

機械学習入門

経済学部 BX584

第13回 パーセプトロンとSVM

線形関数を分類にどう使うか？

- 回帰では関数の出力値をそのまま使っていた
 - 出力値の範囲は $-\infty$ から $+\infty$ まで。
- 出力値を2値分類に使うにはどうすればいいか？
 - 無理やり0から1の範囲に押し込める(1か0かで2クラスに分ける)
 - ロジスティック回帰
 - 符号を使う(正か負かで2クラスに分ける)
 - パーセプトロン、SVM

線形関数とは(正確にはアフィン関数)

- d 次元ベクトルを入力とし、スカラーを出力とする以下のような関数

$$f_w(\mathbf{x}) = b + w_1 x_1 + \cdots + w_D x_D$$

- $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$ とも書ける。
 - ただし $\mathbf{w} = (b, w_1, \dots, w_D)$ 、および、 $\mathbf{x} = (1, x_1, \dots, x_D)$ とする。
- 訓練データを使って (b, w_1, \dots, w_D) を求めるのが機械学習
 - 損失関数が小さくなるように、これらのパラメータを計算機に動かさせる。

線形関数の使いみち

- 回帰 (regression)
 - 分析対象の特定の性質がひとつの数値で表される ← 望ましい出力値
 - $f(x)$ がその数値を表すように線形関数を選ぶ
- 二値分類 (binary classification) ... 多値分類についてはいずれ説明します。
 - 分析対象が2つのグループに分かれている
 - $f(x)$ の値によって2つのグループが区別できるように線型関数を選ぶ

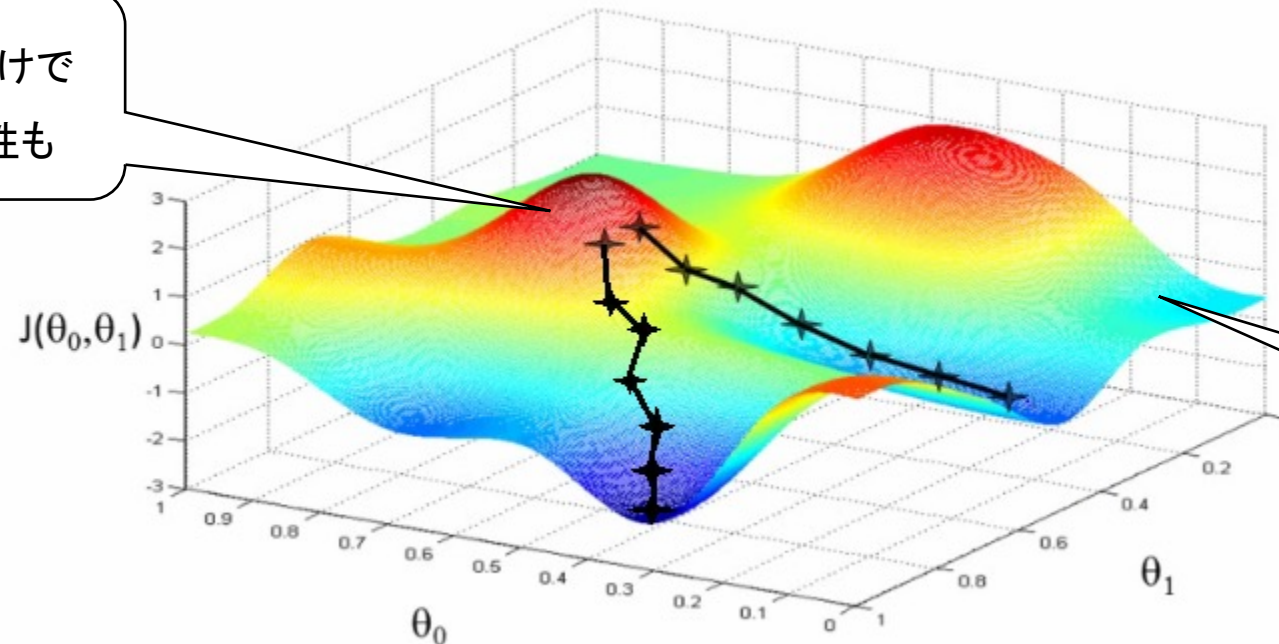
損失関数とは

- それを最小化すると良い線形関数が見つかるような関数
- 「損失 (loss)」= 値が小さいほど良い
 - 無数にある w の集合に、下るほど良いことがあるような地形を持ち込むのが、損失関数。
- 普通は勾配を利用して最小化する
 - 損失関数によって持ち込まれた地形の傾きを調べて(=微分して)下っていく

