と呼ぶ。

超平面（hyperplane)とベクタ

超平面（hyperplane)とは

- 初等幾何学において，n次元空間の超平面とは(n-1)次元の部分空間をいう。その性質として，一つの超平面は全体空間を二つの空間に分割する。これより，次の言い方ができる。
- 3次元空間を分割するのは2次元超平面（平面）である。
- 2次元空間（平面）を分割するのは1次元超平面（直線）である。
- 1次元空間（直線）を分割するのは0次元超平面（点）である。
- 右図で，各直線は平面を二つに分けているので超平面とも称される。



ベクタ(vector)

- 日本の高校数学ではベクトルと称している。英語発音はベクタに近い。
- 方向と大きさを持った量で，位置ベクトル，速度ベクトル，ポインティングベクトルなど多数の種類がある。
- 図のように，空間（平面や3次元空間など）のある1点の位置を原点からの方向と距離，すなわち，原点からのベクトルで表した方が解析や議論で便利な場合がある。このベクトルを位置ベクトルと称する。
- この“位置”を省略して，代わりにサポート（支える）を接頭語に付けたものがサポートベクタである。

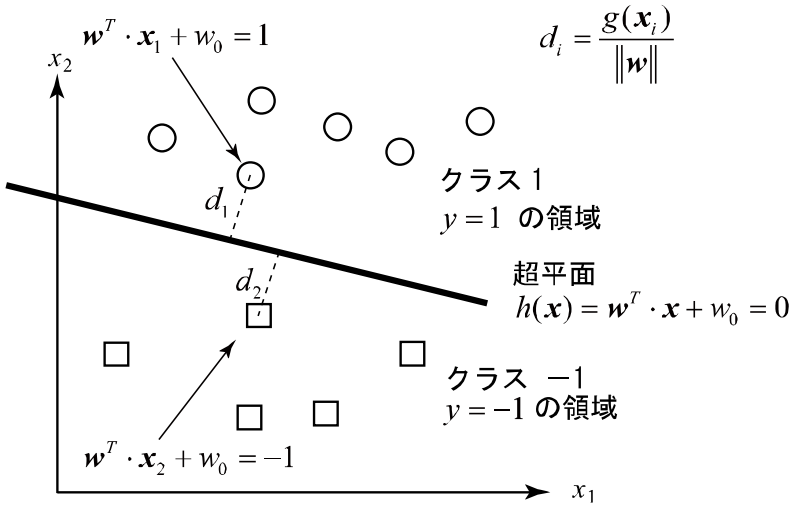
ベクターでなく
ベクタとする
英語発音に近いため

定式化

定義

- 求める超平面 $h(\boldsymbol{x}) = \boldsymbol{w}^T \cdot \boldsymbol{x} + w_0 = 0$
- ここに, $\boldsymbol{x} = [x_1, x_2]^T$, $\boldsymbol{w} = [w_1, w_2]^T$
- x_i (i=1-N)はデータ
- ここでは2次元であるが，N次元に拡張できる。
- 符号関数(sign function)を導入する。

$$\mathrm{sgn}(u) = \begin{cases} 1 & if\ u \geq 0 \\ -1 & if\ u < 0 \end{cases}$$



- 各x_iに対するクラス分類(図のようにクラス1とクラス−1)を表すy_iを次のように定める。

$$y_i = \mathrm{sgn}\left(h(\boldsymbol{x}_i)\right)$$

- 正しい分類を示した場合： $y_i\, h(\boldsymbol{x}_i) > 0$
- 誤った分類を示した場合： $y_i\, h(\boldsymbol{x}_i) < 0$

ポイント

- SMVは超平面h(x)=0から一番近いデータまでの距離（マージン，margin）を最大化すること
- 正しく分類されたデータだけを取り扱う（線形分離可能から）

$$y_i\, h(\boldsymbol{x}_i) > 0$$

- データx_iから超平面までの距離はy_iを敢えて含めて考えて

$$d_i = y_i \frac{h(\boldsymbol{x}_i)}{\|\boldsymbol{w}\|}$$

- SVMの考え方は，次の最小化を果たすx_iを求める

$$\min_i d_i$$

- ここで，**w**とw_0を同じ定数倍しても，超平面は変わらないことに注目すると，右のように条件を追加しても，得られる超平面は同じである。

$$y_i\, h(\boldsymbol{x}_i) = 1$$

定式化

最適なw, w0を求める

- 次の最適化問題に帰着する

$$\arg\max_{\boldsymbol{w}, w_0} \left\{ \frac{1}{\|\boldsymbol{w}\|} \min_i \left[y_i \, h(\boldsymbol{x}_i) \right] \right\}$$

$$\text{subject to} \quad y_i \, h(\boldsymbol{x}_i) \geq 1$$

- 先ほどの議論と，最大・最小の意味を考えると

$$\max_{\boldsymbol{w}, w_0} \left\{ \frac{1}{\|\boldsymbol{w}\|} \min_i \left[y_i \, h(\boldsymbol{x}_i) \right] \right\} \rightarrow \min_{\boldsymbol{w}, w_0} \left\{ \|\boldsymbol{w}\| \right\} \rightarrow \min_{\boldsymbol{w}, w_0} \frac{1}{2} \|\boldsymbol{w}\|^2$$

