構造学習を用いた議事録中の発話の賛否関係の推定

Identification of agreement and disagreement relations between utterances in meetings using structured output learning

森田 一 *1　高村 大也*2　奥村 学*2
Hajime Morita　Hiroya Takamura　Manabu Okumura

*1 東京工業大学 知能システム科学専攻
Department of Computational Intelligence and Systems Science, Tokyo Institute of Technology

*2東京工業大学 精密工学研究所
Precision and Intelligence Laboratory, Tokyo Institute of Technology

We propose a new method, which enables the training of a kernelized structured output model. The structured output learning is gaining popularity in natural language processing. Meanwhile the kernel method is effective in many natural language processing tasks, since it takes into account the combination of features. However, it is computationally difficult to simultaneously use both the structured output learning and the kernel method. Our method avoids this difficulty by transforming the kernel function and enables the kernelized structured output learning.

1. Intrduction

Structured Output Learning とは, インスタンスに対して構造を持つラベルを推定するモデルを学習する機械学習の手法である. その構造は複雑で, ラベルの集合は非常に大きい場合が多い. 構造を持つラベルの例として, 木構造を含むグラフ, シーケンスなどがある. 構造学習は構文解析や文書分類等を含む自然言語処理で重要な役割を担うようになってきている. この構造学習をおこなうアルゴリズムとして, 近年 Support Vector Machine [2] や Passive Aggressive algorithm[1] 等が拡張・提案されている. 構造学習ではラベルを推測するために, 指数的なサイズのラベル集合から最適なラベルを選択する必要がある. このため, ラベルの推測を効率的に行えるかどうかが, 現実的な時間で問題を解けるかどうかに関わってくるが, このラベルの選択には各問題に応じて様々に特化したアルゴリズムが用いられることが多い. 例を挙げると, ラベルがシーケンスであれば viterbi アルゴリズムが使われることが多く [5]. ラベルが係り受け木であれば CKY 等の構文解析のアルゴリズムを使うことができる [4]. これらの学習には, 一般に線形モデルが用いられることが多い.

一方, 自然言語処理では多項式カーネルを用いることで N-gram などの素性間の関係を捉えた学習を行うことが一般的となっている. 構造学習においても, 線形では識別の難しい問題を学習するため, あるいはより識別能力を向上するためにはカーネルを導入することが必要となる. それにも関わらず, 構造学習とカーネルが同時に使われることはあまりない. これは構造学習でカーネルを用いると, 可能性のあるすべてのラベルに対してカーネルを計算する必要があるためである. 本研究では構造学習と多項式カーネルを効率的に同時に扱う手法を提案する. また, 構造学習を自然言語処理のために用いようとするとき, 構造の推定に複数の異なる問題が含まれることがある. このような複数の異なる問題を構造の要素として同時に学習するため, 問題間でクラスの割合の異なる場合を考慮したロス関数を定義する.

MRDA コーパス [3] 中の賛否関係の推定をタスクとしてこの手法を評価する. ラベル集合が大きい場合, 提案手法は従来手法に比べ指数的に早くなることを実験により示す.

2. Passive Aggressive algorithm

連絡先: 森田 一, 神奈川県横浜市緑区長津田町 4259 精密工学研究所 R2-728, morita@lr.pi.titech.ac.jp

Passive Aggressive Algorithm はパーセプトロンベースのオンライン最大マージンアルゴリズムの一種である. 事例数に対して線形時間で動作するため Support Vector Machine(SVM) 等のバッチアルゴリズムと比較して高速で, かつ必要とするメモリも少なくてすむ, といった特徴がある. Crammer らにより, 構造学習への拡張や, カーネルを用いた学習のための派生アルゴリズムがそれぞれ提案されている. このアルゴリズムでは各更新ステップで, 事例 $\mathbf{x}$ を誤分類した場合に正しく分類できるようモデルを最小限更新する. 構造学習を行う場合や, カーネルを使う場合もこれは変わらない. この構造学習を行う場合を Algorithm 1 に示す.

