サポートベクターマシン

サポートベクターマシン

- サポートベクターマシン（support vector machine, SVM）
  - ２値分類のための識別モデル（確率モデルではない）
  - 1990年代に開発され、様々なタスクで高い精度を達成

  - 最大マージン分類器（maximal margin classifier）
    - マージンを最大化することによる高精度分類
  - サポートベクター分類器（support vector classifier）
    - 線形分離可能でない場合でも適用可能
  - サポートベクターマシン
    - カーネルによる非線形分類が可能

最大マージン分類器

- 超平面（hyperplane）
  - p 次元空間中の超平面：$p-1$ 次元の平坦な部分空間
    - 2次元空間中の超平面：直線
    - 3次元空間中の超平面：平面
  - 超平面を表す式
    - 2次元空間の場合：$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$
    - p 次元空間の場合：$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0$
  - 空間は超平面によって2つに分けられる

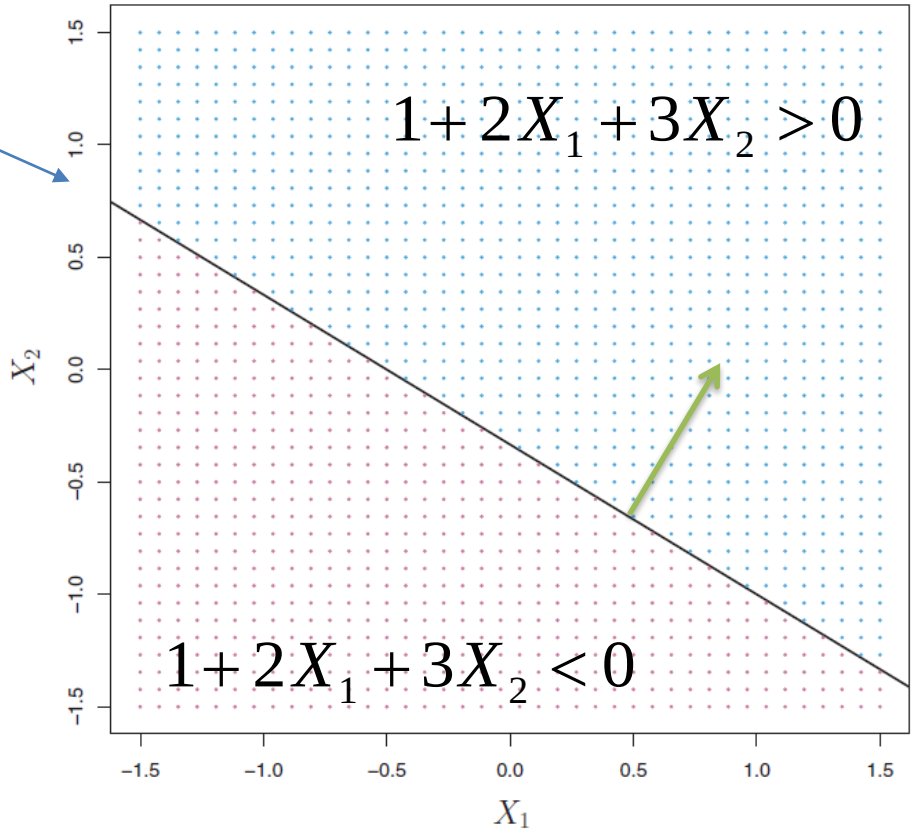
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p > 0$$

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p < 0$$

最大マージン分類器

- 例）2次元空間中の超平面

$1+2X_1+3X_2=0$



最大マージン分類器

- 学習データ
  - $n \times p$ データ行列 **x**

$$x_1 = \begin{pmatrix} x_{11} \\ : \\ x_{1p} \end{pmatrix}, \ldots, x_n = \begin{pmatrix} x_{n1} \\ : \\ x_{np} \end{pmatrix}$$

$$y_1, \ldots, y_n \in \{-1, 1\}$$

クラスを表すラベルは 1 と -1

- テスト事例

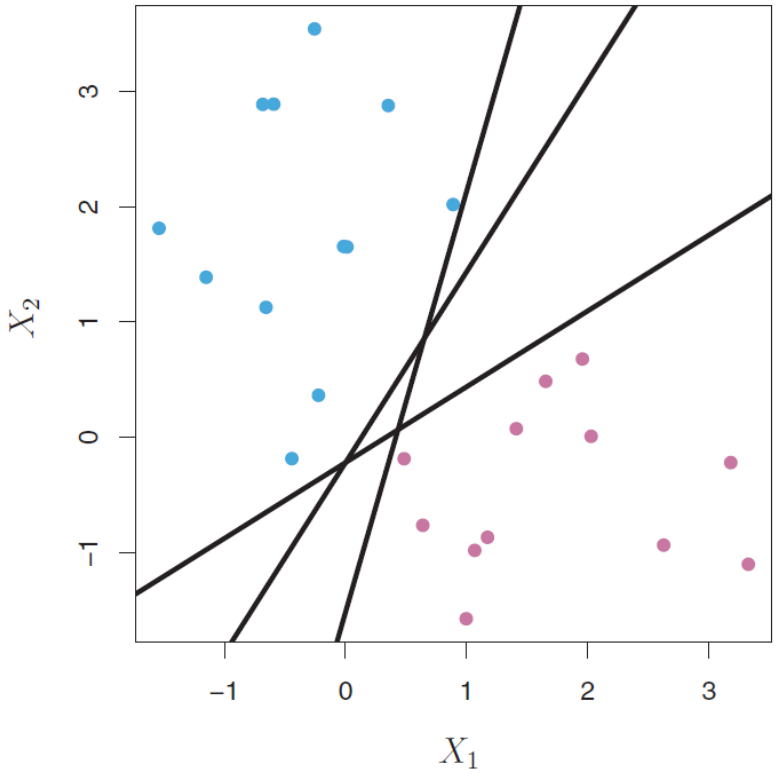
$$x^* = \begin{pmatrix} x_1^* \\ : \\ x_p^* \end{pmatrix}$$

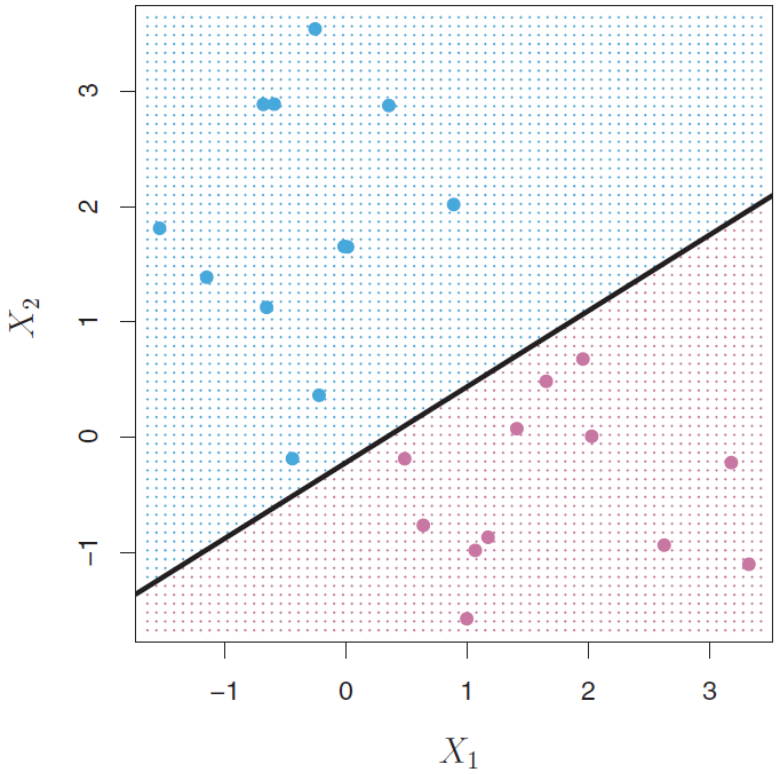
- 目標

  テスト事例を正しく分類する分類器を学習データから作りたい

最大マージン分類器

- 分離超平面（separating hyperplane）
  - 学習事例をすべて正しく分類する超平面





そのような超平面は多数存在

最大マージン分類器

- 分離超平面の性質

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} > 0 \ \text{if} \ y_i = 1$$

