第2章

関連研究

2.1 序言

本研究を構成する重要な要素として，COLLAGREE，議論支援，話題遷移検出，重み付け，分散表現 が挙げられる．本章では各要素について説明しながら関連研究を紹介する．

本章の構成を以下に示す．まず，2.2 節では，COLLAGREE の概要について簡単に述べる．2.3 節では，既存の議論支援に関する研究について述べる．2.4 節では話題に関する本研究の見方や既存の話題に関する研究について述べる．次に，2.5 節では本研究の重要な要素である重み付けについて述べる．2.6 節では本研究のもう 1 つの重要な要素である分散表現及び分散表現を使用した話題関連研究について述べる．最後に，2.7 節で本章のまとめを示す．

2.2 COLLAGREE

2.2.1 概要

COLLAGREE は掲示板のような議論プラットフォームをベースにしており，各ユーザーが自由なタイミングで意見を投稿，返信できる．こうした議論掲示板は基本的には 1 つの議論テーマに対して関連するテーマを扱った複数のスレッドから構成される．スレッドとはある特定の話題・論点に関する 1 つのまとまりを指す．例えば，図 2.1 のようにスレッドを立てたユーザーの発言が親意見となり，他のユーザーが子意見として親意見に返信し，子意見に対して孫意見が存在する場合もある．

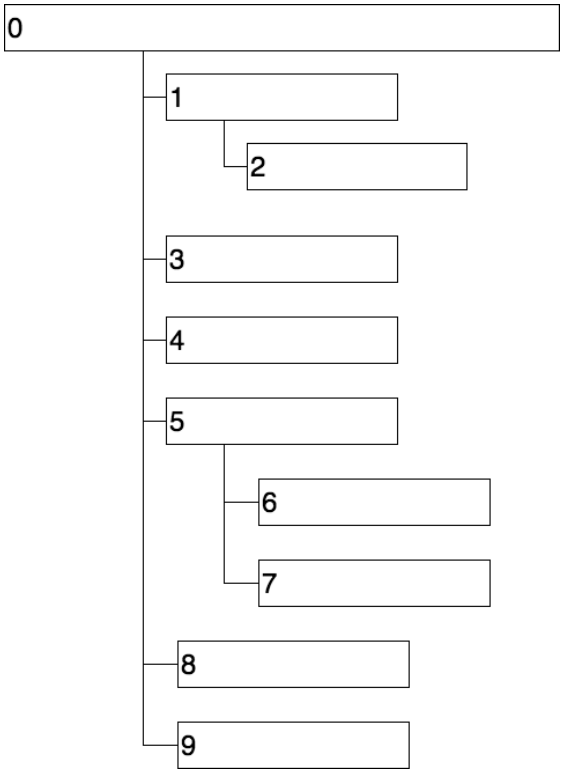

図 2.1: COLLAGREE のスレッドの例

しかし，スレッドや返信は必ずしも正しく使われるとは限らず，単なるチャット

と同様に扱われてしまいスレッドが乱立してしまうこともある．

2.2.2　ファシリテーター

COLLAGREE ではファシリテーターと呼ばれる人物が議論のマネジメントを行っている．ファシリテーターは”促進者”を意味し，議論そのものには参加せず、あくまで中立的な立場から活動の支援を行うようにし，自分の意見を述べたり自ら意思決定をすることはない．基本的な役割として”議論の内容の整理”，”議論の脱線防止”，”意見の促し”等が挙げられる．伊藤ら [?] によってファシリテーターの存在や手腕が合意形成に強く影響を与えることが示されている．

2.3　議論支援

議論は収束を目標として互いの意見を述べ合うことで，何かを決定する際において重要な社会活動である．故に注目が集まり，COLLAGREE 以外でも議論内容の把握支援を行い，議論の進行を支援するための研究が行われている．小谷ら [?] は好意的発言影響度を取り入れた議論支援システムを開発した．システムは議論中の発言の意図や内容に加えて，発言に対するリアクション (同意、非同意、意見) などから議論進行をモニタリングしている．モニタリングの結果を基にして議論の活性化や深化に対して参加者が果たしている役割を”好意的発言影響度”として定量化して表示する．

本研究はファシリテーターに対する支援を目的としており，一般参加者や学習者の議論活性化及び収束に向けた支援が目的ではない．

2.4　話題遷移検出

話題に関連する研究は長い間行われており，古くは「会話分析」という学問に始まる．会話分析は会話を始めとする相互行為の組織 (構造) を明らかにしようとする社会学の研究分野であり，相互行為の分析法も取り扱う．1960 年代に Harvey Sacks と Emanuel A.schegloff によって創案された．話題というものは、本来流れの一時点で簡単に区切ることのできるものではない．しかし、研究によっては話題内容と連鎖組織及び言語形式との関連についての分析を行うことがある．筒井 [?] は話題を区切るのに次のように基準を立てた．

