自然言語処理プログラミング勉強会 6 - 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

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

- x: 入力
- **φ(x)**: 素性関数のベクトル $\{\varphi_1(x), \varphi_2(x), \ldots, \varphi_I(x)\}$
- **w**: 重みベクトル $\{w_1, w_2, \ldots, w_I\}$
- y: 予測値、「yes」なら +1、「no」なら -1
  - sign(v) は「v >= 0」の場合 +1、そうでない場合 -1

オンライン学習

```
create map w
for I 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)
```

- 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

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + y\,\varphi(x)$$

- つまり：
  - y=1 の場合、 φ(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、 φ(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
```

ロジスティック回帰

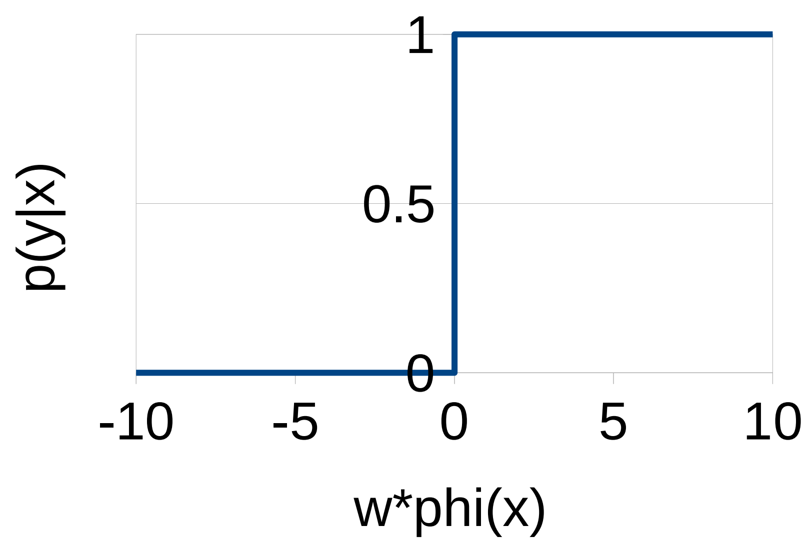
パーセプトロンと確率

- 応用によって確率がほしい時も： $P(y|x)$
  - 予測の信頼性を表現
  - 他システムとの統合が容易に
- しかし、単純なパーセプトロンは yes/no のみ

$$y = \text{sign}(w \cdot \varphi(x))$$

つまり：

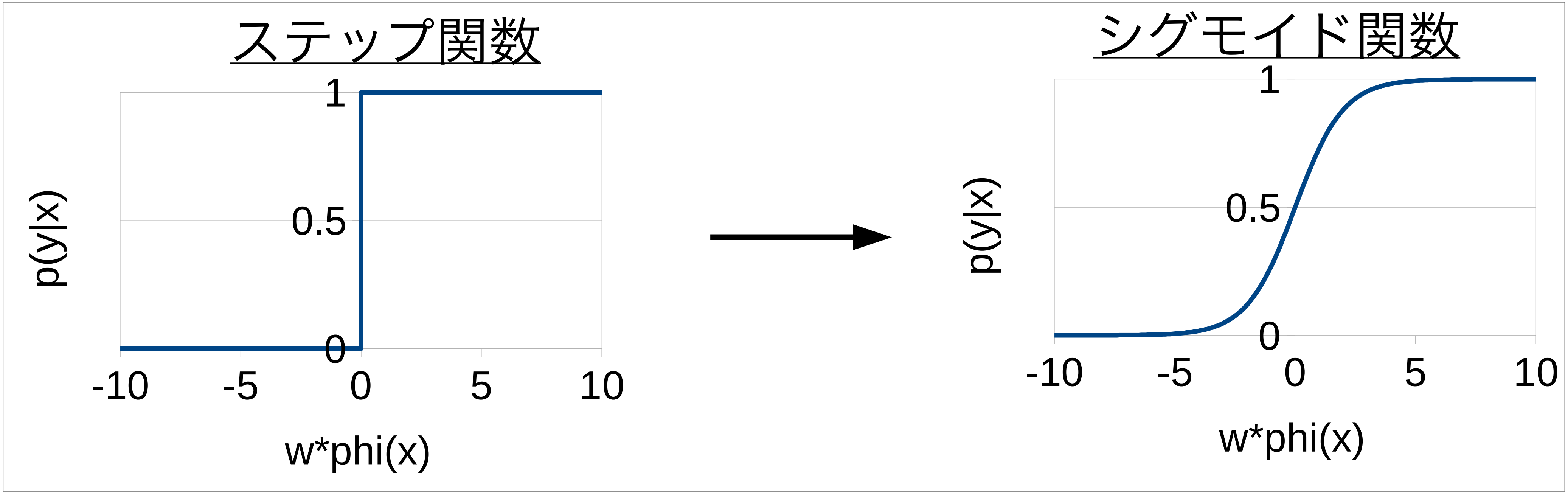
$$P(y=1|x) = 1 \text{ if } w \cdot \varphi(x) \geq 0$$
$$P(y=1|x) = 0 \text{ if } w \cdot \varphi(x) < 0$$



シグモイド関数（ロジスティック関数）

- シグモイド関数はステップ関数を柔らかくしたもの

$$P(y=1|x)=\frac{e^{w\cdot\varphi(x)}}{1+e^{w\cdot\varphi(x)}}$$



- 不確実性（確率）を考慮
- 微分可能

ロジスティック回帰 (logistic regression; LR)

- 条件付き尤度最大化基準で学習
- x が与えられたときの正解 y の条件付き尤度を最大化するパラメータ $\mathbf{w}$ を獲得

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmax}} \prod_i P(y_i \mid x_i; \mathbf{w})$$

- 解き方は？

確率的勾配降下法 (stochastic gradient descent; SGD)

- ロジスティック回帰を含む確率モデルのための学習アルゴリズム