- 最後の係数1/2は2乗形式の最大最小問題を解くとき，べき乗の2を相殺する便宜上のものである。
- 結局次の最適化問題の中の二次計画法を解くことと等価である。

$$\arg\min_{\boldsymbol{w}, w_0} \frac{1}{2} \|\boldsymbol{w}\|^2$$

$$\text{subject to} \quad y_i \, h(\boldsymbol{x}_i) \geq 1$$

- これを解く数値計算法はライブラリに委ねているので説明を行わない。
- これまでの議論において，データは100%分類できる超平面が存在するという仮定を設けていた。
- すなわち，誤まった分類を許さない，という意味でハードマージンと称する。
- これを後に説明するソフトマージンと対比させて用語である。

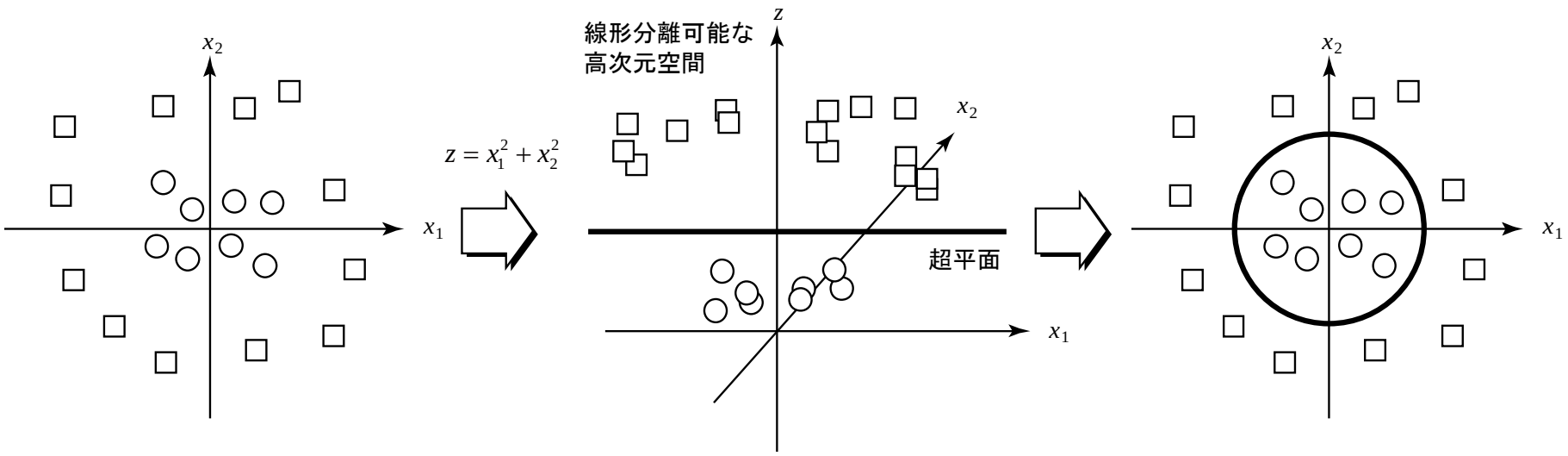
SVMの歴史

- 先ほどの例は線形分離であった
- 線形分離できないデータに対しては？
  - 線形分離できない学習データは，非線形変換で高次元空間へ写像することにより線形分離できる可能性がある。Thomas M. Cover: Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition, IEEE Transactions on Electronic Computers, Volume: EC-14, Issue: 3, Page(s): 326 – 334, 1965, DOI: 10.1109/PGEC.1965.264137
  - カーネルトリックの可能性を示した。Boser, B. E.; Guyon, I. M.; Vapnik, V. N. (1992). "A training algorithm for optimal margin classifiers". Proceedings of the fifth annual workshop on Computational learning theory – COLT '92. p. 144. doi:10.1145/130385.130401

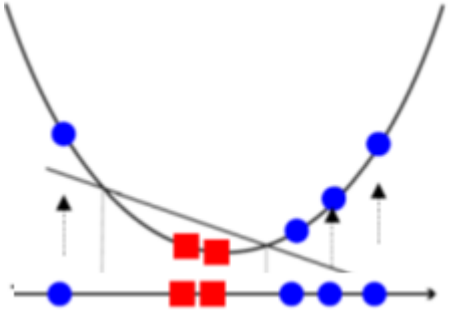
非線形分離に対するアイディア

アイディア

説明図1



説明図2



カーネル関数を用いた定式化

- R^d次元のデータを高次元Hに写す写像関数

$$\varphi_i(\boldsymbol{x}):\ x \in R^d \rightarrow H \quad , \quad (i=1,\cdots,M)$$

$$\boldsymbol{\varphi}(\boldsymbol{x}) = \left\{\varphi_0\left(\boldsymbol{x}\right)=1, \varphi_1\left(\boldsymbol{x}\right),\ \cdots,\ \varphi_M\left(\boldsymbol{x}\right)\right\}$$

これで，次の超平面が線形分離できるものとする

$$h\left(\boldsymbol{\varphi}(\boldsymbol{x})\right) = \boldsymbol{w}^T \cdot \boldsymbol{\varphi}\left(\boldsymbol{x}\right)$$

これより，wは $\{\varphi_0(\boldsymbol{x})=1, \varphi_1(\boldsymbol{x}),\ \cdots,\ \varphi_M(\boldsymbol{x})\}$ で張られるHの部分空間に属する。つまり，

$$\boldsymbol{w} = \sum_{i=1}^{q} \alpha_i \varphi\left(\boldsymbol{x}_i\right)$$