損失関数は地形を持ち込む

- 坂を下る方向にパラメータを変化させる → 極小値に近づく

出発点が少し違うだけで
別の谷に行く可能性も



損失関数によって
持ち込まれた地形

モデルと損失関数（パーセプトロンの場合）

- 線形関数 $f_{\mathbf{w}}(\mathbf{x}) = b + w_1 x_1 + \dots + w_D x_D$ をモデルとして採用
- 損失関数を以下のように決める

$$L(\mathbf{w}) = \sum_{i=1}^N \max(0, -t_i f_{\mathbf{w}}(\mathbf{x}_i))$$

- $\mathbf{x}_i = (x_{i,1}, \dots, x_{i,D})$ は i 番目のデータ点
- t_i は正解ラベルで、 -1 か 1 かのどちらか

例) スпамメールを $t_i = -1$ で、通常メールを $t_i = 1$ で表す。

確率的勾配降下法による損失関数の最小化

- 確率的勾配降下法
 - 各訓練データについて勾配を求め、下る方向にパラメータを更新
- パーセプトロンの場合
 - 個々の x_i の損失関数の値 $L_i = \max(0, -t_i f_w(x_i))$ について...
 - $L_i = -t_i f_w(x_i)$ の場合: $-t_i f_w(x_i)$ を各パラメータで偏微分
 - $L_i = 0$ の場合: 0を偏微分しても0

パーセプトロンでの損失関数の勾配

- $L_i = \max(0, -t_i f_w(x_i)) = -t_i f_w(x_i)$ となる場合のほうを考える

- $\frac{\partial}{\partial b} L_i = \frac{\partial}{\partial b} (-t_i f_w(x_i)) = -t_i \frac{\partial}{\partial b} f_w(x_i)$

- w_1, w_2 等についても同様に偏微分

- $f_w(x_i)$ の偏微分

- $\frac{\partial}{\partial b} f_w(x_i) = \frac{\partial}{\partial b} (b + w_1 x_{i,1} + \dots + w_D x_{i,D}) = 1$

- $\frac{\partial}{\partial w_j} f_w(x_i) = \frac{\partial}{\partial w_j} (b + w_1 x_{i,1} + \dots + w_D x_{i,D}) = x_{i,j} \ (j = 1, \dots, D)$

パーセプトロンの更新式

$$\begin{bmatrix} b \\ w_1 \\ \vdots \\ w_D \end{bmatrix} \leftarrow \begin{bmatrix} b \\ w_1 \\ \vdots \\ w_D \end{bmatrix} - \eta(-t_i) \begin{bmatrix} 1 \\ x_{i,1} \\ \vdots \\ x_{i,D} \end{bmatrix}$$

ロジスティック回帰の更新式(場合分け)

