SIG-SWO-043-07

多様なRDFグラフを用いた機械学習のためのカーネル関数

Kernels for Machine Learning from Various RDF Graphs

荒井 大地[1*]　兼岩 憲[1]
Daichi Arai[1]　Ken Kaneiwa[1]

[1] 電気通信大学大学院 情報理工学研究科 情報・ネットワーク工学専攻
[1] Department of Computer and Network Engineering, Graduate School of Informatics and Engineering, The University of Electro-Communications

Abstract: Web 上に大量の RDF データが公開されるにつれて，RDF グラフに対する機械学習が重要になってきている．RDF グラフを分類やクラスタリングに適用するためにカーネル関数が用いられるが，その相対的な性能は RDF グラフの特性や学習問題によって容易に変動してしまう．本研究ではこの問題を解決するために，多様な RDF グラフや学習問題に対して安定した性能を保つ汎用的なスキップカーネルを提案する．まず，RDF グラフ上のリソースに対する特徴として，PRO の要素を変数化して拡張したスキップを定義する．次に各リソースがもつスキップ集合の共通する要素数をリソース集合ごとの再帰により効率的に計算する方法を示す．最後にその共通する要素数を正規化してスキップカーネルを定義する．提案スキップカーネルと，既存のカーネルである PRO，Walk，Path，Full Subtree，Partial Subtree カーネルに SVM を適用して，4 種類の RDF グラフに対する 10 種類の分類実験を行い，スキップカーネルが他に比べて性能が安定して高いことを示す．

1　はじめに

セマンティック Web では RDF（Resource Description Framework）によりリソースの関係性を記述し，機械可読な構造化データを構築する．RDF は主語，述語，目的語の三つ組であるトリプルの集合であり，RDF グラフと呼ばれるグラフ構造として解釈できる．RDF グラフは近年，LOD(Linked Open Data) として Web 上に広く公開されている．これは，Web ページのメタ情報，辞書データ，統計データ，分子構造の管理など幅広い分野で導入されている．また，人工知能における知識表現の 1 つとしても活用されている．

これまで，RDF グラフに対して機械学習を適用し，属性推定などを行う研究が多くなされてきた．特に，RDF グラフを機械学習に適用するために，グラフ間の類似度を求めるカーネル関数が設計される．Lösch らは 2 つの RDF グラフ間の交差グラフや交差木に基づくカーネル関数群（Walk，Path，Full Subtree，Partial Subtree カーネル）を提案した [9]．また，荒井らは RDF グラフの冗長性に着目し，新たな特徴（PRO）と情報利得率によるフィルタリングを組み合わせた PRO カーネルを提案した [11]．

しかし，RDF に対するカーネル関数では，RDF グラフの違いによって相対的な性能が上下してしまう問題が生じる．例えば，木構造に近い単純な RDF グラフに対して PRO カーネルは Walk カーネルに分類性能で大きく勝るが，閉路を含む複雑な RDF グラフではその差が僅かになってしまう．また，複雑さだけでなく，RDF グラフが手動で生成された整ったものなのか，自動で生成されたノイズを多く含むものなのかによってもカーネル関数の相対的な性能は変わる．加えて，同一の RDF グラフであっても，「ある属性をもつか否か」，「いくつかの属性の内どれをもつのか」，「属性値の大小」など様々な分類問題があり，この違いも性能に大きな影響を与える．

本研究では，以上の問題点を解決するために，以下の 3 点を実現する．

- RDF グラフ上のリソースの性質を良く表す特徴（スキップ）の定義
- リソースがもつスキップの一致数計算の効率化
- 多様な RDF グラフに適用する汎用的かつ効率的なカーネル（スキップカーネル）の設計

まず，RDF グラフ上のリソースに対する完全な特徴集合を定義し，そこから有効な特徴を抽出していく．この特徴抽出は，RDF グラフ上のノードからノードへの経路（Walk）とそれを一部変数化したパターン（Walk パターン）により定義される．それにより，Walk パター

*連絡先：
電気通信大学情報理工学研究科情報・ネットワーク工学専攻
〒 182-8585 東京都調布市調布ヶ丘 1-5-1
E-mail: arai@sw.cei.uec.ac.jp

ンのうち目的語定数が末尾にのみ存在できるスキップという構造をリソースの特徴として定める．また，複数のリソースをまとめた集合に対する特徴集合を効率的に抽出する．これと同様に 2 つのリソース集合がもつ共通のスキップを直接探索することで，スキップの一致数を効率よく計算する．さらに，その共通のスキップ数を適切に正規化したスキップカーネルを提案する．このスキップカーネルの汎用性により，様々な場面で，適切なカーネルを選択しなくても良くなる．