Algorithm 1 Passive Aggressive Algorithm

Input: $S = ((\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), ..., (\mathbf{x}^{(N)}, \mathbf{y}^{(N)})), C$
1: initialize model
2: **for** $iteration = 1, 2, \ldots$ **do**
3: **for** $i = 1, ..., N$ **do**
4: get most violated label $\bar{\mathbf{y}}$
5: **if** $(\bar{\mathbf{y}} \neq \mathbf{y}^{(i)})$ **then**
6: calculate τ from $\rho(\mathbf{y}^{(i)}, \bar{\mathbf{y}})$
7: update model
8: **end if**
9: **end for**
10: **end for**
11: return model

構造学習とカーネル K を同時に使う場合には, モデルはサポートベクトル $(\tau^{(t)}, \mathbf{x}^{(t)}, \mathbf{y}^{(t)})$ の集合 $\mathcal{W}$ で表される. 分類は以下の拡張された最大化問題を解く:

$$\bar{\mathbf{y}} = \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}} \sum_{\{\tau^{(t)}, \mathbf{x}^{(t)}, \mathbf{y}^{(t)}\} \in \mathcal{W}} \tau^{(t)} K(\Phi(\mathbf{x}^{(t)}, \mathbf{y}^{(t)}), \Phi(\mathbf{x}^{(i)}, \mathbf{y})).$$

各反復では次のような最大化を行い, 制約に最も反するラベルを求める:

$$\bar{\mathbf{y}} = \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}} \sum_{\{\tau^{(t)}, \mathbf{x}^{(t)}, \mathbf{y}^{(t)}\} \in \mathcal{W}} \tau^{(t)} K(\Phi(\mathbf{x}^{(t)}, \mathbf{y}^{(t)}), \Phi(\mathbf{x}^{(i)}, \mathbf{y})) + \sqrt{\rho(\mathbf{y}^{(i)}, \mathbf{y})}. \quad (1)$$

その得られたラベル $\bar{\mathbf{y}}$ に対して, 正解事例との間にマージンが十分確保出来るよう係数 τ を決める. 7 行目では τ と $\bar{\mathbf{y}}$ によりモデルを次のように更新する:

$$\mathcal{W} = \mathcal{W} \cup \{-\tau, \mathbf{x}^{(i)}, \mathbf{y}^{(i)}\} \cup \{\tau, \mathbf{x}^{(i)}, \bar{\mathbf{y}}\}.$$

構造学習とカーネルを同時に利用するには, 大きく二つの問題が存在する. 一つはこの式 (1) から明らかなように, サポートベクトルの数が増えるに従い線型モデルの時と比べ分類のための計算量は増加する. 構造学習やオンライン学習アルゴリズムでは, 各反復でインスタンスを分類するため分類時だけでなく学習時にも計算量は増加する.

また, カーネルによって射影された空間上で分類を行うため, ラベルが持つ構造の要素やその素性に対応するスコアを得ることができない. このため, 従来の多くのデコーディングのためのアルゴリズムは用いることが難しい. さらに, 4 行目や分類時の最大化問題は複数のカーネル関数の和に対して最大化する必要があるため, 効率的に解をもとめることもまた難しくなっている.

3. Problem Settings

ここで扱う問題を以下のように定義する.

