

自然言語処理プログラミング勉強会 9 - 識別学習の発展版

Graham Neubig
奈良先端科学技術大学院大学 (NAIST)

復習：識別学習とパーセプトロン

予測問題

x が与えられた時
 y を予測する

今回の例

- Wikipedia 記事の最初の 1 文が与えられた時
- その記事が人物についての記事かどうかを予測

与えられた情報

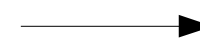
予測

Gonso was a Sanron sect priest (754-827) in the late Nara and early Heian periods.



Yes!

Shichikuzan Chigogataki Fudomyoo is a historical site located at Magura, Maizuru City, Kyoto Prefecture.



No!

- これはもちろん、2 値予測

数学的な定式化

$$\begin{aligned} y &= \text{sign}(\mathbf{w} \cdot \boldsymbol{\phi}(x)) \\ &= \text{sign}\left(\sum_{i=1}^I w_i \cdot \phi_i(x)\right) \end{aligned}$$

- x : 入力
- $\boldsymbol{\phi}(x)$: 素性関数のベクトル $\{\phi_1(x), \phi_2(x), \dots, \phi_I(x)\}$
- \mathbf{w} : 重みベクトル $\{w_1, w_2, \dots, w_I\}$
- y : 予測値、「yes」なら +1、「no」なら -1
 - $\text{sign}(v)$ は「 $v \geq 0$ 」の場合 +1、そうでない場合 -1

オンライン学習

```
create map  $w$ 
for / iterations
  for each labeled pair  $x, y$  in the data
     $\phi = \text{CREATE\_FEATURES}(x)$ 
     $y' = \text{PREDICT\_ONE}(w, \phi)$ 
    if  $y' \neq y$ 
       $\text{UPDATE\_WEIGHTS}(w, \phi, y)$ 
```

- つまり：
 - 各学習事例を分類してみる
 - 間違った答えを返す時に、重みを更新
- 様々なオンライン学習アルゴリズムが存在
 - 最もシンプルで実装しやすいのがパーセプトロン

パーセプトロンによる重み更新

$$w \leftarrow w + y \phi(x)$$

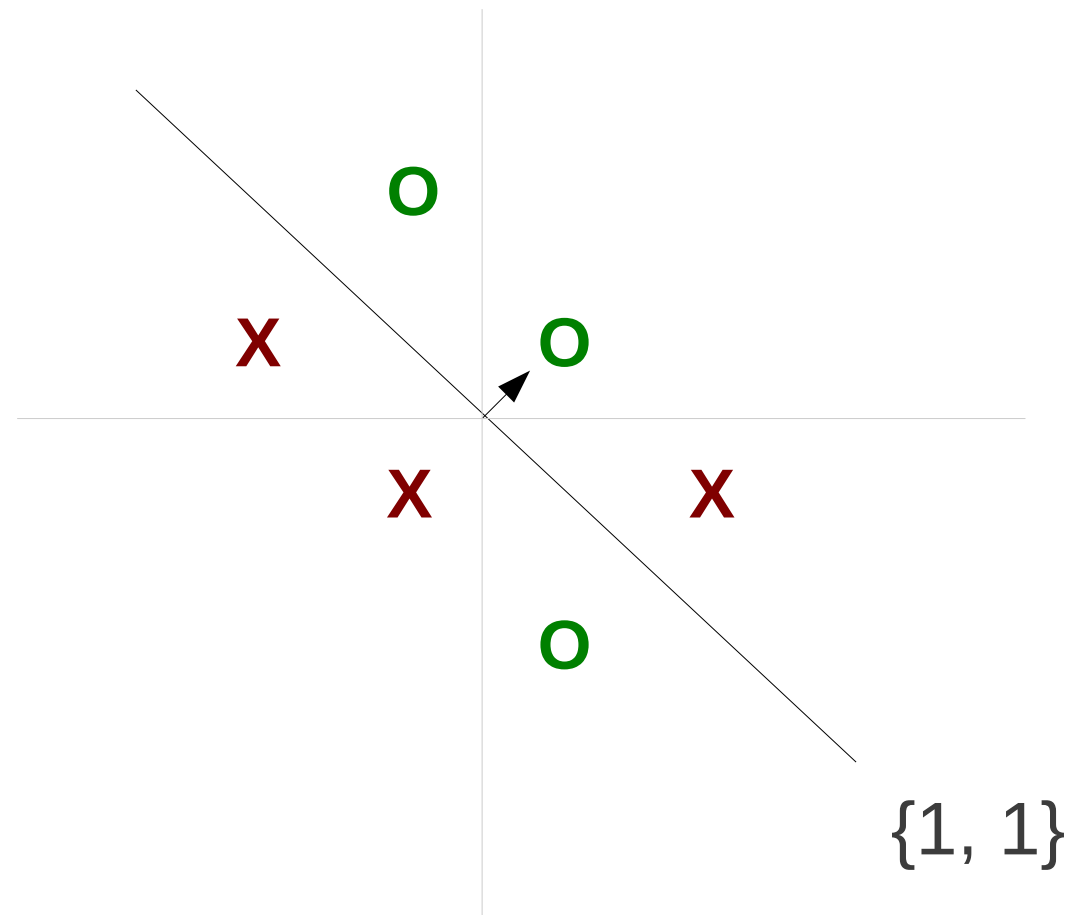
- つまり：
 - $y=1$ の場合、 $\phi(x)$ の素性の重みを増やす
 - 「yes」の事例の素性により大きな重みを
 - $y=-1$ の場合、 $\phi(x)$ の素性の重みを減らす
 - 「no」の事例により小さな重みを
- 更新のたびに、予測性能が向上！

```
UPDATE_WEIGHTS(w, phi, y)  
  for name, value in phi:  
    w[name] += value * y
```