と書ける。ここで，α_i（i = 1,..., q）は適当な係数である。先の評価関数に代入して整理すると

$$\min_{\alpha_1,\ldots,\alpha_n,b} \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$$

$$\text{s.t.}\ \ y_i\left(\sum_{j=1}^{n} \alpha_j \phi(\boldsymbol{x}_j)^T \phi(\boldsymbol{x}_i) + b\right) \ \geq\ 1,\ i = 1,\ldots,n$$

ここで，次のようにカーネル関数kをおく。

$$k(\boldsymbol{x}, \boldsymbol{x}') \ =\ \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}')$$

引用：金森，他：Rで学ぶデータサイエンス（5）パターン認識，共立出版，2011，第11章。

カーネル関数を用いた定式化

利点

- φを考えることなしに，直接カーネル関数k()を考えればよい。
  - φを考えなくていいですよ，φが消えた！という観点から**カーネルトリック**と言われることがある。
- また，φを考えると高次元または無限次元の計算が必要になるが，k()はそれを回避できる。
- 例1　φを考えると6次元，kならば3次元

$$k(\mathbf{x},\mathbf{y}) = (1+\mathbf{x}^T\mathbf{y})^2 = (1+x_1\,y_1+x_2\,y_2)^2$$

$$\mathbf{x} = (x_1, x_2) \text{ and } \mathbf{y} = (y_1, y_2)$$

$$\varphi(\mathbf{x}) = \varphi(x_1, x_2) = (1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2)$$

$$k(\mathbf{x},\mathbf{y}) = (1+\mathbf{x}^T\mathbf{y})^2 = \varphi(\mathbf{x})^T\varphi(\mathbf{y})$$

- 例2　ガウシアンカーネル

$$k(\mathbf{x},\mathbf{y}) = \exp\left(-\gamma\|\mathbf{x}-\mathbf{y}\|^2\right)$$

- これをテイラー展開すると，φは無限次元となることが知られており，大変な計算量となる。

sklearn.svm.SVCのカーネルの種類

ドキュメント

- http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html （下記に転記）

提供カーネル

- kernel = ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’ or a callable
- http://scikit-learn.org/stable/modules/svm.html より

  - linear: $\langle x, x' \rangle$.
  - polynomial: $(\gamma \langle x, x' \rangle + r)^d$. d is specified by keyword `degree`, r by `coef0`.
  - rbf: $\exp(-\gamma \|x - x'\|^2)$. γ is specified by keyword `gamma`, must be greater than 0.
  - sigmoid $(\tanh(\gamma \langle x, x' \rangle + r))$, where r is specified by `coef0`.

- RBF SVM parametersは次に説明がある。
  - http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

注意

- sklearn.svm.SVCがのべているrbf (Radial Basis Function)は，ガウシアンカーネルのことである。一方，rbfは複数の種類があるので，rbf イコール ガウシアンカーネルであると誤らないように。
- see Wikipedia Radial basis function https://en.wikipedia.org/wiki/Radial_basis_function

```
class sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0,
shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None,
verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)
```

データの作成

make_classification

- http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html
- from sklearn.datasets import make_classification
- 例えば，データ数が500で特徴量が4の2値分類データを作る場合，n_sample=500，n_feature=4，n_class=2とすると，(500, 4)次元と(500)次元のnummpy.arrayオブジェクトを返します。そして，n_informative、n_redundant、n_clusters_per_class、flip_yなどでデータ構造や分類の難しさをコントロールする感じです。
- 主要なパラメータ

パラメータ名	説明
n_samples	生成するサンプルの数。
n_features	生成する特徴量の数。
n_informative	目的変数のラベルと相関が強い特徴量(Informative fearture)の数。
n_redundant	Informative featureの線形結合から作られる特徴量(Redundant fearture)の数。
n_classes	分類するクラス数。2なら2値分類問題、3以上なら多値分類問題のデータが作られる。
n_clusters_per_class	1クラスあたりのクラスタ数
class_sep	生成アルゴリズムに関係するパラメータ。後述する生成アルゴリズムにおける超立法体の頂点の距離。
shift	全ての特徴量にshiftを加算する。Noneが指定された場合、[-class_sep, class_sep]の一様乱数を加算する。
random_state	乱数を制御するパラメータ。Noneにすると毎回違うデータが生成されが、整数をシードとして渡すと毎回同じデータが生成される。乱数オブジェクトを渡すことも可能。

```
sklearn.datasets.make_classification(
    n_samples=100,n_features=20, n_informative=2, n_redundant=2,
    n_repeated=0, n_classes=2, n_clusters_per_class=2, weights=None,
    flip_y=0.01, class_sep=1.0, hypercube=True, shift=0.0, scale=1.0,
    shuffle=True, random_state=None)
```

例題：3種のカーネル

- データの生成（2クラス，円状，月状）
- カーネル
  - それぞれに，linear（線形カーネル，内積），rbf（ガウシアンカーネル），poly（多項式カーネル）を適用する。

rbfカーネル: radial basis function kernel

SVM_HardMargine

```
X, y = make_classification( n_samples=100, n_features=2, n_informative=2, n_redundant=0,
                            n_classes=2, n_clusters_per_class=1,
                            class_sep=2.0, # 大きいほどクラス分離の距離が大きい
                            shift=None,
                            random_state=5) # 整数を与えると乱数の再現性がある
plt.scatter(X[:,0], X[:,1], c=y, cmap=cm_bright, edgecolors='k')
plt.colorbar()
if FLAG_fig: plt.savefig('fig_SVM_HM_LN_01.png')
plt.show()
```

