2008 8/3@PFI本郷オフィス

機械学習による自然言語処理チュートリアル

～PerceptronからCRFまで～

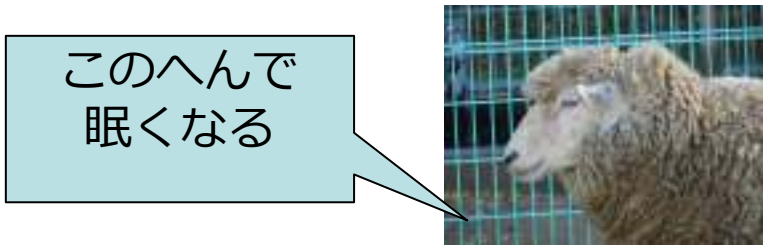
岡野原　大輔

東京大学

Preferred Infrastructure

目次

- 自然言語処理 紹介
- 機械学習導入
- パーセプトロン
- バッチ学習 (最大エントロピー法)
- 過学習/正則化
- 多クラス分類
- 系列分類 (CRF, Structured Perceptron)



自然言語処理 (1/2)

- 言語情報をコンピュータで処理する
  - コンピュータ言語の研究との対比で*自然言語*
  - 世界最初のコンピュータの出現の頃から自動翻訳は試みられている。コンピューターサイエンスの中でも歴史の長い分野
  - 近年ビジネス的にも成功, Googleなどなど
- 非常に幅広い分野と接触する、境界領域
  - 処理する手法 = 言語学, 数学, 統計学, 機械学習, アルゴリズム,データマイニング, データ構造, 物理
  - 処理する対象 = 情報検索, ウェブ,音声情報, ユーザーインターフェース
  - 言語が絡んでたら何でもいい

自然言語処理 (2/2)

- 分野例
  - 構文解析（係り受け解析）
  - 意味解析, SRL
  - 機械翻訳（統計的機械翻訳, ルールベース）
  - 情報検索
  - 固有表現抽出、文書分類、単語クラスタリング、
- 有名な学会、ジャーナル
  - Computational Linguistic, ACL, EMNLP, CoLing, NAACL, EACL, IJCNLP, などなど
  - 「自然言語処理の学会」 http://hillbig.cocolog-nifty.com/do/2008/04/post_fe44.html
  - ACL anthology

自然言語処理で
機械学習を使う

自然言語処理
ルールベース編

- 人手でルールをひたすら書いて処理する
- 例：スパム排除ルール
  - if (s =~ /H|奥様|楽しかったね/)
    then スパム
- 単純だが強力な手法で昔から現在でも活躍
  - 現時点でも日英機械翻訳などはルールベースの方が精度が良い

自然言語処理
ルールベース編

- 長所
  - 内部処理が分かりやすい
  - 人手で調整可能
  - 頑張れば頑張るほど精度が上がる
- 短所：
  - コストが大きい(最初は精度が伸びるが、そこから伸びない、メンテナンスコスト）、
  - 新しいドメインに弱い、
  - ルールに専門知識が必要（職人芸）、
  - 一貫性が無い（時間、人によってルールが変わる）

自然言語処理
機械学習編

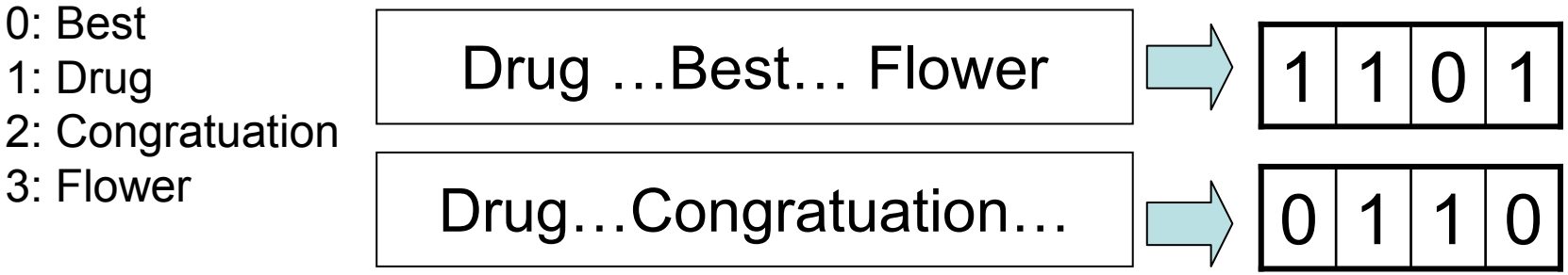
- 人手でルールを書く代わりに正解の訓練データを用意し、そこから正解を導くルールを導出する
  - 自然言語処理の場合、ルールは「単語」や「品詞」の出現などを利用
- ルールの候補集合を決め，その中の効くルールは学習で自動で探す
  - どのようにルールを作るか＝テンプレート

機械学習の前に
ルールベース vs 機械学習

- 何か問題を解こうとする場合は、むやみに機械学習を使わず検討してみる
  - 訓練データを得るためのコスト 対
  - ルールを書くためのコスト
- ルールベースの方が良い場合も多い
  - 問題のスケール、要求される精度、問題に対してもってる知識などでよく吟味
  - ルールの多くは素性という形で機械学習に反映させることもできるが、複雑な場合は人がやったほうがいい（文法変換などは数千行のperlでかいたそう）

ベクトル表現

- 多くの自然言語情報は疎な高次元のデータとして扱うことができる
  - 文書のBOW表現では、Vを単語種類数とした時，各単語に0～V-1の順番に番号を振る
  - i番目の単語が出現したらV_i=1とするV次元のベクトルとして扱える



各値を決めるのに使ったルールを素性（そせい、または特徴）、この高次元ベクトルを素性ベクトル（特徴ベクトル）と呼ぶ。

線形識別器

- 分類問題：入力**x**を二値(y=+1,-1)に分類
  - 例：文書をスパム(+1)orそれ以外(-1)に分類

$$S(\mathbf{x}) = \mathbf{w}^T \mathbf{x} = w_0 x_0 + w_1 x_1 + \ldots w_m x_m$$

  - S(x)≧0 ⇒ y=+1
  - S(x)<0 ⇒ y=-1 と判断する
  - w_i>0ならば、x_iが出現したらスパム度合いが増す（$|w_i|$が大きければ大きいほど効く）
- このような**w**をどうやって求めるか

重みベクトル**w**を求める

- 訓練データ($\mathbf{x}_i$,y_i) (y_i=1 or -1)を正しく分類できるような**w**を求める
  - 全てのiで $y_i\mathbf{w}^T\mathbf{x}_i>0$ をみたすようにする
  - 訓練データを分類できるのだから、未知のテストデータも分類できるのだろう
    （という期待。真面目には汎化性能とかを議論しなければならない）

オンライン学習・バッチ学習

- オンライン学習
  - 訓練データを1例ずつ見て**w**を更新する
  - 長所：あらかじめ全部の訓練例をもたなくても良い。速い（一般に訓練データ数に線形の計算量）
  - 短所：バッチに比べ適当、訓練例の順番に偏りが大きい場合に精度が不安定
- バッチ学習
  - 訓練データを全部見てから**w**を求める
  - 長所：精度が良い、理論的保証も強い(汎化性能）
  - 短所：重い場合が多い

オンライン学習
パーセプトロン [Rosenblatt 57]

- $\mathbf{w}$ = {0,0,….0}に初期化
- loop
  - (x_i, y_i) 訓練データをランダムにとってくる
  - s := $y_i \mathbf{w}^T \mathbf{x}_i$　　//$\mathbf{w}^T \mathbf{x}_i$=現在の予測
  - if (s < 0)　　// 現在の予測が外れた
    - **w := w** + $y_i \mathbf{x}_i$ // **w** + $\alpha y_i \mathbf{x}_i$, 0<α<1とする場合
  - endif
- end

なぜ学習できるのか

- 更新後の重みベクトル

  $\mathbf{w}' := \mathbf{w} + y_i \mathbf{x}_i$

- これを使って、もう一度間違えた訓練例$(\mathbf{x}_i, y_i)$を分類すると

  $\mathbf{w}'^{\mathrm{T}} \mathbf{x}_i$

  $= (\mathbf{w} + y_i \mathbf{x}_i)^{\mathrm{T}} \mathbf{x}_i$

  $= \underbrace{\mathbf{w}^{\mathrm{T}} \mathbf{x}_i}_{\text{先程間違えた予測結果}} + y_i \underbrace{\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_i}_{\text{常に正}}$

wは正解が出るように更新される

Perceptronの収束定理
[Block 62][Novikoff 62]