平均化パーセプトロン

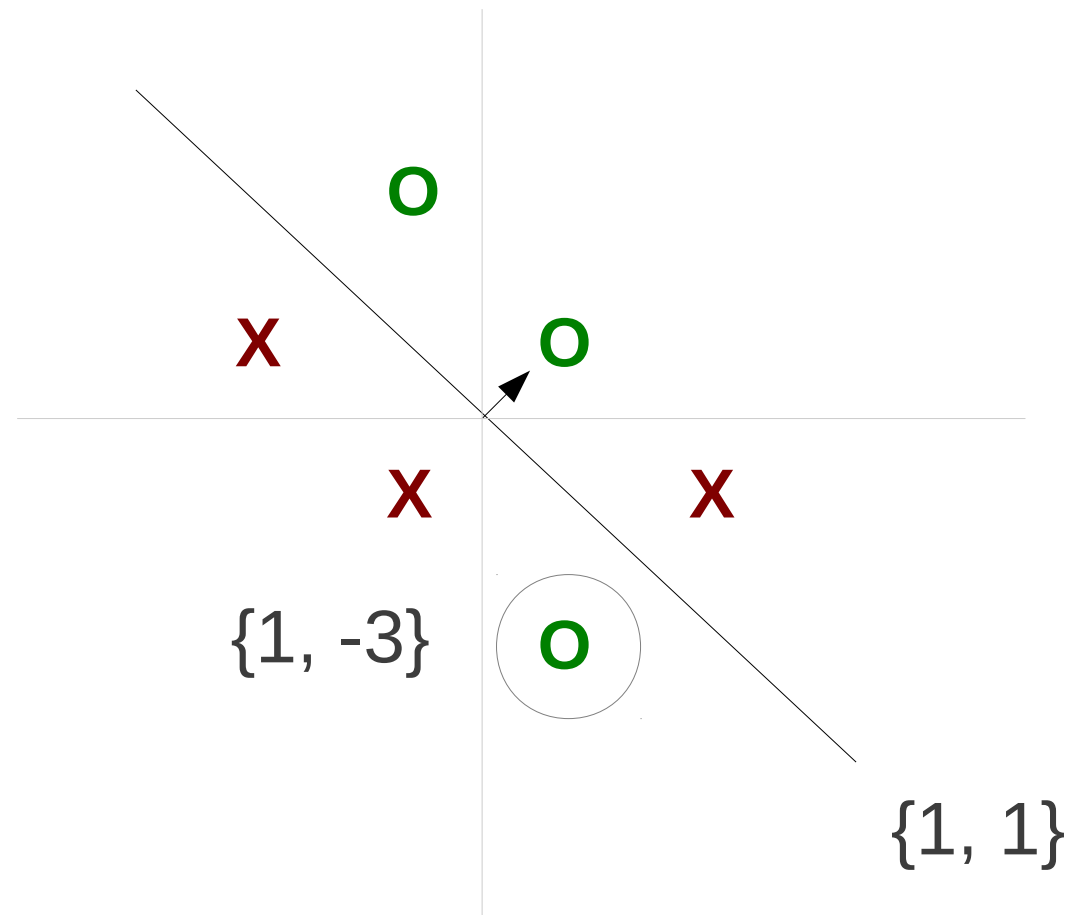
パーセプトロンの安定性問題

- 線形分離不可能なデータに対してパーセプトロンは不安定：



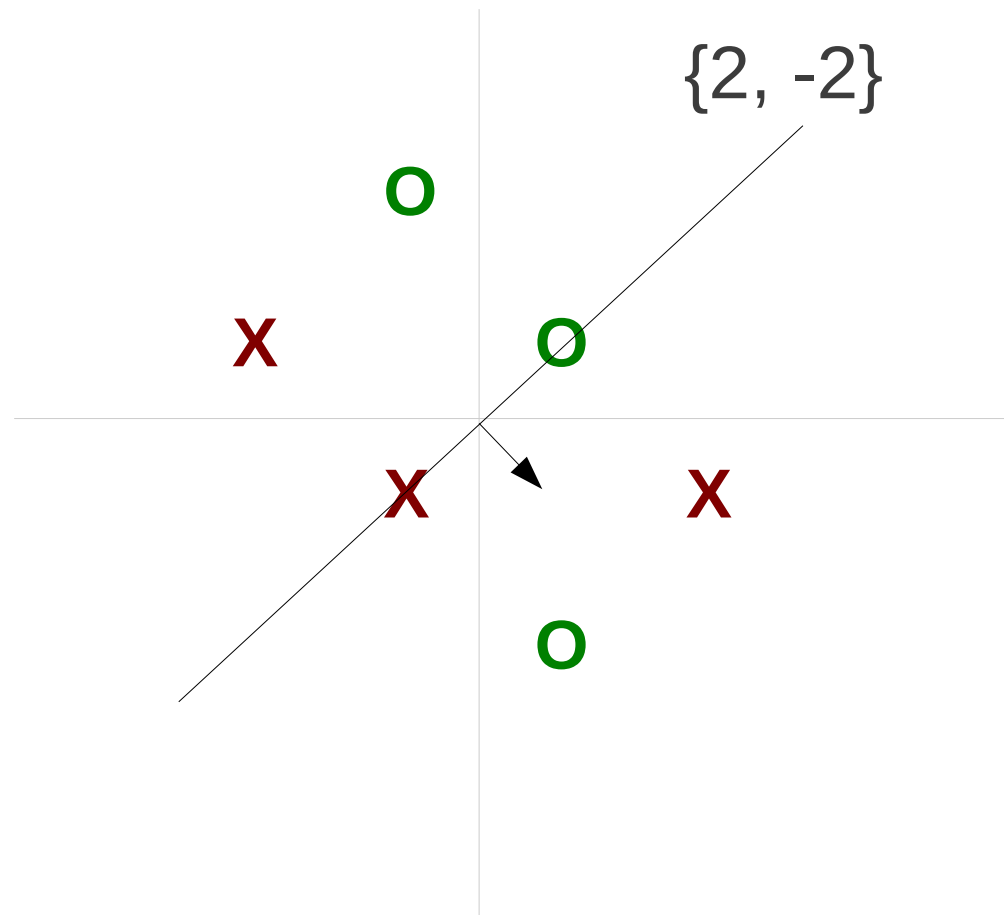
パーセプトロンの安定性問題

- 線形分離不可能なデータに対してパーセプトロンは不安定：



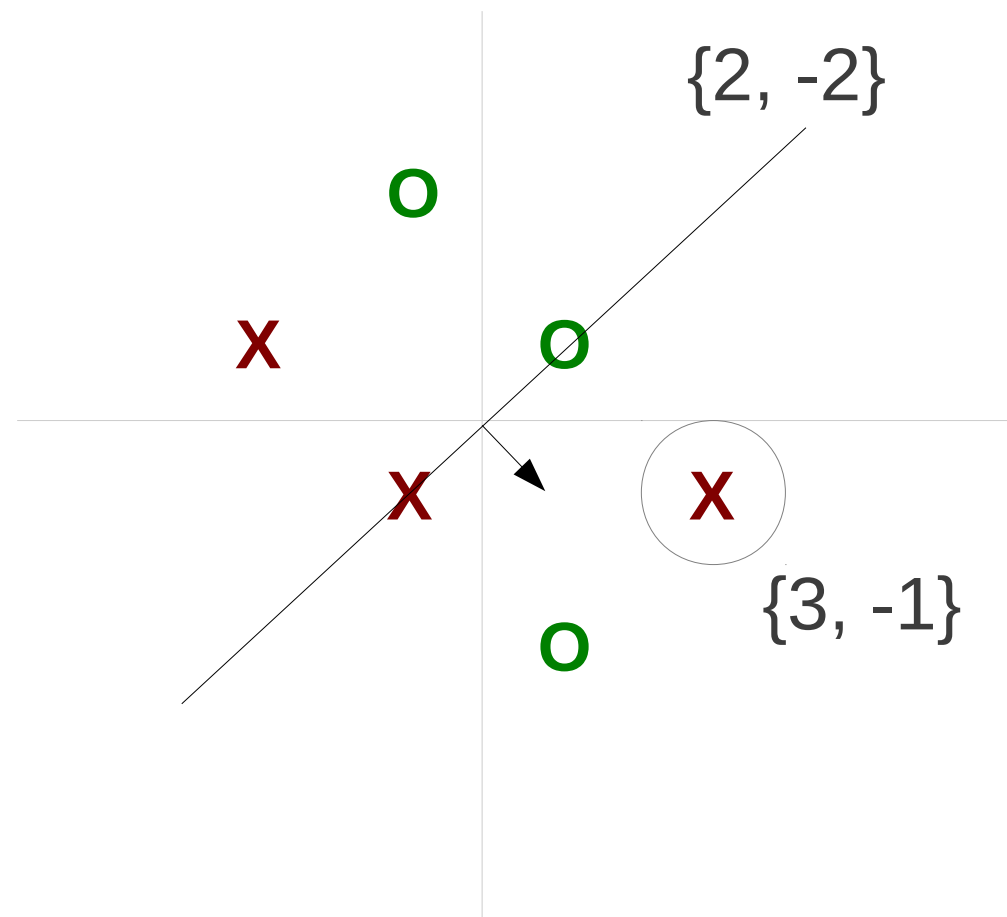