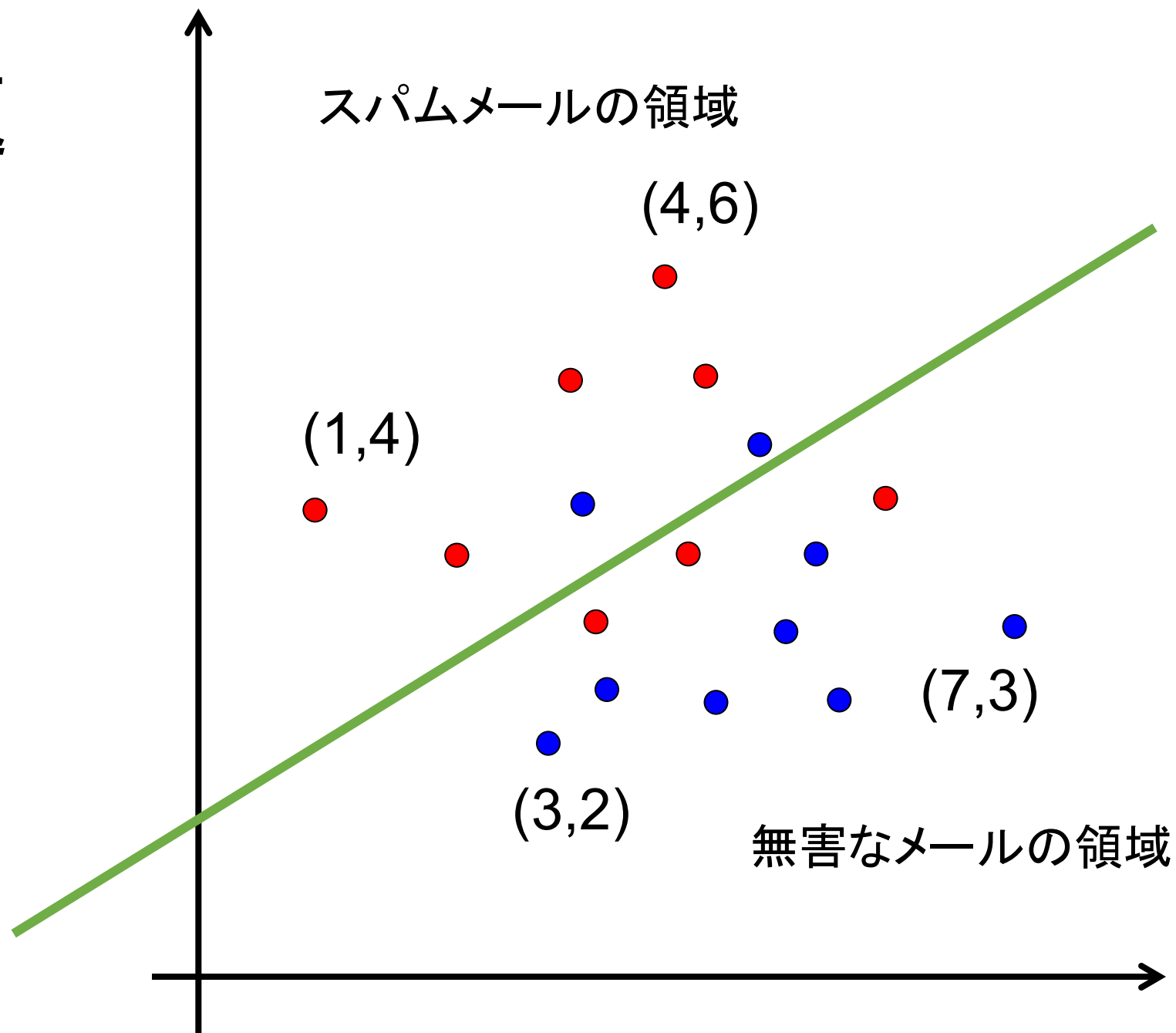
- $t_i = 1$ のとき

$$\begin{bmatrix} b \\ w_1 \\ \vdots \\ w_D \end{bmatrix} \leftarrow \begin{bmatrix} b \\ w_1 \\ \vdots \\ w_D \end{bmatrix} + \eta(1 - g(\mathbf{x}_i)) \begin{bmatrix} 1 \\ x_{i,1} \\ \vdots \\ x_{i,D} \end{bmatrix}$$

- $t_i = 0$ のとき

$$\begin{bmatrix} b \\ w_1 \\ \vdots \\ w_D \end{bmatrix} \leftarrow \begin{bmatrix} b \\ w_1 \\ \vdots \\ w_D \end{bmatrix} - \eta g(\mathbf{x}_i) \begin{bmatrix} 1 \\ x_{i,1} \\ \vdots \\ x_{i,D} \end{bmatrix}$$

例題



scikit-learnでパーセプトロンを実装

```
from sklearn import linear_model
```

```
clf = linear_model.Perceptron()
```

```
x = [[1,4], [4,6], [3,2], [7,3]]
```

```
y = [-1, -1, 1, 1]
```

```
clf.fit(x, y)
```

```
print(clf.coef_)
```

```
print(clf.intercept_)
```

SVMとパーセプトロン

- 線形関数 $f_{\mathbf{w}}(\mathbf{x}) = b + w_1x_1 + \cdots + w_Dx_D$ をモデルとして採用
- SVMで最小化する関数は下記のとおり

$$L(\mathbf{w}) = \min_{\mathbf{w}} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \max(0, 1 - t_i f_{\mathbf{w}}(\mathbf{x}_i))$$