1. それまで話題となっていた対象や事態とは異なる，新しい対象や事態への言及
2. すでに言及された対象や事態の異なる側面への言及
3. すでに言及された対象や事態の異なる時間における様相への言及
4. すでに言及された対象や事態について，それと同種の対象や事態への言及
5. すでに言及された個別の対象や事態の一般化

本研究では上記の基準を参考に話題変化の判定の評価の基準を設けている．

話題の遷移に基づいた文章の分割は人間によるテキスト全体の内容の把握の容易化や複数のテキストに対する自動分類や検索の精度向上のために研究されている．以下では話題の遷移に関する関連研究について述べる．

2.4.1　テキストセグメンテーション

テキストセグメンテーションは複数のトピックが混合的に書かれている非構造的である文書をトピックに応じて分割する手法である．

TextTiling

Hearst ら [?] は複数の単語を連結した 2 つのブロックをテキストの初めから末尾まで動かしていきブロック間の類似度を計算する手法 (TextTiling) を提案した．類似度はブロック間で共通して使われる単語数などで計算する．図 2.2 に手法の簡単な流れを示す．

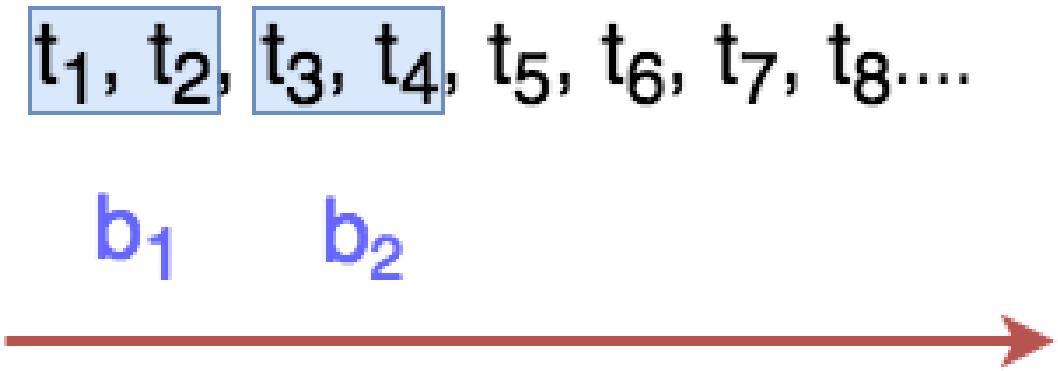

図 2.2: TextTiling の流れ

計算された類似度をグラフにすると図 2.3 のようになる．縦軸はブロック間の類似度，横軸はブロック間の番号を表し，グラフ中の番号は段落番号，垂直線は選出された文の境界位置を表す．グラフは各ブロック間の類似度を繋いだものと，更に平滑化したものの 2 つが描かれている．グラフの谷のような場所は左右のブロック間の類似度が下がる場所を表し，使われる単語が変わったこと，すなわちトピックが変わったことを意味し，谷の付近に境界を設けることで文書をトピックごとに区切ることを可能としている．