- Q.直前の訓練例だけを分類できるよう更新して、全ての訓練例を分類できるようになるのか？
- A. 訓練例が線形分離可能ならできる
- 定理：もし訓練データ $(\mathbf{x}_i, y_i)$ $(i = 1 \ldots m)$ $(|\mathbf{x}_i| < R)$が、ある重みベクトル$\mathbf{u}$ $(|\mathbf{u}| = 1)$ でマージンγで分類可能なら$(y_i(\mathbf{u}\mathbf{x}_i) \geqq \gamma)$、アルゴリズムの更新回数は多くても $(R/\gamma)^2$
  - $\mathbf{x}$の次元数によらないことにも注意

Perceptronの収束定理の証明

$\mathbf{w}_k$ をk回目の更新前の重みベクトルとおく

$\mathbf{w}_{k+1}\mathbf{u} = \mathbf{w}_k\mathbf{u} + y_i(\mathbf{u}\mathbf{x}_i) \geqq \mathbf{w}_k\mathbf{u}+\boldsymbol{\gamma}$

これより $\mathbf{w}_{k+1}\mathbf{u} \geqq k\boldsymbol{\gamma}$ $(\mathbf{w}_0=\mathbf{0})$

また、

$|\mathbf{w}_{k+1}|^2 = |\mathbf{w}_k|^2+2y_i(\mathbf{w}_k\mathbf{x}_i)+|\mathbf{x}_i|^2 \leqq |\mathbf{w}_k|^2+R^2$

これより $|\mathbf{w}_{k+1}|^2 \leqq kR^2$

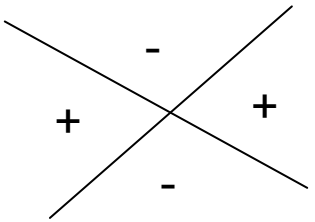
w_kで間違った訓練例なので値は負

上記二つより

$kR^2 \geqq |\mathbf{w}_{k+1}|^2 \geqq (\mathbf{w}_{k+1}\mathbf{u})^2 \geqq k^2\boldsymbol{\gamma}^2 \Rightarrow (R/\gamma)^2 \geqq k$

Perceptron補足

- 線形識別可能で無い場合は使えないとの批判 [Minsky 69]
  - データにノイズがある場合やxorなど
  - 廃れてしまった・・　⇒　最近復活！



- カーネルトリックを使える [Y. Freund 98]
- 後述の構造学習などでは、非常に強力な学習手法であることが分かってきた[Collins 02, Daume 06]
- 自然言語処理分野をはじめ多くの分野で再度、使われるようになってきた

頭が疲れてきたかもなので
ちょっと休憩

続いて　OLL実践編

OLLを使って
実際に分類をしてみよう

- OLLではパーセプトロンを初め様々なオンライン学習手法がサポートされている

```
>wget http://oll.googlecode.com/files/oll-0.01.tar.gz
>cd oll-0.01; ./configure; make; (make install)
>oll_train P trainfile modelfile
```

(パーセプトロンで学習し、学習結果を`modelfile`に保存する)

```
>oll_test testfile modelfile
```

(学習結果を`modelfile`から読み込み、`testfile`を分類する)

訓練・テストデータの フォーマット

- 各行が1訓練例に対応
  - 最初に正例なら+1,負例なら-1を書き、その後に素性番号：値の列を並べる

+1 0:1.0 201:2.2 744:-0.3 15:3.0 ….

-1 47:2.0 66:0.1 733:1.0 500:1.0

+1 3:1.0 201:2.2 300:-0.3 15:0.3

文書分類を作ってみよう（C++ですいません）

```
map<string, int> str2id;
// 文字列から素性番号への連想配列
vector<string>    id2str;
// 素性番号から文字列への逆引き


int getID(const string& str,
          map<string, int>& str2id){
    map<string, int>::const_iterator it
       = str2id.find(str);
    if (it != str2id.end()){
       return it->second;    // 登録済みのIDを返す
     } else {                //登録されてない
        const int newID = str2id.size();
       str2id[str] = newID; // 登録
        id2str.push_back(str);
       return newID;         // 新しいiDを返す
     }
 }
```

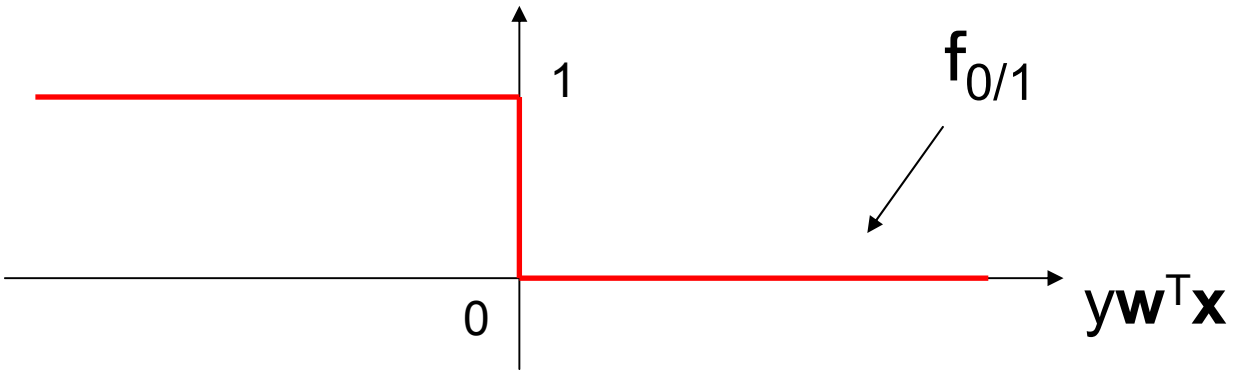
スペース区切り済みのドキュメントから素性ベクトルを作る

```
int doc2fv(const string& doc, const int y,
           ofstream& outf){
   istringstream is(doc); // docの単語はスペースで区切られ
   ている。きられて無い場合はmecabやら辞書の最長一致をしてください
   string chunk;
   vector<int> IDs;
   while (is >> chunk){
      IDs.push_back(getID(chunk));
   }
   sort(IDs.begin(), IDs.end());
   IDs.erase(unique(IDs.begin(), IDs.end()),
                    IDs.end()); // 重複を除く
   outf << y << " "; // ラベル
   for (size_t i = 0; i < IDs.size(); i++){
      outf << IDs[i] << ":1 ";
   }
   outf << endl;
}
```

学習
バッチ編

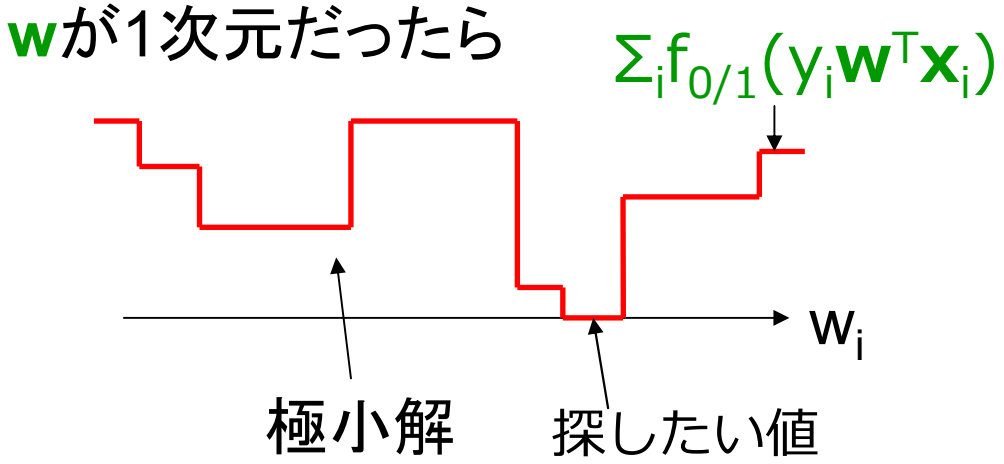
バッチ学習編

- 全部のデータを真面目にみてもっと精度を高くしよう
- $f_{0/1}$: 0/1損失関数
  - $f_{0/1}(a) = 1$ もし $a<0$,
    $= 0$ それ以外
  - $\Sigma_i f_{0/1}(y_i \mathbf{w}^T \mathbf{x}_i)$ を最小化するような$\mathbf{w}$を求める