- パーセプトロンの損失関数は下記のとおり

$$L(\mathbf{w}) = \sum_{i=1}^N \max(0, -t_i f_{\mathbf{w}}(\mathbf{x}_i))$$

SVMとパーセプトロンの違い

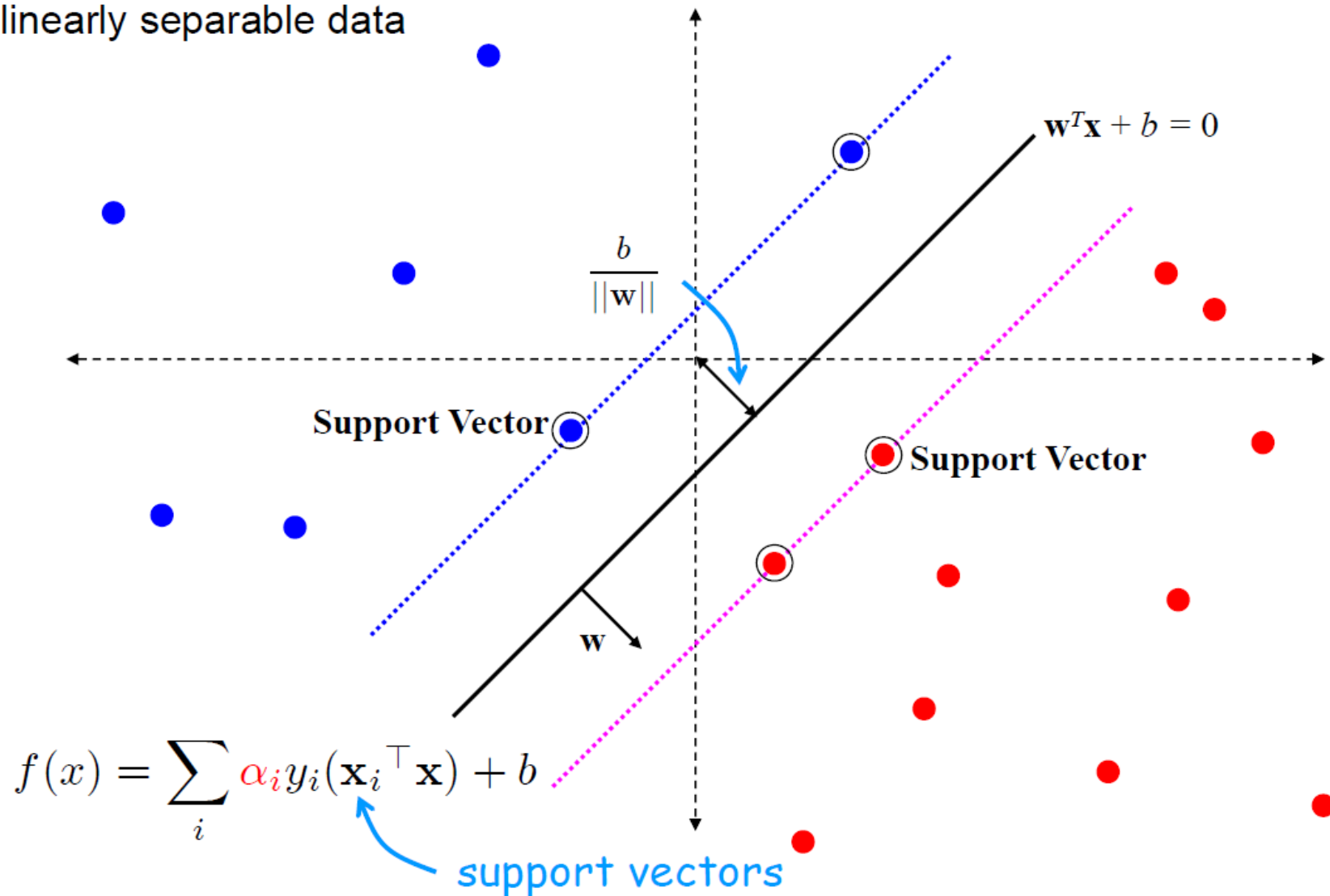
- SVMにはL2ノルムによる正則化の項がある
 - 正則化項と損失関数の和を最小にする
- 損失関数がパーセプトロンと似ている
- しかしパーセプトロンより「厳しい」損失関数になっている
 - $t_i f_w(x_i)$ の値が1以上にならないと、損失が0にならない
 - パーセプトロンでは、 $t_i f_w(x_i)$ の値が0以上になれば、損失は0になっていた
 - つまり、符号が一致する以上のことを要求する損失関数になっている