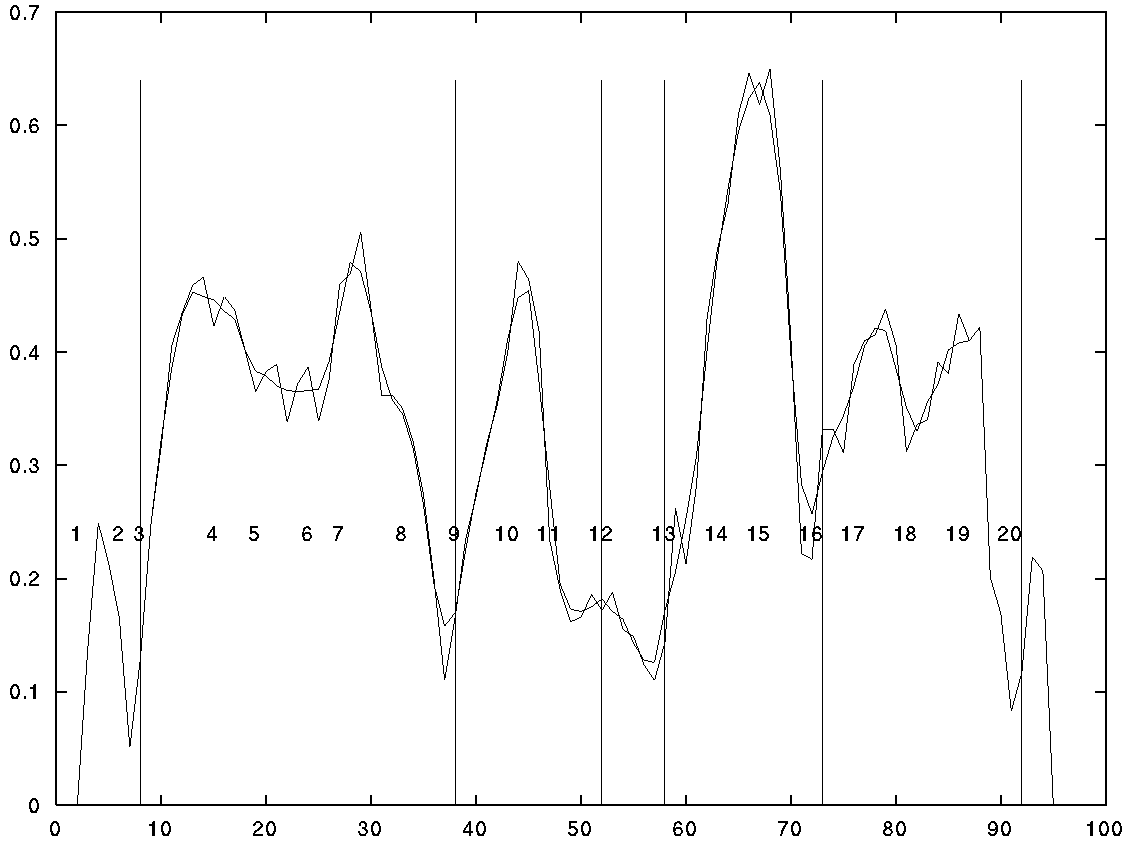

図 2.3: TextTiling のグラフ

テキストセグメンテーションを使用した関連研究

別所 [?] は単語の共起頻度行列を特異値分解で次元を削減して作成した単語の概念ベクトルを用いて新聞記事の分割を行っている．ブロック間の類似度を計算する際にブロック中の単語の内，自立語にのみ概念ベクトルを付与し左右のブロックの和ベクトル (または重心ベクトル) の余弦測度を求め，類似度 (または結束度) としている．

テキストセグメンテーションは基本的にある地点 A で話題に沿って分割をするか考える際に，地点 A より先の情報を使うことができる．すなわち，ある程度の文章が現れてから話題が変わったかどうかの判定をしており，リアルタイムでの動作を想定していない．本研究はリアルタイムでの動作を想定し，ある発言 B が話題の変化を起こすかどうか判定する際に B より先の情報を使うことなく判定している点で異なる．

2.4.2 トピックモデル

トピックモデルは，確率モデルの一種にあたり，文章中の「単語が出現する確率」を推定している．単語が出現する確率をうまく推定することができれば，似たような単語が出てくる文章が把握できる．すなわち，トピックモデルとは「文書における単語の出現確率」を推定するモデルといえる．

ユニグラムモデル

図は四角の箱が文書を表し，中身の色がトピックを表す．ユニグラムモデルは，すべてのボックスが同じ青色である．全ての文書の単語は，一つのトピックから生成されたものと仮定するモデルである．

図 2.4: ユニグラムモデル

混合ユニグラムモデル

混合ユニグラムモデルは，各ボックスの色が異なっている．つまり，各文書に一つのトピックがあり，そのトピックから文書の単語が生成されると仮定するモデルである．

図 2.5: 混合ユニグラムモデル

LDA

LDA(Latent Dirichlet Allocation) では，ボックスの中で色が異なっている．つまり，各文書は複数のトピックで構成されていて，各トピックの単語分布を合算した形で単語が生成されていると仮定を行う．

図 2.6: LDA

トピックモデルは基本的に学習の際にトピックの数を指定する必要があり，指

定に伴いトピック数が制限されてしまう．すなわち，トピックの数によっては変化に対応できない話題が存在することがあり得る．本研究は分散表現を用いており，トピック数の指定がない点で異なる．

2.5 重み付け

重み付けは情報検索を主とする分野で使われる方法で，蓄積された情報中の語の索引語，すなわち特定の情報の特徴を表し検索の手掛かりとなる語，としての重要度を数値的に表現し，それぞれの語の重要度に応じて重みを付け，集計して総合評価を出す手法である．出された総合評価はスコア等と呼ばれる．一般的に語の重要度は整数または実数値で与えられる．自動的な重み付けにおいては語の出現頻度情報や出現位置を利用して数値を与える．本研究で行う重み付けは下記の 2 種類の重み付けを基にしている．

2.5.1 Okapi BM25

Okapi BM25[?] は出現頻度情報を用いる重み付け手法の代表の 1 つである．Okapi BM25 では TF(Term Frequency 単語の出現頻度)[?]，IDF(Inverse Document Frequency 逆文書頻度)[?]，DL(Document Length 文書長) の 3 つを用いて重要度の計算を行う．各用語について説明を行う．

TF

TF は単語の出現頻度を表し，文書中において出現頻度の高い単語は重要であるという考え方に基づく．ある単語 t_i の文書 D_j 中における出現頻度重み $tf_{i,j}$ は式 (2.1)

のようにして求められる．

$$tf_{i,j} = \frac{n_{ij}}{\sum_k n_{kj}} \tag{2.1}$$

ここで n_{ij} は文書 d_j における単語 t_i の出現回数，$\sum_k n_{kj}$ は文書 d_j におけるすべての単語の出現回数の和である．