パーセプトロンの安定性問題

- 線形分離不可能なデータに対してパーセプトロンは不安定：



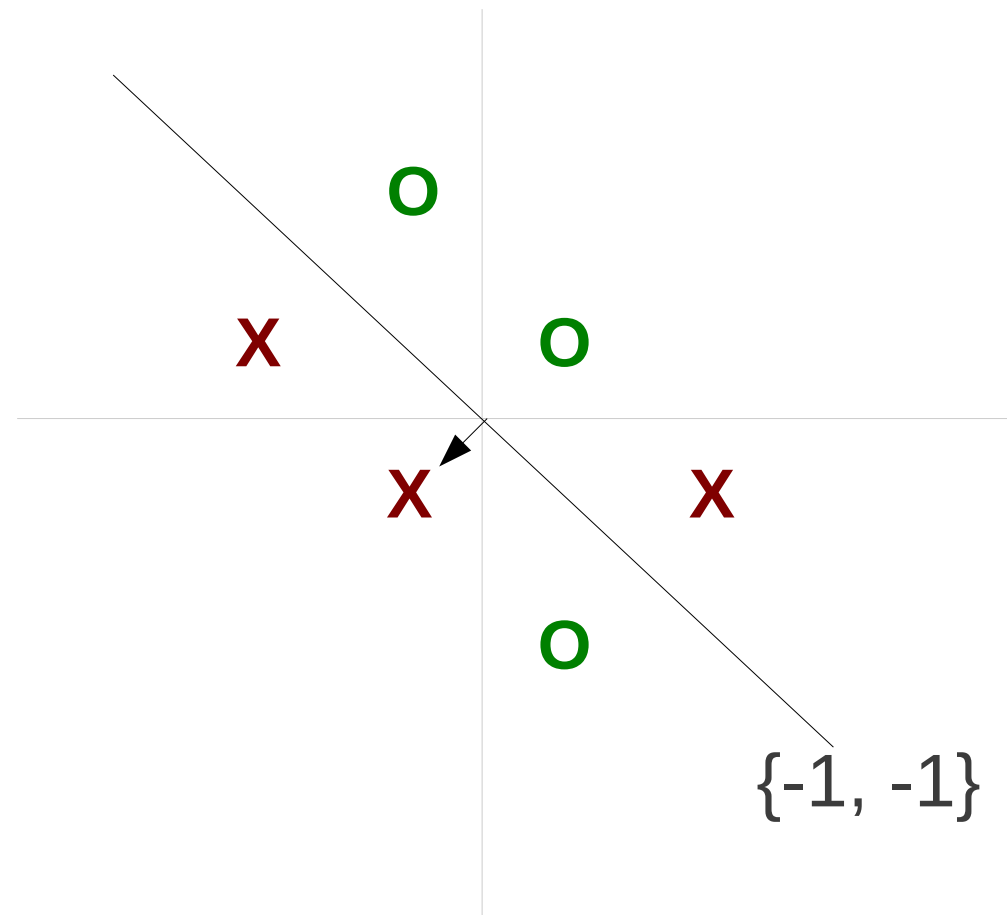
パーセプトロンの安定性問題

- 線形分離不可能なデータに対してパーセプトロンは不安定：



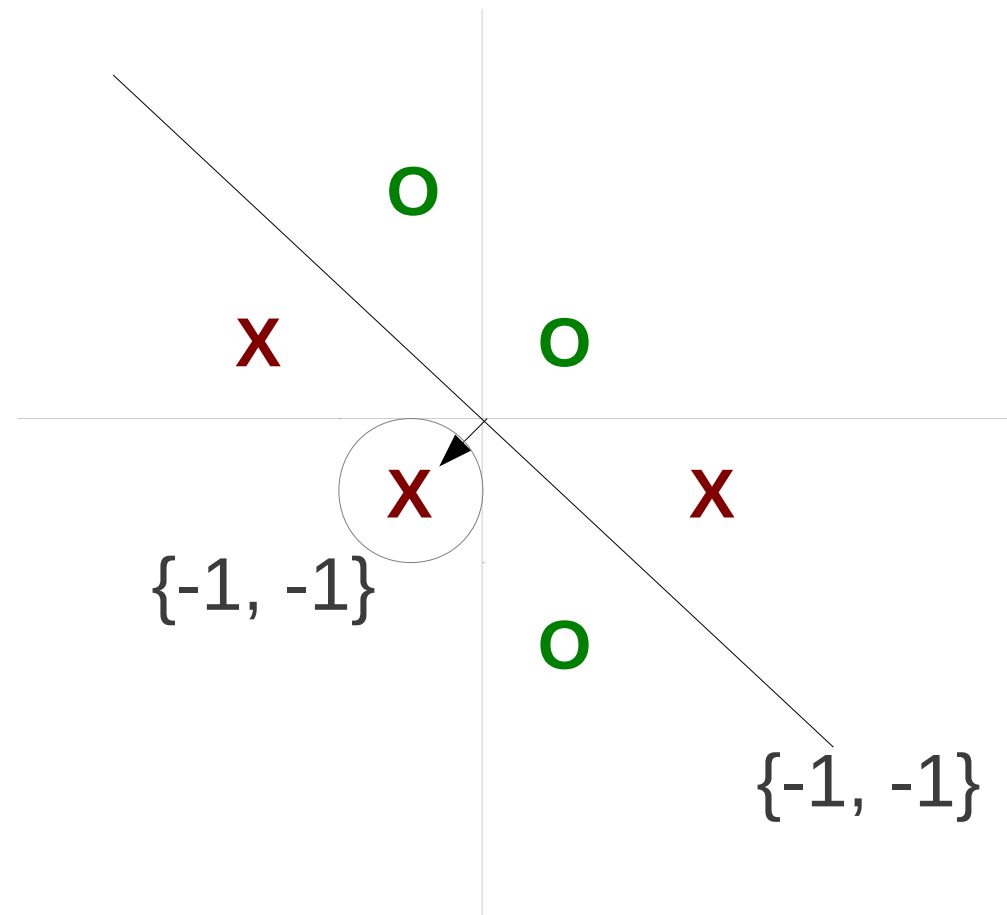
パーセプトロンの安定性問題