```
w = 0
for I iterations
    for each labeled pair x, y in the data
        w += α * dP(y|x)/dw
```

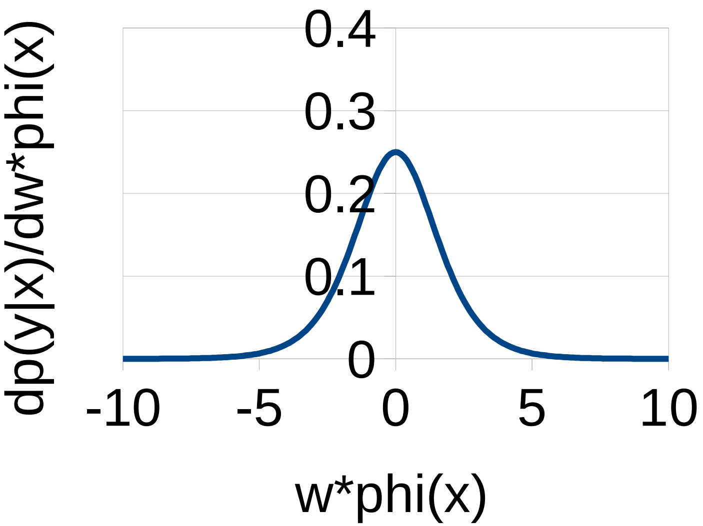
- つまり
  - 各学習例に対して勾配を計算（ y の確率が上がる方向）
  - その方向へ、学習率 α をかけた分だけ動かす

シグモイド関数の勾配

- 確率の微分

$$\frac{d}{dw} P(y=1|x) = \frac{d}{dw} \frac{e^{w\cdot\varphi(x)}}{1+e^{w\cdot\varphi(x)}}$$

$$= \varphi(x) \frac{e^{w\cdot\varphi(x)}}{(1+e^{w\cdot\varphi(x)})^2}$$



$$\frac{d}{dw} P(y=-1|x) = \frac{d}{dw} \left(1 - \frac{e^{w\cdot\varphi(x)}}{1+e^{w\cdot\varphi(x)}}\right)$$

$$= -\varphi(x) \frac{e^{w\cdot\varphi(x)}}{(1+e^{w\cdot\varphi(x)})^2}$$

例：最初の更新

- α=1 を設定、 **w=0** と初期化

x = A site , located in Maizuru , Kyoto　　　y = -1

$$\mathbf{w}\cdot\varphi(x)=0 \qquad \frac{d}{d\,w}P(y=-1|x) = -\frac{e^{0}}{(1+e^{0})^{2}}\varphi(x)$$