SVMで最小化する関数がこうなっている理由

- 説明は割愛します
- 概略だけ・・・
 - マージン最大化という考え方を導入する
 - データ集合を線形分離する(線形関数が表す超平面で二分割する)とき、超平面の両側の一定の幅(マージン)にはデータが入らないようにする
 - 現実にはデータ集合を線形分離できない場合もあるので、違反を許す
 - これをソフトマージンと呼ぶ
 - 以上のように考えると、さきほどの式が得られる

Support Vector Machine

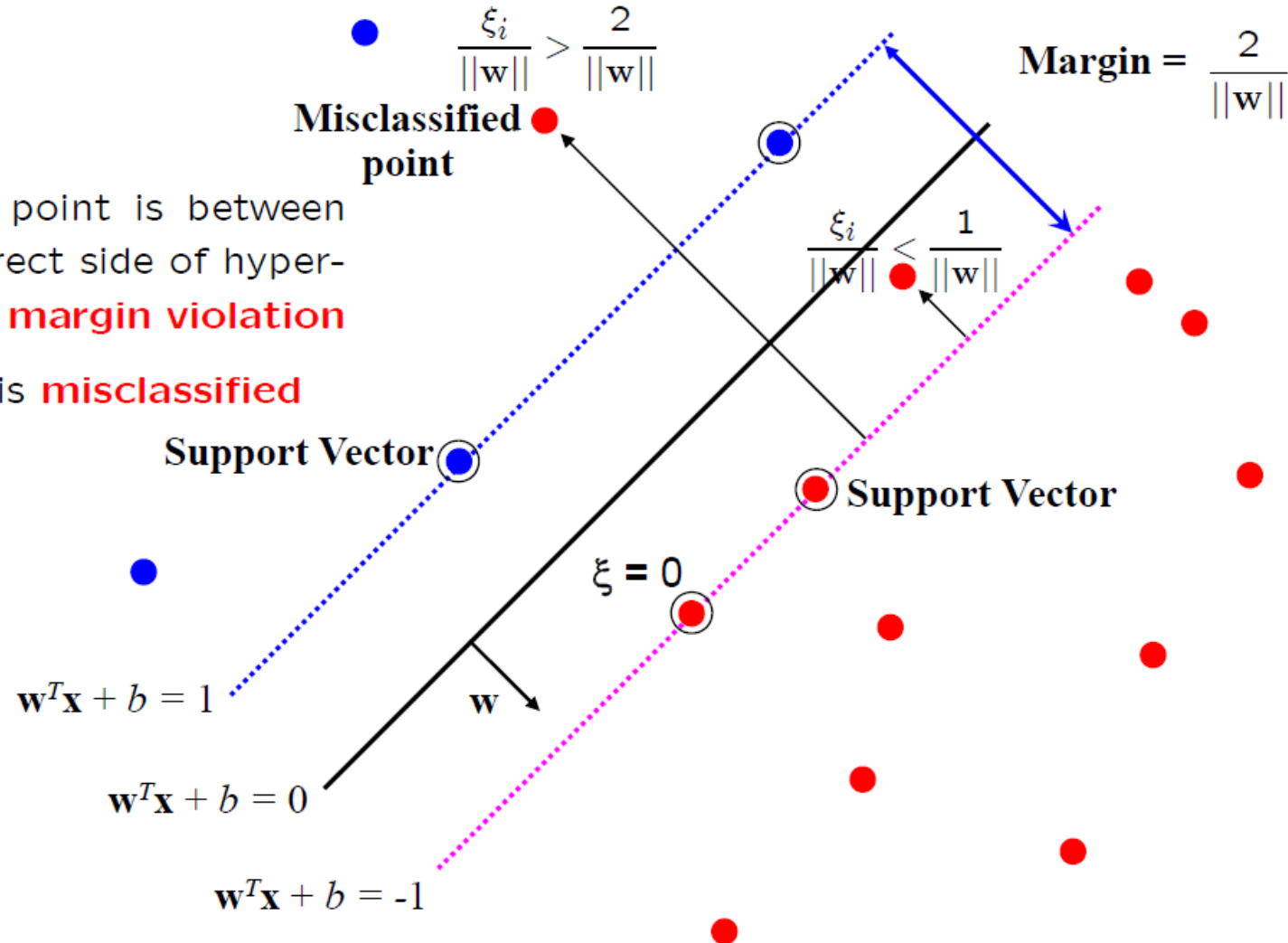
linearly separable data



Introduce “slack” variables

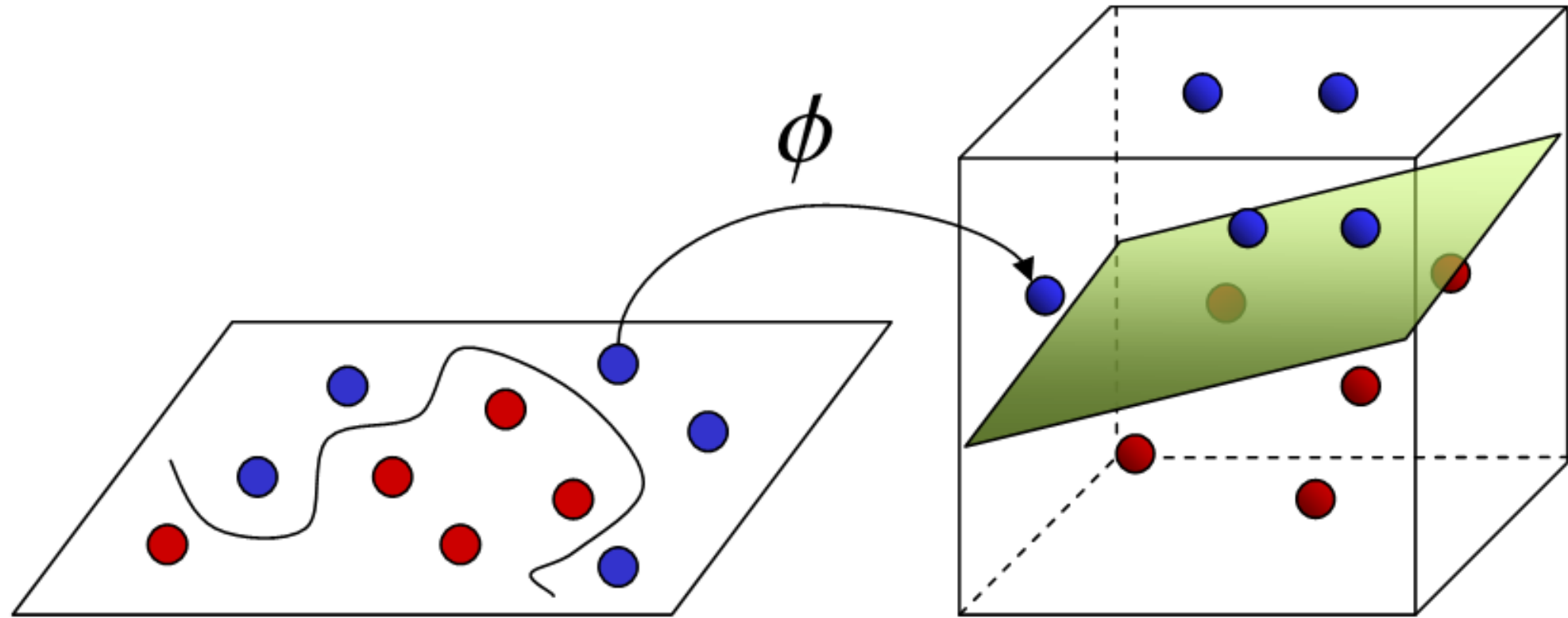
$$\xi_i \geq 0$$

- for $0 < \xi \leq 1$ point is between margin and correct side of hyper-plane. This is a **margin violation**
- for $\xi > 1$ point is **misclassified**



SVMのおもしろい使い方

- カーネル・トリックを使うと「曲がった面」でデータを二分割できる
 - 非線形なモデルになる
- カーネル・トリックでは入力データの次元をわざと上げる
 - 入力データを特徴空間と呼ばれる高次元の空間へマッピングする
 - その高次元空間で内積が計算できれば、その空間でSVMを使える
 - 高次元空間では線形分離。もとの入力空間に戻すと、「曲がった面」で分離。
- 「曲がった面」が使えるので「より自由な」分類ができる
 - 過学習の話は？