- ラベル $\mathbf{y}$
  ここではラベルとして二値の長さ m のベクトルを仮定する. グラフやシーケンスなどの多くのデータ構造はバイナリベクトルとして表せる.
- インスタンス $\mathbf{x}$
  本稿では $\mathbf{x}$ はインスタンスを表す素性のベクトルとする.
- Φ 関数
  Algorithm 1 では, ラベルと事例から素性ベクトルを生成するために Φ 関数が使われている. 本論文では Φ 関数として以下のような関数を定義する:
  $\Phi(\mathbf{y}, \mathbf{x}) = (y_1\mathbf{x}) \oplus (y_2\mathbf{x}) \oplus \cdots \oplus (y_m\mathbf{x})$.
  ただし, $\oplus$ はベクトルの連結を表す演算子とし, ラベル長を m, ラベルの各要素を y_i とする. 例えば,
  $\mathbf{y} = (1,0,1,0), \mathbf{x} = (1,2,3,4)$ とすれば,
  $\Phi(\mathbf{y}, \mathbf{x}) = (1,2,3,4,\ 0,0,0,0,\ 1,2,3,4,\ 0,0,0,0)$
  となる.
- ロス関数
  Algorithm 1 の 4, 5 行目に用いられる誤分類のコストを表すロス関数 $\rho(\mathbf{y}, \mathbf{y}')$ として, 二値のベクトルであるラベルに対して 3 種類のロス関数を定義する. 詳しくは 5. 節で解説する.

4. Transformation of Polynomial Kernel

自然言語処理にカーネル学習器を用いる場合, 単語の組み合わせや係り受けの組み合わせ等を扱うため, 多項式カーネルが用いられることが多い. ここではラベル長は m とし, カーネル $K(\mathbf{v}, \mathbf{v}')$ は p 次の多項式カーネルを表すとする, このカーネルを変形することで計算量を抑える工夫を行う.

4.1 Φ 関数とカーネルの統合

カーネルを計算するためには, その都度 Φ 関数で生成するか, 事例ベクトルを保持する必要がある. しかし, 事例ベクトルは $N2^m$ 存在するため, 全ての事例ベクトルを保持することは困難である. このため, ここでは Φ 関数とカーネルを統合し, Φ 関数を経由せずカーネルを計算する方法について説明する.

ここで順を追って Φ 関数により生成されたベクトル同士のカーネルを計算してみる. 簡単のため多項式カーネルの定数項は省くが, 定数項がある場合も同様に計算できる. p 次の多項式カーネルは次のように展開することが出来る:

$$\begin{aligned}
& K(\Phi(\mathbf{y}, \mathbf{x}), \Phi(\mathbf{y}', \mathbf{x}')) \\
&= K(y_1 \cdot \mathbf{x} \oplus y_2 \cdot \mathbf{x} \oplus \ldots \oplus y_m \cdot \mathbf{x}, \\
&\quad {y'}_1 \cdot \mathbf{x}' \oplus y_2 \cdot \mathbf{x}' \oplus \ldots \oplus y_m \cdot \mathbf{x}') \\
&\quad \text{カーネルを展開する} \\
&= ((y_1 \cdot \mathbf{x} \oplus y_2 \cdot \mathbf{x} \oplus \ldots \oplus y_m \cdot \mathbf{x}) \cdot \\
&\quad ({y'}_1 \cdot \mathbf{x}' \oplus {y'}_2 \cdot \mathbf{x}' \oplus \ldots \oplus {y'}_m \cdot \mathbf{x}'))^p \\
&\quad \text{カーネル空間内であれば内積は各ラベル} \\
&\quad \text{要素に対応する区間ごとに行える} \\
&= \{(y_1 \cdot {y'}_1)(\mathbf{x} \cdot \mathbf{x}') + (y_2 \cdot {y'}_2)(\mathbf{x} \cdot \mathbf{x}') \\
&\quad + \ldots + (y_m \cdot {y'}_m)(\mathbf{x} \cdot \mathbf{x}')\}^p \\
&\quad \text{素性ベクトルの積をくくり出す} \\
&= ((\mathbf{y} \cdot \mathbf{y}')(\mathbf{x} \cdot \mathbf{x}'))^p.
\end{aligned}$$

このように, ラベル同士の積と素性ベクトル同士の積でカーネルを表すことができるため, 実は Φ 関数を介さずに多項式カーネルを計算することが出来る. ここで, Φ 関数と統合されたカーネルを K_{ex} とする:

$$K_{ex}(\mathbf{y}, \mathbf{x}, \mathbf{y}', \mathbf{x}') = ((\mathbf{y} \cdot \mathbf{y}')(\mathbf{x} \cdot \mathbf{x}'))^p. \quad (2)$$

