自然言語処理プログラミング勉強会 3 - パーセプトロンアルゴリズム

Graham Neubig
奈良先端科学技術大学院大学 (NAIST)

予測問題

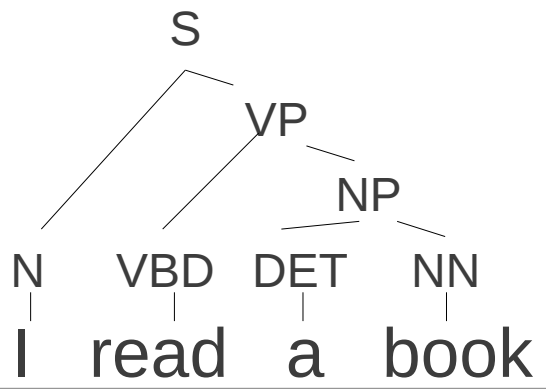
x が与えられた時
y を予測する

予測問題

x が与えられた時　　y を予測する

本のレビュー	「良い」評価なのか？	2 値予測
Oh, man I love this book!	yes	（選択肢が２つ）
This book is so boring...	no	

ツイート	書かれた言語	多クラス予測
On the way to the park!	English	（選択肢が数個）
公園に行くなう！	Japanese	

文	構文木	構造化予測
I read a book		（選択肢が膨大）

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

予測方法

どうやって予測するか

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

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

どうやって予測するか

「 priest 」を含む→
人物の可能性が高い

「 (<#>-<#>) 」を含む →
人物の可能性が高い

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

「 site 」を含む →
人物の可能性
が低い

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

「 Kyoto Prefecture 」を含む →
人物の可能性が低い

様々な情報を組み合わせる

- 予測に利用する情報は「素性」と呼ぶ

「 priest 」を含む　「 (<#>-<#>) 」を含む
「 site 」を含む　「 Kyoto Prefecture 」を含む

- 書く素性に重みが振られており、「 yes 」の可能性が高ければ高いほど重みが正の値になる

$w_{\text{contains “priest”}} = 2$　$w_{\text{contains “(<#>-<#>)”}} = 1$
$w_{\text{contains “site”}} = -3$　$w_{\text{contains “Kyoto Prefecture”}} = -1$

- 新しい事例が入ってきたら、重みの和で答えを予測

Kuya (903-972) was a priest born in Kyoto Prefecture. → 2 + -1 + 1 = 2

- 重み付き和が 0 以上であれば「 yes 」そうでなければ「 no 」

数学で言うと

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

- 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

素性関数の例： 1-gram 素性

- 「事例において、ある単語が何回現れるか？」

x = A site , located in Maizuru , Kyoto

$\varphi_{\text{unigram “A”}}(x) = 1$ $\varphi_{\text{unigram “site”}}(x) = 1$ $\varphi_{\text{unigram “,”}}(x) = 2$

$\varphi_{\text{unigram “located”}}(x) = 1$ $\varphi_{\text{unigram “in”}}(x) = 1$

$\varphi_{\text{unigram “Maizuru”}}(x) = 1$ $\varphi_{\text{unigram “Kyoto”}}(x) = 1$

$\varphi_{\text{unigram “the”}}(x) = 0$ $\varphi_{\text{unigram “temple”}}(x) = 0$ 残りはすべて 0

…

- 便宜のため、素性 ID(φ_1) の代わりに、素性の名前 ($\varphi_{\text{unigram “A”}}$) を利用

重み付き和の計算

x = A site , located in Maizuru , Kyoto

$\varphi_{\text{unigram “A”}}(x)$	= 1		$w_{\text{unigram “a”}}$	= 0		0 +
$\varphi_{\text{unigram “site”}}(x)$	= 1		$w_{\text{unigram “site”}}$	= -3		-3 +
$\varphi_{\text{unigram “located”}}(x)$	= 1		$w_{\text{unigram “located”}}$	= 0		0 +
$\varphi_{\text{unigram “Maizuru”}}(x)$	= 1		$w_{\text{unigram “Maizuru”}}$	= 0		0 +
$\varphi_{\text{unigram “,”}}(x)$	= 2	*	$w_{\text{unigram “,”}}$	= 0	=	0 +
$\varphi_{\text{unigram “in”}}(x)$	= 1		$w_{\text{unigram “in”}}$	= 0		0 +
$\varphi_{\text{unigram “Kyoto”}}(x)$	= 1		$w_{\text{unigram “Kyoto”}}$	= 0		0 +
$\varphi_{\text{unigram “priest”}}(x)$	= 0		$w_{\text{unigram “priest”}}$	= 2		0 +
$\varphi_{\text{unigram “black”}}(x)$	= 0		$w_{\text{unigram “black”}}$	= 0		0 +
		…				…
						=
						-3 → No!