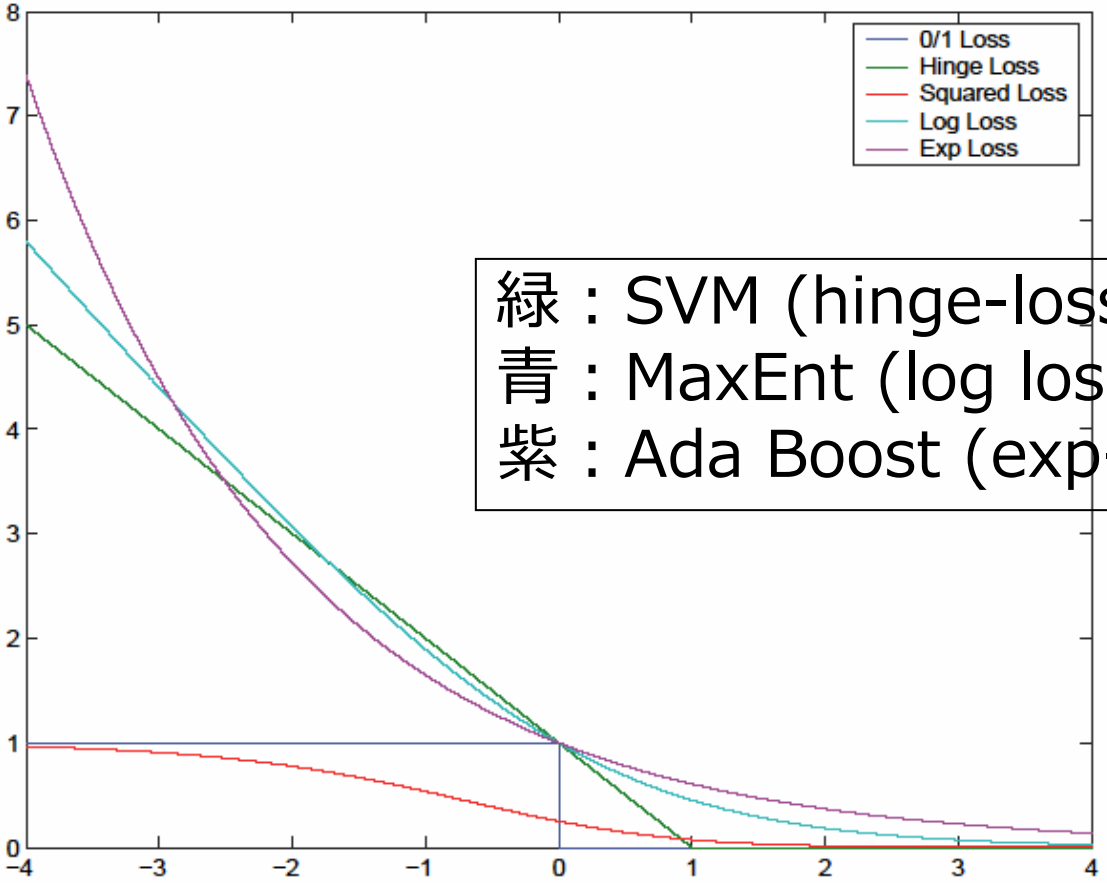
重みベクトル**w**を求める(続)
バッチ学習編

- $\Sigma_i f_{0/1}(y_i \mathbf{w}^T \mathbf{x}_i)$を最小にする**w**を探す
  - 数値最適化問題
- この最適化問題は実は難しい！
  - 関数$f_{0/1}(a)$は微分不可能で凸でもない（極小解が複数存在する）



様々な損失関数による$f_{0/1}$の近似

凸（穴が無い、Hessian行列が正定値）または微分可能（滑らか）な関数



緑：SVM (hinge-loss) $[1-y\mathbf{w}^T\mathbf{x}]_+$

青：MaxEnt (log loss) $-\log(1+\exp(-y\mathbf{w}^T\mathbf{x}))$

紫：Ada Boost (exp-loss) $\exp(-y\mathbf{w}^T\mathbf{x})$

線形識別器　まとめ

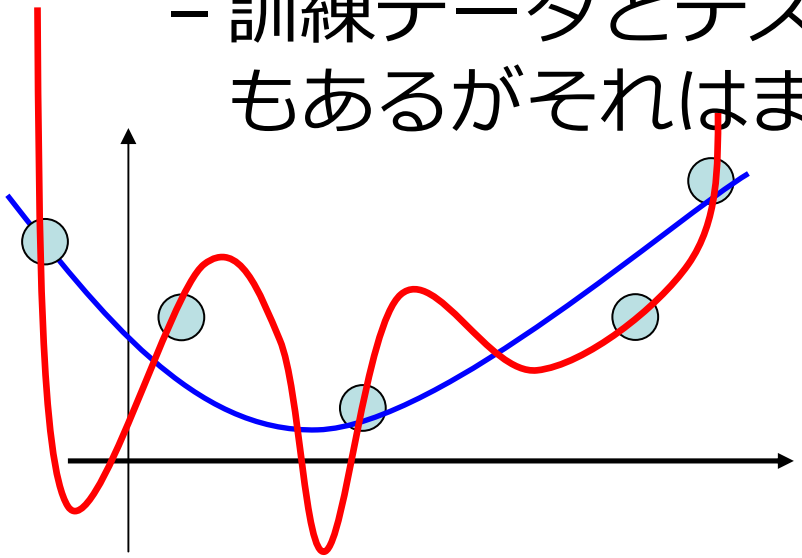
- 訓練データを用意 $(\mathbf{x}_i, y_i)$ $(y_i \in \{-1, +1\})$
- 学習：損失関数Lを使って
  $\Sigma_i L(\mathbf{x}_i, y_i, \mathbf{w})$が最小となる$\mathbf{w}$を探す
- 推定：分類したいデータ$\mathbf{x}_{new}$に対し
  $s = \mathbf{w}^T \mathbf{x}_{new}$を求め
  - $s>0 \Rightarrow y=1$,
  - $s<0 \Rightarrow y=-1$　と推定する

- 非常に多くの分類器がこれに属する
  - SVM, 最大エントロピー法、NB、Adaboostなど

過学習/正則化

過学習

- 訓練データは分類できるようになっても、テストデータが分類できない場合がある
- 大抵、訓練データにオーバーフィットしている場合が多い
  - 訓練データとテストデータの性質が違う場合もあるがそれはまた別の話



回帰の例:
青は2次曲線
赤は8次曲線

赤は全ての点を通っているが
スムーズではない(オーバーフィット)

正則化

- 過学習を防ぐように**w**にペナルティを与える
  - 絶対値が大きいw_iにペナルティを与える
  - L_2 ノルム $|\mathbf{w}|^2=w_1^2+w_2^2+\cdots. w_m^2$
  - L_1 ノルム $|\mathbf{w}| =|w_1|+|w_2|+\cdots+|w_m|$
  - 実はこの二つがものすごい差を生む（今回は話しません）
- 訓練例を正しく分類しつつ過学習しない学習

  $\Sigma_i L(\mathbf{x}_i,y_i,\mathbf{w})+C|\mathbf{w}|^2$
  - Cはトレードオフパラメーター Cが大きいと過学習をしないが訓練データにfitしない、Cが小さいと訓練データにfitするが過学習する危険性がある
  - 最適なCは別データを利用して推定するか、ベイズ推定

線形識別器のまとめ（改）

- 訓練データを用意$(\mathbf{x}_i, y_i)$ $(y_i = 1 \text{ or } -1)$
- **学習**：損失関数Lを使って
  $\Sigma_i L(\mathbf{x}_i, y_i, \mathbf{w}) + C|\mathbf{w}|^2$が最小となる$\mathbf{w}$を探す
  （$C|\mathbf{w}|^2$：正則化）
- **推定**：分類したいデータ$\mathbf{x}_{new}$に対し
  $s = \mathbf{w}^T\mathbf{x}_{new}$を求め
  - $s>0 \Rightarrow y=1$,
  - $s<0 \Rightarrow y=-1$　と推定する