本論文の構成は以下の通りである．2 章では，本研究の基礎的な概念である RDF グラフ，クエリとその解決，カーネル関数，既存の RDF グラフカーネルの概要を示す．3 章では，PRO を拡張したスキップという特徴を定義し，リソースがもつスキップ集合を効率的に構築する方法を示す．4 章では，スキップをリソースの特徴としたスキップカーネルを提案する．5 章では，スキップカーネルと既存の RDF グラフカーネルの性能比較実験を行い，スキップカーネルの汎用性を示す．最後に 6 章で結論を述べる．

2 準備

2.1 RDF グラフ

まず，RDF グラフを定義する [7]．ここで U を URI 参照の集合，B を空白ノードの集合，L をリテラルの集合とする．主語 $s \in U \cup B$，述語 $p \in U$，目的語 $o \in U \cup B \cup L$ の 3 つ組 (s, p, o) を RDF トリプルと呼ぶ．主語 s をリソース，述語 p をプロパティ，目的語 o をプロパティ値とも呼ぶ．RDF トリプルの有限集合 $G \subseteq (U \cup B) \times U \times (U \cup B \cup L)$ を RDF グラフと呼ぶ．RDF グラフは，主語と目的語を頂点，述語を主語から目的語への有向辺とした有向グラフである．これによりリソース間の関係性を記述でき，構造化されたデータを形成する．

図 1 に RDF グラフの例を示す．ここでは Java と C# に関するリソースの関係を示しており，例えば「Java はプログラミング言語である」ことや「C#はマイクロソフトが開発している」こと，「Java と C#は互いに影響し合っている」ことなどが分かる．

2.2 クエリとその解決

ここでは [12] に基づき，クエリとその解決について説明する．RDF トリプルの各要素に（?から始まる）変数を許したものをトリプルパターンを呼ぶ．すなわち，変数の集合を V とすると，トリプルパターンは $(U \cup B \cup V) \times (U \cup V) \times (U \cup B \cup L \cup V)$ の要素である．

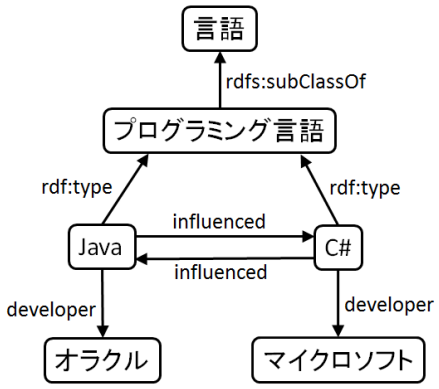

図 1: RDF グラフの例

クエリとは，トリプルパターンの有限列 $q_1, q_2, \ldots, q_n$ であり，$Q = q_1.q_2.\cdots q_n.$ と表される．

変数の集合 V から $U \cup B \cup L$ への部分写像 $\theta : V \to (U \cup B \cup L)$ を代入と呼ぶ．クエリへの代入の適用は，以下のように記述する．

$$\begin{aligned} \theta?x &= \theta(?x) \ \ (?x \in V) \\ \theta e &= e \ \ (e \in U \cup B \cup L) \\ \theta(s,p,o) &= (\theta s, \theta p, \theta o) \\ \theta(q_1.q_2.\cdots q_n.) &= (\theta q_1.\theta q_2.\cdots \theta q_n.) \end{aligned}$$

なお，2 つの代入 θ_1，θ_2 の合成 $\theta_2 \circ \theta_1$ は，$\theta_2 \circ \theta_1(?x) = \theta_2(\theta_1(?x))$ により定義される．

クエリ $Q = q_1.q_2.\cdots q_n.$ に対し，$i = 1, 2, \ldots, n, \theta q_i \in G$ であるような代入 θ を RDF グラフ G における Q の解決と呼ぶ．G における Q の解決集合を以下に定義する．ただし，Sub は全ての代入から成る全体集合とする．

$$\Theta(Q) = \{\theta \in Sub \mid \bigwedge_{i=1,2,\ldots,n} \theta q_i \in G\}$$

2.3 RDF グラフカーネル

カーネルとは対称性と半正定値性をもつ関数である [3]．カーネル関数は非常に柔軟な枠組みであり，分類だけでなく，主成分分析や正準相関分析，クラスタリングなどにも適用できる [1, 4, 10]．構造データに対しては畳み込みカーネル [5] があり，部分構造のカーネルの和として再帰的に定義される [8]．畳み込みカーネルの例には，構文解析木カーネル [2] やそれを拡張したラベル付き順序木に対するカーネル [6] などがある．