IDF

IDF は逆文書頻度を表し，多くの文書において出現頻度の高い単語は重要ではないという考え方に基づく．IDF は多くの文書に出現する語，すなわち一般的な語の重要度を下げ，特定の文書にしか出現しない単語の重要度を上げる役割を果たす．ある単語 t_i の逆文書頻度重み idf_i は式 (2.2) のようにして求められる．

$$idf_i = \log \frac{|D|}{|\{d : d \ni t_i\}|} \tag{2.2}$$

ここで $|D|$ は総文書数，$|\{d : d \ni t_i\}|$ は単語 t_i を含む文書数である．

DL

DL は文書長を表し，ある単語の出現回数が同じ 2 つの文書について，総単語数の少ない文書と多い文書では，前者のほうがより価値があるという考え方に基づく．ある文書 d_j の文書長重み ndl_j は式 (2.3) のようにして求められる．

$$ndl_j = \frac{dl_j}{ave(dl)} \tag{2.3}$$

ここで dl_j は文書 d_j の総単語数，$ave(dl)$ はすべての文書の平均 dl を表す．

上記の 3 つの重みを用いて Okapi BM25 は (2.4) のように単語 t_i の文書 D_j 中にお

ける統合重み cw_{ij} を求める．

$$cw_{ij} = \frac{tf_{i,j} \cdot idf_i \cdot (k_1 + 1)}{k_1 \cdot (1 - b + b \cdot ndl_j) + tf_{i,j}} \tag{2.4}$$

ここで定数 k_1 と b について説明する．２つの定数はどちらもチューニングの役割を果すもので k_1 は単語の出現頻度による影響を，b は文書の長さによる影響を調節する．

2.5.2 LexRank

Erkan ら [?] によって考案された LexRank は Google の PageRank[?] を使用した文章要約アルゴリズムで，文の類似度を計算して次の 2 つの基準に基いて文の重要度を計算する．

1. 多くの文に類似する文は重要な文である．

2. 重要な文に類似する文は重要な文である．

LexRank でいう類似度は簡単にいえば 2 文がどれだけ共通の単語を持つかということを表し，文を TF-IDF を用いてベクトル化して Cosine を求めることで類似度としており，式 2.5 に沿ってベクトル x と y の Cosine が計算される．

$$\text{idf} - \text{modified} - \text{cosine}(x, y) = \frac{\sum_{w \in x,y} tf_{w,x} tf_{w,y} (idf_w)^2}{\sqrt{\sum_{x_i \in x} (tf_{x_i,x} idf_{x_i})^2} \times \sqrt{\sum_{y_i \in y} (tf_{y_i,y} idf_{y_i})^2}} \tag{2.5}$$

文の間の Cosine 類似度をグラフとして可視化すると図 2.7 のようになる．各エッジは文の間の Cosine 類似度を表し，$dXsY$ は文書 X の Y 番目の文を示す．