出力で確率値を出したい場合

- スコアに確率をだしてつけたい
  - yが０と1になる確率値はそれぞれいくつ？
  - 確信度として後の処理で使いたい
  - スコアそのままでは正規化されていない
- 確率値に必要な条件
  - 出力値は正の実数（条件1 ）
  - 可能な出力値の確立を足し合わせたら1（条件2）

出力で確率値を出したい場合 (続)
Logistic回帰

- 条件1を満たすために$\mathbf{w}^T\mathbf{x}$をexpに入れる

$$s(y) = \exp(y\mathbf{w}^T\mathbf{x}) > 0$$

- 条件2を満たすために正規化すると

$$p(y) = s(y)/(s(1)+s(-1))$$

$$= 1/(1+\exp(-2y\mathbf{w}^T\mathbf{x}))$$

- この形の確率モデルをLogistic回帰と呼ぶ（最大エントロピーモデルと一致）

$$p(y \mid \mathbf{x};\mathbf{w}) = \frac{1}{1+\exp(-y\mathbf{w}^T\mathbf{x})}$$

最大エントロピーモデル（後で読んでね）

- 先程は強引な導出をしたが、別の意味づけもできる
- 各素性に関するラベルの符号付の期待値を$f_k=\Sigma_i y_i x_{ik}/N$ とおく
- x,y上の確率分布$p(x,y)$で各素性の期待値が一致するような分布でエントロピー（$\int_{x,y} p(x,y)\log p(x,y)\,dx\,dy$)が最大となる確率分布は先程のモデルの最尤推定結果と一致する

Logistic回帰の学習

- 訓練データの生成確率を最大にするような**w**を求める

$$\arg\max_{\mathbf{w}} \prod_i p(y_i \mid \mathbf{x}_i; \mathbf{w})$$

$$= \arg\max_{\mathbf{w}} \sum_i \log p(y_i \mid \mathbf{x}_i; \mathbf{w})$$

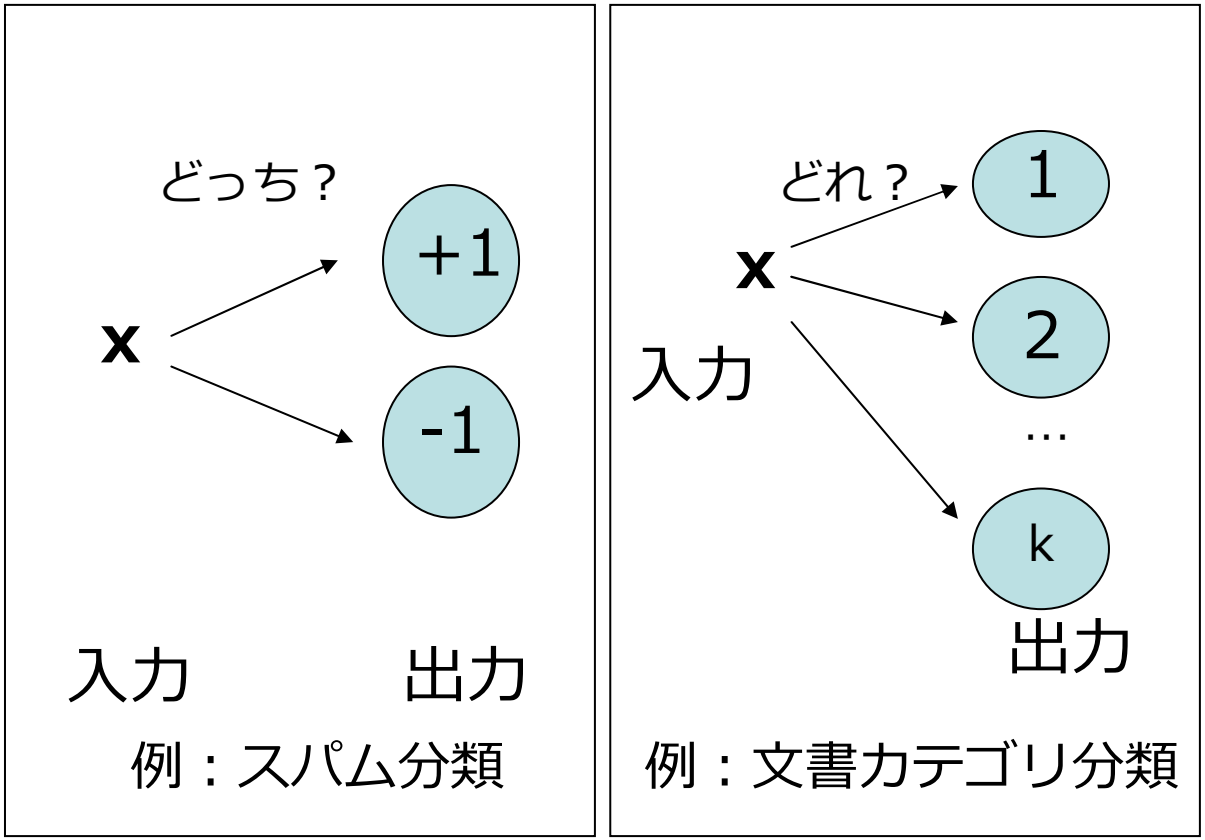
$$= \arg\min_{\mathbf{w}} -\sum_i \log(1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i))$$

log loss損失関数！　先程見ましたね

多クラス分類

多クラス分類

ちょっと難しそう・・



多クラス分類

- 入力**x** 出力 {1,2,…,m}のどれか
- 重みベクトル**w**をクラス種類数分用意
  - $\mathbf{w}_1 \cdots \mathbf{w}_m$
- $\mathbf{w}^T_y\mathbf{x}$ が一番大きかったyを推定結果とする
  - これを $\mathrm{argmax}_y \mathbf{w}^T_y\mathbf{x}$ とかく
- そのような$\mathbf{w}_1 \cdots \mathbf{w}_m$をどのように求めるのか？

パーセプトロンによる学習

- $\mathbf{w}_1 \cdots \mathbf{w}_m$ を全部{0,0,…,0}に初期化
- loop
  - (x_i, y_i) 訓練データをランダムにとってくる
  - $y^* := \mathrm{argmax}_y \mathbf{w}_y^{\top} \mathbf{x}_i$ (現在の予測結果)
  - if ($y^* \neq y_i$) // 現在の予想が外れた

    $\mathbf{w}_{yi} := \mathbf{w}_{yi} + \mathbf{x}_i$

    $\mathbf{w}_{y^*} := \mathbf{w}_{y^*} - \mathbf{x}_i$ // 正解じゃないのに

    //スコアが高いやつには罰を！
  - endif
- end

多クラス版最大エントロピーモデル

$$p(y \mid \mathbf{x}; \mathbf{w}) = \frac{1}{Z(\mathbf{x})} \exp(\mathbf{w}_y^T \mathbf{x})$$

