

SVM

(support vector machine)

正田 備也

masada@rikkyo.ac.jp

パーセプトロンとSVM

線形モデルによる2値分類

- 確率を使う
 - 線形モデルの出力を確率（0～1の値）に変換
 - 変換にはシグモイド関数を使う
- 別の方法はないか？・・・符号（+か-か）を使う
 - 実はロジスティック回帰も符号で分けている（後述）
- 符号を直接使うように、損失関数を変更する

パーセプトロン

- 2値分類に使われる機械学習の手法のひとつ
- モデルは線形モデル

$$f(\mathbf{x}_i) = a_0 + a_1x_{i,1} + \cdots + a_dx_{i,d}$$

- 損失関数を以下のように設定する
 - y_i は、 i 番目のデータ点が属するクラスを**1**か**-1**で表す、target値。

$$L(a_0, a_1, \dots, a_d) = \sum_{i=1}^N \max(0, -y_i f(\mathbf{x}_i))$$

なぜこの損失関数で2値分類できるのか？

- $y = 1$ の場合

- 線形関数の出力 $f(\mathbf{x})$ が正だと、 $\max(0, -yf(\mathbf{x})) = 0$
- 線形関数の出力 $f(\mathbf{x})$ が負だと、 $\max(0, -yf(\mathbf{x})) = -f(\mathbf{x}) > 0$

- $y = -1$ の場合

- 線形関数の出力 $f(\mathbf{x})$ が正だと、 $\max(0, -yf(\mathbf{x})) = f(\mathbf{x}) > 0$
- 線形関数の出力 $f(\mathbf{x})$ が負だと、 $\max(0, -yf(\mathbf{x})) = 0$

勾配を使ったパラメータ更新の3種類

- バッチ勾配降下法
 - 一度に訓練データすべてを使って勾配を計算
- ミニバッチ勾配降下法 ← 最もよく使われる
 - 訓練データのうち数十～数百のサンプルを使って勾配を計算
- SGD（確率的勾配降下法）
 - 訓練データからひとつずつサンプルを使って勾配を計算

SGD (stochastic gradient descent)

- 訓練データのひとつひとつについて勾配を求める、パラメータを更新していく方法
 1. パラメータをランダムに初期化
 2. 訓練データをランダムシャッフル
 3. for each サンプル in 訓練データ:
 1. 一つのサンプルについて勾配を計算
 2. その勾配に小さな数 α を掛けて現在のパラメータから引き算する
 4. 計算が収束していなければ2. に戻る

パーセプトロンのSGDの更新式

- 実際に偏微分の計算をすると、以下のようなになる

$$\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{bmatrix} \leftarrow \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{bmatrix} - \alpha(-y_i) \begin{bmatrix} 1 \\ x_{i,1} \\ \vdots \\ x_{i,d} \end{bmatrix}$$

パーセプトロンのSGDの更新式

- $y_i = 1$ のとき

$$\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{bmatrix} \leftarrow \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{bmatrix} + \alpha \begin{bmatrix} 1 \\ x_{i,1} \\ \vdots \\ x_{i,d} \end{bmatrix}$$

- $y_i = -1$ のとき

$$\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{bmatrix} \leftarrow \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{bmatrix} - \alpha \begin{bmatrix} 1 \\ x_{i,1} \\ \vdots \\ x_{i,d} \end{bmatrix}$$