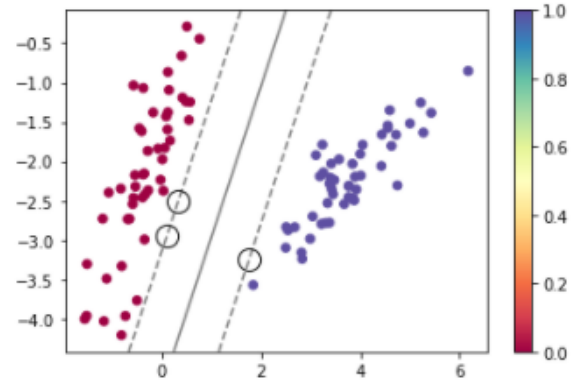
moons http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html
circles http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_circles.html

例題：3種のカーネル

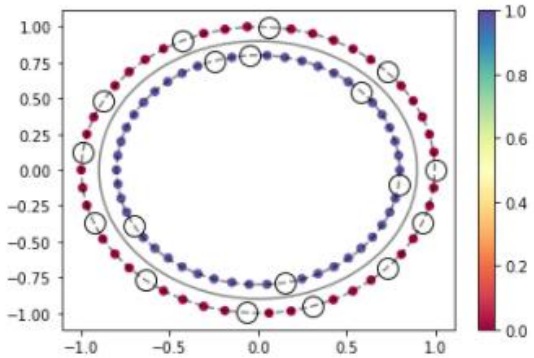
シミュレーション結果

- 三つの結果とも，データ全てに対して分離が正しく行われている，すなわち，ハードマージンの例であることがわかる。
- サポートベクタは白抜き丸で表現されている。
- 超平面は実線，それから距離±1を破線で示している。
- クラス0を赤，1を青で表現している。
- 1番目は，直線，すなわち線形分離である
- 2番目と3番目は，高次元空間での線形分離が行えて，もとの空間では超平面は曲線で表されている。

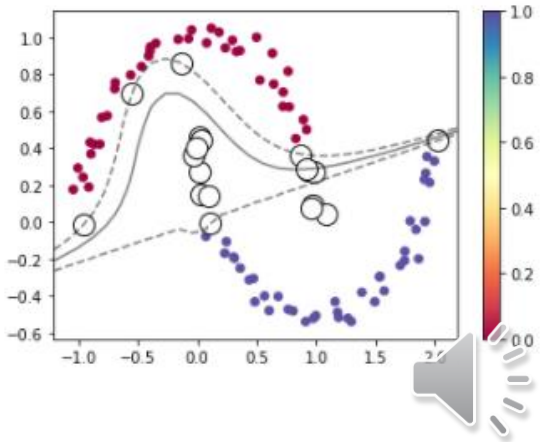
2クラス(0, 1)のデータ生成



Circles, 円状データ



Moons, 月状データ



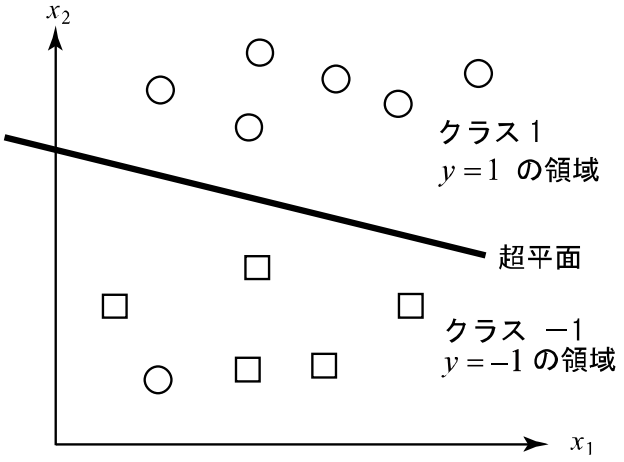
サポートベクターは白丸で表現されている

ソフトマージンの考え方

- **ハードマージン：100%分離できるものを対象とする**
- **ソフトマージン：他のクラスへの混入を認めるが，その代わりにペナルティを科す**
  - 図では，丸印のデータが一つクラス−1に混入している。
  - これを無理やり分離する超平面（曲線にならざるを得ない）を見つけるのではなく，この混入を認めるという考え方
- **定式化**
  - 上記の考え方は，スラック変数ξ_iをペナルティ
  - Cをマージンを破るコストの重み
    （ハードマージンでは$C = \infty$）
  - 次の最適化問題を解く

$$\arg\min_{\boldsymbol{w}, w_0} \left\{ \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N} \xi_i \right\}$$

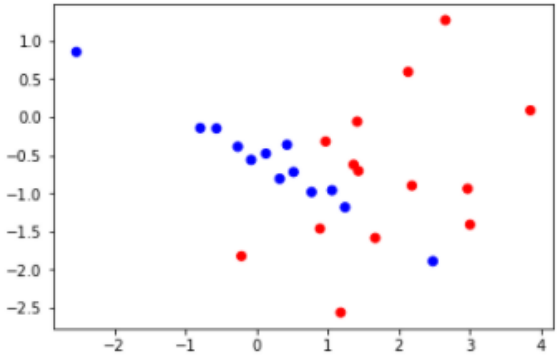
$$\text{subject to} \quad \xi_i \geq 1 - y_i\, h(\boldsymbol{x}_i), \quad \xi_i \geq 0 \quad (i = 1, \cdots, N)$$