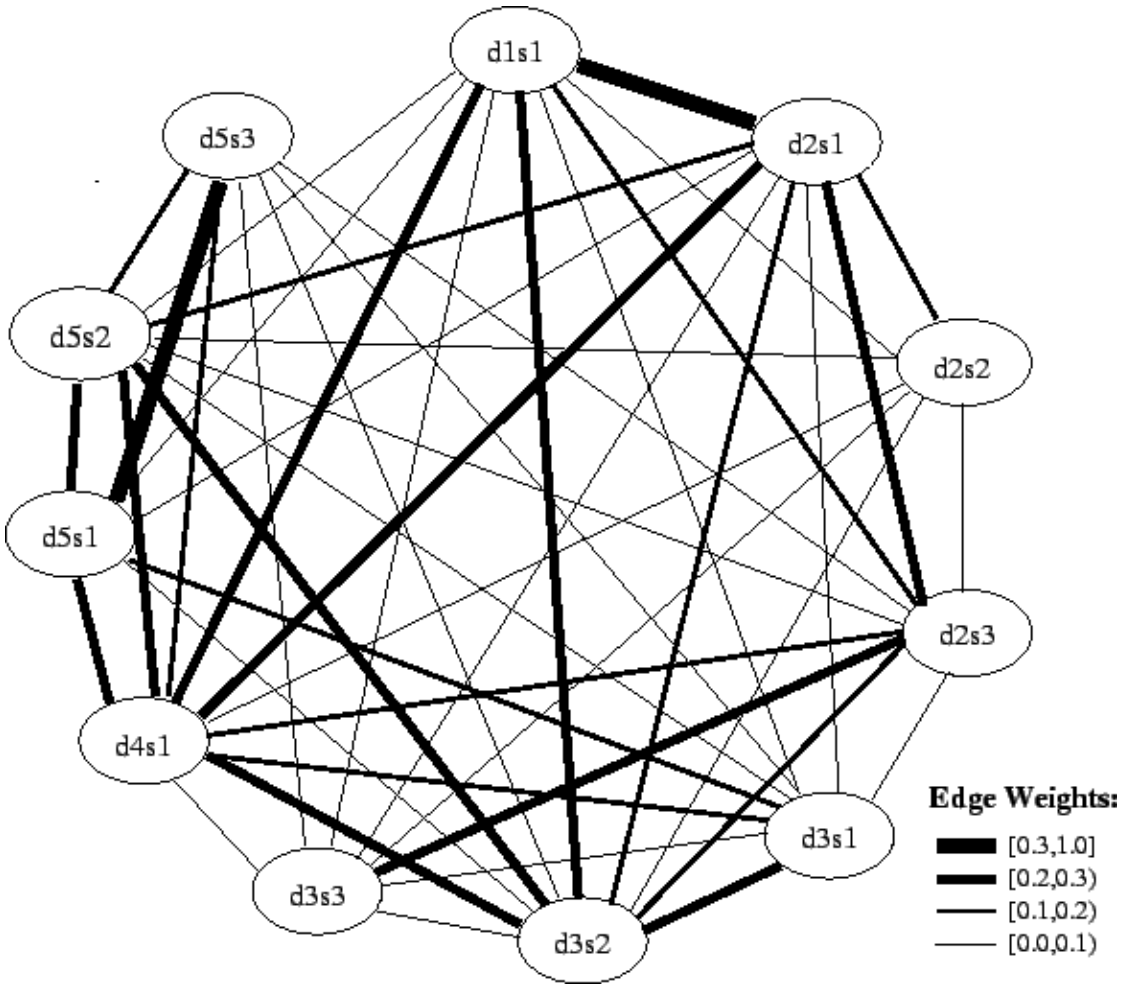

図 2.7: 類似度グラフの例

その後，Cosine 類似度が閾値を超えたかどうかを基に隣接行列が作成される．隣接行列はグラフを表現するために用いられる行列で，あるノード v と w の間にエッジの有無が行列の (v, w) 成分に割り当てられる．隣接行列の各要素を類似している文の数で割り，確率行列に変換した後，**Algorithm1** に従って行列の固有ベクトル $\boldsymbol{p}$ が計算される．求められた固有ベクトルが LexRank スコアとなる．

Algorithm 1 べき乗法の計算アルゴリズム

```
1: Input : 確率的かつ既約かつ非周期的な行列 M
2: Input : 行列サイズ N, 誤差許容値 ϵ
3: Output: 固有ベクトル p
4: procedure powerMethod(M, N,ϵ)
5:     p_0 = (1/N)1;
6:     t = 0;
7:     repeat
8:         t = t + 1;
9:         p_t = M^T p_{t-1};
10:        δ = ||p_t − p_{t-1}||;
11:    until δ < ϵ
12:    return p_t;
```

LexRank スコアを計算する一連のアルゴリズムを **Algorithm2** に示す．7∼14 行目では隣接行列が作成されており，15∼18 行目では LexRank が計算されている．

2.6 分散表現

分散表現は単語を低次元または高次元の実数ベクトルで表現する技術で Harris[**?**] が提唱した”分布仮説”(”同じ文脈で出現する単語は類似した味を持つ傾向があり，単語はその単語とともに出現する単語等によって特徴づけられる．”という考え方) に基づく．

Algorithm 2 LexRank スコアの計算アルゴリズム

```
1: Input : n 個の文からなる配列 S, コサイン類似度の閾値 t
2: Output : 各文の LexRank スコアを格納した配列 L
3: Array  CosineMatrix[n][n];
4: Array  Degree[n];
5: Array  L[n];
6: procedure LexRank(S, t)
7:     for i ← 1 to n do
8:         for j ← 1 to n do
9:             CosineMatrix[i][j] = idf-modified-cosine(S[i], S[j]);
10:            if CosineMatrix[i][j] > t then
11:                CosineMatrix[i][j] = 1;
12:                Degree[i] + +;
13:            else
14:                CosineMatrix[i][j] = 0;
15:    for i ← 1 to n do
16:        for j ← 1 to n do
17:            CosineMatrix[i][j] = CosineMatrix[i][j]/Degree[i];
18:    L = PowerMethod(CosineMatrix, n, ε );
19:    return L
```