SVM

参考: https://www.ism.ac.jp/~fukumizu/ISM_lecture_2006/svm-ism.pdf

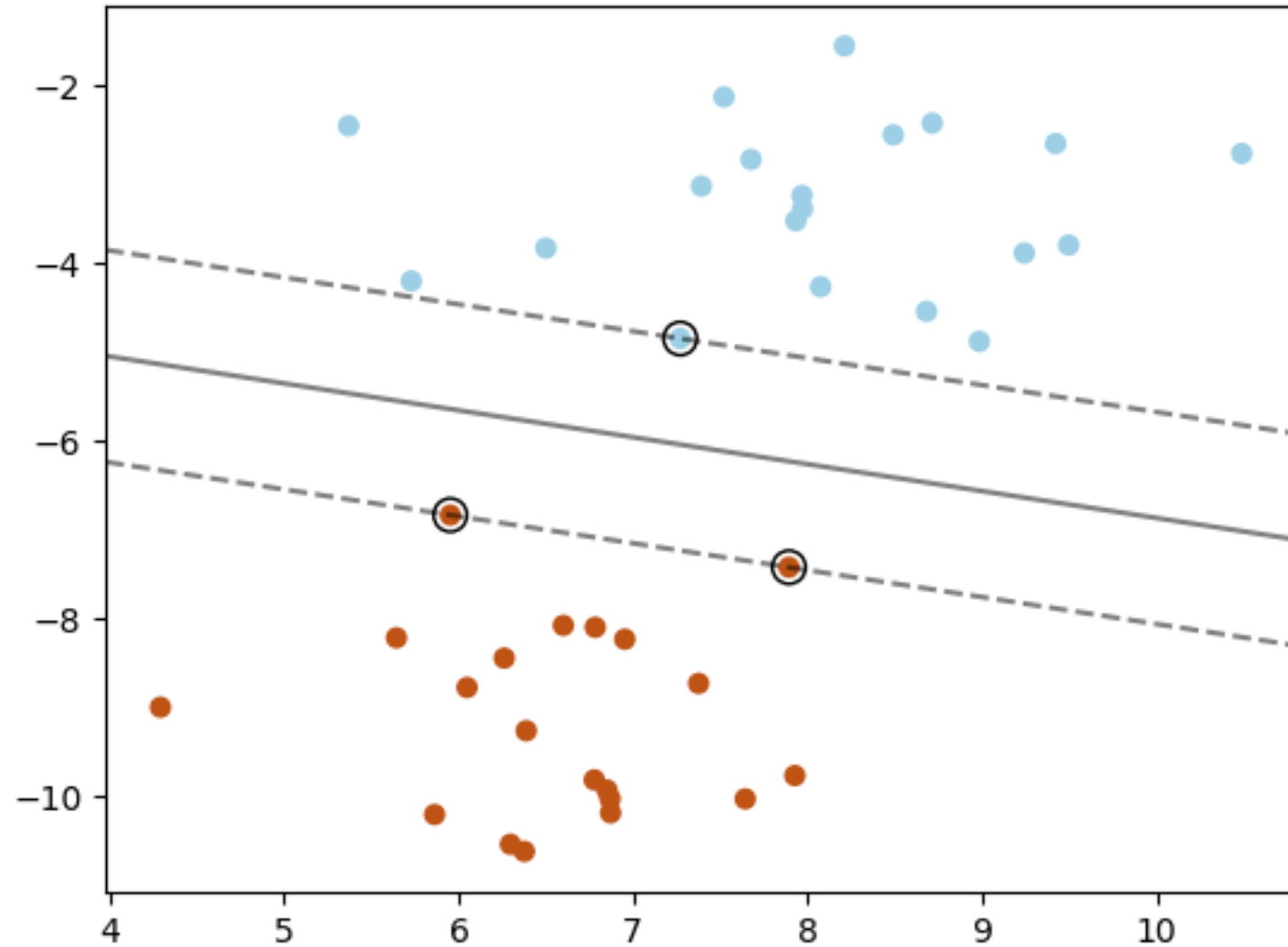
$$f(\mathbf{x}_i) = a_0 + a_1 x_{i,1} + \cdots + a_d x_{i,d}$$

- SVMで最小化する関数

$$l(a_0, \dots, a_d) = \frac{1}{2} \sum_{j=1}^d a_j^2 + C \sum_i \max(0, 1 - y_i f(\mathbf{x}_i))$$