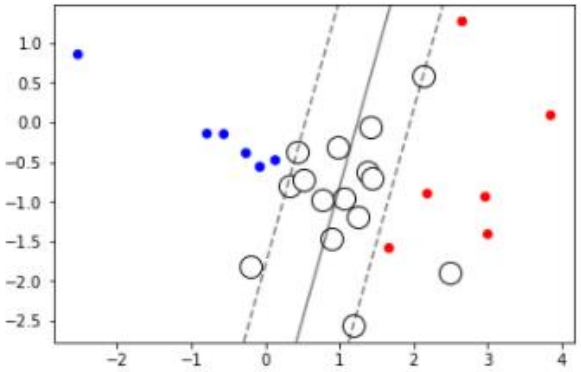


$$\begin{cases} \xi_i = 0 & \text{マージン内で正しく分類} \\ 0 < \xi_i \leq 1 & \text{マージン境界を超えるがほぼ正しく分類} \\ \xi_i > 1 & \text{マージン境界を越えて誤分類} \end{cases}$$

例題：ソフトマージンの例

- **線形分離できない例を対象として，次を見る**
  - クラス分類器の性能評価





トレーニングデータに対する評価

```
In [ ]:
1  y_train_est = clf.predict(X_train)
2  print("推定値: %s" % y_train_est)
3  print("真値   : %s" % y_train)
```

```
推定値: [0 0 0 1 1 1 1 1 0 1 1 1 1 0 0 1 0 0 0 1 0 1 1 0 1 0 1]
真値   : [1 0 0 1 0 1 1 1 0 1 0 1 1 0 0 1 0 0 0 0 0 1 1 1 1 0 1]
```

上記の図とy_train_est, y_trainの内容を見て，誤りは5つ，内訳は青が3つ，赤が2つ誤認識

SVM_SoftMargine

```
X, y = make_classification( n_samples=100, n_features=2, n_informative=2, n_redundant=0,
                            n_classes=2, n_clusters_per_class=1,
                            class_sep=0.4, # 大きいほどクラス分離の距離が大きい
                            shift=None,
                            random_state=5) # 整数を与えると乱数の再現性がある
```

例題：ソフトマージンの例

トレーニングデータに対する性能評価

```
1 print('トレーニングデータ　正解率', clf.score(X_train, y_train))
```

トレーニングデータ　正解率 0.814814814815

トレーニングデータに対する評価，下記のaccuracy と上記のclf.score()は同じ計算

```
1 print('confusion = ¥n %s' % confusion_matrix(y_train, y_train_est)) # 混同行列
2 print('accuracy = %f ' % accuracy_score(y_train, y_train_est))  # 正答率
3 print('precision = %f ' % precision_score(y_train, y_train_est)) # 適合率
4 print('recall = %f ' % recall_score(y_train, y_train_est)) # 再現率
5 print('F-measure = %f' % f1_score(y_train, y_train_est)) # F-値
```

```
confusion =
 [[10  3]
 [ 2 12]]
accuracy = 0.814815
precision = 0.800000
recall = 0.857143
F-measure = 0.827586
```

以前に述べた，識別器の性能評価を参照
この性能が良いか，悪いのか？
この数字がNotebookと異なる場合には，
Notebookを優先

```
1 print( classification_report(y_train, y_train_est)) # 正答率 (accuracy) が無いことに注意
```

```
             precision    recall  f1-score   support

          0       0.83      0.77      0.80        13
          1       0.80      0.86      0.83        14

avg / total       0.82      0.81      0.81        27
```

上記の結果の見方，"0", "1"はクラス名で，それぞれの立場での 評価指標が示されている。 上記の一つ一つの評価では，"1"の立場で見ていることと同じである。また，F-measureとf1-scoreは同じ量を示す。

例題：ソフトマージンの例

テストデータに対する性能評価

```
1 y_test_est = clf.predict(X_test)
2 print("推定値: %s" % y_test_est)
3 print("真値  : %s" % y_test)
```

```
推定値: [0 1 1]
真値  : [0 0 1]
```

データからクラスを推定する識別器，正解率が約67%
この性能が良いか，悪いのか？

```
1 print('テストデータ          正解率', clf.score(X_test, y_test))
2 print('confusion = ¥n %s' % confusion_matrix(y_test, y_test_est))
3 print('accuracy = %f ' % accuracy_score(y_test, y_test_est))
4 print('precision = %f ' % precision_score(y_test, y_test_est))
5 print('recall = %f ' % recall_score(y_test, y_test_est))
6 print('F-measure = %f' % f1_score(y_test, y_test_est))
```

```
テストデータ          正解率 0.666666666667
confusion =
 [[1 1]
 [0 1]]
accuracy = 0.666667
precision = 0.500000
recall = 1.000000
F-measure = 0.666667
```

```
1 print( classification_report(y_test, y_test_est))
```

```
             precision    recall  f1-score   support

          0       1.00      0.50      0.67         2
          1       0.50      1.00      0.67         1

avg / total       0.83      0.67      0.67         3
```