- 線形分離不可能なデータに対してパーセプトロンは不安定：



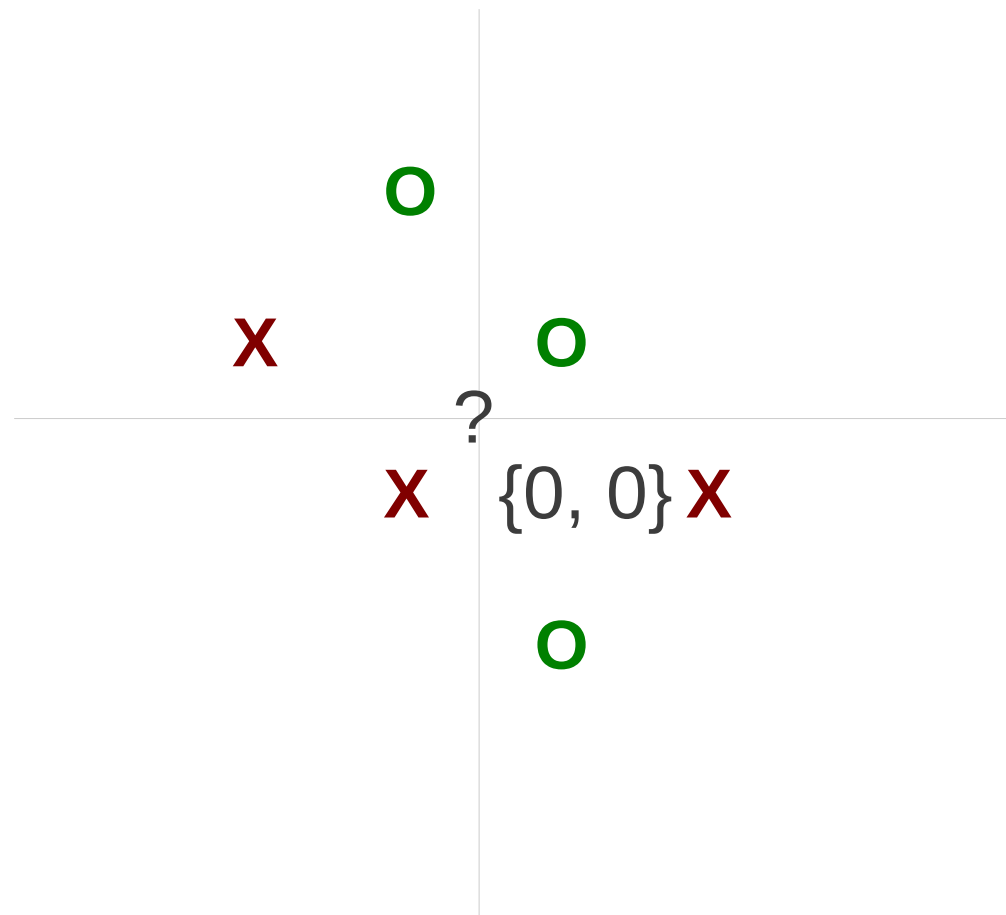
パーセプトロンの安定性問題

- 線形分離不可能なデータに対してパーセプトロンは不安定：



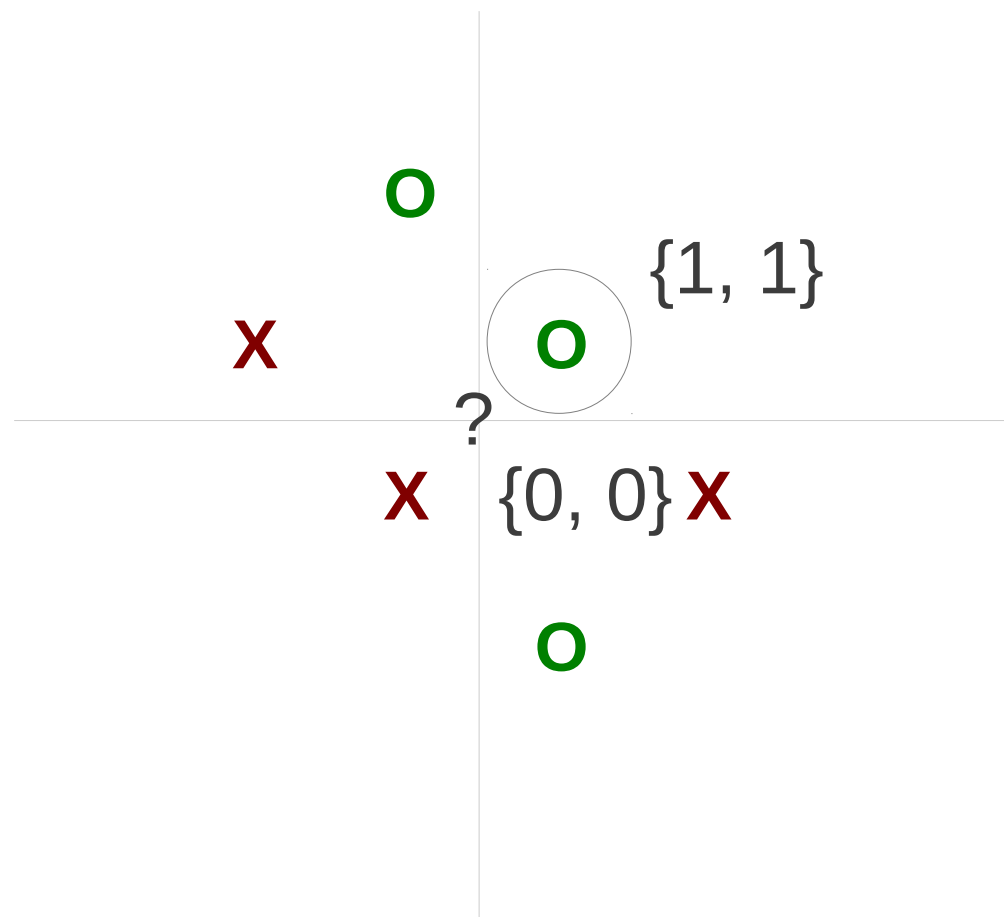
パーセプトロンの安定性問題

- 線形分離不可能なデータに対してパーセプトロンは不安定：