- パーセプトロンで最小化する関数

$$l(a_0, \dots, a_d) = \sum_i \max(0, -y_i f(\mathbf{x}_i))$$



<https://scikit-learn.org/stable/modules/svm.html>

SVMとパーセプトロンの違い

- $\max(0, 1 - y_i f(\mathbf{x}_i))$ に注目
 - ヒンジロス(hinge loss)と呼ばれる損失関数
 - $y_i f(\mathbf{x}_i)$ が1以上にならないと損失がゼロにならない
 - パーセプトロンでは、 $y_i f(\mathbf{x}_i)$ が正なら損失はゼロになっていた。
- 係数の2乗の和も同時に最小化
 - これはマージン最大化に由来する項
 - 一種の正則化ともみなせる

(復習) ロジスティック回帰

- ロジスティック回帰の損失関数はクロスエントロピーだった

$$-\{y_i \log p_i + (1 - y_i) \log(1 - p_i)\}$$

- 正解 y_i が1のとき： $-\log p_i = -\log \frac{1}{1+e^{-f(x_i)}} = \log(1 + e^{-f(x_i)})$

- 正解 y_i が0のとき： $-\log(1 - p_i) = -\log\left(1 - \frac{1}{1+e^{-f(x_i)}}\right) = \log(1 + e^{f(x_i)})$

- target値 y_i が**1**か**-1**であるとき、まとめて次のように書き換えられる

$$\log(1 + e^{-y_i f(x_i)})$$

y_i が 1 か -1 で表される場合に
分類器の学習時に最小化する関数

- SVM

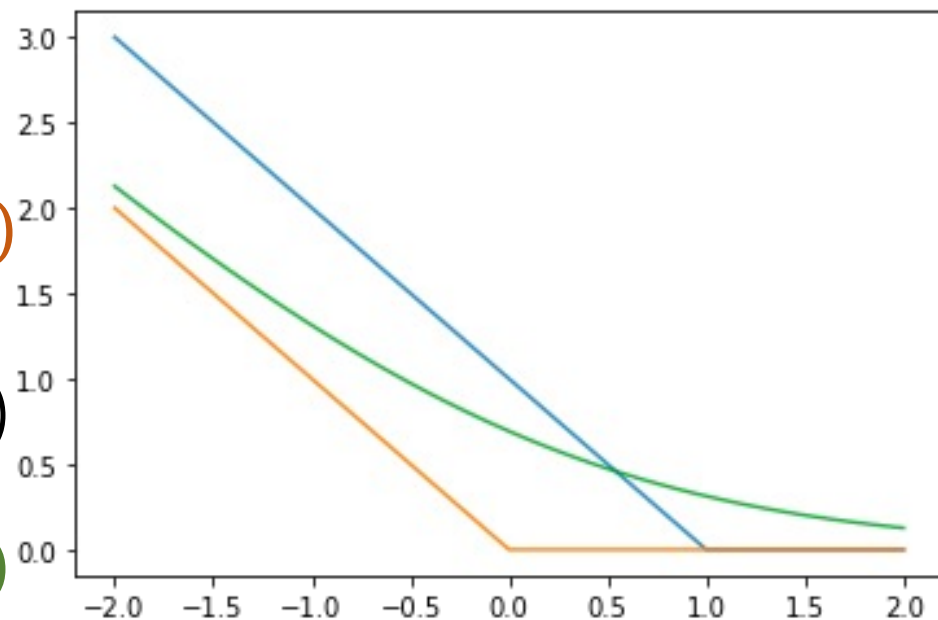
$$\frac{1}{2} \sum_{j=1}^d a_j^2 + C \sum_i \max(0, 1 - y_i f(\mathbf{x}_i))$$

- パーセプトロン

$$\sum_i \max(0, -y_i f(\mathbf{x}_i))$$

- ロジスティック回帰

$$\sum_i \log(1 + e^{-y_i f(\mathbf{x}_i)})$$



カーネルトリック

パーセプトロンでのパラメータ推定

- パーセプトロンの更新式をよく見ると・・・

$$\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{bmatrix} \leftarrow \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{bmatrix} - \alpha(-y_i) \begin{bmatrix} 1 \\ x_{i,1} \\ \vdots \\ x_{i,d} \end{bmatrix}$$

- 推定された $\hat{\mathbf{a}} = (a_1, \dots, a_d)$ は、訓練データ \mathbf{x}_i の線形結合！

$$\hat{\mathbf{a}} = \sum_{i=1}^N w_i y_i \mathbf{x}_i$$

- なお、初期値はゼロベクトルだったとする。

Perceptron Representer Theorem

Theorem 12 (Perceptron Representer Theorem).