例題：ソフトマージンの例

Cの影響

- 小さくすると，混入しやすくなる。
- C→∞ならば，ハードマージン
- Cをだんだんと小さくして，0.1にするとどうなるか，各自で確かめてみよう。

概要

パターン認識の目的

- クラス分類器を求めること
- 限られたデータから，どのようにトレーニングデータとテストデータを振り分ける
- どのカーネルが良いか？　それに付随するパラメータの値をどのように決めるか？

方針

- 交差検証法とカーネルの種類，およびパラメータを段階的に変えた総当たり戦

交差検証法

model_selection.cross_val_scoreの使用例

- 下記に示す。
- ただし，単にデータを5分割したクラス分類器を求めただけである。
- どの分割されたデータが良いのか？
- また，パラメータは？

SVM_CrossValidation

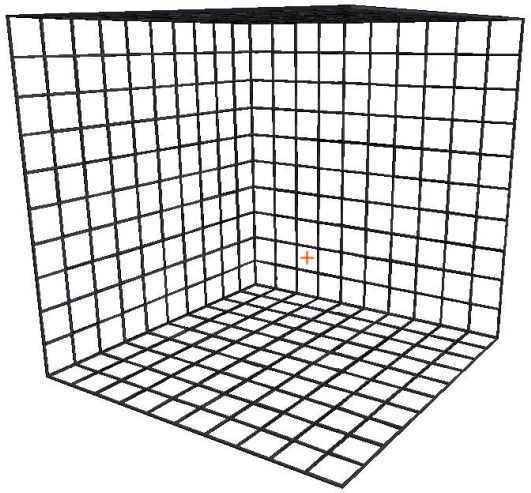
```
from sklearn.model_selection import cross_val_score
X, y = make_classification(…)
clf = svm.SVC(kernel='linear', C=1)
scores = cross_val_score(clf, X, y, cv=5, scoring='accuracy')
print(scores)
print("Accuracy: %0.2f (+/- %0.2f) for 95 %% confidence interval" % (scores.mean(),
scores.std() * 2))

[ 0.8625  0.95    0.9375  0.8875  0.9125]
Accuracy: 0.91 (+/- 0.06) for 95 % confidence interval
```

グリッドサーチ

考え方

- 良いクラス分類器を見出すには，カーネルの種類と　それに付随するパラメータの値を決定しなければならない。
- 考え方の一つ　⇒　パラメータの総当たり戦
  - 例：a = { a1, a2, a3 },　b = {b1, b2},　c = {c1, c2, c3, c4}、必ずa, b, cから一つずつ選ぶとすれば、
    3 × 2 × 4 = 24 通り　（総当たり）
- パラメータの総当たり戦で行うことは，あたかも格子状（グリッド，Grid）を隈なく探索（サーチ,search）することからグリッドサーチ（Grid Search）と称している。

SVM_GridSearch



グリッドサーチ

使い方

SVM_GridSearch

```
# グリッドサーチ用パラメータを設定
parameters = {'kernel':('linear', 'rbf'), 'C':[0.1, 1.0, 10.0], 'gamma':[0.01, 0.1, 1.0, 10.0]}
svc = svm.SVC()
```

総当たり

```
# グリッドサーチを実行
clf = GridSearchCV(svc, parameters, cv=5)
clf.fit(X, y)
```

```
GridSearchCV(cv=5, error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'kernel': ('linear', 'rbf'), 'C': [0.1, 1.0, 10.0], 'gamma': [0.01, 0.1, 1.0, 10.0]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

Grid serachは計算時間がかかるので，開発途中では，パラメータ数を少なく。ColaboはJupyterより早い例が多い。

- グリッドサーチを行い，
- Accuracy （正答率，ディフォルト値）に基づき，
  - scoring='param' を用いて，他の指標に変えられる。paramの種類は次を参照
  - http://scikit-learn.org/stable/modules/model_evaluation.html
- ベストパラメータを求めた結果を示している。

グリッドサーチ

- ベストパラメータと性能評価を示す

```
# 最適パラメータを表示
print(clf.best_score_)
print(clf.best_params_)
```

```
0.8775
{'C': 1.0, 'gamma': 1.0, 'kernel': 'rbf'}
```

カーネルはrbf，そのパラメータ値が示された

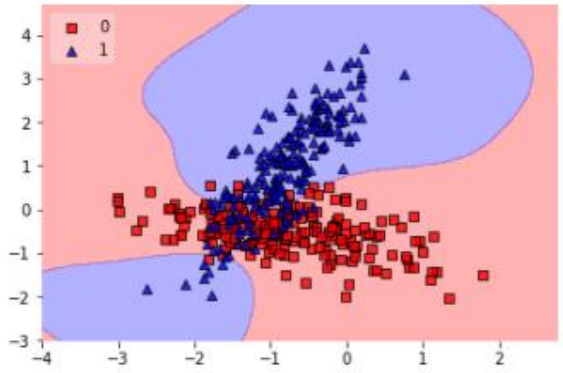
```
# 最適パラメータによる識別器を全データに適用，テストデータは後述
bst_clf = clf.best_estimator_
eval = bst_clf.predict(X)plot_decision_regions(X,y, clf=bst_clf, res=0.02, legend=2)
print(accuracy_score(y, eval))
print(classification_report(y, eval))
```