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} < 0 \ \text{if} \ y_i = -1$$

つまり、すべての i に対して $y_i\left(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip}\right) > 0$

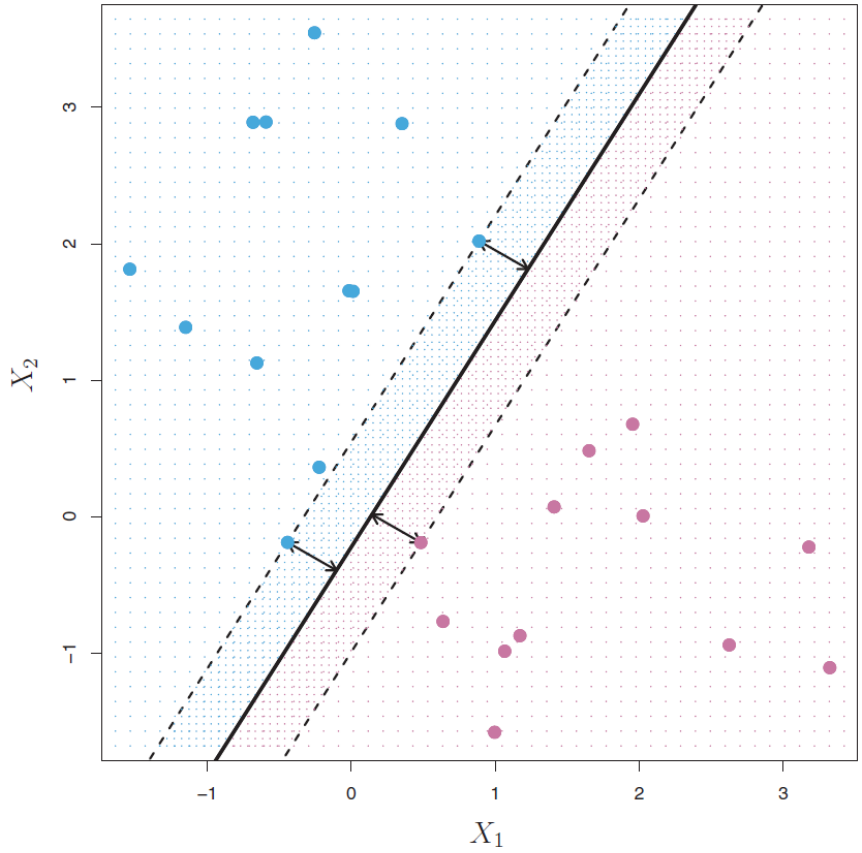
- テスト事例の分類

$$f\left(x^*\right) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + ... + \beta_p x_p^*$$

の正負に従って判定

最大マージン分類器

- 最大マージン超平面
  - 分離超平面は無数に存在
    - どのような超平面が望ましい？
  - マージン（margin）
    - 最も近い学習事例までの距離
  - 最大マージン超平面
    - マージンが最大になる超平面
  - サポートベクター
    - 超平面に最も近い事例
      - 右の例では3つ（青2、赤1）
    - 超平面を「サポート」している



最大マージン分類器

- 最大マージン分類器の構築
  - 学習事例: $x_1,\ldots,x_n$
  - クラスラベル: $y_1,\ldots,y_n$

最大マージン超平面
$\beta_0,\beta_1,\ldots,\beta_p$ を求める

- 最適化問題

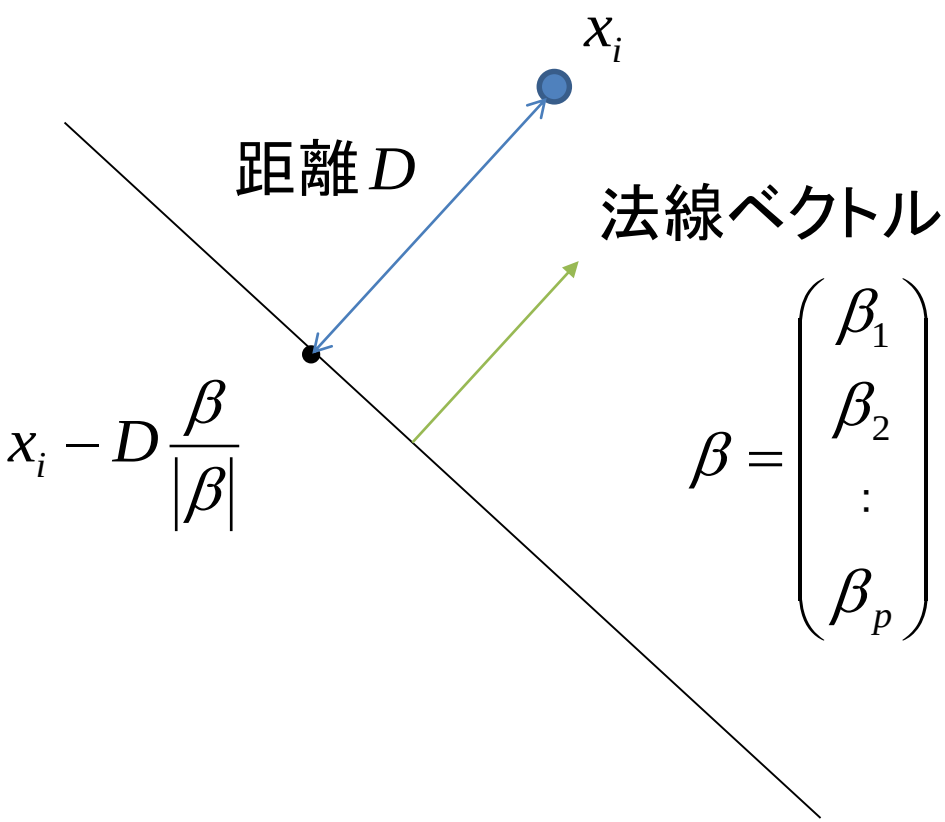
$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\text{maximize}}\, M$$

$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1$$

$$y_i\left(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}\right) \geq M \quad \forall i = 1, \ldots, n$$