$$= -0.25\,\varphi(x)$$

$$\mathbf{w} \leftarrow \mathbf{w} + -0.25\,\varphi(x)$$

$w_{\text{unigram “Maizuru”}} = -0.25$

$w_{\text{unigram “,”}} = -0.5$

$w_{\text{unigram “in”}} = -0.25$

$w_{\text{unigram “Kyoto”}} = -0.25$

$w_{\text{unigram “A”}} = -0.25$

$w_{\text{unigram “site”}} = -0.25$

$w_{\text{unigram “located”}} = -0.25$

例： 2 回目の更新

x = Shoken , monk born in Kyoto y = 1

-0.5 -0.25 -0.25

$$\mathbf{w}\cdot\varphi(x)=-1 \qquad \frac{d}{dw}P(y=1|x) = \frac{e^{1}}{(1+e^{1})^{2}}\varphi(x) = 0.196\,\varphi(x)$$

$$\mathbf{w} \leftarrow \mathbf{w} + 0.196\,\varphi(x)$$

$w_{\text{unigram “Maizuru”}}$ = -0.25
$w_{\text{unigram “,”}}$ = -0.304
$w_{\text{unigram “in”}}$ = -0.054
$w_{\text{unigram “Kyoto”}}$ = -0.054

$w_{\text{unigram “A”}}$ = -0.25
$w_{\text{unigram “site”}}$ = -0.25