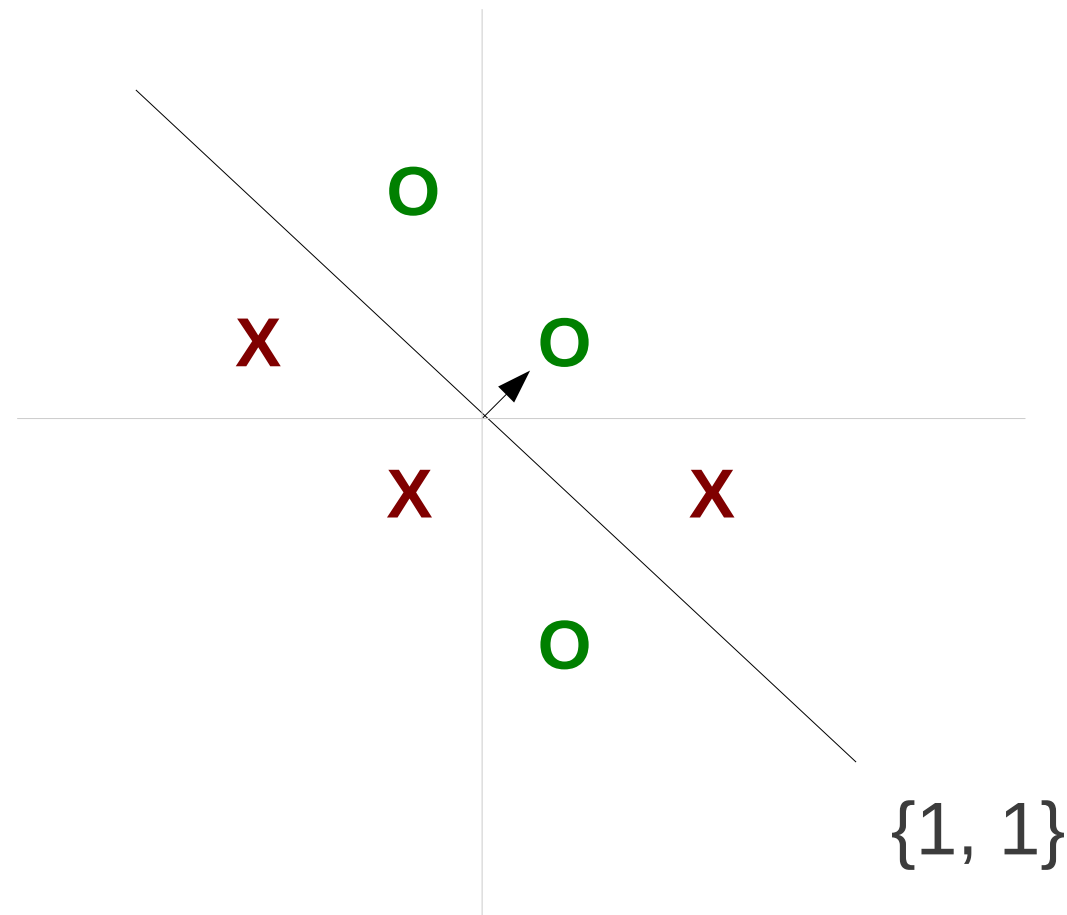
パーセプトロンの安定性問題

- 線形分離不可能なデータに対してパーセプトロンは不安定：



パーセプトロンの安定性問題

- 線形分離不可能なデータに対してパーセプトロンは不安定：



パーセプトロン学習の結果：

- 永遠に収束しない重みの羅列
- 学習を止めるタイミングで重みが大きく左右される

悪くない ...

非常に悪い!