最大マージン分類器

- M とマージン
  （$y_i = 1$ の場合）



分離超平面 $\beta_0 + \beta^T X = 0$

左図の黒点は超平面上にあるので

$$\beta_0 + \beta^T \left(x_i - D\frac{\beta}{|\beta|} \right) = 0$$

$$\beta_0 + \beta^T x_i - D|\beta| = 0$$

$$\frac{1}{|\beta|}\left(\beta_0 + \beta^T x_i\right) = D$$

$|\beta| = 1$ なら $\beta_0 + \beta^T x_i = D$

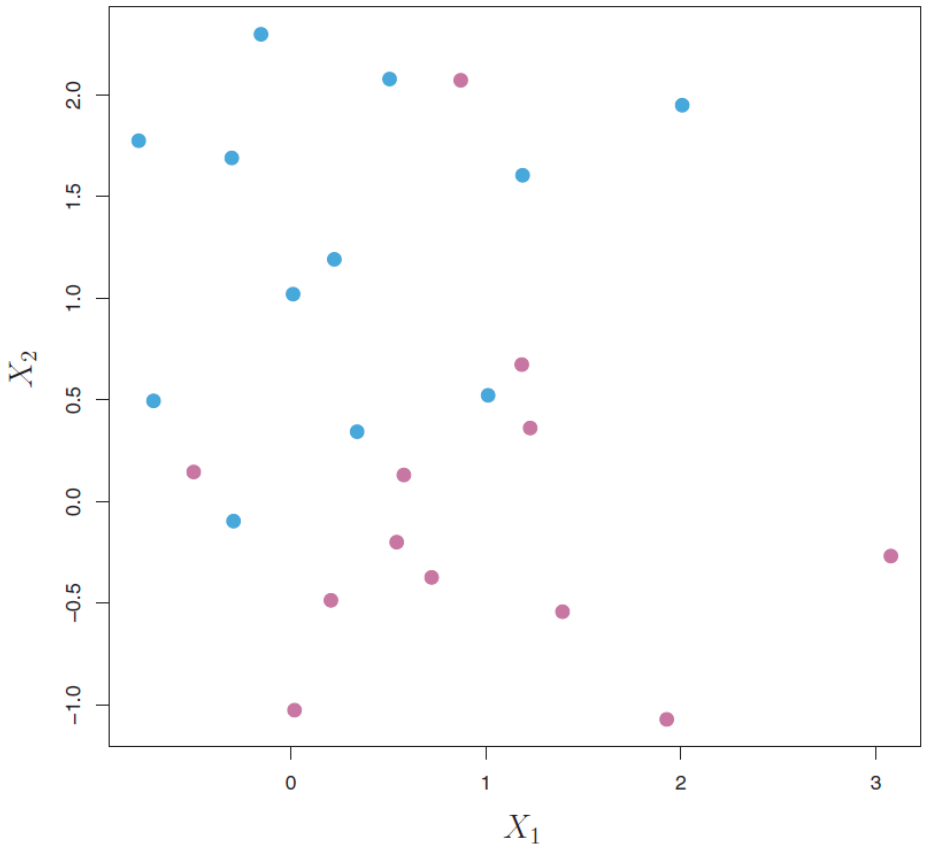
つまり $\sum_{j=1}^{p} \beta_j^2 = 1$ であれば

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}$$

は超平面までの距離を表す

最大マージン分類器

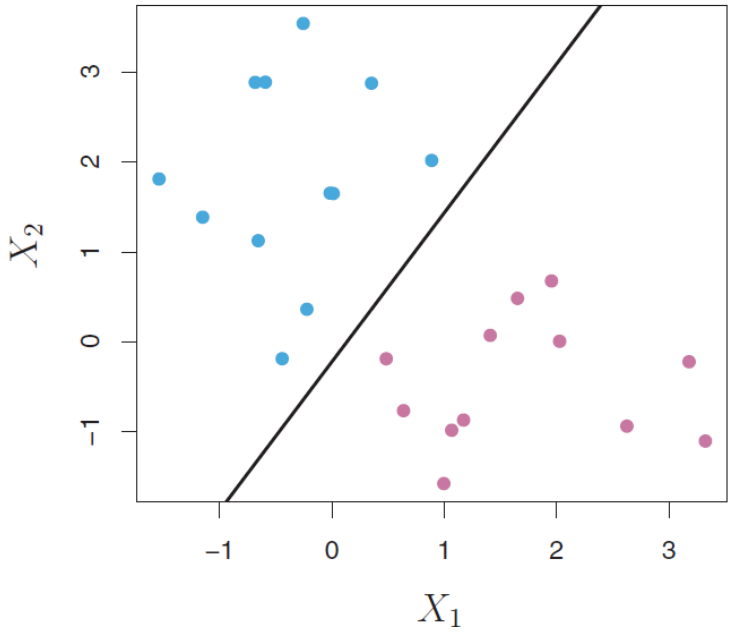
- 分離超平面が存在しない場合

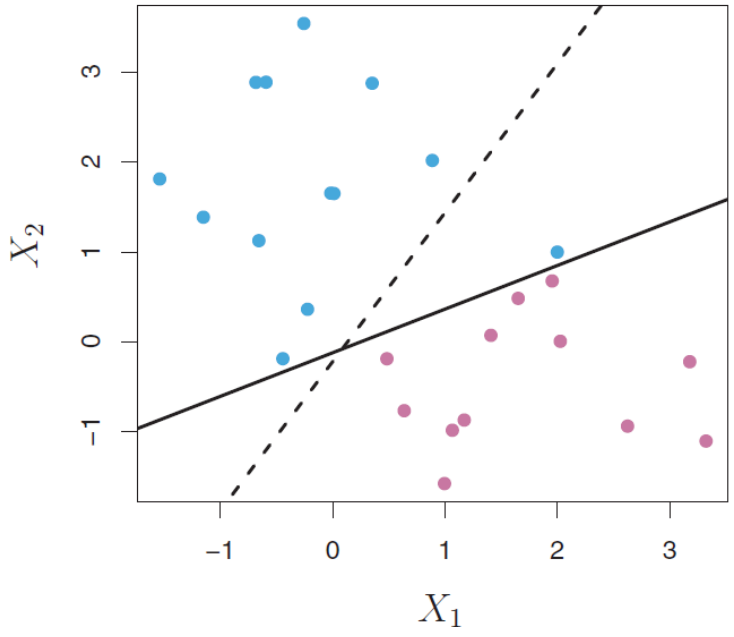


→ ソフトマージンによる
サポートベクター分類器

サポートベクター分類器

- 最大マージン分類器の問題点
  - 分離超平面が存在する場合でも、学習データの変化に敏感すぎるという問題あり

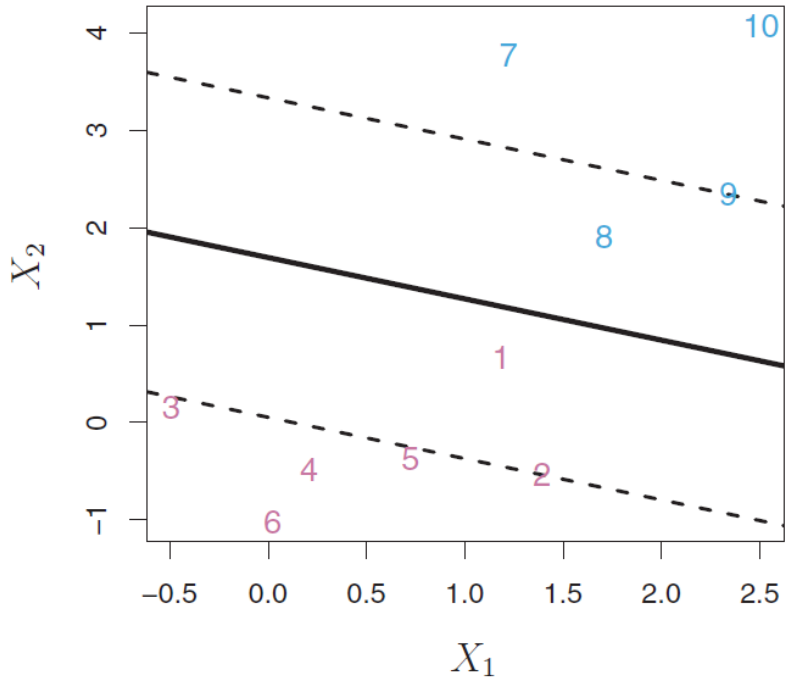




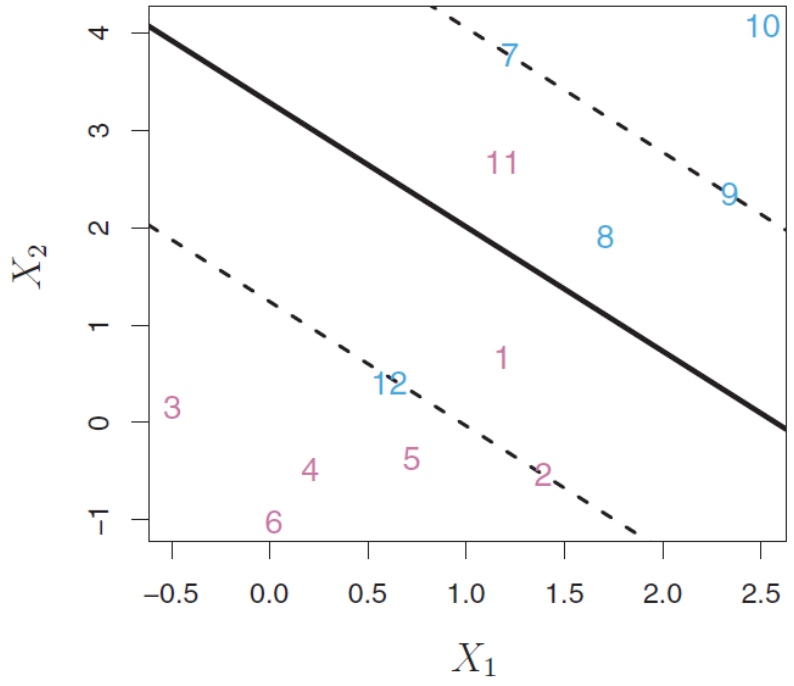
学習事例（青）がひとつ追加されただけで超平面が大きく変化

サポートベクター分類器

- サポートベクター分類器
  - マージンが小さい事例や正しく判別されない事例の存在をある程度許すことにより、ロバストな分類器を構築



事例1と事例8はマージンの内側に入ってしまっている



事例11と事例12はクラスの判別も間違う

サポートベクター分類器

- 最適化問題

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\varepsilon_1,\ldots,\varepsilon_n}{\text{maximize}} \quad M$$