$w_{\text{unigram “located”}}$ = -0.25

$w_{\text{unigram “Shoken”}}$ = 0.196
$w_{\text{unigram “monk”}}$ = 0.196
$w_{\text{unigram “born”}}$ = 0.196

学習率の決定

- 学習率 α をどうやって設定？
- 徐々に減らしていくのが基本（収束が保証）

$$\alpha = \frac{1}{C + t}$$

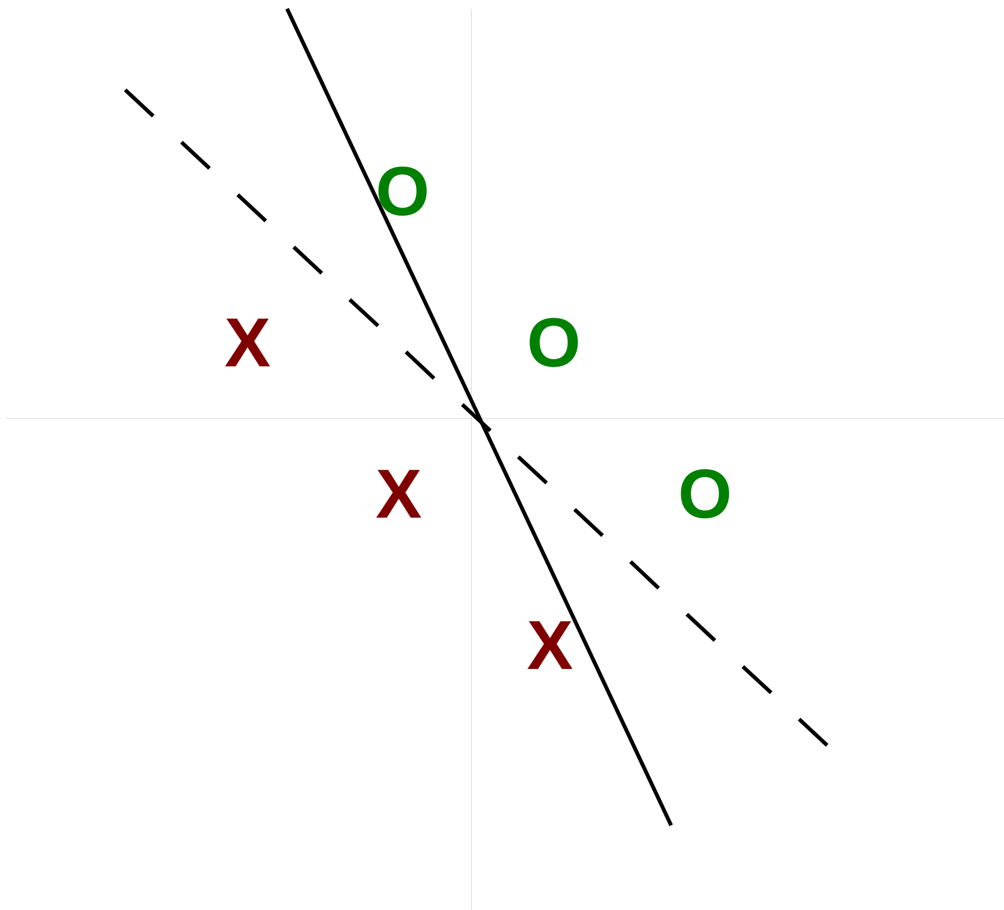
パラメータ　　　　サンプル数

- または：学習データに含まれない開発データを用意し、開発データの尤度が下がったら学習率を減らす

マージンを考慮した識別

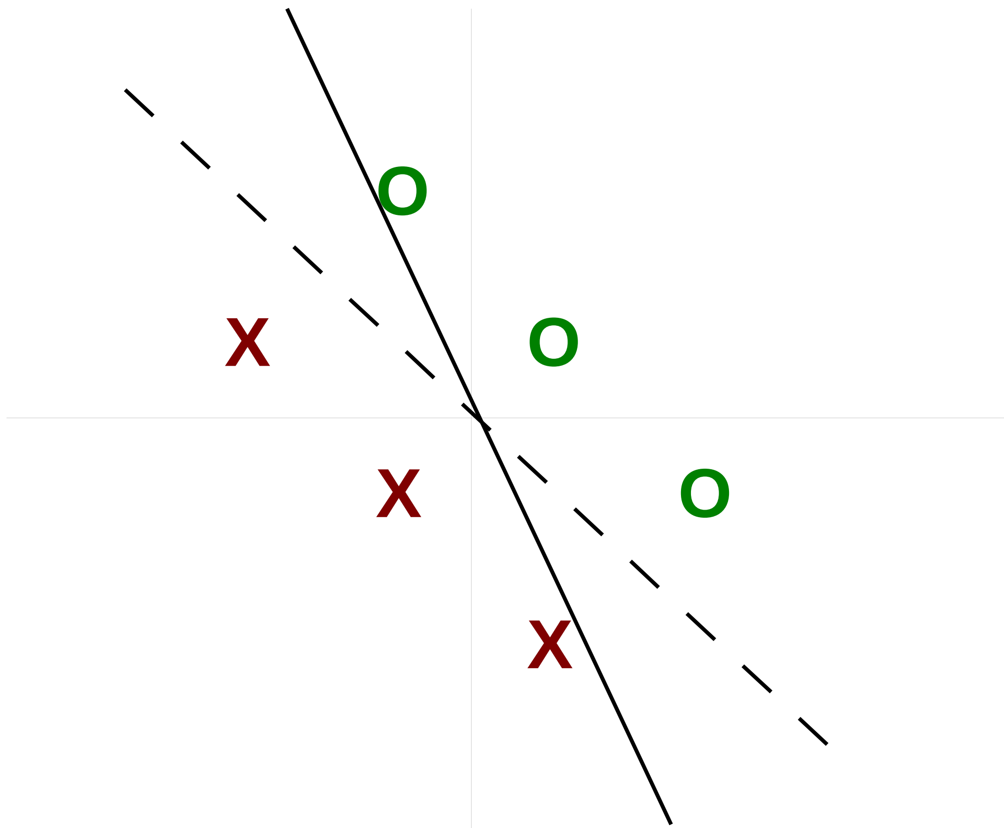
精度が同一の識別平面の選択方法

- 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

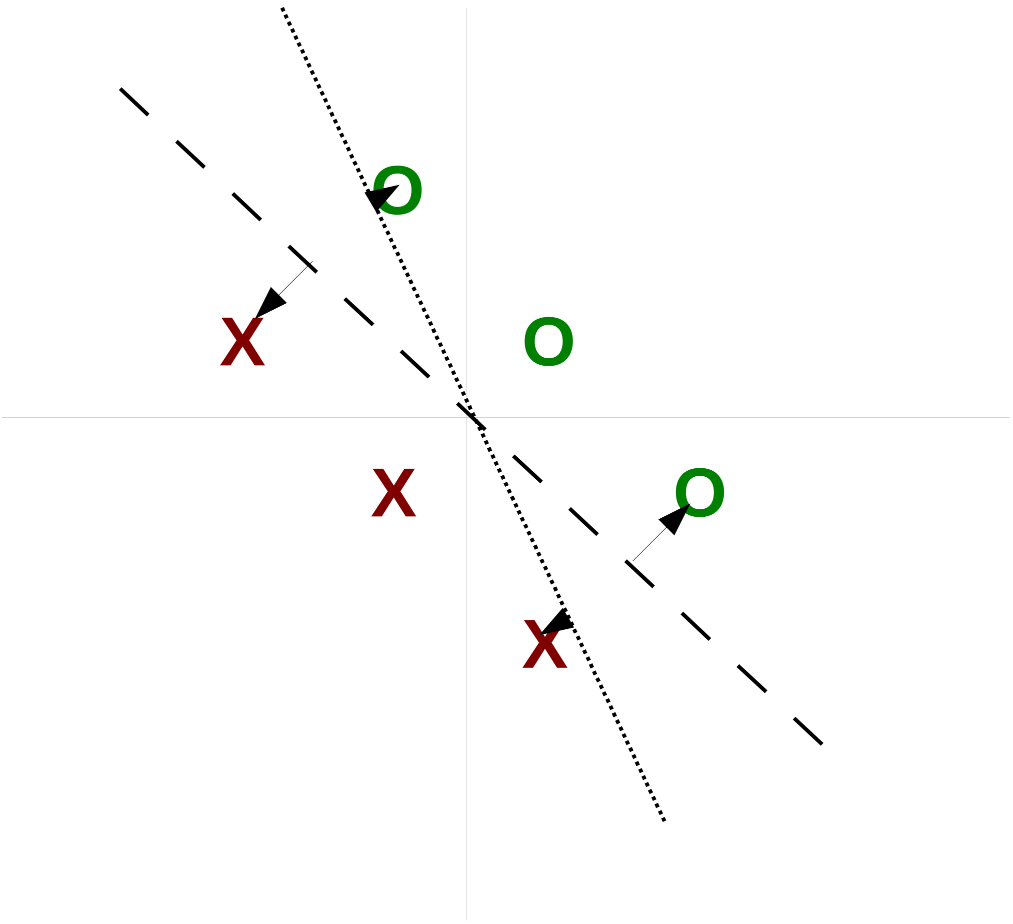
- どの線が良い？直線か点線か？



- 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 I iterations
    for each labeled pair x, y in the data
        phi = create_features(x)
        val = w * phi * y
★       if val <= margin
            update_weights(w, phi, y)
```