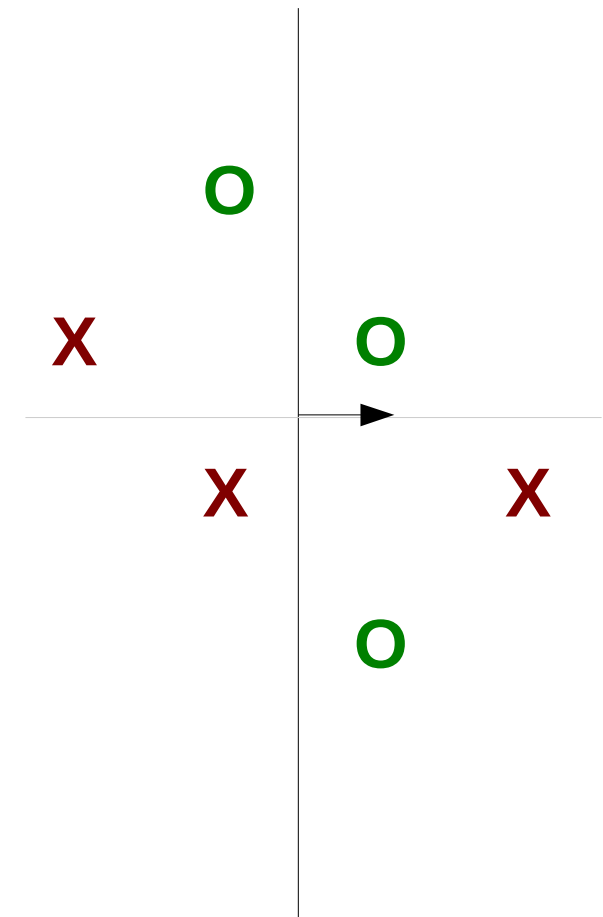
{1, 1}
{2, -2}
{-1, -1}
{0, 0}
{1, 1}
{1, 1}
{1, 1}
{2, -2}
{-1, -1}
{0, 0}
{1, 1}
{1, 1}
{1, 1}

...

平均化パーセプトロン

- 重みの平均を取るだけ！

$\text{average}(\begin{matrix} \{1, 1\} \\ \{2, -2\} \\ \{-1, -1\} \\ \{0, 0\} \\ \{1, 1\} \\ \{1, 1\} \\ \{1, 1\} \\ \{1, 1\} \\ \{2, -2\} \\ \{-1, -1\} \\ \{0, 0\} \\ \{1, 1\} \\ \{1, 1\} \\ \{1, 1\} \\ \dots \end{matrix}) \rightarrow \{0.67, 0\}$



平均化パーセプトロンのコード

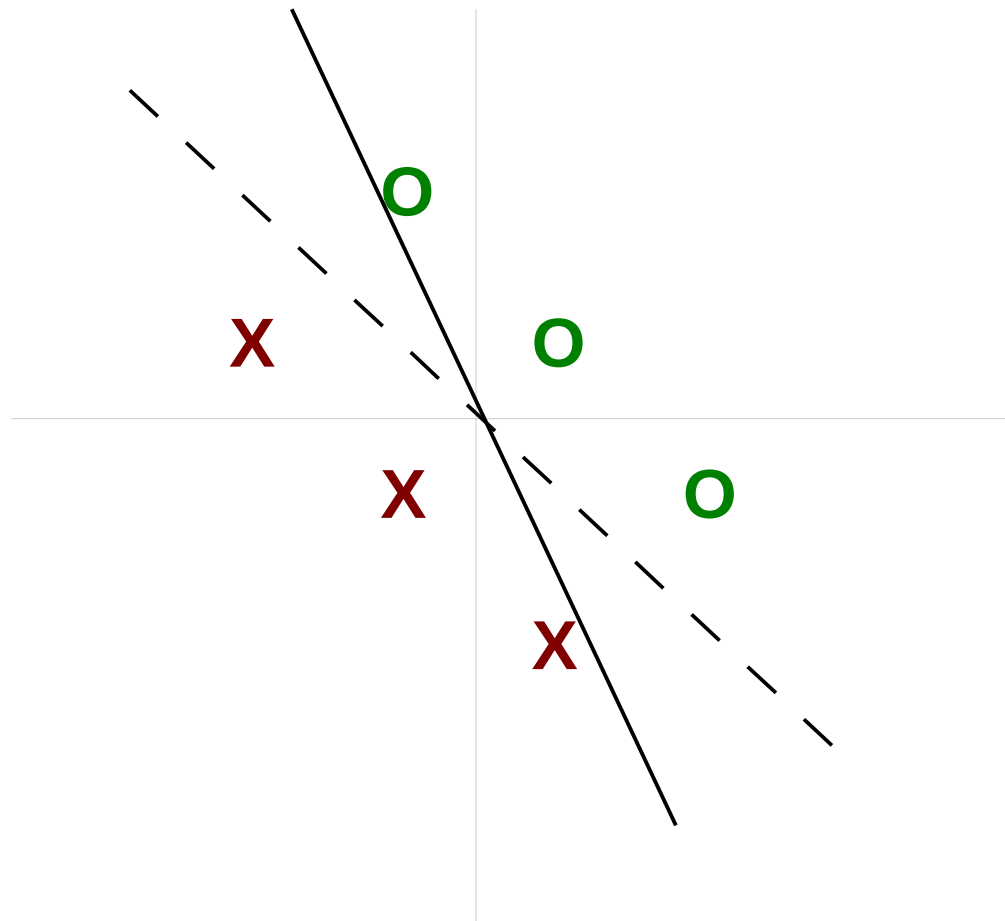
```
create map w
★ create map avg
★ set updates = 0
for l iterations
  for each labeled pair x, y in the data
    phi = CREATE_FEATURES(x)
    y' = PREDICT_ONE(w, phi)
    if y' != y
      UPDATE_WEIGHTS(w, phi, y)
★   updates += 1
★   avg = (avg * (updates-1) + w) / updates
```