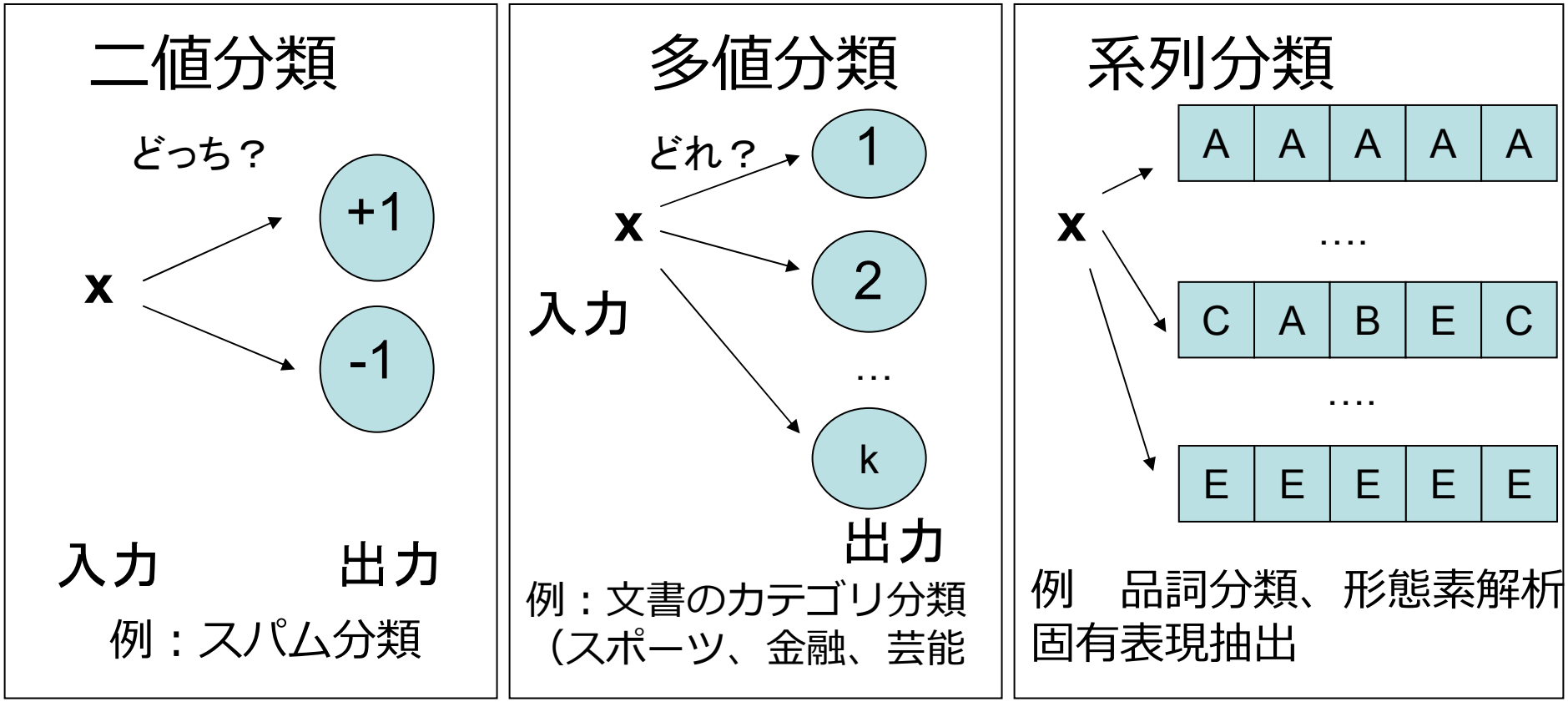
$$Z(\mathbf{x}) = \sum_{y'} \exp(\mathbf{w}_{y'}^T \mathbf{x})$$

- 推定は$\mathrm{argmax}_y \mathbf{w}_y^\top \mathbf{x}$ で行なう点で同じ
- 学習は訓練データの生成確率を最大にするような**w**を求める
- 後でCRFの時にこれがでてきます

系列分類問題
（構造出力学習）

系列分類問題とは

- 分類対象が系列（ラベル列）



系列分類問題（続）

- 分類対象が系列（ラベル列）
  - 一般にk個の多値分類に分解できる
  - 多値分類で、$|Y|^k$個の候補から探すと思ってもよい
- 品詞分類
  - 各単語に付く品詞を分類する
- 固有表現抽出
  - 各単語が固有表現抽出の先頭(B)か、途中か(I)、それ以外か(O)の分類問題として解く
- 形態素解析
  - 形態素ラティス上を選ぶ問題
- 画像認識、時系列解析など多分野でも使われている

固有表現抽出

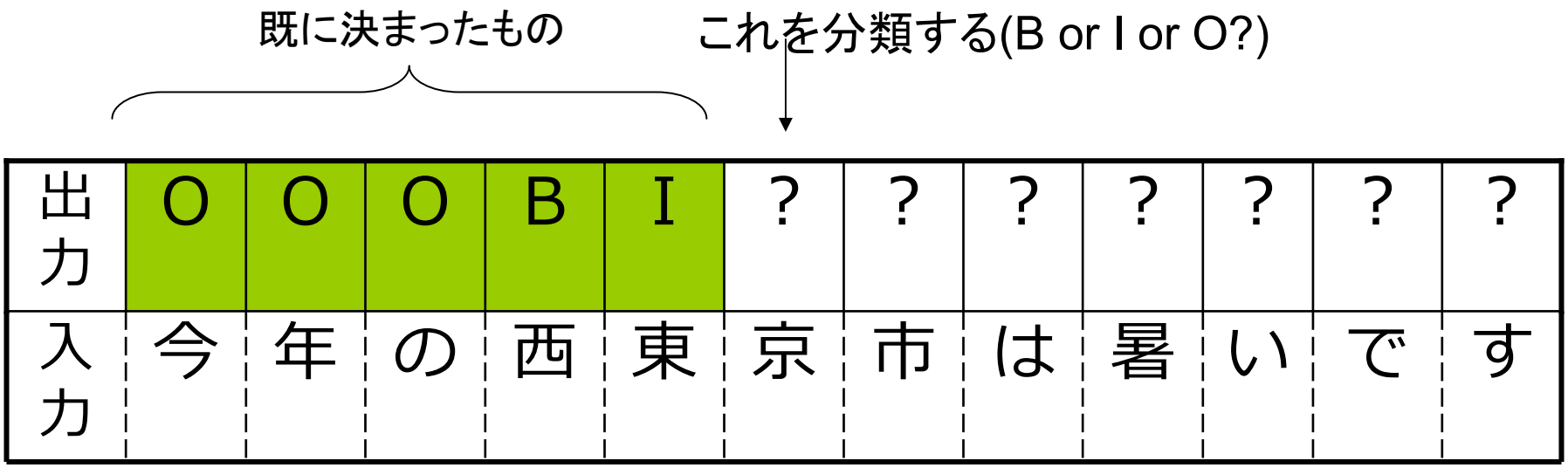
- タスク：人名や地名などの固有表現を文中から抜き出す
  - 訓練データさえ用意できえれば何でも
- ラベルを各文字に割り当てる問題
  - B: 固有表現の先頭
  - I：固有表現の途中
  - O：固有表現以外