直感的な説明は, 図 1 を参照してもらいたい. カーネルを統合す

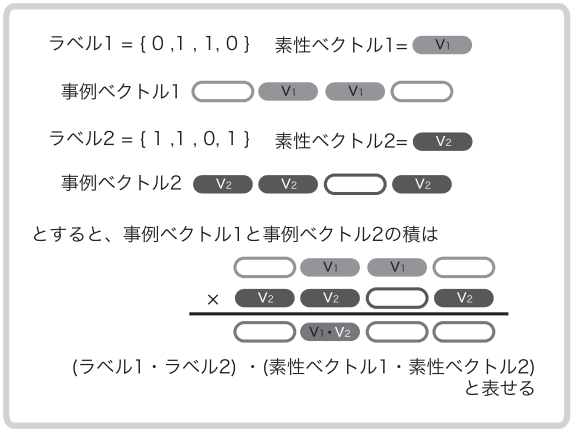

図 1　Φ 関数とカーネルの統合の概要

ることにより, ラベル長に関わらず素性ベクトル間の内積のみの計算でカーネルを計算することが出来る.

Φ 関数とカーネルの統合をしない場合, カーネルの計算には, 計算量とメモリともに(ラベル長) $\times$ (素性ベクトル長) だけのコストがかかる. 一方この拡張を行うことにより, 計算する要素数は(ラベル長) + (素性ベクトル長) となる. また, ラベルを考えずにインスタンス間のカーネルのみをキャッシュすればよいため, キャッシュ効率も大きく向上する.

4.2 二次多項式カーネルの展開

さらに, 出来る限りカーネルを呼び出す回数を減らすため, カーネルを展開することを考える. カーネルを 2 次の多項式カーネルに限定すると, 統合されたカーネルを $\mathbf{y}$ の要素である y_i の組み合わせごとに分割することができる. ここでも多項式カーネルの定数項は省くが, 定数項がある場合も同様に計算を行うことが出来る. 統合されたカーネル K_ex を以下のように分割する:

$$\begin{aligned}
& K_{ex}(\mathbf{y}, \mathbf{x}, \mathbf{y}', \mathbf{x}') \\
&= ((\mathbf{y} \cdot \mathbf{y}')(\mathbf{x} \cdot \mathbf{x}'))^2 \\
&= ((\mathbf{y} \cdot \mathbf{y}')^2 (\mathbf{x} \cdot \mathbf{x}')^2) \\
&\quad y \text{ を要素 } y_i \text{ごとに分割する} \\
&= (({y_1} \cdot {y'}_1) + \cdots + ({y_m} \cdot {y'}_m))^2 (\mathbf{x} \cdot \mathbf{x}')^2 \\
&\quad \text{二乗を展開する} \\
&= \sum_{i|i \le m} y_i {y'}_i (x \cdot x')^2 + \sum_{i,j|i \ne j,\ i,j<m} y_i^2 {y'}_j^2 (x \cdot x')^2.
\end{aligned}$$

ここで, 事例 $(\mathbf{y}, \mathbf{x})$ とモデルに含まれるサポートベクトルとのカーネルを計算し, 事例ベクトルのスコアを計算することを考えると,

$$\sum_{(\tau^{(t)},\mathbf{y}^{(t)},\mathbf{x}^{(t)})\in\mathcal{W}} \tau^{(t)} K_{ex}(\mathbf{y},\mathbf{x},\mathbf{y}^{(t)},\mathbf{x}^{(t)})$$

$$= \sum_{(\tau^{(t)},\mathbf{y}^{(t)},\mathbf{x}^{(t)})\in\mathcal{W}} \left\{ \sum_{\{i,j|i\neq j,\ i,j<m\}} \tau_t y_i y_j y_i^{(t)} y_j^{(t)} (\mathbf{x}\cdot\mathbf{x}^{(t)})^2 + \sum_{\{i|i<m\}} \tau^{(t)} y_i y_i^{(t)} (\mathbf{x}\cdot\mathbf{x}^{(t)})^2 \right\}$$