- 全ての更新で平均も更新

マージンを考慮した識別

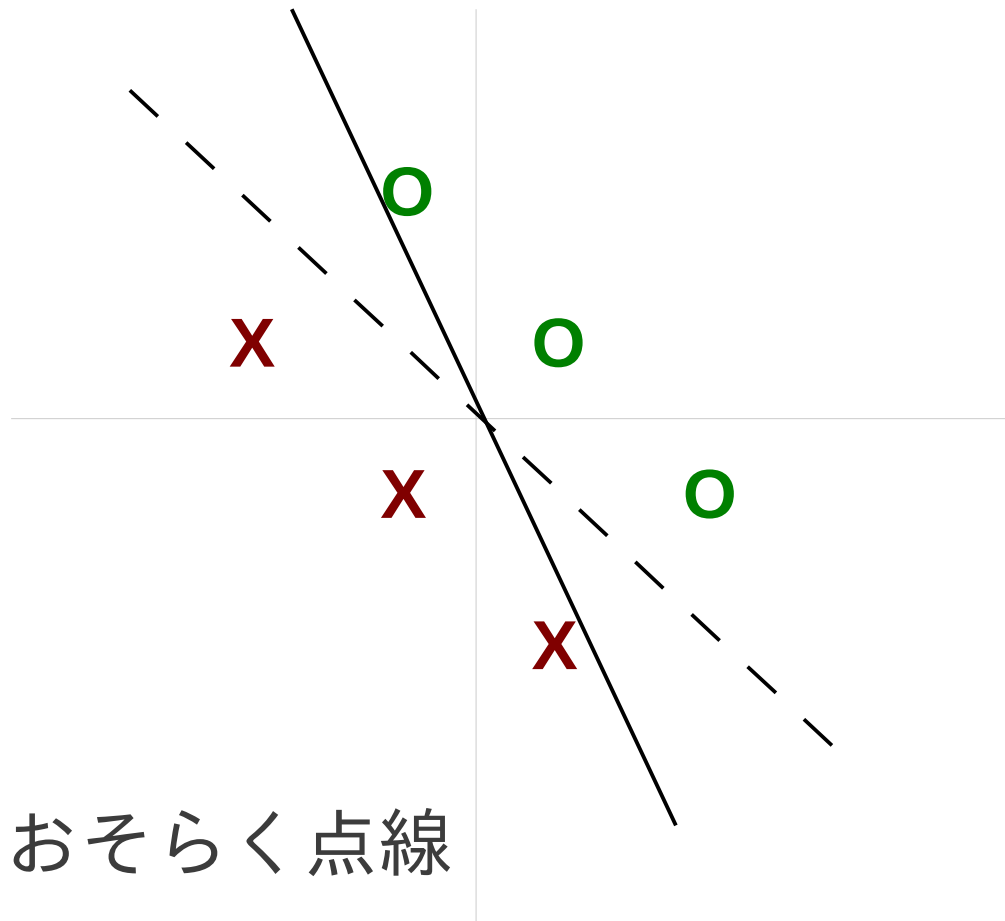
精度が同一の識別平面の選択方法

- どの線が良い？直線か点線か？



精度が同一の識別平面の選択方法

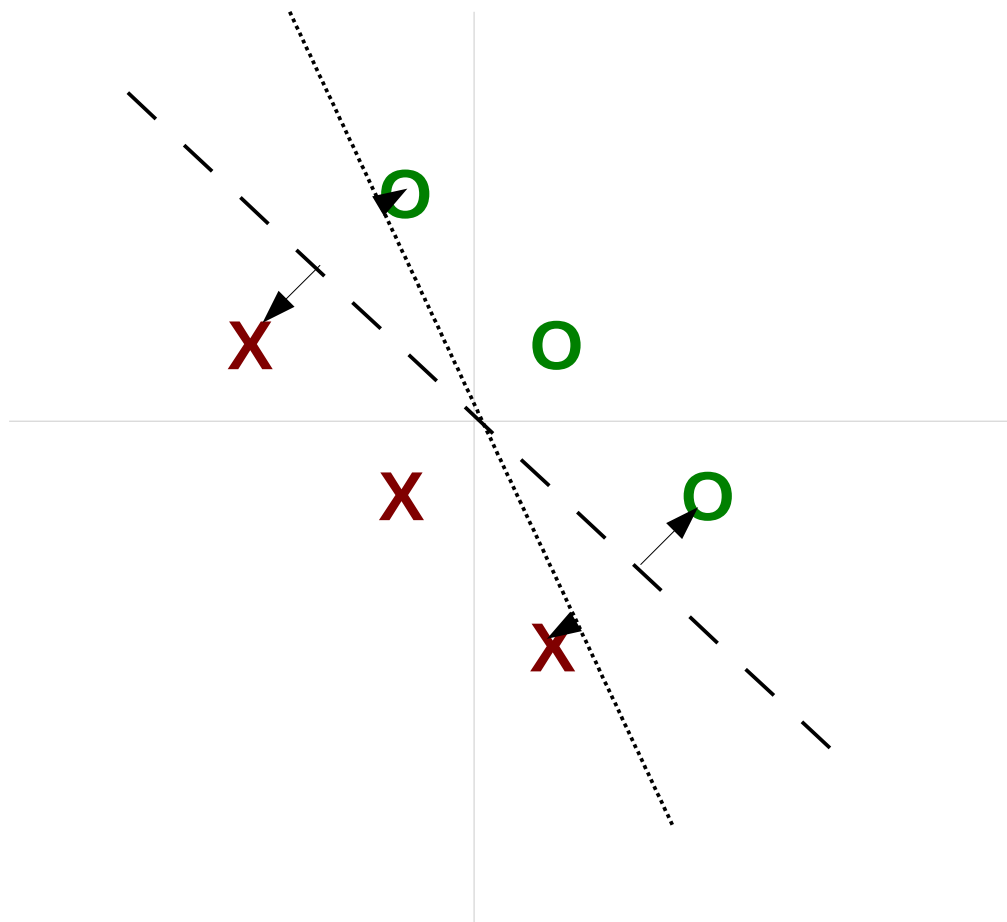
- どの線が良い？直線か点線か？



- 答え：おそらく点線
- 理由：マージンがより大きいから

マージンとは？

- 識別平面と事例の間の最短距離



サポートベクトルマシン (Support Vector Machine: SVM)

- 最も有名な分類器
 - ハードマージン：マージンを直接最大化
 - ソフトマージン：誤りを少々許す
- 多くの場合はバッチ学習を利用
 - バッチ学習：全ての例で統計を計算してから更新
→精度が少々高い、より安定した学習
 - オンライン学習：事例ごとに更新
シンプル、省メモリ、収束が速い
- SVM のバッチ学習についての資料
<http://disi.unitn.it/moschitti/material/Interspeech2010-Tutorial.Moschitti.pdf>
- バッチ学習ライブラリ：
LIBSVM, LIBLINEAR, SVMLite

マージンを用いたオンライン学習

- 誤りだけでなく、一定のマージン以内の場合でも更新

```
create map  $w$ 
for / iterations
  for each labeled pair  $x, y$  in the data
     $\phi = \text{CREATE\_FEATURES}(x)$ 
     $val = w * \phi * y$ 
    if  $val \leq \text{margin}$ 
       $\text{UPDATE\_WEIGHTS}(w, \phi, y)$ 
```