地名抽出問題の正解例

出力	O	O	O	B	I	I	I	O	O	O	O	O
入力	今	年	の	西	東	京	市	は	暑	い	で	す

系列分類
順次適用モデル

- 前から順に分類問題を独立に解いていく
  - 直前の分類結果も素性として利用できる



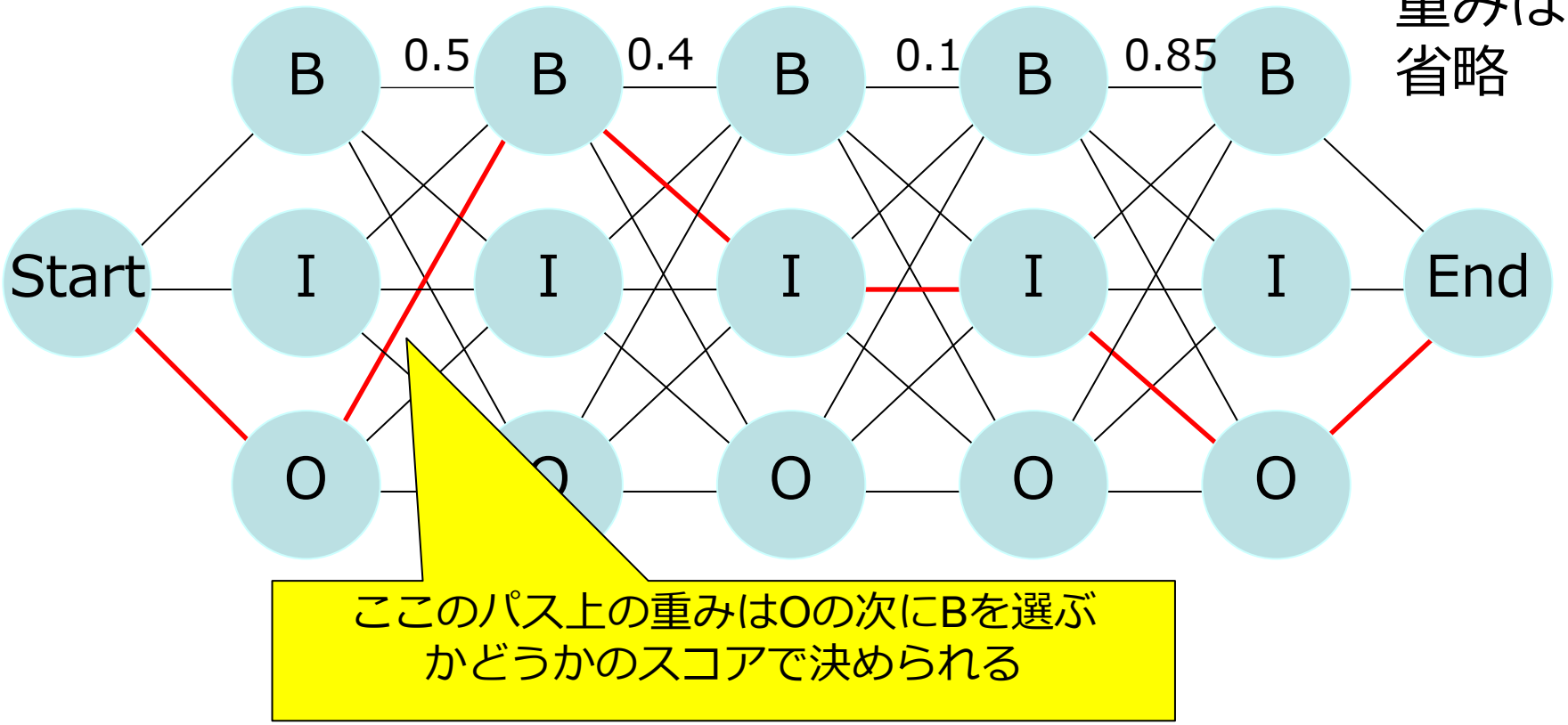
使える情報：　入力（その位置の文字＝京、一つ前の文字＝東、一つ後の文字＝市、一つ前のラベル＝I）
HMMとは違い、入力情報全部がどの位置でも使えることに注意

系列分類、解き方

- 問題点
  - 後ろの情報を自由に使えない
    例：人名抽出タスクで「様」という単語がきたらその直前の漢字列は人名だが、前から順番に見た時にOと判定してしまっていた
- HMMのようにViterbiを使って推定する
  - 全候補中の最もよいラベル列を、動的計画法で効率よく求める
  - MEMMのように自由に素性を使える

系列問題
グラフ上の最大パスを求める問題

- 各候補がグラフ上の頂点に対応し、各枝には分類結果の重みがついている

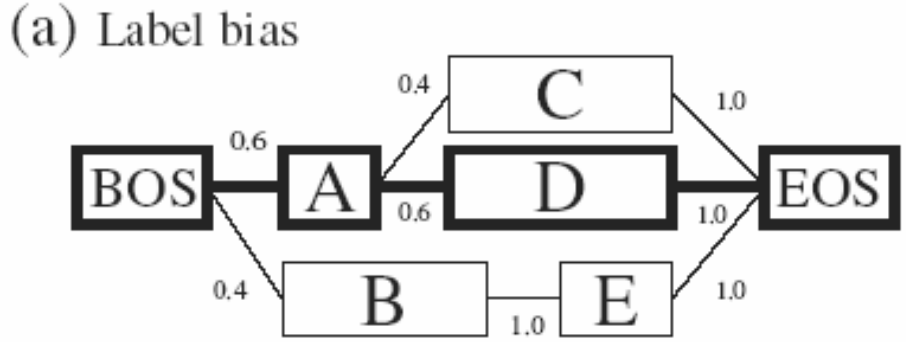


Viterbi復号

- グラフ上の最大重みパスは動的計画法によって解くことができる
  - 各位置での候補数がk、系列長がNの時$O(k^2N)$時間で解くことができる
  - 各重みを最大エントロピー法による確率値で求めたものを特にMEMM (Maximum Entropy Markov Model)と呼ぶ
  - 確率値のlogをとったものを重みにすれば、通常の最大重みパスのアルゴリズムで解ける

MEMMなどの順次適用モデルの問題点1. label bias

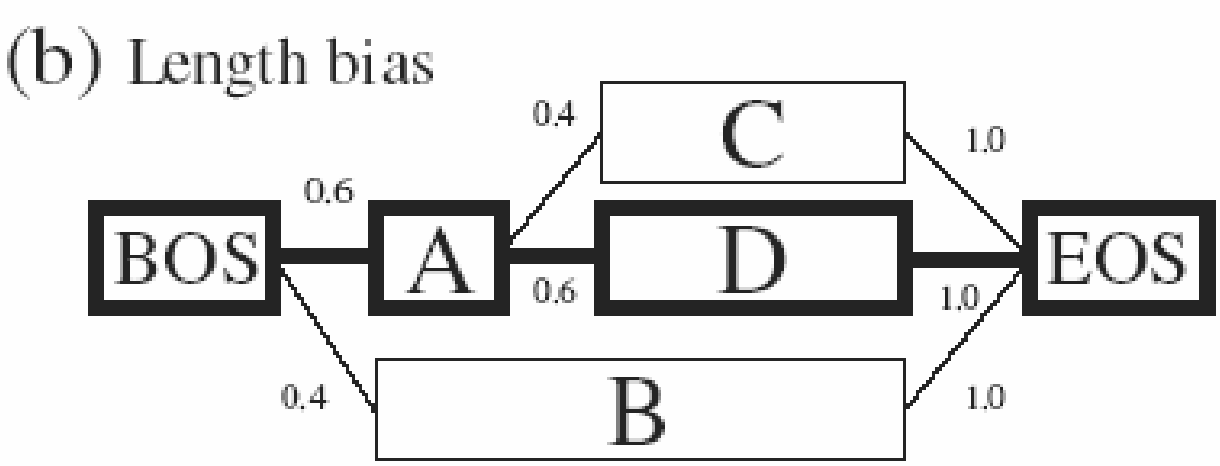
- 簡単な経路（曖昧性の小さい経路）が存在した場合にそちらが選ばれやすい
- 学習時に正解系列とその周辺だけが考慮されるので、知らない系列の確率値は非常に不安定になる



[Kudo＋ 04]より

MEMMなどの順次適用モデルの問題点2. length bias

- ラベルをつける単位が等しくない場合（形態素解析など）、短い経路が選ばれやすい



$P(A, D \mid x) = 0.6 * 0.6 * 1.0 = 0.36$

$P(B \mid x) = 0.4 * 1.0 = 0.4$ $\quad P(A,D|x) < P(B|x)$

[Kudo＋ 04]より

条件付確率場
CRF (Conditional Random Fields)