RDF グラフ上のリソースに対するカーネル関数は，そのリソースが持つ特徴の一致数により計算できる．既存のカーネル関数として，Walk カーネル，Path カーネ

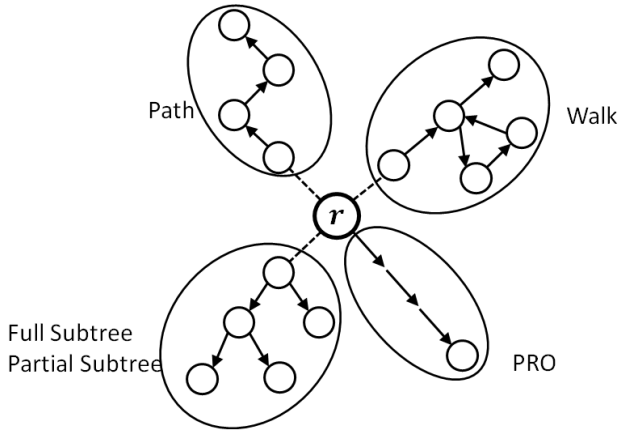

図 2: リソース r に対する既存の特徴

ル，Full Subtree カーネル，Partial Subtree カーネルといった交差グラフや交差木に基づくカーネル関数群 [9]，PRO カーネル [11] などがあり，これらはそれぞれ，Walk，Path，Full Subtree，Partial Subtree，PRO を特徴として，その一致数を計算している．

Walk は RDF グラフ G におけるリソース近傍に存在する経路である．

定義 2.3.1 (Walk) 目的語と主語が一致する d 個のトリプル列 $(o_0, p_1, o_1), (o_1, p_2, o_2), \ldots, (o_{d-1}, p_d, o_d)$ を長さ d の Walk と呼ぶ（統一的な定義のため，始点も目的語として扱う）．これは単に $2d+1$ 個の要素列

$$\mathbf{e} = \langle o_0, p_1, o_1, p_2, o_2, \ldots, p_d, o_d \rangle$$

とも記述する．特にリソース o_0 を始点とする長さ *0* の Walk は $\langle o_0 \rangle$ である．なお，入れ子となった要素列は外側の要素列に再帰的に展開され，連なる要素列は *1* つの要素列に連結される．例えば $\langle e_1, \langle e_2, e_3 \rangle, e_4 \rangle$ と $\langle e_1, e_2 \rangle \langle e_3, e_4 \rangle$ はいずれも $\langle e_1, e_2, e_3, e_4 \rangle$ となる．

Path は Walk から閉路を除いたもの，Full Subtree はリソース近傍の子孫を最大限もつ部分木，Partial Subtree はリソース近傍の単純な部分木である．また，PRO はリソースに接続する述語列と末尾の目的語の組である．Walk を含め，これらの特徴を図 2 に示す．

既存のカーネルの中でも，PRO カーネルは分類性能が特に高いが，木構造に近い単純な RDF グラフではなく閉路を含む複雑な RDF グラフに対してその性能の低下が確認されている．

3 リソースの特徴抽出

本研究では，RDF グラフ上の多様性や複雑性に適応したカーネル関数を設計する．そのために，リソースの本質を良く表す特徴を定義し，それを効率的に抽出する方法を示す．

3.1 リソースの特徴

RDF グラフ G に対し，$G^d(r) \subseteq G$ をリソース r から深さ d で到達可能な RDF トリプルの集合とする．このとき，リソース r の完全な特徴は $G^d(r)$ の全部分構造である．ただし，d は $G^d(r)$ が r を表す上で適当な値とする．ここで，データ点の空間を X，その特徴の空間を F として，あるデータからそれがもつある特徴への写像 $ex : X \to F$ を特徴抽出と呼ぶ．全特徴抽出を EX とすると，リソース r の完全な特徴集合 $F_c^d(r)$ は以下のように表される．

$$F_c^d(r) = \{ex(G^d(r)) \mid ex \in EX\}$$

ただし，$F_c^d(r)$ は非常に膨大な集合であり，本質的にリソース r とは関係の薄い構造も多く抽出される．

リソースと関係の強い特徴を扱うため，対象リソースを始点とする Walk から特徴を抽出する．ここで，RDF グラフ G 上のリソース r を始点とする長さ d 以下の Walk 集合 $Walk^d(r)$ により，リソース r の特徴集合 $F_{walk}^d(r)$ を以下のように表す．

$$F_{walk}^d(r) = \{ex(w) \mid w \in Walk^d(r), ex \in EX\}$$

次に，Walk の任意の要素を互いに異なる変数に置き換えた **Walk パターン**を導入する．まず，要素列 $\langle e_1, e_2, \ldots, e_n \rangle$ の任意の位置と互いに異なる変数を対応づける部分写像 $v : \mathbb{N} \to V$（**変数化関数**と呼ぶ）を $v(i) = ?x_i$ と定める．ここで，変数化関数の全体集合を Var と表す．また，任意の要素列による全体集合を E とし，変数化関数 v を要素 e_i と要素列 $\mathbf{e} = \langle e_1, e_2, \ldots, e_n \rangle$ に対する関数 $v : E \to E$ へ拡張する．

$$\begin{aligned} v(e_i) &= \begin{cases} e_i & (i \notin Dom(v)) \\ v(i) & (i \in Dom(v)) \end{cases} \\ v(\mathbf{e}) &= \langle v(e_1), \ldots, v(e_n) \rangle \end{aligned}$$