$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1$$

$$y_i\left(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}\right) \geq M\left(1-\varepsilon_i\right) \quad \forall i = 1, \ldots, n$$

$$\varepsilon_i \geq 0, \quad \sum_{i=1}^{n} \varepsilon_i \leq C$$

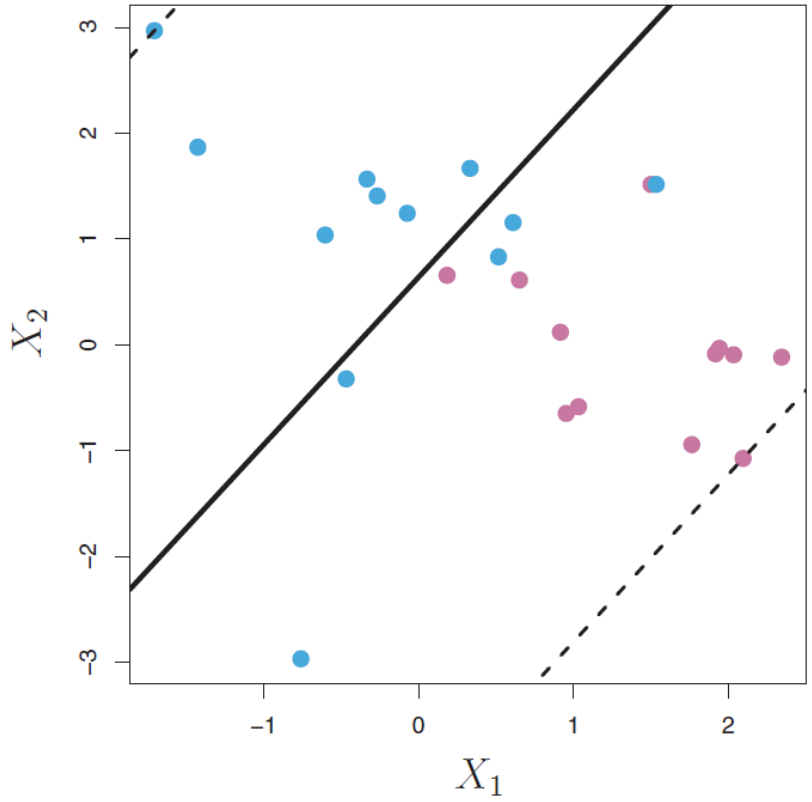
スラック変数（slack variable）
- $\varepsilon_i = 0$: 違反なし
- $\varepsilon_i > 0$: マージンの制約に違反
- $\varepsilon_i > 1$: 超平面の反対側