During a run of the perceptron algorithm, the weight vector is always in the span of the (assumed non-empty) training data.

Hal Daumé III. A Course in Machine Learning. Chapter 11.

<http://ciml.info/>

パーセプトロンでの予測

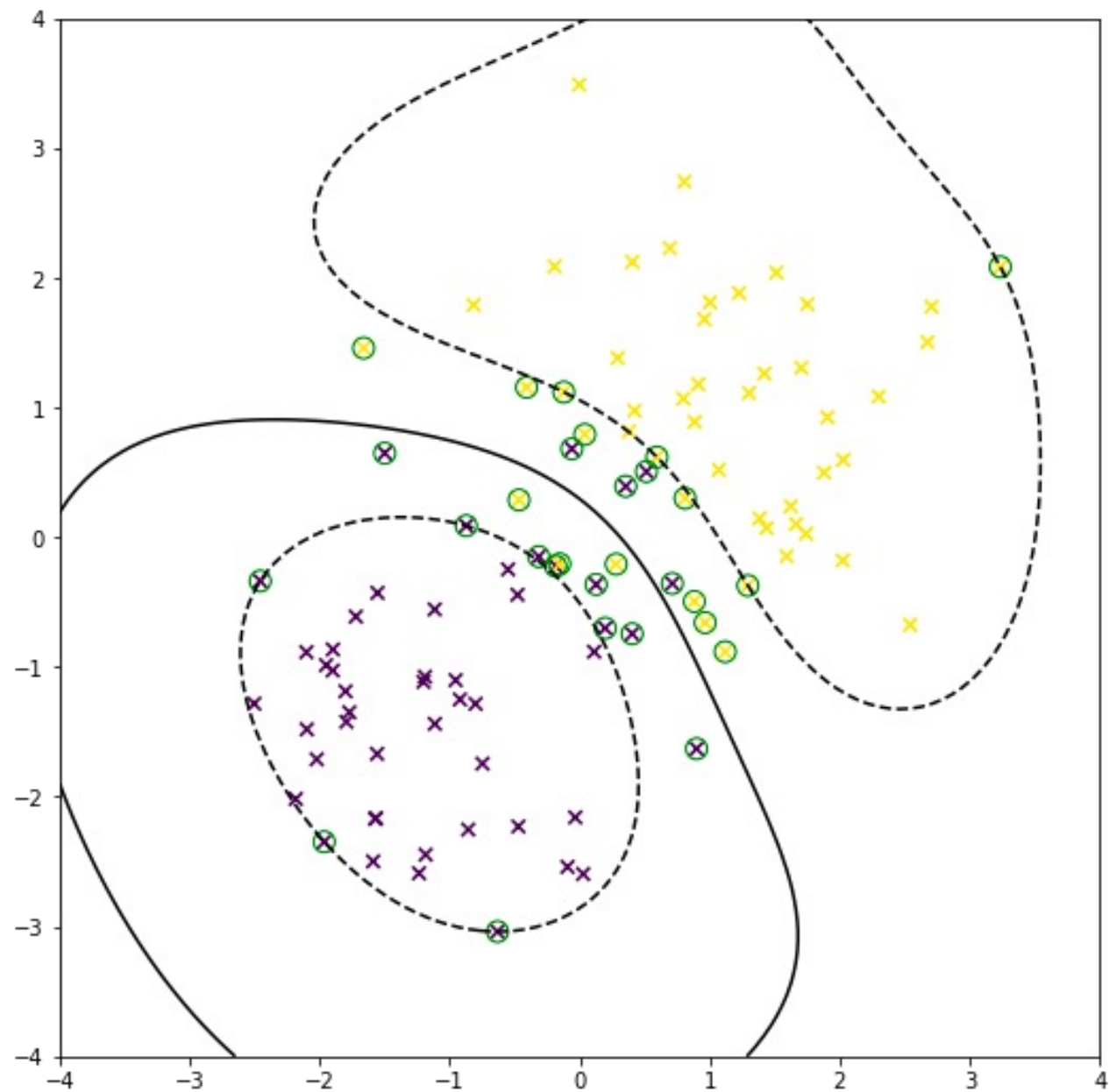
- 未知の入力 $\mathbf{x}' = (x_1', \dots, x_d')$ はどちらのクラス？
 - $\hat{a}_0 + \hat{\mathbf{a}}^T \mathbf{x}' \geq 0$ ならば、 $y = 1$ のほうのクラス
 - $\hat{a}_0 + \hat{\mathbf{a}}^T \mathbf{x}' < 0$ ならば、 $y = -1$ のほうのクラス
 - ただし \hat{a}_0 は推定された切片とする。
- つまり、下の値の符号によってクラスを予測する