ここで $\gamma_{ij}^{(t)} = \tau^{(t)} y_i^{(t)} y_j^{(t)} (\mathbf{x}\cdot\mathbf{x}^{(t)})^2$ と置くと

$\mathbf{y}$ について $\gamma_{ij}^{(t)}$ は定数であるので

$$= \sum_{\{i,j|i\neq j,\ i,j<m\}} y_i y_j \sum_{\{t|(\tau^{(t)},\mathbf{y}^{(t)},\mathbf{x}^{(t)})\in\mathcal{W}\}} \gamma_{ij}^{(t)} + \sum_{\{i|i\leq m\}} y_i^2 \sum_{\{t|(\tau^{(t)},\mathbf{y}^{(t)},\mathbf{x}^{(t)})\in\mathcal{W}\}} \gamma_{ii}^{(t)}$$

ここでさらに $\gamma_{ij} = \sum_{\{t|(\tau^{(t)},\mathbf{y}^{(t)},\mathbf{x}^{(t)})\in\mathcal{W}\}} \gamma_{ij}^{(t)}$ と置くと

$$= \sum_{\{i,j|i\neq j,\ i,j<m\}} y_i y_j \gamma_{ij} + \sum_{\{i|i\leq m\}} y_i^2 \gamma_{ii}$$

このように, 全てのサポートベクトルをパラメータ γ としてまとめることができ, 同じ素性ベクトル $\mathbf{x}$ とモデルに対しては $\mathbf{y}$ が変わるごとにカーネルを呼び出すことなく γ からスコアを計算することができる.

4.3 分類損失最大化ラベル

Algorithm 1 の 4 行目では, 最大化問題を解いているが, このカーネルの展開を利用し, 効率的に分類損失を最大化するラベル $\bar{\mathbf{y}}$ を求める. 分割したカーネルを使い, $\bar{\mathbf{y}}$ を求めるアルゴリズムを Algorithm 2 に示す. アルゴリズム中の, $\mathbf{1}$ は要素が全て 1 のラベル, $\tau^{(k)}$ は各サポートベクトルの重みである. $\mathbf{sv}_{ki} = \Phi(\mathbf{e}_i, \mathbf{SV}_k)$ は k 番目のサポートベクトルと i 番目の要素のみ 1 となっているラベルから Φ 関数により生成される事例ベクトルを表す. これ以降, 定数項を含めて γ の計算を行うため $y^{(0)}$ は常に 1 であると定義し, ラベル長 m のラベルは暗黙に定数項 $y^{(0)}$ を含むとする.

γ の計算に $n(m+1)^2$ 回のカーネル呼び出しが必要となるが, 各ラベルについては γ を係数とした多項式の計算を行うだけで評価できる. このため, ラベル全ての評価についてカーネルの呼び出し回数は $n(m+1)^2$ 回のみですむ. また, Passive Aggressive Algorithm では増補的にモデルを学習するため, 一度モデルに加えられたサポートベクトルの重みが変わることはない. このため, γ を各事例ごとに保持しておけば, γ を新しくモデルに追加されたサポートベクトルについて更新することができ, さらに計算量を抑えることが出来る.

4.4 Analysis

ここでは, 学習に必要な計算量について考察する. まず, 事例数を N, ある時点でモデルに含まれるサポートベクトル数を h, 最終的なサポートベクトル数を H, H のサポートベクトルを得るために必要な反復回数を I, ラベル長を m とする. 学習中, 訓練データに対する誤分類率が一定と仮定すると, 1 つのサポートベクトルを得るために必要な事例の分類回数 λ は, $\lambda = \frac{N\,I}{H}$ となる.

Algorithm 2 for finding $\bar{\mathbf{y}}$.