・違反を合計でどれだけ許すか
・交差検証などでチューニング

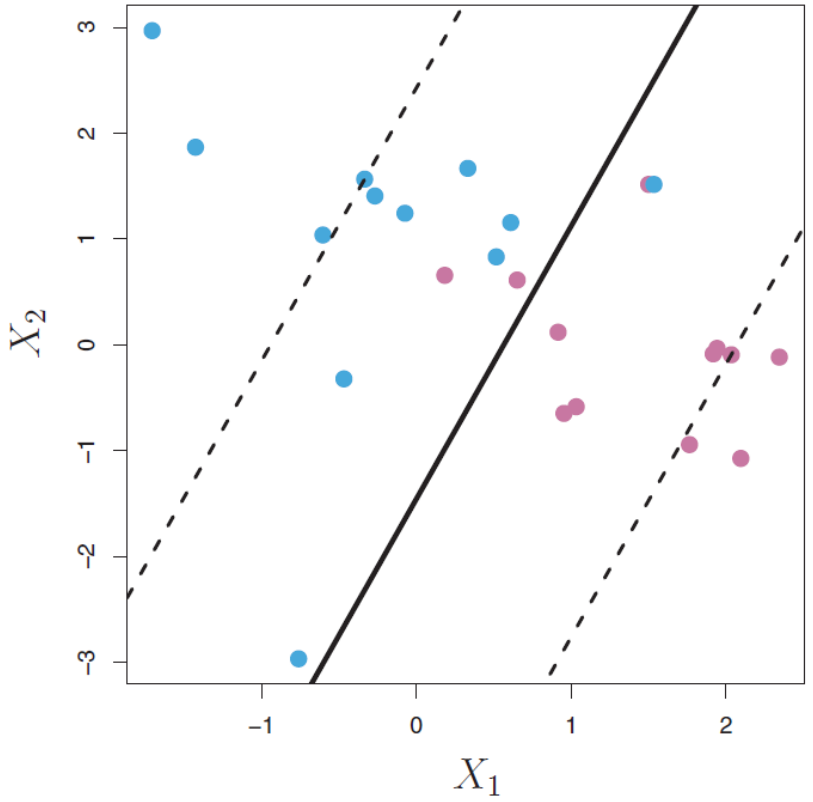
サポートベクター分類器

- C の値による違い

C が大きい場合



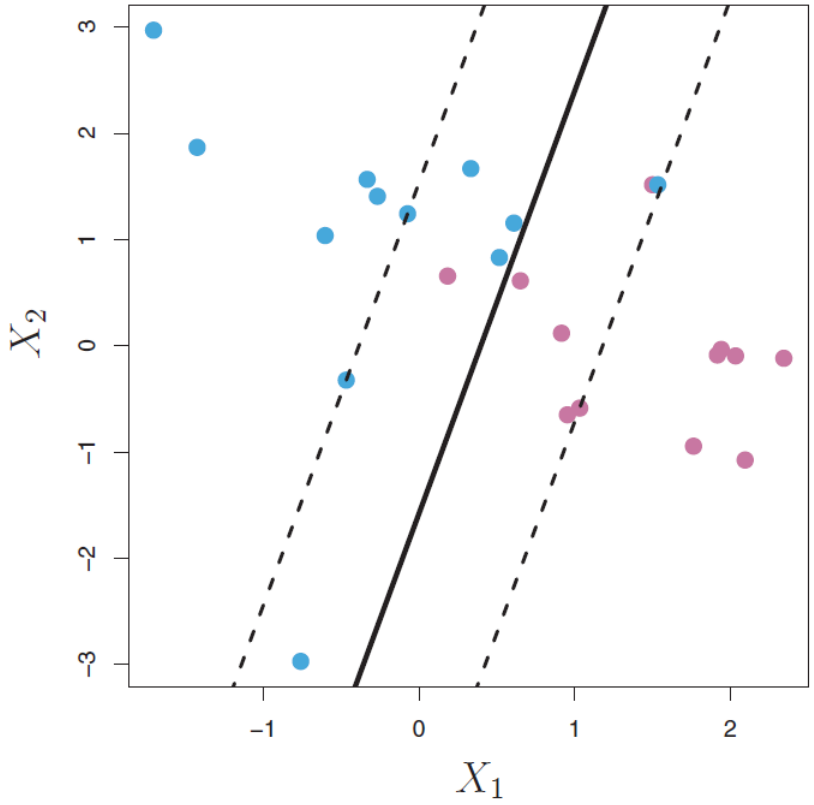
C を少し小さくした場合



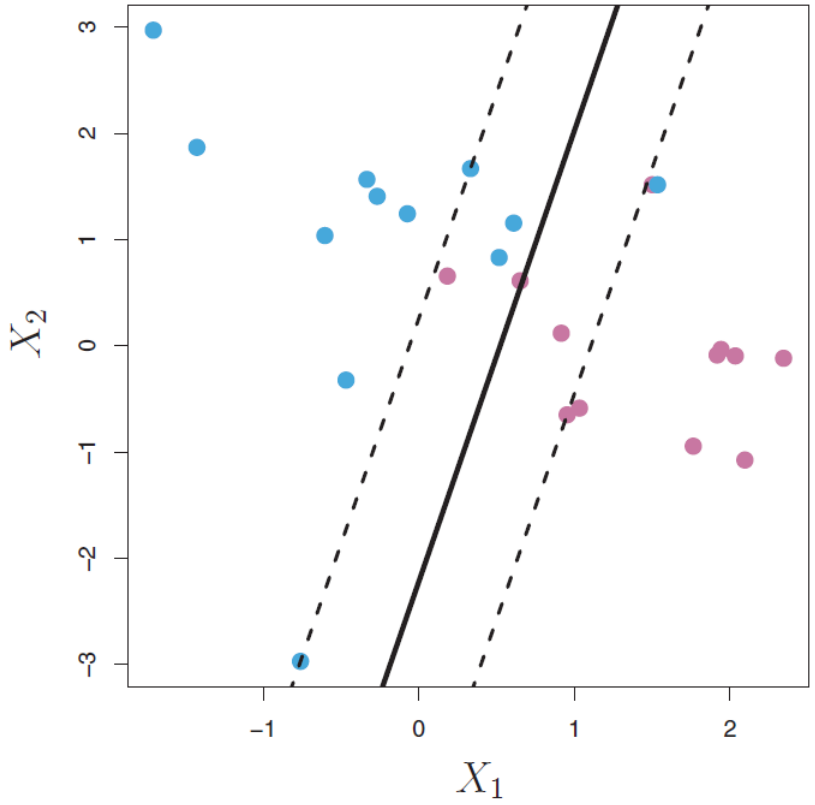
サポートベクター分類器

- C の値による違い

C を小さくした場合



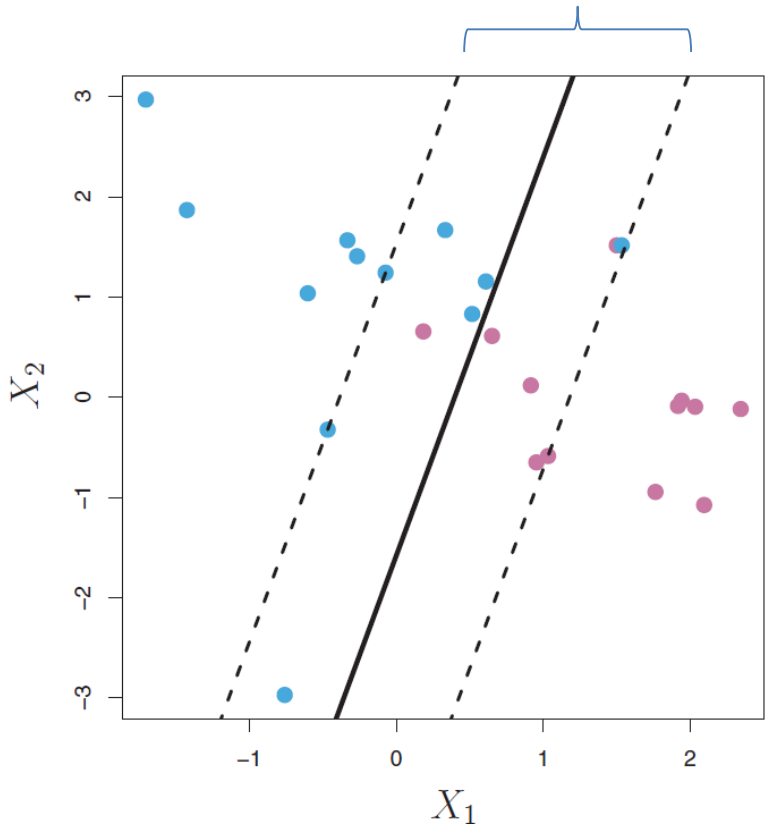
C をさらに小さくした場合



サポートベクター分類器

- サポートベクター

この領域に入っている事例がサポートベクター



- LDA 等と異なり、サポートベクター以外の事例はテスト事例の分類に影響なし
- *C* を大きくするとサポートベクターの数が増える
- The Bias-Variance trade-off
  - *C* が大きい: バリアンス小、バイアス大
  - *C* が小さい: バリアンス大、バイアス小

Python実習

- データ準備
  - 茶色の点（-1）: (0, 0) を中心に正規分布で10個
  - 水色の点（+1）: (1, 1) を中心に正規分布で10個

```
>>> import numpy as np
>>> np.random.seed(5)
>>> X = np.random.randn(20, 2)
>>> y = np.repeat([1, -1], 10)
>>> X[y == -1] = X[y == -1] + 1
>>> import matplotlib as mpl
>>> import matplotlib.pyplot as plt
>>> plt.scatter(X[:,0], X[:,1], s=70, c=y, cmap=mpl.cm.Paired)
>>> plt.xlabel('X1')
>>> plt.ylabel('X2')
>>> plt.show()
```

線形分離可能ではないことに注意

Adapted from http://www.science.smith.edu/~jcrouser/SDS293/labs/lab15-py.html

Python実習

- プロット用関数の読み込み
  - ダウンロード
    - ファイル　lec12.py
      - http://www.logos.t.u-tokyo.ac.jp/~tsuruoka/lecture/sml/lec12.py
    - 上記ファイルをホームディレクトリに保存（Windowsの場合）
  - ホームディレクトリに移動

```
>>> import os
>>> os.chdir(os.path.expanduser("~"))
```

  - plot_svc関数をインポート

```
>>> from lec12 import plot_svc
```

Python実習

- プロット用関数

```
def plot_svc(svc, X, y, h=0.02, pad=0.25):
    x_min, x_max = X[:, 0].min()-pad, X[:, 0].max()+pad
    y_min, y_max = X[:, 1].min()-pad, X[:, 1].max()+pad
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.2)

    plt.scatter(X[:,0], X[:,1], s=70, c=y, cmap=mpl.cm.Paired)
    # Support vectors indicated in plot by vertical lines
    sv = svc.support_vectors_
    plt.scatter(sv[:,0], sv[:,1], c='k', marker='x', s=100, linewidths='1')
    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)
    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.show()
    print('Number of support vectors: ', svc.support_.size)
```

Python実習

- サポートベクター分類器
  - SVC()
    - 引数C: マージン制約違反に対するペナルティの大きさ

```
>>> from sklearn.svm import SVC
>>> svc = SVC(C=1, kernel='linear')
>>> svc.fit(X, y)
```

- プロット
  - サポートベクターがXで表示される

```
>>> plot_svc(svc, X, y)
```

Python実習

- サポートベクター
  - support_： サポートベクターのインデックス
  - support_vectors_： サポートベクター
  - n_support_： サポートベクターの数

```
>>> svc.support_
 :
>>> svc.support_vectors_
 :
>>> svc.n_support_
 :
```

Python実習

- Cost を小さくしてみる

```
>>> svc2 = SVC(C=0.1, kernel='linear')
>>> svc2.fit(X, y)
>>> plot_svc(svc2, X, y)
```

マージンが広がり、サポートベクターの数が増える

Python実習

- 交差検証でチューニング
  - GridSearchCV()

```
>>> from sklearn.model_selection import GridSearchCV
>>> tuned_parameters = [{'C': [0.001, 0.01, 0.1, 1, 5, 10, 100]}]
>>> clf = GridSearchCV(SVC(kernel='linear'), tuned_parameters,
cv=10, scoring='accuracy')
>>> clf.fit(X, y)
```