```
0.8825
             precision    recall  f1-score   support

          0       0.85      0.94      0.89       200
          1       0.93      0.83      0.88       200

avg / total       0.89      0.88      0.88       400
```

上記のクラス分類器を用いた性能評価

```
plot_decision_regions(X,y, clf=bst_clf, res=0.02, legend=2)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x22686dd2588>
```



プロット図を簡単に描ける便利なパッケージ

```
from mlxtend.plotting import plot_decision_regions
```

グリッドサーチ

注意

- ユーザが指定したカーネルと，離散値のパラメータの組合せ，かつ，与えられたデータの中で，正答率が最も高い組合せを最善（best）な組合せとして示しているだけで，最適と言っていないことに注意すること。
- すなわち，他のカーネル，他のパラメータ値やデータを与えるとこの最善は変わるかもしれないことを念頭に置いて，得られたクラス分類器を使用せざるを得ない。

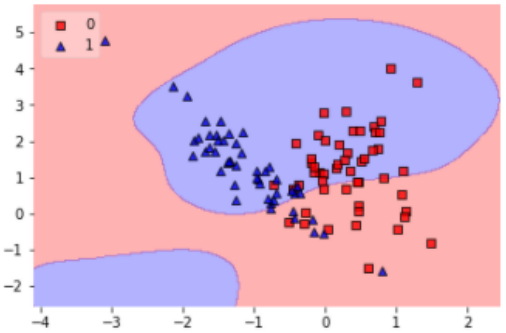
- 実際，同じ属性のデータを新たに発生させる。（これが先のトレーニングデータと全く同じ属性か否かは議論の余地がある）
- これに求めたクラス分類器を適用すると，先のトレーニングデータの場合ほどの正答率は得られない
- グラフを見ても，数値では正答率は先の例よりも低下しているが，これが先の分類度より，本当に，低いと断定することは難しい。
- この分類を突き詰めて考察するには，やはり，データの属性の物理的・科学的観点からの考察，かつ，データとクラスとの因果性（またはメカニズム）を追求するのが王道と考える。
- この属性や因果性に考察が行い難い場合に，パターン認識のアルゴリズムを適用して，その結果を考察せざるを得ないのであろう。

```
X_test, y_test = make_classification(n_features=2, n_redundant=0, n_informative=2,
                                     n_clusters_per_class=1, n_samples=100, n_classes=2,
                                     random_state=3, #元のデータと異なる乱数を発生
                                     class_sep=0.7,  #分離性
                                     shift=None)
```

```
y_test_est = bst_clf.predict(X_test)
print(accuracy_score(y_test, y_test_est))
print(classification_report(y_test, y_test_est))
plot_decision_regions(X_test,y_test, clf=bst_clf, res=0.02, legend=2)
```

```
0.58
             precision    recall  f1-score   support

          0       0.71      0.33      0.45        52
          1       0.54      0.85      0.66        48

avg / total       0.63      0.58      0.55       100
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x22686df74a8>
```



値は、各自で異なるので、各自のNotebookの内容を考察すること

多クラス分類の基本的アイディア

クラス分類問題で，クラスの数が3以上の場合を多クラス（マルチクラス; multi-class)と呼んで区別することがある。クラス数がkの多クラス問題（k>2）を，2クラス分類器で解く場合には，一対他分類器や一対一分類器がよく利用される.

一対他分類器（one-versus-rest classifier), one-vs-all ともいう

- i=1,…,k−1の各クラス i それぞれについて，クラス i なら 1を，その他のクラスなら 0 に分類する2値分類器を求める。クラス k は，k−1個の分類器が全て 0 を出力すれば，クラスkと分かる，という考え方を導入している。
- 複数の分類器が 1 を出力したとき，最終的な解をどれにするか決定できない場合があるので，推定確率を最も高くするような工夫を行う。

一対一分類器（one-versus-one classifier) one-vs-oneともいう

- k個のクラスから2つのクラス(i, j) (i≠j)の組合せk(k−1)/2個について，クラス i と j とを2クラス分類を考える（k(k-1)/2個の2クラス分類問題）。最終的なクラスは多数決によって決める。

参考

- http://scikit-learn.org/stable/modules/multiclass.html

例：Iris

多変量解析で説明したデータと同じ（3クラス）

SVM_Multiclass_Iris

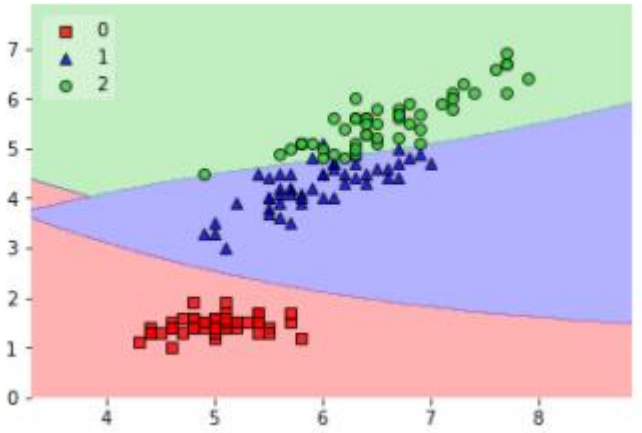
iris変量の内，花びらの長さと幅だけを取り出すために，
iris.dataの2番目と3番目の要素だけを抽出してXの配列(150 x 2)に格納
ラベル(花びらの種類)を y(150 x 1)に格納