(正しい分類結果は常に $w * \phi * y > 0$)
 $\text{margin} = 0$ の場合はパーセプトロンと等しい

正則化

大きい分類器と小さい分類器

- 次の事例に対して：

-1 he saw a bird in the park
+1 he saw a robbery in the park

- どの分類器が良い？

分類器1

he +3
saw -5
a +0.5
bird -1
robbery +1
in +5
the -3
park -2

分類器2

bird -1
robbery +1

大きい分類器と小さい分類器

- 次の事例に対して：

-1 he saw a bird in the park
+1 he saw a robbery in the park

- どの分類器が良い？

分類器1

he +3
saw -5
a +0.5
bird -1
robbery +1
in +5
the -3
park -2

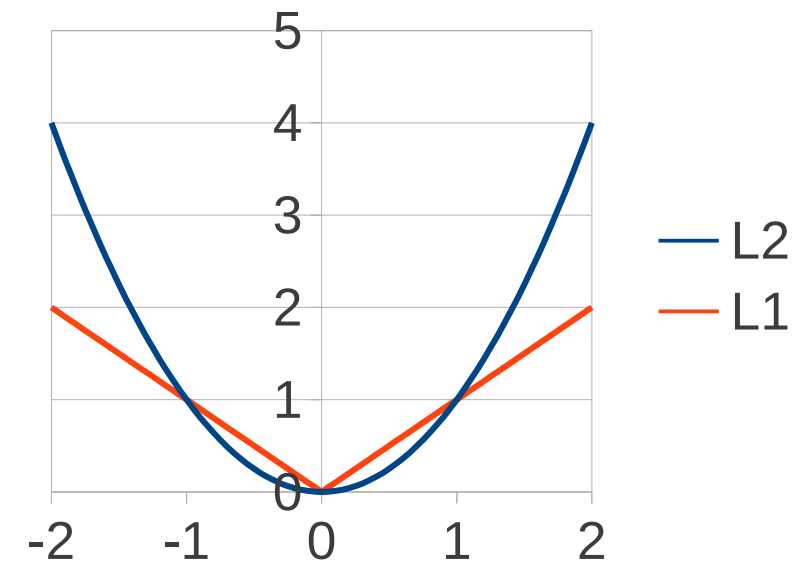
分類器2

bird -1
robbery +1

たぶん分類器2!
無関係な情報を使用しないから

正則化

- モデルに重みを追加することに対する罰則
- L2 正則化 :
 - 大きな重みに対して大きな罰則
小さな重みに対して小さな罰則
 - 精度が少々高い
- L1 正則化 :
 - 大小に関わらず同等の罰則
 - 多くの重みが 0 になるため
小さなモデルが学習可能



オンライン学習で L1 正則化

- 更新のたびに、重みから定数 c を引く

```
UPDATE_WEIGHTS( $w$ ,  $\phi$ ,  $y$ ,  $c$ )
```

```
★ for name, value in  $w$ :
```

```
★   if ABS(value) <  $c$ :
```

```
★        $w$ [name] = 0
```

```
★   else:
```

```
★        $w$ [name] -= SIGN(value) *  $c$ 
```

```
for name, value in  $\phi$ :
```

```
     $w$ [name] += value *  $y$ 
```

絶対値 < c ,
→ 0 に設定

値 > 0,
 c を引く

値 < 0,
 c を足す

例

- 順番に正則化 (R) 更新 (U) 正則化 (R) 更新 (U)

正則化係数 : $c=0.1$
 更新 : 1 回と 5 回に $\{1, 0\}$
 3 回に $\{0, -1\}$

	R_1	U_1	R_2	U_2	R_3	U_3
変更 :	$\{0, 0\}$	$\{\underline{1}, 0\}$	$\{-\underline{0.1}, 0\}$	$\{0, 0\}$	$\{-\underline{0.1}, 0\}$	$\{0, \underline{-1}\}$
w:	$\{0, 0\}$	$\{1, 0\}$	$\{0.9, 0\}$	$\{0.9, 0\}$	$\{0.8, 0\}$	$\{0.8, -1\}$
	R_4	U_4	R_5	U_5	R_6	U_6
変更 :	$\{-\underline{0.1}, \underline{0.1}\}$	$\{0, 0\}$	$\{-\underline{0.1}, \underline{0.1}\}$	$\{\underline{1}, 0\}$	$\{-\underline{0.1}, \underline{0.1}\}$	$\{0, 0\}$
w:	$\{0.7, -0.9\}$	$\{0.7, -0.9\}$	$\{0.6, -0.8\}$	$\{1.6, -0.8\}$	$\{1.5, -0.7\}$	$\{1.5, -0.7\}$

効率の問題

- 素性の数：
 - 各文 (ϕ): 10~1000
 - 全体 (w): 1,000,000~100,000,000

```
UPDATE_WEIGHTS( $w$ ,  $\phi$ ,  $y$ ,  $c$ )
  for  $name$ ,  $value$  in  $w$ :
    if ABS( $value$ ) <=  $c$ :
       $w[name]$  = 0
    else:
       $w[name]$  -= SIGN( $value$ ) *  $c$ 
  for  $name$ ,  $value$  in  $\phi$ :
     $w[name]$  +=  $value$  *  $y$ 
```

このループは
非常に遅い!

効率化のトリック

- 正則化は重みの使用時に行う！

```
GETW(w, name, c, iter, last)
  if iter != last[name]:          # 重みが古くなっている
    c_size = c * (iter - last[name])
    if ABS(w[name]) <= c_size:
      w[name] = 0
    else:
      w[name] -= SIGN(w[name]) * c_size
    last[name] = iter
  return w[name]
```

- これは「遅延評価」というやり方

正則化係数の選び方

- 正則化係数 c は大きな影響を及ぼす
- 大きい
 - モデルサイズ：小
 - 学習データ精度：低
 - 過学習：少
- 小さい
 - モデルサイズ：大
 - 学習データ精度：高
 - 過学習：多
- 交差検定などで様々な値を試して決定：
 - 例： 0.0001, 0.001, 0.01, 0.1, 1.0

演習課題

演習課題

- **実装** : L1 正則化とマージンで学習を行う train-svm
- **学習** `data-en/titles-en-train.labeled`
- **テスト** `data-en/titles-en-test.word`
- **比較** `c=0.0001` で精度を測り、パーセプトロンと比較
 - `script/grade-prediction.py data-en/titles-en-test.labeled your_answer`
- **チャレンジ** :
 - 様々な正則化係数を試す
 - 効率化を実装

Thank You!