- 問題を部分問題に分けずに，一つのモデルで表す

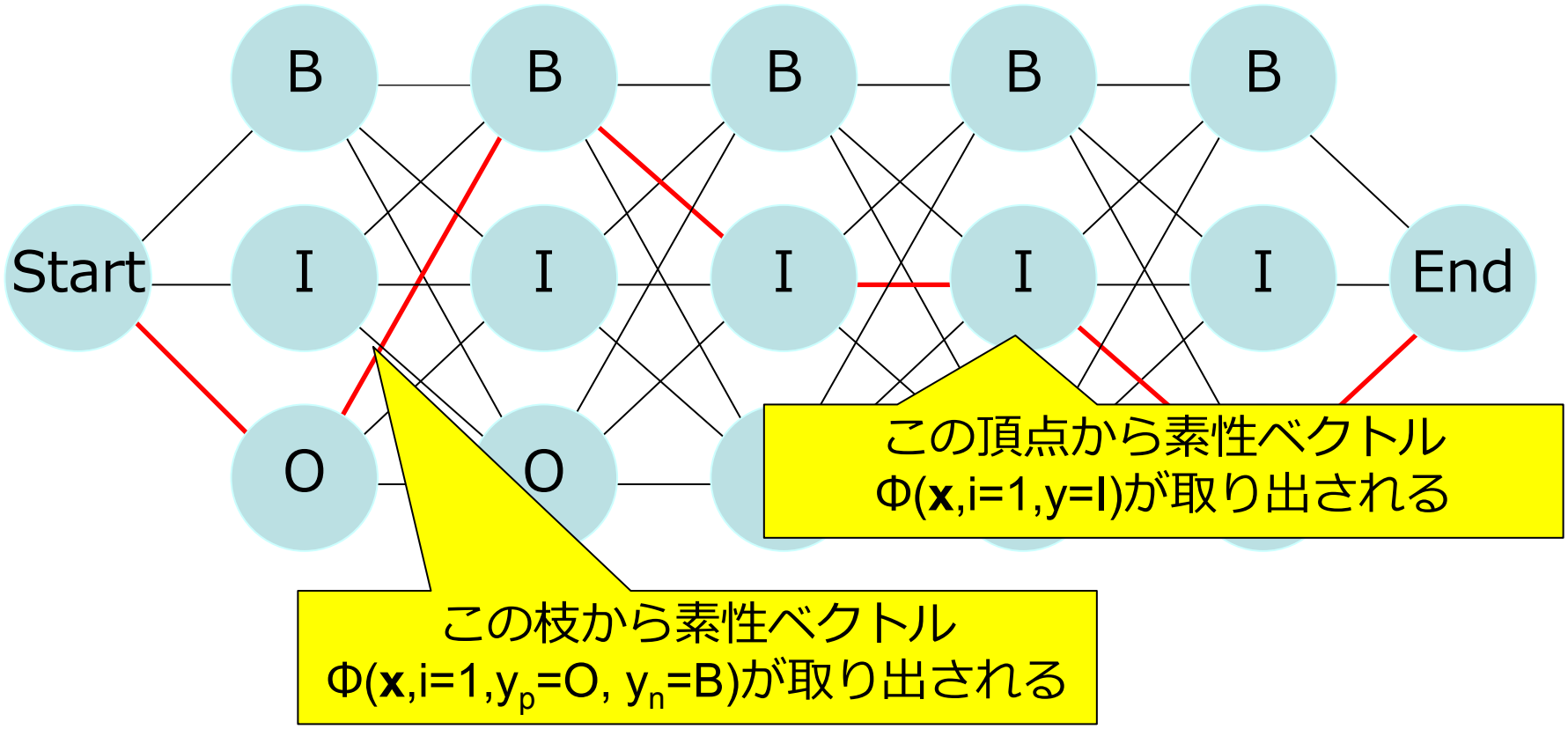
$$p(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp(\mathbf{w}^T \phi(\mathbf{x}, \mathbf{y}))$$

$$Z(\mathbf{x}) = \sum_{\mathbf{y}' \in O(\mathbf{x})} \exp(\mathbf{w}^T \phi(\mathbf{x}, \mathbf{y}'))$$

y'はx上の全てのパス！

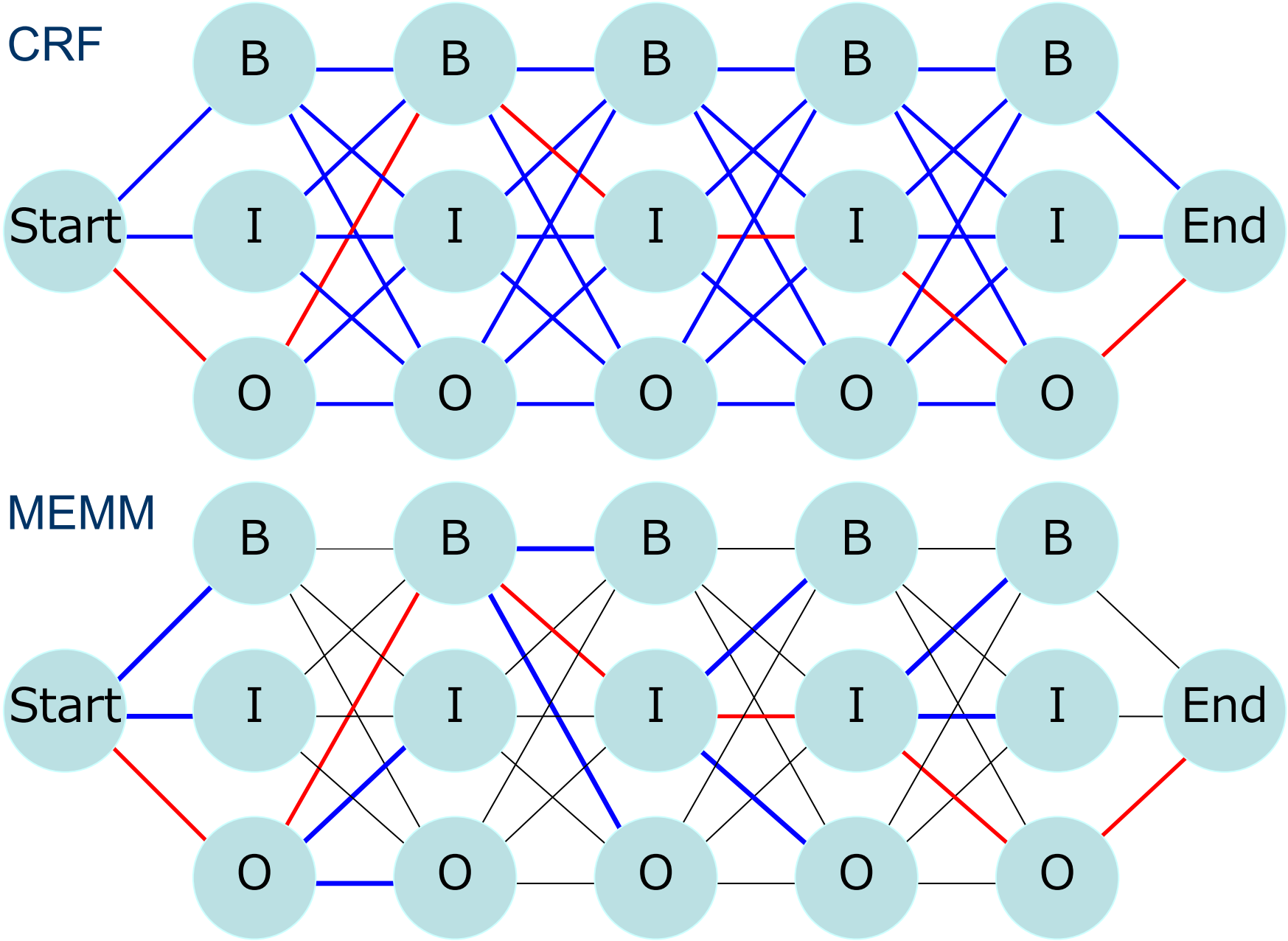
- 但しΦ(**x**,**y**)は、系列**y**=$y_1 y_2 \ldots y_k$上のパスの全ての素性ベクトルを足し合わせたもの
  - o(**x**)は**x**上に定義される全ての系列集合
- 先程の二つの問題（label bias, length bias)問題を自然に解消

CRF



先程は、とられた素性ベクトルをそれぞれの中でexpの中に入れるモデル

赤線が正解例　青線が負例



負例は正解系列の周辺のみ！　⇒全く知らない系列の推定が不安定

素性

- 頂点からの素性ベクトル
  - ラベル×入力の全ての情報
- 枝からの素性ベクトル
  - 枝の両端のラベル×入力の全ての情報
- 例：
  - ラベル×その位置の入力単語
  - 直前の単語×直後の単語×直前と直後のラベル
    など

CRFの特徴　まとめ

- 確率値が出せる
- 一番予測値が良いパスだけでなくN-best（上位N個）も推定できる
- 精度が安定して良い
  - 訓練時に全てのパスが考慮されるので
- 枝（隣接するラベル対）、頂点（ラベル）で任意の素性を定義できる
  - 入力はどの位置でも全て使える。より複雑なモデルでは学習、推定が難しくなる（詳しくはGraphical Modelで調べてみてください）

パーセプトロンによる学習

- この系列問題もパーセプトロンで学習できてしまう！ [Collins 02]
- 確率値は出せず、1-bestだけしか出ないが学習はすごく速くて簡単
  - N-bestも一応出せるが、訓練時に1位以下のパスは考慮しないので、CRF++に比べ信頼できない