予測の擬似コード

```
PREDICT_ALL(model_file, input_file):
    load w from model_file   # w[name] = w_name となるように
    for each x in input_file
        phi = CREATE_FEATURES(x)    # phi[name] = φ_name(x) となるように
        y' = PREDICT_ONE(w, phi)    # sign(w*φ(x)) を計算
        print y'
```

1 つの事例に対する予測の擬似コード

```
PREDICT_ONE(w, phi)
    score = 0
    for each name, value in phi          # score = w*φ(x)
        if name exists in w
            score += value * w[name]
    if score >= 0
        return 1
    else
        return -1
```

素性作成の擬似コード (例： 1-gram 素性)

```
CREATE_FEATURES(x):
    create map phi
    split x into words
    for word in words
        phi["UNI:"+word] += 1    # 「UNI: 」を追加して 1-gram を表す
    return phi
```

- この関数を変更し、他の素性を簡単に導入できる
  - 2-gram ？
  - その他の素性？

重みの学習：
パーセプトロンアルゴリズム

重みの学習

- 人手で重みを付与するのが困難
  - 有用な素性の数は膨大
  - 重みをむやみに変更すると予期しない影響
- その代わり、ラベル付きデータから学習

y	x
1	FUJIWARA no Chikamori (year of birth and death unknown) was a samurai and poet who lived at the end of the Heian period .
1	Ryonen (1646 - October 29 , 1711) was a Buddhist nun of the Obaku Sect who lived from the early Edo period to the mid-Edo period .
-1	A moat settlement is a village surrounded by a moat .
-1	Fushimi Momoyama Athletic Park is located in Momoyama-cho , Kyoto City , Kyoto Prefecture .

オンライン学習

```
create map w
for / 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

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

- 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

→ 更新のタビに、予測性能が向上！

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

例：最初の更新

- **w=0 として初期化**

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

$$\boldsymbol{w}\cdot\boldsymbol{\varphi}(x)=0 \qquad y'=\mathrm{sign}(\boldsymbol{w}\cdot\boldsymbol{\varphi}(x))=1$$

$$y' \neq y$$

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

$w_{\text{unigram “Maizuru”}} = -1$
$w_{\text{unigram “,”}} = -2$
$w_{\text{unigram “in”}} = -1$
$w_{\text{unigram “Kyoto”}} = -1$
$w_{\text{unigram “A”}} = -1$
$w_{\text{unigram “site”}} = -1$
$w_{\text{unigram “located”}} = -1$

例：２回目の更新

x = Shoken , monk born in Kyoto　　y = 1

-2　-1　-1

$$\boldsymbol{w}\cdot\boldsymbol{\varphi}(x)=-4 \qquad y'=\mathrm{sign}(\boldsymbol{w}\cdot\boldsymbol{\varphi}(x))=-1$$

$$y'\neq y$$

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

$w_{\text{unigram “Maizuru”}}$	= -1	$w_{\text{unigram “A”}}$	= -1	$w_{\text{unigram “Shoken”}}$	= 1
$w_{\text{unigram “,”}}$	= -1	$w_{\text{unigram “site”}}$	= -1	$w_{\text{unigram “monk”}}$	= 1
$w_{\text{unigram “in”}}$	= 0	$w_{\text{unigram “located”}}$	= -1	$w_{\text{unigram “born”}}$	= 1
$w_{\text{unigram “Kyoto”}}$	= 0				

演習問題

演習問題

- ２つのプログラムを作成
  - train-perceptron: パーセプトロンを用いた分類器学習
  - test-perceptron: 重みを読み込み、予測を 1 行ずつ出力
- テスト： train-perceptron
  - 入力： test/03-train-input.txt
  - 正解： test/03-train-answer.txt

演習問題

- data-en/titles-en-train.labeled でモデルを学習
- data-en/titles-en-test.word のラベルを予測
- 評価スクリプトで分類器の精度を計算
  - script/grade-prediction.py data-en/titles-en-test.labeled your_answer
- 上級編：
  - モデルが間違えた箇所を見て、間違えた理由について考察する
  - 新しい素性を導入し、精度への影響を計る