- 最良のモデルのパラメータ

```
>>> clf.cv_results_
 :
>>> clf.best_params_
 :
```

Python実習

- テストデータで評価
  - テストデータの作成

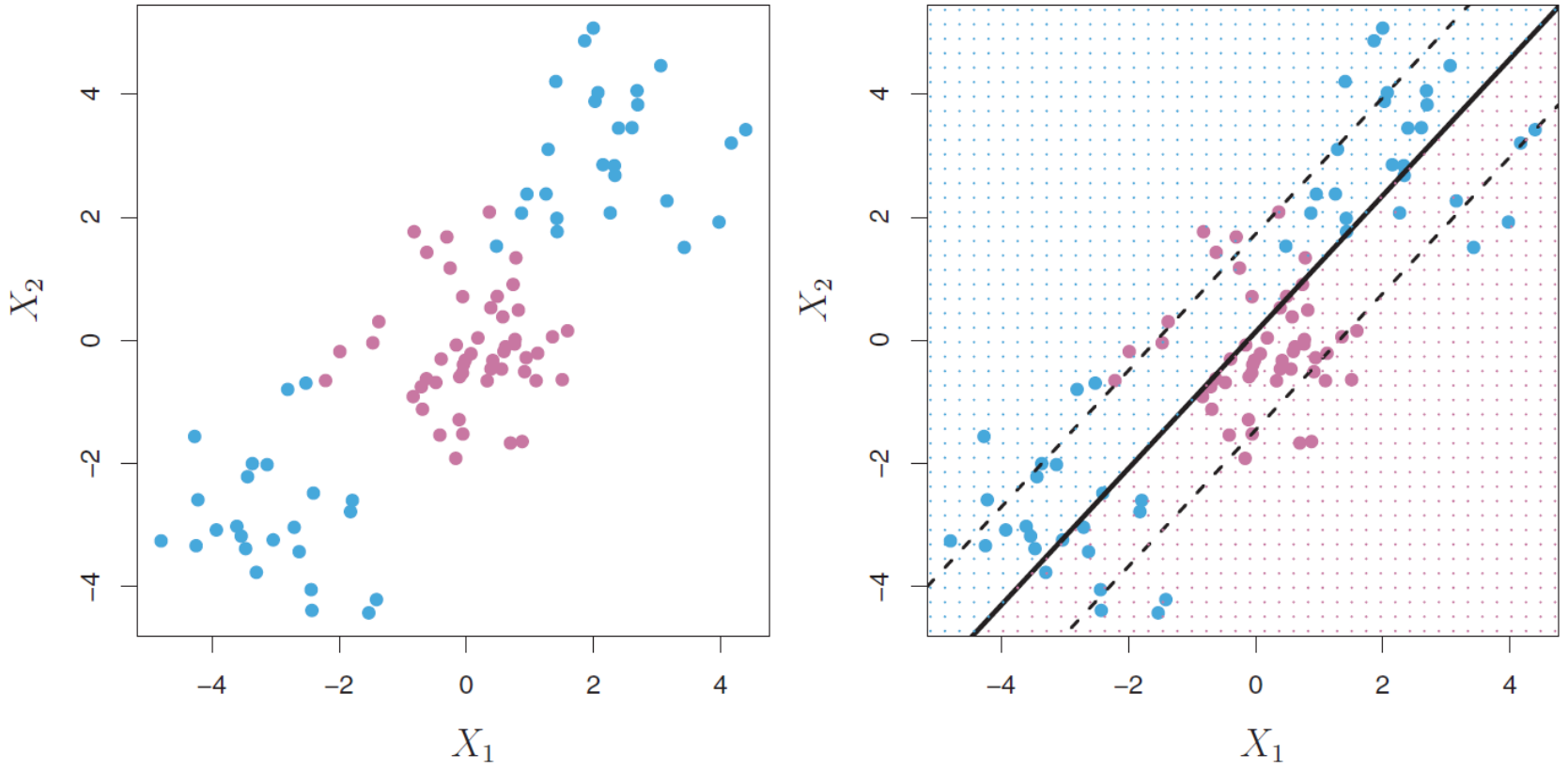
```
>>> np.random.seed(1)
>>> X_test = np.random.randn(20, 2)
>>> y_test = np.random.choice([-1, 1], 20)
>>> X_test[y_test == 1] = X_test[y_test == 1] - 1
```

  - 評価

```
>>> svc2 = SVC(C=0.001, kernel='linear')
>>> svc2.fit(X, y)
>>> y_pred = svc2.predict(X_test)
>>> import pandas as pd
>>> from sklearn.metrics import confusion_matrix
>>> pd.DataFrame(confusion_matrix(y_test, y_pred),
index=svc2.classes_, columns=svc2.classes_)
 :
```

サポートベクターマシン

- サポートベクター分類器でうまくいかない例



サポートベクターマシン

- 非線形な境界による分類
  - 特徴量空間の拡張によるアプローチ

$$X_1, X_2, \ldots, X_p \rightarrow X_1, X_1^2, X_2, X_2^2, \ldots, X_p, X_p^2$$

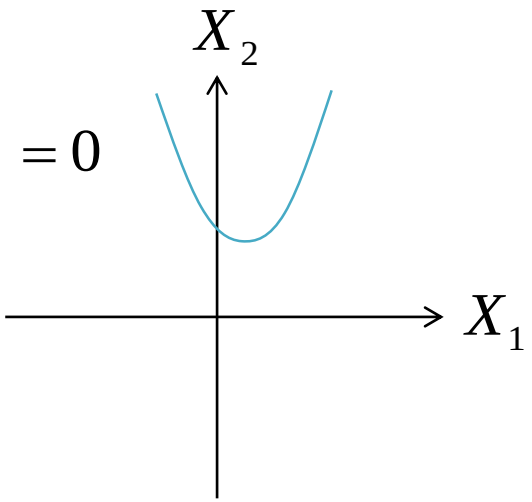
$$X_1, X_2, \ldots, X_p \rightarrow X_1, X_1^2, X_1^3, X_2, X_2^2, X_2^3, \ldots, X_p, X_p^2, X_p^3$$

$$X_1, X_2, \ldots, X_p \rightarrow X_1, \ldots, X_p, X_1X_2, X_1X_3, \ldots, X_1X_p, X_2X_3, \ldots$$

  - 例

決定境界

$$5 + 2X_1 - X_1^2 + X_2 = 0$$



- 非線形な境界は作れるが特徴量の数が大きく増える

サポートベクターマシン

- 事例ベクトル間の内積

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j}$$

- サポートベクター分類器はテスト事例と学習データの事例間の内積の和で表現できる

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i y_i \langle x, x_i \rangle$$

  - x_i がサポートベクターなら $\alpha_i > 0$、それでなければ $\alpha_i = 0$

参考

主問題と双対問題

主問題

$$\underset{\beta_0,\beta}{\text{maximize}}\, M$$

$$\text{subject to } \sum_{j=1}^{p}|\beta|^2 = 1$$

$$y_i\left(\beta_0 + \beta^T x_i\right) \geq M \quad \forall i = 1, \ldots, n$$

$$\underset{\beta_0,\beta}{\text{minimize}}\,|\beta|^2$$

$$\text{subject to } y_i\left(\beta_0 + \beta^T x_i\right) \geq 1 \quad \forall i = 1, \ldots, n$$

双対問題

$$\underset{\alpha}{\text{maximize}} \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{i'=1}^{n} y_i y_{i'} \alpha_i \alpha_{i'} \langle x_i, x_{i'} \rangle$$

$$\text{subject to } \alpha_i \geq 0 \quad \forall i = 1, \ldots, n$$

$$\sum_{i=1}^{n}\alpha_i y_i = 0$$

内積のみ

解の対応関係

$$\beta = \sum_{i=1}^{n}\alpha_i y_i x_i$$

$$\beta_0 = -\frac{\max_{i:y_i=-1}\beta^T x_i + \min_{i:y_i=1}\beta^T x_i}{2}$$

$$f(x) = \beta_0 + \beta^T x$$

$$= \beta_0 + \sum_{i=1}^{n}\alpha_i y_i \langle x, x_i \rangle$$

サポートベクターマシン

- カーネル（kernel）
  - 内積の計算をすべてカーネル関数に置きかえる
  - SVMでよく使われるカーネル

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}$$

・・・ 線形カーネル
（内積そのもの）

$$K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^{p} x_{ij} x_{i'j}\right)^d$$

・・・ 多項式カーネル

$$K(x_i, x_{i'}) = \exp\left(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2\right)$$

・・・ RBF カーネル
（Gaussian カーネル）

サポートベクターマシン

- 多項式カーネルの例（$p = 2, d = 2$ の場合）

$$K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^{2} x_{ij} x_{i'j}\right)^2$$
$$= (1 + x_{i1}x_{i'1} + x_{i2}x_{i'2})^2$$
$$= 1 + 2x_{i1}x_{i'1} + 2x_{i2}x_{i'2} + 2x_{i1}x_{i2}x_{i'1}x_{i'2} + x_{i1}^2 x_{i'1}^2 + x_{i2}^2 x_{i'2}^2$$