```
# Loading some example data
iris = datasets.load_iris()
X = iris.data[:, [0, 2]]
y = iris.target
```

グリッドサーチで探索した結果

```
# 最良パラメータを表示
print(clf.best_score_)
print(clf.best_params_)
```

```
0.966666666667
{'C': 0.1, 'decision_function_shape': 'ovo', 'gamma': 10.0, 'kernel': 'poly'}
```



上記のグラフを見て，分類する超平面が複雑すぎると感じても，次に新しいデータが取得したとき，視覚的にどのクラスに入るかが，直ちにわかるという利点を有する。

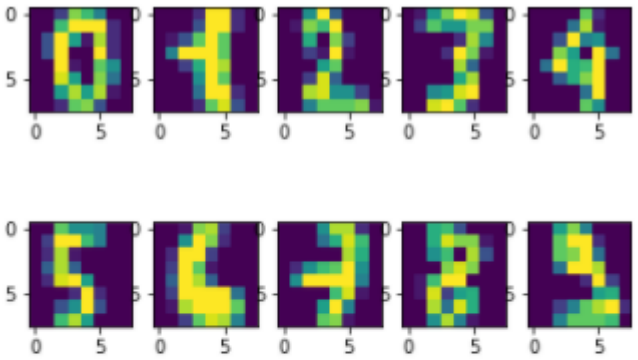
例：手書き数字

SVM_Multiclass_Digits

データの内容

handwritten digits 0-9
http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html
Each datapoint is a 8x8 image of a digit.
Classes 10
Samples per class ~180
Samples total 1797
Dimensionality 64
Features integers 0-16

クラスの数が10



```
(8, 8)
[[ 0.   0.   0.   2.  13.   0.   0.   0.]
 [ 0.   0.   0.   8.  15.   0.   0.   0.]
 [ 0.   0.   5.  16.   5.   2.   0.   0.]
 [ 0.   0.  15.  12.   1.  16.   4.   0.]
 [ 0.   4.  16.   2.   9.  16.   8.   0.]
 [ 0.   0.  10.  14.  16.  16.   4.   0.]
 [ 0.   0.   0.   0.  13.   8.   0.   0.]
 [ 0.   0.   0.   0.  13.   6.   0.   0.]]
```

データの分割とパラメータ候補

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=0)
# グリッドサーチ用パラメータを設定
parameters = {'kernel':('linear', 'rbf'), 'C':[0.1, 1.0, 10.0],
              'gamma':[0.01, 0.1, 1.0, 10.0], 'decision_function_shape':('ovo', 'ovr')}
```

例：手書き数字

実行結果

```
# グリッドサーチを実行
clf = GridSearchCV(svm, parameters, cv=5)
clf.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'kernel': ('linear', 'rbf'), 'C': [0.1, 1.0, 10.0], 'gamma': [0.01, 0.1, 1.0, 10.0], 'decisio
n_function_shape': ('ovo', 'ovr')},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

```
# 最適パラメータを表示
print(clf.best_score_)
print(clf.best_params_)
```

```
0.976485148515
{'C': 0.1, 'decision_function_shape': 'ovo', 'gamma': 0.01, 'kernel': 'linear'}
```

例：手書き数字

実行結果

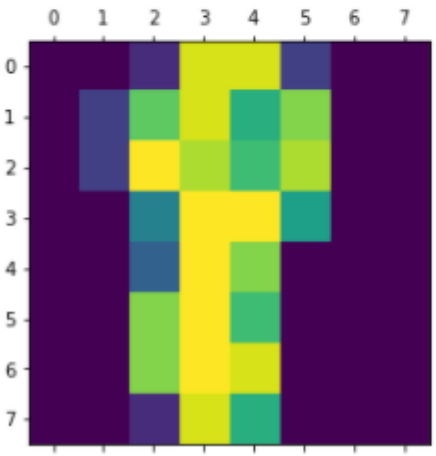
```
y_test_pred = clf.predict(X_test)
print("Accuracy Score = %f ¥n" % accuracy_score(y_test, y_test_pred))
print("Classification report for classifier ¥n %s"  % classification_report(y_test, y_test_pred))
```

```
Accuracy Score = 0.977778

Classification report for classifier
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        11
          1       0.91      1.00      0.95        20
          2       1.00      1.00      1.00        16
          3       1.00      1.00      1.00        10
          4       0.91      1.00      0.95        10
          5       0.95      1.00      0.98        21
          6       1.00      0.96      0.98        25
          7       1.00      0.95      0.97        20
          8       1.00      0.96      0.98        23
          9       1.00      0.96      0.98        24

avg / total       0.98      0.98      0.98       180
```

idが11番目のデータ，実際と推定の結果は？
時間がかかるので，各自で予習・復習内で

推定

```
# idで指定したデータから推定した数字を見る
id=11
dat = np.array([X_test[id]])
print("Estimated Number is %d " % clf.predict(dat))
print("Real      Number is %d " % y_test[id])

plt.matshow(X_test[id].reshape(8,8))
plt.show()
```



クラス分類の実践的手順

1. データの前処理(標準化，正規化)を行う
2. データの分類，必要ないものを除外するなど
3. クラス分類実行
4. パラメータやデータを含めた評価，この評価が思わしくない場合には1．に戻り再検討する。