$$\hat{a}_0 + \hat{\mathbf{a}}^T \mathbf{x}' = \hat{a}_0 + \left(\sum_{i=1}^N w_i y_i \mathbf{x}_i \right)^T \mathbf{x}' = \hat{a}_0 + \sum_{i=1}^N w_i y_i \mathbf{x}_i^T \mathbf{x}'$$

- 内積を計算できさえすればクラスの予測ができるということ。

高次元化

- 入力ベクトル $\mathbf{x} = (x_1, x_2)$ を、例えば下の写像 φ で変更してみる
$$\varphi(\mathbf{x}) = (\sqrt{2}x_1x_2, x_1^2, x_2^2)$$
 - 2次の項を含んでいる。
- これを入力ベクトルとして使う
- モデルの表現力が増した
 - 平面ではなく、曲面で分割できるようになる。
 - 別の関数を φ として使った例を、次のスライドに示した。
- でも、必要な計算の量も増えてしまった
 - 次元が上がった空間で内積の計算をしないといけないため。



https://nbviewer.jupyter.org/github/ctgk/PRML/blob/master/notebooks/ch07_Sparse_Kernel_Machines.ipynb

上を単独で動かせるようにしたもの: https://github.com/tomonari-masada/course2022-sml/blob/main/RBF_kernel_example01.ipynb

写像 φ のうまい選び方

- 予測に必要なのは、入力ベクトルどうしの内積だけだった
- $\varphi(\mathbf{x}) = (\sqrt{2}x_1x_2, x_1^2, x_2^2)$ で高次元化したら内積はどうなる？

$$\varphi(\mathbf{x})\varphi(\mathbf{x}') = \left(\sum_{j=1}^d x_j x_j' \right)^2 = (\mathbf{x}^T \mathbf{x}')^2$$

- 計算の手間が元の入力ベクトルどうしの内積とほぼ同じで済む
- 写像 φ をうまく決めると、計算の手間は増やさずに、モデルの表現力だけを上げることができる

内積の便利さ

- ベクトルに関する処理には、内積さえ計算できれば実行できるものが結構ある
 - ベクトルの長さを求める（＝自分自身との内積のルート）
 - ベクトル間の距離を求める（＝引き算して長さを求める）
 - 複数のベクトルの重心（平均）から別のベクトルまでの距離
 - 複数のベクトルの重心そのものは、内積だけでは求められない。
- 主成分分析