拡張された特徴量

$$\phi(x_i) = \begin{pmatrix} 1 \\ \sqrt{2}x_{i1} \\ \sqrt{2}x_{i2} \\ \sqrt{2}x_{i1}x_{i2} \\ x_{i1}^2 \\ x_{i2}^2 \end{pmatrix} \quad \phi(x_{i'}) = \begin{pmatrix} 1 \\ \sqrt{2}x_{i'1} \\ \sqrt{2}x_{i'2} \\ \sqrt{2}x_{i'1}x_{i'2} \\ x_{i'1}^2 \\ x_{i'2}^2 \end{pmatrix}$$

拡張された特徴量空間での
内積と同じ結果！

$$\langle \phi(x_i), \phi(x_{i'}) \rangle$$
$$= 1 + 2x_{i1}x_{i'1} + 2x_{i2}x_{i'2} + 2x_{i1}x_{i2}x_{i'1}x_{i'2} + x_{i1}^2 x_{i'1}^2 + x_{i2}^2 x_{i'2}^2$$

サポートベクターマシン

- サポートベクターマシン（support vector machine）
  - 非線形カーネルを用いたサポートベクター分類器

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i y_i K(x, x_i)$$

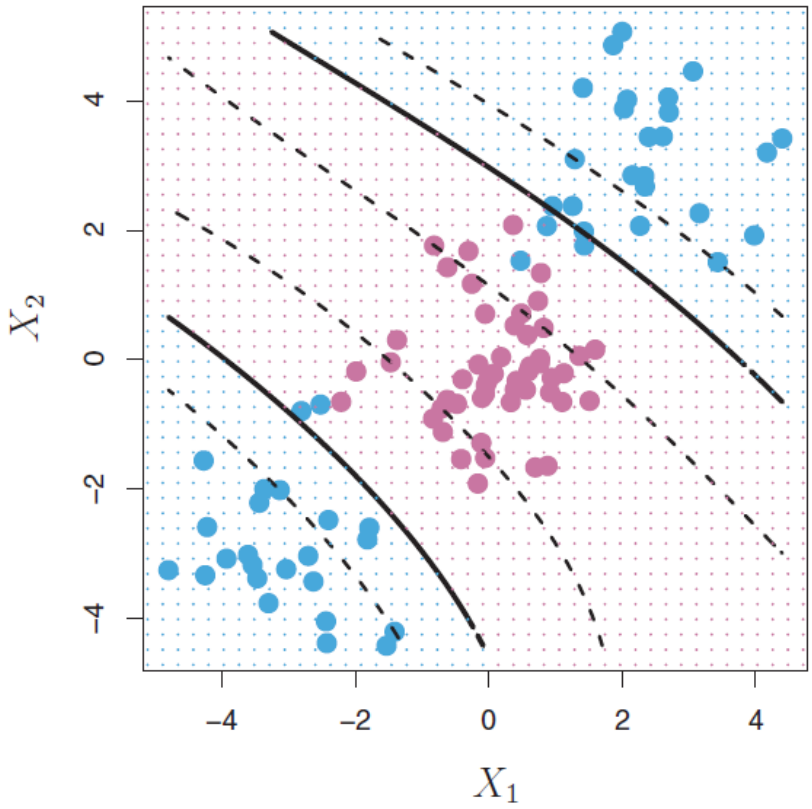
S : サポートベクターのインデックスの集合

  - 明示的に特徴量を増やすことなく非線形な決定境界を構成することができる

サポートベクターマシン

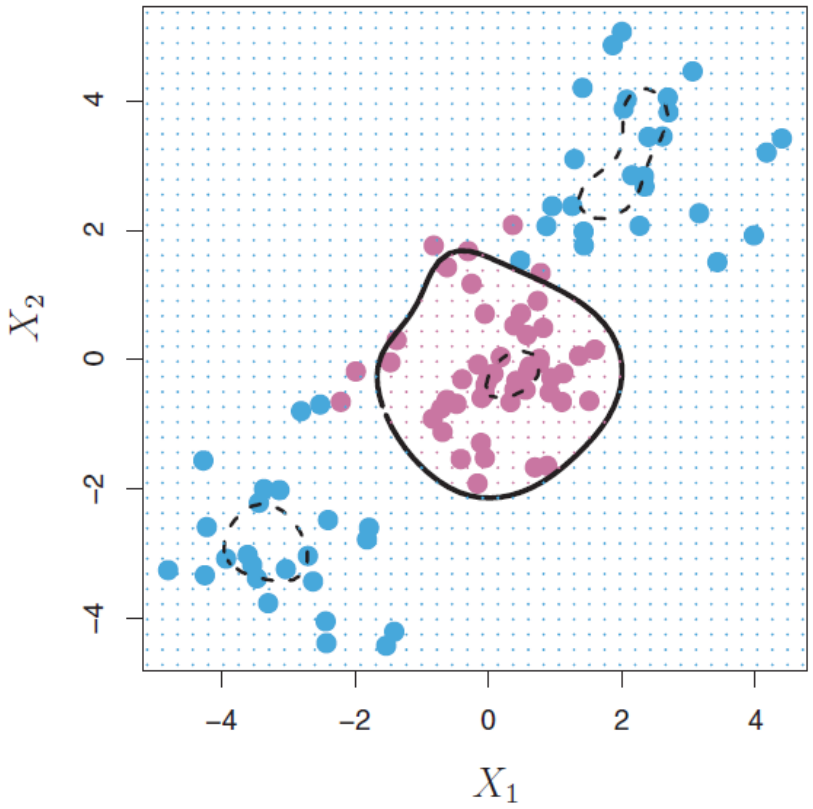
多項式カーネル（3次）

$$K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^{p} x_{ij} x_{i'j}\right)^3$$



RBF カーネル

$$K(x_i, x_{i'}) = \exp\left(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2\right)$$



Python実習

- サポートベクターマシン
  - 学習データの作成

```
>>> from sklearn.model_selection import train_test_split
>>> np.random.seed(8)
>>> X = np.random.randn(200, 2)
>>> X[:100] = X[:100] + 2
>>> X[101:150] = X[101:150] - 2
>>> y = np.concatenate([np.repeat(-1, 150), np.repeat(1,50)])
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.5,
random_state=2)
```

  - プロット

```
>>> plt.scatter(X[:,0], X[:,1], s=70, c=y, cmap=mpl.cm.Paired)
>>> plt.xlabel('X1')
>>> plt.ylabel('X2’)
>>> plt.show()
```

Python実習

- 学習
  - RBFカーネル
    - ハイパーパラメータ: cost および gamma を指定

```
>>> svm = SVC(C=1.0, kernel='rbf', gamma=1)
>>> svm.fit(X_train, y_train)
>>> plot_svc(svm, X_test, y_test)
```

正しく分類されていない事例もある

Python実習

- Cost を大きくしてみる

```
>>> svm2 = SVC(C=100, kernel='rbf', gamma=1.0)
>>> svm2.fit(X_train, y_train)
>>> plot_svc(svm2, X_test, y_test)
```

学習エラーは低下
不規則な決定境界 → 過学習の可能性

Python実習

- 交差検証でチューニング
  - cost と gamma に関してグリッドサーチ

```
>>> tuned_parameters = [{'C': [0.01, 0.1, 1, 10, 100],
                        'gamma': [0.5, 1,2,3,4]}]
>>> clf = GridSearchCV(SVC(kernel='rbf'), tuned_parameters, cv=10,
scoring='accuracy')
>>> clf.fit(X_train, y_train)
>>> clf.best_params_
 :
```