Input: $\mathbf{x} = \Phi(\mathbf{1}, \mathbf{x}), \tau, \mathcal{W} = \{\mathbf{sv}_1, \ldots, \mathbf{sv}_n\}$
Input: $\mathbf{true\_y}$ //正解となるラベル (学習時のみ)
1: **for all** $\{(i,j)|0 < i \leq j \leq m\}$ **do**
2: $\gamma_{ij} = \sum_{\mathbf{sv}_k\in\mathcal{W}} \tau^{(t)} \beta_{ij} K(\mathbf{sv}_{ki}, \mathbf{x}) K(\mathbf{sv}_{kj}, \mathbf{x})$
$\beta_{ij} = \begin{cases} 1 & if(i=j) \\ 2 & otherwise \end{cases}$
3: **end for**
4: **for** $0 \leq i \leq m$ **do**
5: $\gamma_{0i} = \gamma_{i0} = \sum_{\mathbf{sv}_k\in\mathcal{W}} \tau^{(t)} K(\mathbf{sv}_{ki}, \mathbf{x})$ // 定数項の処理
6: **end for**
7: $\bar{y} = \underset{(y_1,\ldots,y_m)\in\{0,1\}^m}{\operatorname{argmax}} \sum_{\forall i,\forall j\in(0\leq i\leq j\leq m)} \gamma_{ij} y_i y_j + \sum_{i=1}^{n} \gamma_{i0} y_i$ //分類時
8: $\bar{y} = \underset{(y_1,\ldots,y_m)\in\{0,1\}^m}{\operatorname{argmax}} \sum_{\forall i,\forall j\in(0\leq i\leq j\leq m)} \gamma_{ij} y_i y_j + \sum_{i=1}^{n} \gamma_{i0} y_i + \rho(\mathbf{true\_y}, \mathbf{y})$ //学習時
9: return $(\bar{\mathbf{y}})$

- カーネルを展開しない場合
  各事例の評価に $h2^m$ 回のカーネル呼び出しが必要となり, 1 つのサポートベクトルを得るためにカーネルを呼び出す回数は $h\lambda$ 回となる. このとき, H 個のサポートベクトルを得るまでに必要なカーネルの呼び出し回数は, $\sum_{h=1}^{H} h\lambda 2^m = NI(H+1)2^{m-1}$ となる.
- カーネルを展開する場合
  各事例の評価に必要なカーネルは各イテレーションで追加されたサポートベクトルについて計算すればいいので, 平均して $\frac{H(m+1)^2}{I}$ 回カーネルを呼び出す必要がある. このとき, H 個のサポートベクトルを得るまでに必要なカーネルの呼び出し回数は, $H\frac{H(m+1)^2}{I}\lambda = NH(m+1)^2$ となる.

5. ロス関数

構造学習ではラベル間の距離はロス関数として与えられる. これにより, "近い間違い" と "遠い間違い" を表すことができるようになり, 近い間違いには小さいペナルティ, 遠い間違いには大きいペナルティを設定することが出来る. 出力としてバイナリのベクトルをとるため, 以下の 2 つのロスを定義する. ロス関数のパラメータ s はロス関数をスケールするためのパラメータとなっている.

- 平均ロス

$$\rho_{average}(\mathbf{y}, \mathbf{y}') = \frac{1}{m} \sum_{i=1}^{m} \begin{cases} 0 & if\ y^{(i)} = y'^{(i)} \\ s & otherwise \end{cases} \quad (3)$$

  $s\times$(ラベル中の異なる要素の数) を, ラベル長で割った値を返す. これは, ラベルの要素を一つ間違うよりも, 複数のラベルの要素を間違う方がより遠い間違いであるとすることにあたる. 近い間違いに対しては小さいマージン, 遠い間違いに対しては大きなマージンをとるようにすることで, より自然な学習ができる.
- 非対称ロス

$$\rho_{asymmetric}(\mathbf{y}, \mathbf{y}') = \frac{1}{m} \sum_{i=1}^{m} \begin{cases} 0 & y^{(i)} = y'^{(i)} \\ s\cdot j_i & y^{(i)} = 1, y^{(i)} \neq y'^{(i)} \\ s & otherwise \end{cases} \quad (4)$$