2.6.1 単語文脈行列

単語文脈行列は分散表現の最も基本的な形式で図 2.8 のような形の行列で表される．

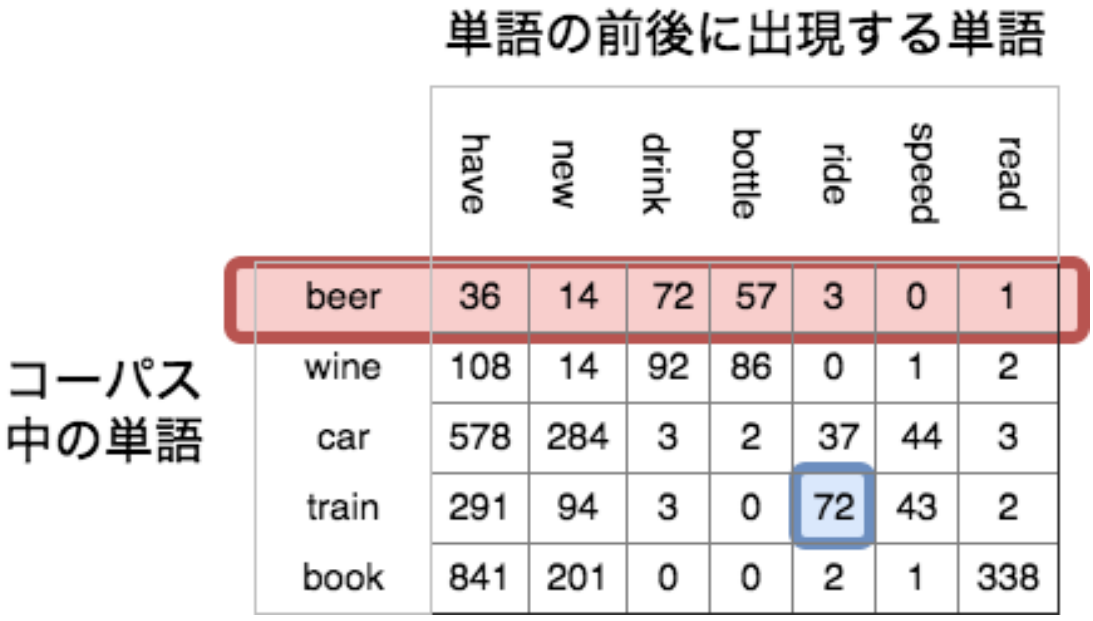

図 2.8: 単語文脈行列

行列中の各要素 M_{ij} は単語 i と文脈 j の共起頻度を表しており，例として青い四角は train と ride が 72 回共起したことを表している．各行 M_i は単語 i の意味ベクトルを表し，例として赤い四角は”beer”の単語ベクトルを表している．

また，単語の類似度を式 2.6 のように単語の意味ベクトルのコサイン類似度で求めることができる．

$$cos\theta = \frac{M_i \cdot M_j}{|M_i||M_j|} \tag{2.6}$$

式 2.6 を図 2.8 の行列に適応させると beer と wine のコサイン類似度は約 0.941，beer と train のコサイン類似度は約 0.387 となり，train よりも wine の方が beer に類似していることが分かる．

2.6.2 word2vec

2.6.1 節で説明した単語文脈行列から得られる単語ベクトルは単語の類似度を求めることは出来たが，他の数学的処理には対応していなかった．Mikolov ら [?] が開発した word2vec はニューラルネットワークを用いて分散表現の生成を行う手法で，CBOW と skip-gram の 2 種類の学習方法が存在する．図 2.9，図 refFig:skip-gram はそれぞれ CBOW と skip-gram の構造を表す．