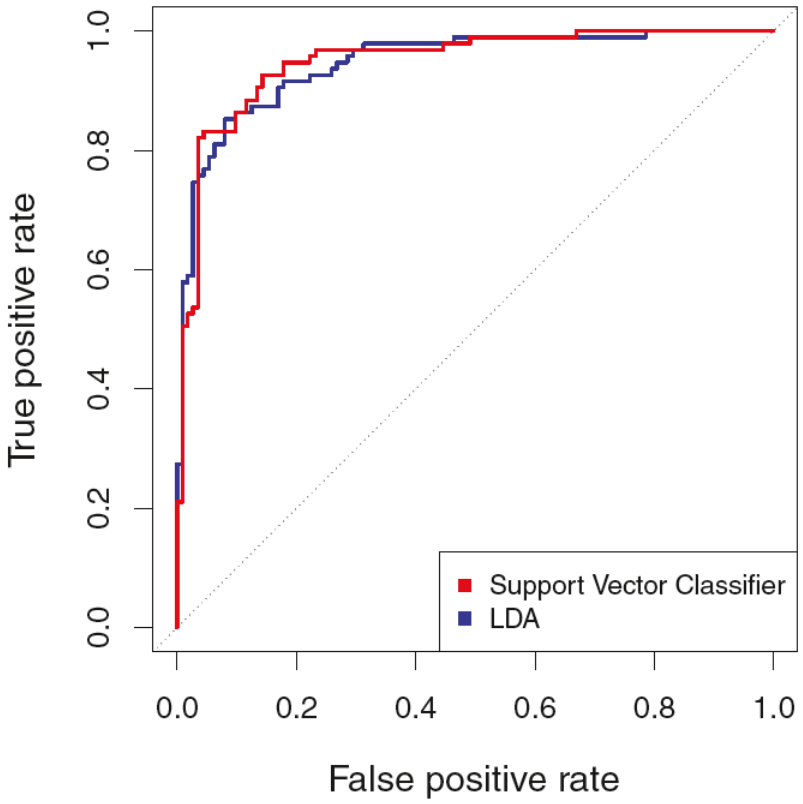
- テストデータで評価

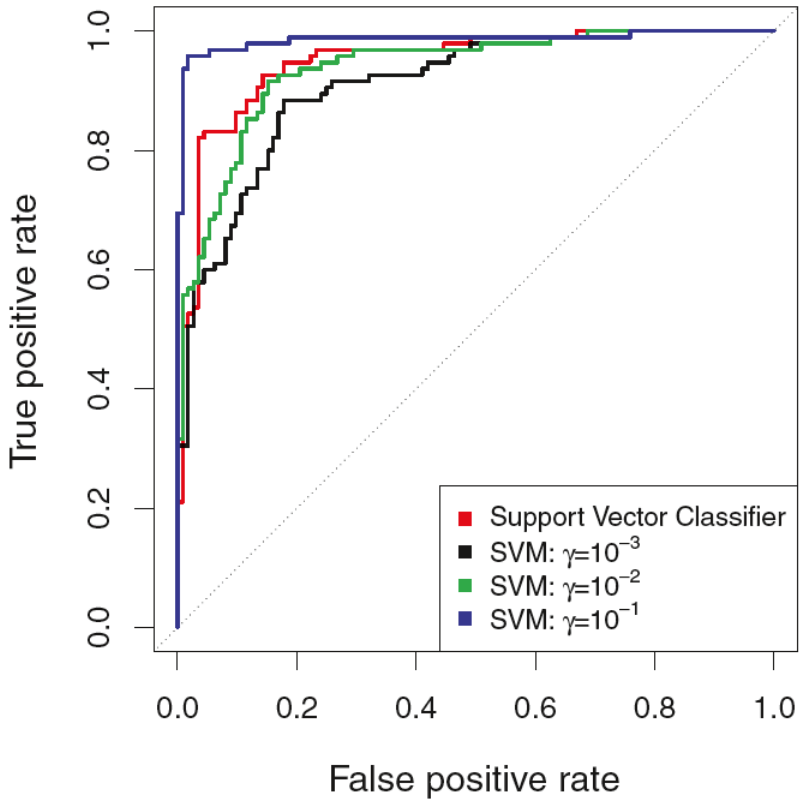
```
>>> plot_svc(clf.best_estimator_, X_test, y_test)
>>> confusion_matrix(y_test, clf.best_estimator_.predict(X_test))
>>> clf.best_estimator_.score(X_test, y_test)
```

SVMによる分類の例

- Heart データ
  - 心疾患の有無を13の特徴量によって予測

学習データでの精度（ROC曲線）



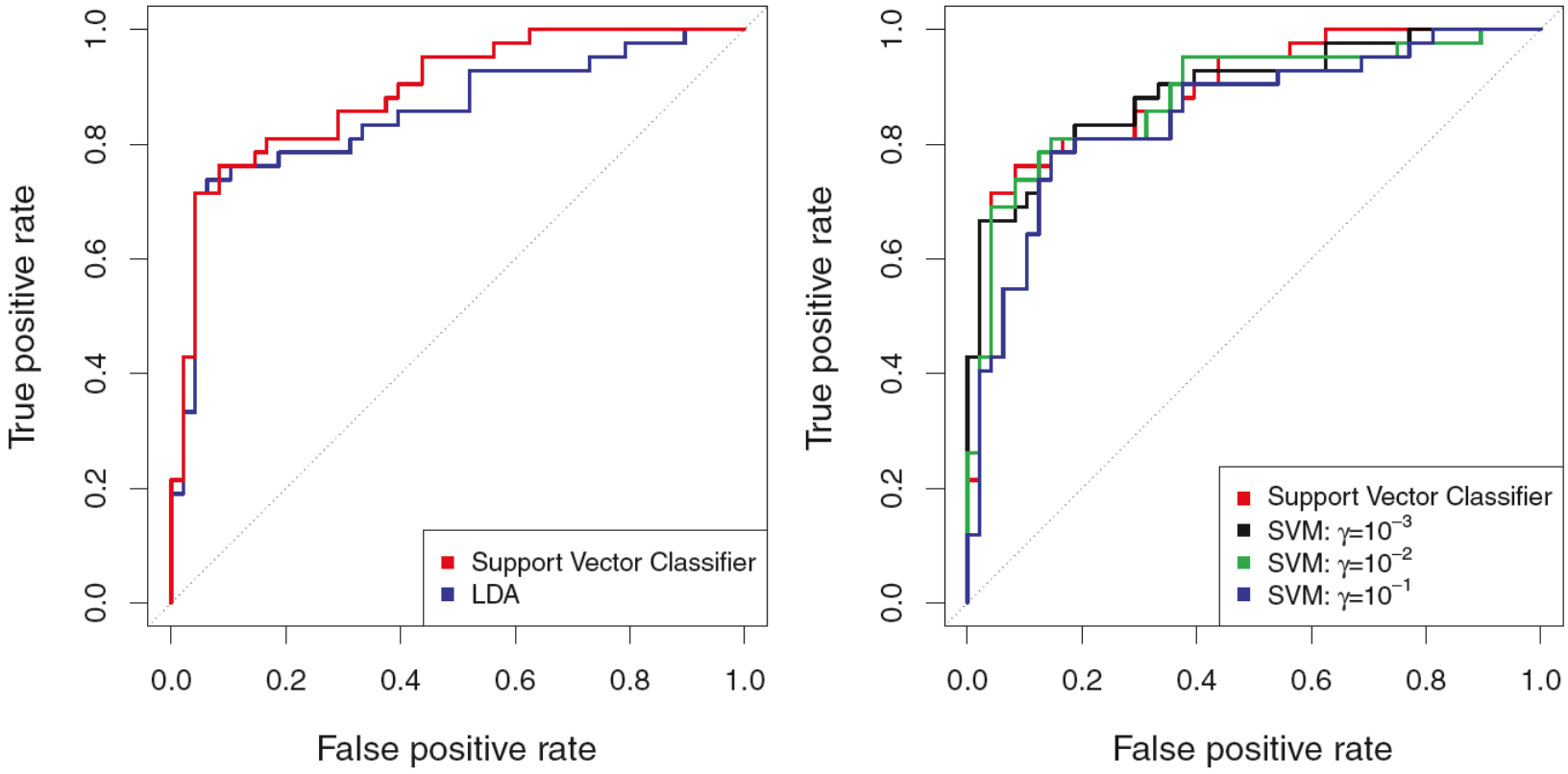


RBFカーネル（$\gamma=10^{-1}$）が最も良く見えるが・・

SVMによる分類の例

- Heart データ

テストデータでの精度（ROC曲線）



→ RBFカーネル（$\gamma=10^{-1}$）では過学習

SVMによる多クラス分類

- SVM は2値分類のための手法

- 出力のクラスが3つ以上の場合
  - One-Versus-One 分類
    - すべてのクラスの組み合わせに対して2値分類モデルを作成
    - 最終的な出力は多数決で決める
  - One-Versus-All 分類
    - それぞれのクラスに対して、そのクラスであるか否かを判別する2値分類モデルを作成
    - マージンが最も大きいクラスを採用