パーセプトロンによる学習

- $\mathbf{w}_1 \cdots \mathbf{w}_m$ を全部{0,0,…,0}に初期化
- loop
  - (x_i, y_i) 訓練データをランダムにとってくる
  - $\mathbf{y}^* := \mathrm{argmax}_y \mathbf{w}^T \mathbf{\Phi(x,y)}$ (現在の予測系列)
  - if $(y^* \neq y_i)$ // 現在の予想が外れた
    - $\mathbf{w}_{yi} := \mathbf{w}_{yi} + \mathbf{\Phi(x,y)}$
    - $\mathbf{w}_{k^*} := \mathbf{w}_{k^*} - \mathbf{\Phi(x}$
  - endif
- end

素性がラベル間（枝素性）のような localなものしか使って無いならargmax はViterbiアルゴリズムで処理可能 そうでない場合は、ビームサーチや Gibbs Samplingで近似解を推定する

パーセプトロンまとめ

- どんなに複雑な問題でもargmax操作さえ効率的にできれば、

  $y' = \mathrm{argmax}_y \mathbf{w}^T \Phi(x,y)$

  $w := w + y$

  $w := w - y'$

  として学習できる
- 全列挙が難しくてもargmaxができる問題も多数存在する
  - 例：二部グラフの最大重みマッチングは全列挙は頂点数の指数時間かかるがargmaxは線形計画法で線形時間で解ける

CRF++を使って系列分類を行なおう

CRF++ http://crfpp.sourceforge.net/ Taku Kudo

訓練データのフォーマット

- M個の要素からなる系列問題はM行からなり、各問題は空行で区切られる
  - 各行は任意個数の列を持つことができる。全ての行は同じ個数の列を持つ必要がある
  - 最終列が正解を持つ

訓練データ例

```
He                  PRP   B-NP
reckons     VBZ     B-VP
the         DT      B-NP
current     JJ      I-NP
account     NN      I-NP
deficit     NN      I-NP
will        MD      B-VP
narrow      VB      I-VP
to                  TO    B-PP
only        RB      B-NP
#                   #     I-NP
1.8         CD      I-NP
billion     CD      I-NP
in                  IN    B-PP
September   NNP     B-NP
.                   .     O

He                  PRP   B-NP
reckons     VBZ     B-VP  ..
```

1列目　単語
2列目　品詞
3列目　ラベル
(学習したい
目標)

素性テンプレート

- %x[i,j]
  - 現在の位置からの相対位置でi行目のj番目の列の要素を指す
- 工夫した使い方
  - 組み合わせもできる
    %x[-2,1]/%x[-1,1] //2つ前の品詞と1つ前の品詞の組み合わせ
- 先頭のU/B ラベル情報
- Example中にあるテンプレートを使うと簡単です

CRF++の日本語固有表現のテンプレート例

```
# Unigram
U00:%x[-2,0]
U01:%x[-1,0]
U02:%x[0,0]
U03:%x[1,0]
U04:%x[2,0]
U05:%x[-1,0]/%x[0,0]
U06:%x[0,0]/%x[1,0]

U10:%x[-2,1]
U11:%x[-1,1]
U12:%x[0,1]
U13:%x[1,1]
U14:%x[2,1]
U15:%x[-2,1]/%x[-1,1]
U16:%x[-1,1]/%x[0,1]
U17:%x[0,1]/%x[1,1]
U18:%x[1,1]/%x[2,1]

U20:%x[-2,1]/%x[-1,1]/%x[0,1]
U21:%x[-1,1]/%x[0,1]/%x[1,1]
U22:%x[0,1]/%x[1,1]/%x[2,1]

U30:%x[-2,2]
U31:%x[-1,2]
U32:%x[0,2]
U33:%x[1,2]
U34:%x[2,2]
U35:%x[-2,2]/%x[-1,2]
U36:%x[-1,2]/%x[0,2]
U37:%x[0,2]/%x[1,2]
U38:%x[1,2]/%x[2,2]

U40:%x[-2,2]/%x[-1,2]/%x[0,2]
U41:%x[-1,2]/%x[0,2]/%x[1,2]
U42:%x[0,2]/%x[1,2]/%x[2,2]

U50:%x[0,1]/%x[0,2]

# Bigram
B
```

U=ラベル1個
B=ラベル2個の組み合わせ

固有表現を作ってみる (1/4)

- CRF++をダウンロードしてくる

```
>tar xvzf CRF++-0.51.tar.gz
>cd CRF++-0.51; ./configure; make; (make
  install)
>cd example/JapaneseNE
>./exec.sh
```

学習とテストをしてくれる

```
>../../crf_learn -f 3 -c 4.0 template
  train.data model
>../../crf_test -m model test.data
```

実際にモデルを適用する

固有表現を作ってみる (2/４)

- 自分で1から作るならmecabなどと組み合わせてやる

```
西東京　名詞,固有名詞,地域,一般,*,*,西東京,ニシトウキョウ,ニシトーキョー
市　　名詞,接尾,地域,*,*,*,市,シ,シ
は　　助詞,係助詞,*,*,*,*,は,ハ,ワ
暑い　　形容詞,自立,*,*,形容詞・アウオ段,基本形,暑い,アツイ,アツイ
です　　助動詞,*,*,*,特殊・デス,基本形,です,デス,デス
ね　　助詞,終助詞,*,*,*,*,ね,ネ,ネ
EOS
```

- 使えそうな最初の2つをくっけるプログラムを作る

```
八丁堀　　名詞-固有名詞-地域 O
店　　名詞-接尾-一般　　O
【　　記号-括弧開-*　　O
八丁堀　　名詞-固有名詞-地域 O
】　　記号-括弧閉-*　　O
17　　名詞-数-*　　O
時　　名詞-接尾-助数詞　O
〜　　記号-一般-*　　O
翌　　接頭詞-数接続-*　O
```

固有表現を作ってみる (3/４)

正解データを頑張って作る
　100例ぐらい作れば結構いい精度出る

八丁堀　名詞-固有名詞-地域　　O
で　　助詞-格助詞-一般　　O
連日　　名詞-副詞可能-*　　O
人気　　名詞-一般-*　　O
の　　助詞-連体化-*　O
和食　　名詞-一般-*　　B-STORE
居酒屋　名詞-一般-*　　I-STORE
。　　記号-句点-*　　O

学習と推定 (4/4)

- templateをそのまま使う
  - 3列構成ならそのまま使える

../../crf_learn -f 3 -c 4.0 template train.data model

>../../crf_test -m model test.data

適当に整形すればうまくみえます

質問/回答 (1/4)

- Q. パーセプトロンの更新式では発散することはないのか
- A. 収束定理が示すとおり線形分離可能なら有限回数で終わる。また線形分離可能で無い場合でも一定の収束性能は示せる

- Q. カーネルトリックをパーセプトロンでどのように使えるのか
- A. 重みベクトルをwの代わりに各訓練データで間違えた回数をα_iでもっていて$w^T x$の代わりに$\Sigma_i \alpha_i y_i K(x_i, x)$を使えばできる

質問/回答 (2/4)

- Q. 多クラス分類をこれ以外の手法で実現できないか
- A. 全てのクラスの組み合わせで二値分類の問題を作って多数決を取る方法、（１個のクラス対残り全部）を全部作って一番スコアが高いのを選ぶ方法（one versus rest)、トーナメント方式にする方法、エラー符号を作って、それでハミング距離が一番近いのを選ぶ方法などいろいろある。

質問/回答 (3 /4)

- Q. MEMMとCRFはどちらの方が速いのか
- A. MEMMでは各Stateで正規化が必要なのでその分遅い。ただMEMMでは前からgreedyに展開など工夫はいくらでもできる
- Q. MEMMは良く出る系列の周辺だけ集中して学習しているのでMEMMの方がよいのではないか
- A. 一概に言えないが一般的なラベル数ではCRFでも学習としては問題ない。ただラベル数や系列数が非常に大きい場合は周辺パスのノイズの影響が無視できない。Searn [Daume06]などが対処している

質問/回答 (4/4)

- Q. CRFで利用する素性の種類数は爆発するのでは
- A. CRFでは枝素性が隣接するラベルの組み合わせ×入力に関する素性種類数なので、ラベル数がk、入力種類素性数がmの時k^2m個の素性が使われる。ラベル種類数が多い場合は非常に大きいので、頂点に付く素性しか使わないことが多い