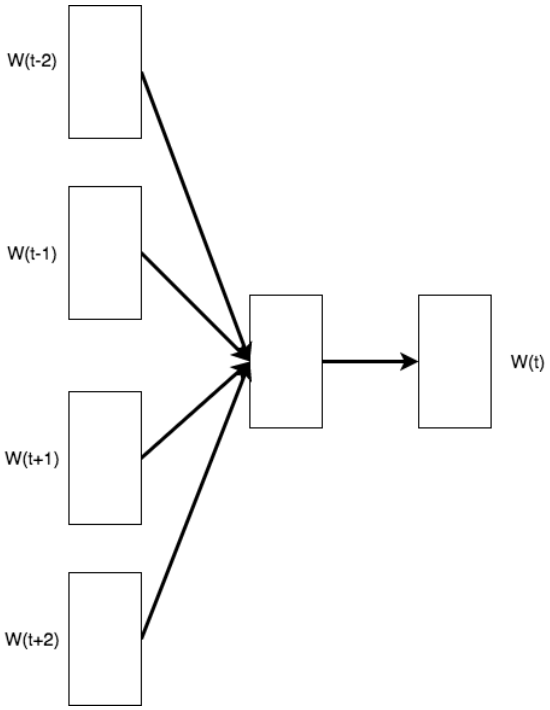

図 2.9: CBOW

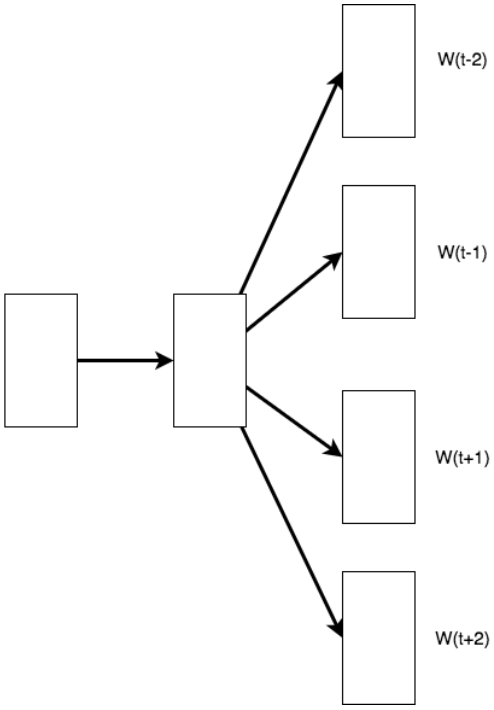

図 2.10: skip-gram

CBOW は周辺の単語 $W(t-2)\cdots W(t+2)$ を入力として，現在の単語 $W(t)$ を予測することを目指して学習する．逆に skip-gram は現在の単語を入力として，周辺の単語を予測することを目指して学習する．どちらの手法でも中間層と単語の変換処理が行われており，学習によって得られる単語と中間層での数値が対応した行列が単語の分散表現モデルとなる．

word2vec が従来の手法と比べて大きく異なる点は　ニューラルネットによる学習で単語の類似度の計算に加えて，ベクトルの加減算が単語の意味の加減算に対応しているということである．また，従来の手法を用いた単語ベクトルよりも類推精度が高いことが Levy ら [?] によって示されている．例えば，$X = vector(\text{"}biggest\text{"}) - vector(\text{"}big\text{"}) + vector(\text{"}small\text{"})$ として計算されたベクトル X に最も類似したベクトルを探すことで biggest が big に類似しているのと同じ意味で small に類似している単語 smallest を見つけることができる．ただし，分散表現モデルが十分に訓練されていることが前提である．

2.6.3 fastText

fastText[?][?] は word2vec を発展させた手法でより大きな語彙や多くの稀な単語に対応することができ，学習の速度を上昇させることにも成功している．

fastText は skip-gram モデルを採用しており，学習の際に単語だけでなく部分語 (単語を構成する文字のまとまり) についても考慮する．以下に単語 w_t が与えられた時に予測した文脈単語 w_c の間のスコア関数を示す．

$$s(w_t, w_c) = \boldsymbol{u}_{w_t}^{\mathrm{T}} \boldsymbol{v}_{w_c} \tag{2.7}$$

$$s(w_t, w_c) = \sum_{g \in \mathcal{G}_w} \boldsymbol{z}_g^{\mathrm{T}} \boldsymbol{v}_{w_c} \tag{2.8}$$

式 2.7 は fastText 以前の手法でのスコア関数を表し，式 2.8 は fastText でのスコア関数を表す．$\boldsymbol{u}_{w_t}$，$\boldsymbol{v}_{w_c}$ はそれぞれ単語 w_t，w_c を実数ベクトルで表したもので，式 2.8 において $\mathcal{G}_w$，$\boldsymbol{z}_g$ はそれぞれ単語 w の n-gram の集合と n-gram g を実数ベク

トルで表現したものを表す．式 2.7 では単語と文脈単語の間のスカラー積をスコアとしているが，式 2.8 では単語の n-gram と文脈単語の間のスカラー積の合計をスコアとしている．式 2.8 の手法を用いることで従来のモデルでは考慮されていなかった”活用形”を考慮できるようになった．例として，単語 go と goes と going は全て go の活用形であるが字面は異なるので従来のモデルでは異なる単語として学習されていたが，fastText では部分語である”go”を 3 つ全てで学習することで意味の近しい単語として学習することが可能となることが挙げられる．

本研究では分散表現の手法として fastText を使用している．

2.6.4 分散表現を使用した話題関連研究

分散表現は関連語を導出できることから分散表現を話題関連に用いる研究が行われている．中野ら [?] は分散表現を用いて雑談対話システムでのシステム側の応答生成を行っている．図 2.11 に話題展開システムの構造を示す．

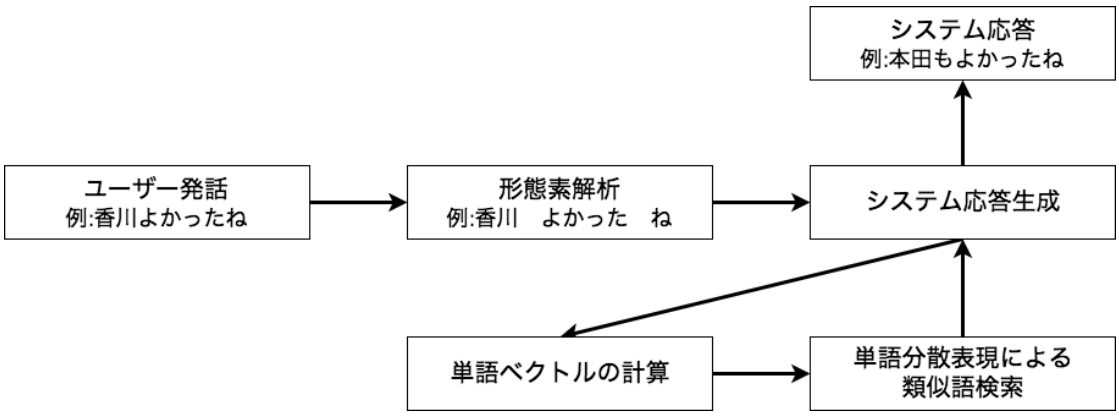

図 2.11: 話題展開システムの構造

システムではユーザ発話を形態素解析して検出された単語を単語分散表現による類似語検索から得られた結果と入れ替えることでシステム応答の生成を行って

いる．

また，Li ら [?] は分散表現を用いて Twitter のツイートをトピックカテゴリに分類する分類器 TweetSift を提案している．図 2.11 にトピックカテゴリの予測のワークフローを示す．