(正しい分類結果は常に w * phi * y > 0)
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	分類器 2
he +3	bird -1
saw -5	robbery +1
a +0.5	
bird -1	
robbery +1	
in +5	
the -3	
park -2	

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

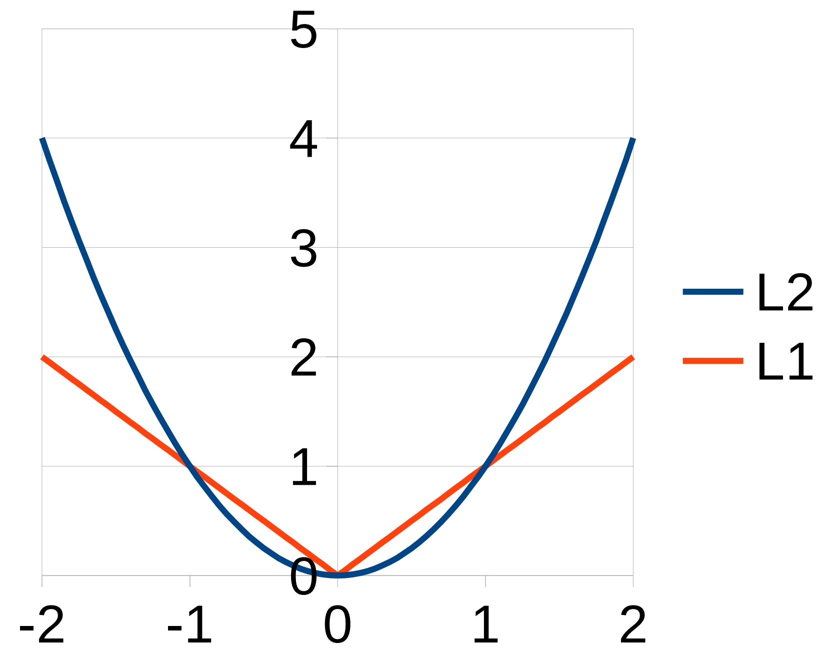
- どの分類器が良い？

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

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}	{1, 0}	{-0.1, 0}	{0, 0}	{-0.1, 0}	{0, -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 :	{-0.1, 0.1}	{0, 0}	{-0.1, 0.1}	{1, 0}	{-0.1, 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 (phi): 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!