Input Space

Feature Space

<https://datascience.stackexchange.com/questions/17536/kernel-trick-explanation>

カーネル・トリック

- 元の入力ベクトル x を、関数 ϕ で変換して、高次元のベクトル $\phi(x)$ を得る
 - 変換した後のベクトル $\phi(x)$ が属する空間を「特徴空間」と呼ぶ。
- 元の空間にあるどの2つのベクトル x と z についても、 $\phi(x)$ と $\phi(z)$ との内積 $\kappa(x, z) = \langle \phi(x), \phi(z) \rangle$ のことをカーネルと呼ぶ
 - 関数 ϕ がどんな関数か分かっていたら、当然この計算はできる。
 - 関数 ϕ がどんな関数か分かていなくても、 $\kappa(x, z)$ が内積になっていればよい。
- ベクトルに関する多くの演算は内積だけを使って表わしなおせる
 - よって、内積の代わりにカーネル関数を使えば、特徴空間で様々な演算がおこなえる。

カーネルの例

$$\kappa(\mathbf{x}, \mathbf{z}) = 2x_1x_2z_1z_2 + x_1^2z_1^2 + x_2^2z_2^2$$

これは、以下のように2つのベクトルの内積として書き直せる。

$$\begin{aligned}\kappa(\mathbf{x}, \mathbf{z}) &= 2x_1x_2z_1z_2 + x_1^2z_1^2 + x_1^2z_2^2 \\ &= \langle (x_1^2, x_1^2, \sqrt{2}x_1x_2), (z_1^2, z_1^2, \sqrt{2}z_1z_2) \rangle\end{aligned}$$

つまり、特徴空間への写像として $\phi(\mathbf{x}) = (x_1^2, x_1^2, \sqrt{2}x_1x_2)$ というものを選べば、

$\kappa(\mathbf{x}, \mathbf{z})$ は特徴空間での内積になっている。だから、 $\kappa(\mathbf{x}, \mathbf{z})$ はカーネルになっている。

(\mathbf{x} と \mathbf{z} の関数なら何でもカーネルになるわけではない。)

カーネルの何が嬉しいか？

$$\begin{aligned}\kappa(\mathbf{x}, \mathbf{z}) &= 2x_1x_2z_1z_2 + x_1^2z_1^2 + x_1^2z_2^2 \\ &= \langle (x_1^2, x_2^2, \sqrt{2}x_1x_2), (z_1^2, z_2^2, \sqrt{2}z_1z_2) \rangle\end{aligned}$$

$\phi(x)$ を適当に決めれば、いくらでも上のようなカーネルは作れる。

しかし、空間の次元を上げるのと一緒に、計算量も増えてしまったら、あまり嬉しくない。

上のカーネル関数が嬉しいのは、計算の手間が元の空間での内積とほぼ同じだから。実際、

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle^2$$

こういうカーネルをSVMなどでよく使う。

内積の便利さ

- ベクトルに対する多くの演算は内積さえ分かれば実行できる
 - ベクトルの長さ: $\|x\| = \sqrt{\langle x, x \rangle}$
 - 二つのベクトル間の距離: $\|x - z\|^2 = \langle x, x \rangle - 2\langle x, z \rangle + \langle z, z \rangle$
 - 多くのベクトルの平均(重心)から特定のベクトルまでの距離
 - 多くのベクトルの平均(重心)自体は、それらのベクトルの内積では表わせない
 - 主成分分析
 - SVM
- 内積をカーネルで置き換えれば特徴空間での演算に変身する

例：RBFカーネル

$$\kappa(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{z}\|^2}{2\sigma^2}\right)$$

元の空間にある2つのベクトル \boldsymbol{x} と \boldsymbol{z} を、ある関数 ϕ で特徴空間に移し、ベクトル $\phi(\boldsymbol{x})$ と $\phi(\boldsymbol{z})$ を得る。

$\phi(\boldsymbol{x})$ と $\phi(\boldsymbol{z})$ の内積が上の関数 $\kappa(\boldsymbol{x}, \boldsymbol{z})$ に一致するような、そういう ϕ があることを示せる。

つまり、上の関数はカーネルになっている。

このカーネルは、元の空間で2つのベクトルの距離を計算するのと同じ手間で計算できる。

最後の課題

- MNISTデータの0の画像と0以外の画像を分類しよう
- scikit-learnのパーセプトロンやSVMを使うこと
 - パーセプトロンでは、penaltyとCの設定を変更して検証データ上で評価
 - SVM (sklearn.svm.LinearSVC) では、Cの設定を変更して検証データで評価
- テストデータを使った評価でロジスティック回帰と比較しよう