ポジティブな要素をネガティブへと間違った場合にそれぞれ j_i 倍の値を返す. ラベルの要素に, ポジティブな要素とネガティブな要素で出現頻度に大きな偏りがある場合, どちらかに偏った学習がおこなわれ, どちらかが大きすぎるマージンをとり, もう片方が十分なマージンをとれず, 丸暗記的な学習がなされてしまうことがある. この偏りを修正し適正なマージンを確保するため, ラベルの要素ごとに異なるパラメータ j_i を与えることで、ポジティブな要素とネガティブな要素で間違った場合のペナルティに差をつけている.

6. 実験

提案手法の有効性を確かめるため, 賛否関係の推定をタスクとした実験を行う. 賛否関係の推定とは, 各発話について賛成を示すかどうか, 否定を示すかどうか, 各発話間にリンクがあるかどうか, を推定する問題である.

6.1 実験設定

本研究では MRDA コーパス [3] を用いて実験を行う. このコーパスは ICSI で集められた, 複数人による 75 のミーティングの書き起こしテキストと音声データを含む. ミーティングは ICSI の様々な研究チームにより毎週, 平均 6.5 人, 一時間以内で行われる. このコーパス中の全発話に対して意図タグ, 発話者, 隣接応答ペアタグ等がラベル付けされている. 本研究ではこの意図タグや隣接応答ペアタグを元に賛否と発話間のリンクを定義した.

今回の実験では 3 発話間に区間を区切った賛否関係を対象とする. 3 発話間の賛否関係を表すラベル長 m は, 定数項をのぞいて $m = 9$ となる. 各ラベルは, 各発話が賛成を示すか否か, 否定を示すか否か, 各発話間にリンクがあるか否かを表す. 賛否関係中の各発話を, 過去に発話されたものから順に 1 発話目, 2 発話目, 3 発話目と呼ぶ. 提案手法では, 区間を 1 発話ずつスライドさせて, 学習及び分類を行っている. つまり, ある事例で 2 発話目だった発話は, 次の事例では 1 発話目となる.

分類に使う素性としては提案手法, ベースラインともに, 次の素性を前後 3 発話ずつ, 合計 7 発話間分を用いる. 発話内容を表す素性として, 発言語数, uni-gram, bi-gram, tri-gram, 先頭 2 語の 5 種, 発話間の関係を表す素性として, 同一話者かどうか, 発言間の時間間隔 の 2 種を用いた.

この問題は, 賛成, 反対, 発話間のリンクの有無, それぞれのクラスに属する事例数が大きく偏っているため, 正解率ではうまく評価できない. このため, F 値を用いてすべての評価を行う. 訓練データに 12499 事例, テストデータに 9200 事例を用いている. カーネルは全て二次多項式カーネルとし, 定数項は 1 に固定して実験を行う.

6.2 ロスの効果

ロス関数を変更し実験を行った結果を表 1 に示す. 非対称ロスの結果を平均ロスと比べると, 全てのラベル要素について良くなっている. これはポジティブな要素とネガティブな要素の出現頻度の偏りを, j の値を変えることで間違った時のペナルティに差を付け, 吸収できたためだと考えられる. 賛成とリンクでも向上が見られるが, 特にポジティブな例が少ない否定の分類において顕著に性能が向上している.

6.3 カーネルの効果

賛成と否定, リンクでそれぞれ最適な重みを選び, 線型モデルとの比較を行う. 今回は, 各要素ごとに次のようにパラメータを指定した:

$j_{\text{賛成}} = 3, j_{\text{否定}} = 10, j_{\text{リンク}_{1-2,2-3}} = 6, j_{\text{リンク}_{1-3}} = 5$. また, ロス関数の倍率 s は 5 とした. 以上のパラメータを用いて学習した提案手法と, 線形モデルで学習を行った場合を比較する.