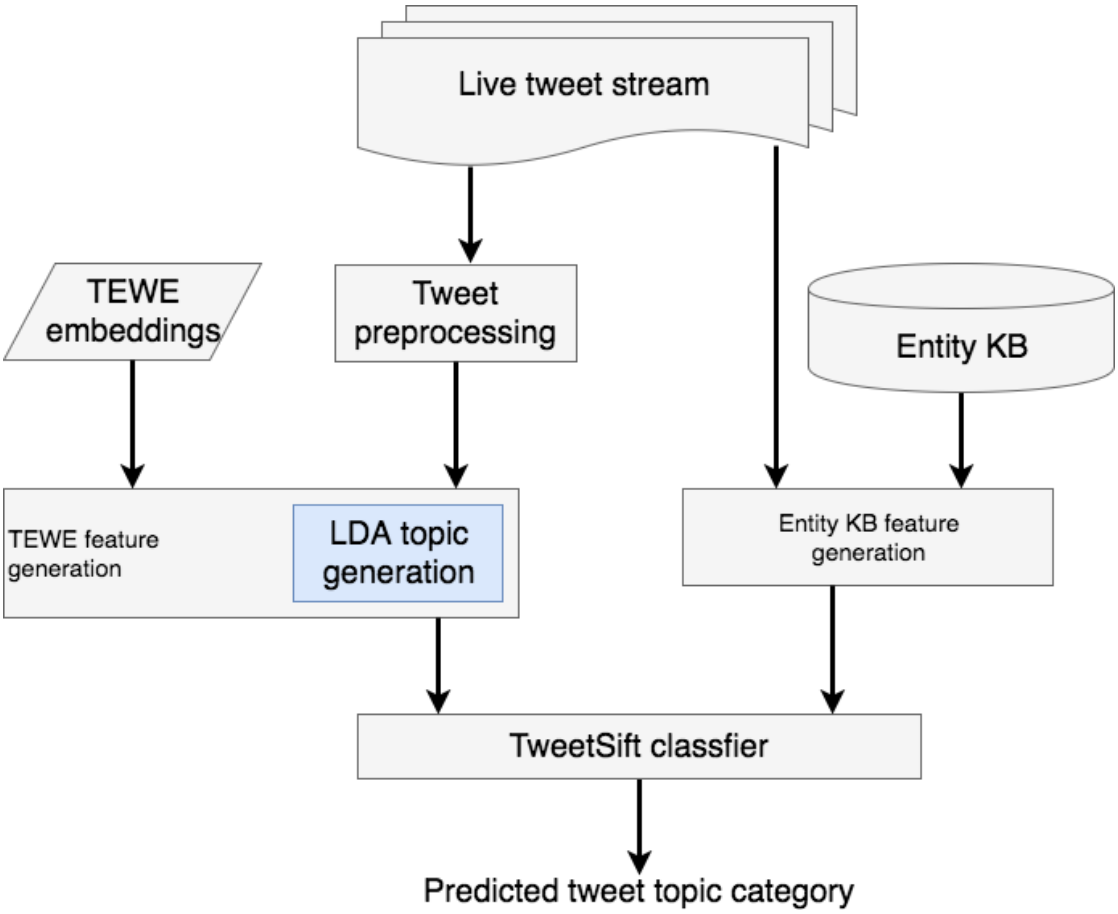

図 2.12: TweetSift によるトピック予測のワークフロー

図の右のフローではツイートデータとスクレイピングで作成された知識ベースを用いて知識ベース特徴を生成している．図の左のフローでは前処理を行ったツイートと作成済み分散表現モデルを用いて分散表現特徴を生成している．２つの特徴を用いて SMO(Sequential Minimal Optimization) によってトピックが予測されている．Li らの研究で用いられる分散表現は学習の際に単語だけでなく，LDA

を用いて予測した単語のトピックも使用されている．

本研究はWeb上での議論を対象としており，対話やSNSを対象としていない点で異なる．また，中野らの研究とは文の生成を行わない点でも異なり，Liらの研究とは分散表現の学習の際にトピックを用いていない点でも異なる．

2.7 結言

本章では，COLLAGREEや他の議論プラットフォームでの議論支援研究を紹介し，本研究との違いを説明した，また，本研究と類似した研究や本研究での重要な要素，及び要素を使用した関連研究についても説明し，本研究の立ち位置を明らかにした．