カーネル

- 入力ベクトルを写像 φ で移した先の空間での内積を与える関数を、カーネル関数と呼ぶ

- $\varphi(\mathbf{x}) = (\sqrt{2}x_1x_2, x_1^2, x_2^2)$ のカーネル関数 k は以下の通り。

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}')^2$$

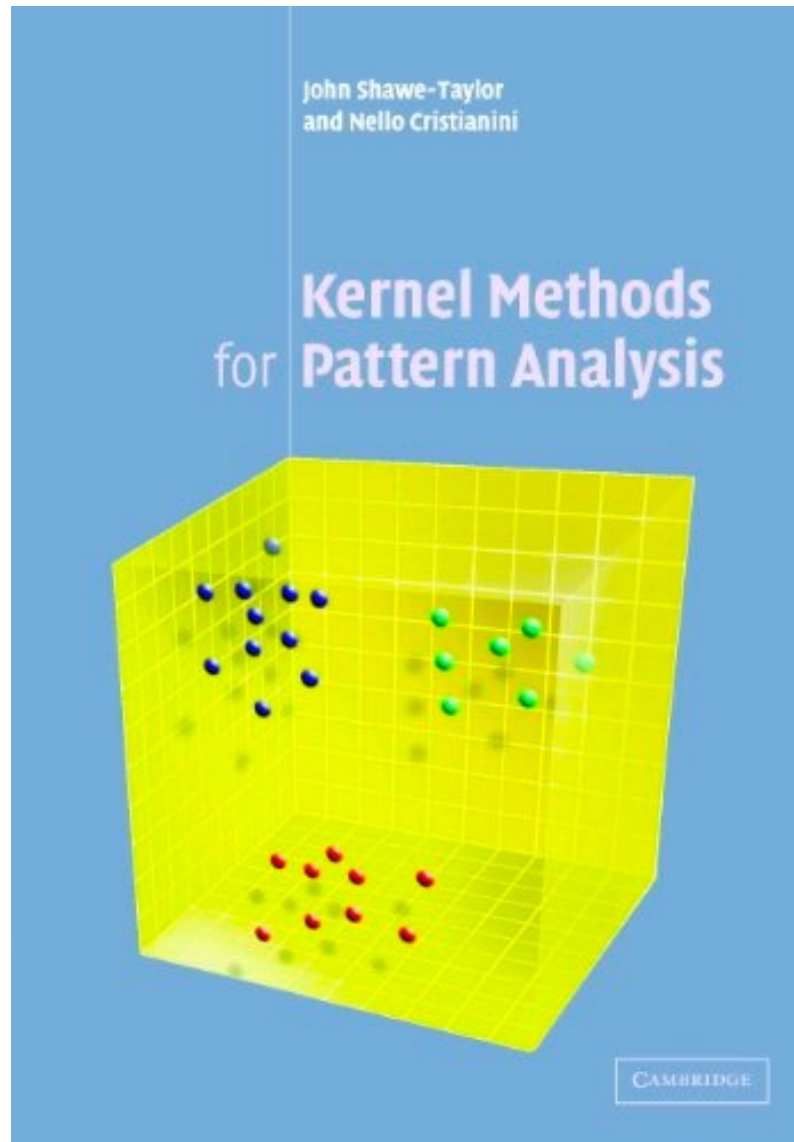
- 3つ前のスライドのカーネル関数（RBFカーネルと呼ばれる）は、以下。

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \sum_{j=1}^d (x_j - x_j')^2)$$

- 参考: <http://pages.cs.wisc.edu/~matthewb/pages/notes/pdf/svms/RBFFKernel.pdf>

カーネル・トリック

- 写像 ϕ を、feature mapと呼ぶ
- うまく写像を選ぶと、高次元化した後の空間での内積が、元の空間での内積と同じぐらいの手間で計算できる
 - feature mapが明示的に計算される必要はない
- カーネル・トリック
 - 入力ベクトルをfeature mapで高次元空間に移す
 - 移した先の空間(feature space)で、SVMや主成分分析をおこなう
 - feature spaceでの超平面は、元の空間では曲面になっている
 - より柔軟な分類や次元圧縮がおこなえる。



<https://kernelmethods.blogs.bristol.ac.uk/>

今日の課題

- Wikipediaから米国の俳優のエントリをクロールして取得
 - spaCyというツールで、特定の品詞を削除し、lemmatization
 - lemmatization = 活用変化をなくして原型にする
1. テストセットに含まれるエントリについて、それが男性俳優のエントリか女性俳優のエントリかを、予測してみる
 2. 分類に特に効いている単語に映画界でのジェンダー・ステレオタイプのようなものが現れているか否かを調べてみる