表 1　ロス関数の性能に与える影響

ロス関数	0/1	平均	非対称 (j=**3**)	$j=$ **10**	$j=$ **50**	$j=$ **100**
1 発話目 賛成	0.337	0.394	0.490	**0.526**	0.500	0.479
2 発話目 賛成	0.170	0.343	0.462	**0.510**	0.497	0.460
3 発話目 賛成	0.097	0.237	0.414	**0.514**	0.507	0.464
1 発話目 否定	0.016	0.000	0.134	0.245	**0.320**	0.287
2 発話目 否定	0.000	0.032	0.032	0.201	**0.306**	0.277
3 発話目 否定	0.000	0.000	0.076	0.258	**0.320**	0.316
1-2 発話間 リンク	0.030	0.074	0.305	0.472	**0.508**	0.445
2-3 発話間 リンク	0.021	0.088	0.366	**0.529**	0.518	0.456
1-3 発話間 リンク	0.012	0.083	0.277	0.431	**0.450**	0.413

表 2　線形モデルとの比較

学習器	提案手法	線形
賛成	**0.512**	0.501
否定	**0.316**	0.263
2-3 発話間 リンク	0.550	0.494
1-3 発話間 リンク	0.454	0.418

6.4 時間性能評価

また, カーネルの拡張・展開を行う場合と行わない場合の実行時間を評価する実験を行う. イテレーション回数は 15 回とし, ロス関数には平均ロス, ロス関数の倍率 s には 10 倍を用いる. 実行時間を計測し, 比較した結果を表 6.4 に示す. カーネルの非展開, 展開の場合ともに実行時間はおおよそ (事例数) × (サポートベクトル数) に比例している. この実行時間の差は, オーバーヘッドを差し引いて考慮すると, カーネルの呼び出し回数が 512 倍, カーネルあたりの計算量が 9 倍の, あわせて 13500 倍という理論上の計算量の差とおおむね一致している.

表 3　実行時間

訓練データの事例数	50	100	150
サポートベクトルの数	118	183	294
カーネル非変形	15523s	56227s	139590s
カーネル変形	3.19s	8.14s	28.65s
倍率	4866 倍	6907 倍	4872 倍

7. Conclution

本研究では賛否関係の推定を構造学習の問題として捉えることで, 発話間のリンクと発話の示す賛否の推定という, 異なる関連した推定を同時に行う手法を提案した. 提案手法は, Passive Aggressive Algorithm というオンライン最大マージン学習に対して拡張を行っており, 高速かつメモリ効率よく学習を行うことが可能である. 発話間の賛否関係を推定するというタスクで実験を行い, 理論的な計算量を確認し, 手法の有効性を確認した.

参考文献

[1] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, Yoram Singer, Online Passive-Aggressive Algorithms. *Journal of Machine Learning research*, Vol.7, Mar, pp.551–585, 2006.

[2] Ioannis Tsochantaridis, Tohomas Hofmann, Thorsten Joachims, Yasemin Altun, Support Vector Learning for Interdependent and Structured Output Spaces. In *Proceedings of the twenty-first international conference on Machine learning* , pp.823–830, 2004.

[3] Elizabeth Shriberg, Raj Dhillon, Sonali Bhagat, Jeremy Ang, and Hannah Carvey, The ICSI meeting recorder dialog act (MRDA) corpus. In *Proceedings of the 5th SIGdial Workshop on Discourse and Dialogue*, pp.97–100, 2004.

[4] Ryan McDonald, Koby Crammer , Fernando Pereira, Online Large-Margin Training of Dependency Parsers. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics* , pp.91–98 ,2005.

[5] Jiampojamarn Sittichai , Cherry Colin , Kondrak Grzegorz, Joint Processing and Discriminative Training for Letter-to-Phoneme Conversion. In Proceedings of ACL-08: HLT, pp.905–913, 2008.