以上の変数化関数を用いて，Walk パターンを定める．

定義 3.1.1 (Walk パターン) 任意の Walk w に対し，任意の $v \in Var$ による $v(w)$ を Walk パターンと呼ぶ．

$Dom(v) = \emptyset$ の場合，Walk パターン $v(w)$ は Walk w である．Walk と同じく，$2d+1$ 個の要素列からなる Walk パターンは d 個のトリプルパターン列を記述する．ここで，Walk パターン wp の長さ（トリプルパターンの数）を $|wp|$ と表す．

リソース r の特徴を長さ d 以下の Walk パターンとすると，その特徴集合 $F_{wp}^d(r)$ は以下で表される．

$$F_{wp}^d(r) = \{v(w) \mid w \in Walk^d(r), v \in Var\}$$

$F^d_{wp}(r)$ は r の特徴となる全ての Walk パターンを含むだけでなく，r の無駄な特徴も含む．例えば，r が Walk パターン $\langle \mathbf{e}, o\rangle$ をもち，末尾の o が Walk パターン $\mathbf{e}'$ をもつとする（すなわち，$\langle \mathbf{e}, o\rangle \in F^d_{wp}(r)$ かつ $\mathbf{e}' \in F^d_{wp}(o)$）．このとき，$r$ は自明に $\langle \mathbf{e}, o, \mathbf{e}'\rangle$ という Walk パターンをもつ．したがって，特徴集合に $\langle \mathbf{e}, o\rangle$ が含まれれば，$\langle \mathbf{e}, o, \mathbf{e}'\rangle$ は意味のない冗長な特徴である．これに基づき，Walk パターンに含まれる目的語定数は，特徴の末尾にしか現れない制限を課す．また，末尾の 2 要素が変数の Walk パターン $\mathbf{e}_l = \langle e_1, e_2, \ldots, e_{n-2}, ?x_{n-1}, ?x_n\rangle$ は末尾のトリプルパターンが $(e_{n-2}, ?x_{n-1}, ?x_n)$ となり，これは「e_{n-2} が存在する」ことを意味するだけである．そのため，$\mathbf{e}_s = \langle e_1, e_2, \ldots, e_{n-2}\rangle$ とほぼ同じ特徴であり，$\mathbf{e}_s$ があれば $\mathbf{e}_l$ は不要である．したがって，末尾の 2 要素は少なくとも一方が定数とする．このように制限した Walk パターンを**スキップ**と呼ぶ．

スキップの中で，最も変数の少ない具体的な特徴は，次に定義する PRO である．

定義 3.1.2 (PRO) 任意の Walk w に対し，$Dom(v) = \{1, 3, \ldots, 2|w| - 1\}$ を満たす $v \in Var$ による $v(w)$ を PRO と呼ぶ．

したがって，長さ d 以下の PRO を特徴としたリソース r の特徴集合 $F^d_{pro}(r)$ は以下のように表される．

$$F^d_{pro}(r) = \{v(w) \mid w \in Walk^d(r), v \in Var, Dom(v) = \{1, 3, \ldots, 2|w| - 1\}\}$$

この定義により，PRO の目的語定数は末尾にのみ存在する．したがってスキップは，末尾の 2 要素の少なくとも一方が定数のままであるように PRO を変数化した Walk パターンである．

定義 3.1.3 (スキップ) 任意の PRO pro に対し $\{2|pro|, 2|pro| + 1\} \setminus \{0\} \not\subseteq Dom(v)$ を満たす任意の $v \in Var$ による $v(pro)$ をスキップと呼ぶ（図 3 に例を示す）．

長さ d 以下のスキップを特徴としたリソース r の特徴集合 $F^{var,d}_{skip}(r)$ を以下のように表す．

$$F^{var,d}_{skip}(r) = \{v(pro) \mid pro \in F^d_{pro}(r), v \in Var, \{2|pro|, 2|pro| + 1\} \setminus \{0\} \not\subseteq Dom(v)\}$$

ここで，$F^d_{pro}(r) \subseteq F^{var,d}_{skip}(r)$ であるため，スキップは PRO を拡張した特徴であり，より汎用的に扱えるカーネル関数を設計できる．

3.2 特徴抽出の効率化

前節のように，Walk を変数化して $F^{var,d}_{skip}(r)$ を構築するには，時間計算量が非常に大きくなる．実際，RDF グラフ上の目的語（主語）の種類数を $N = |O|$，述語の種類数を $M = |P|$ とすると以下の定理が成立する．

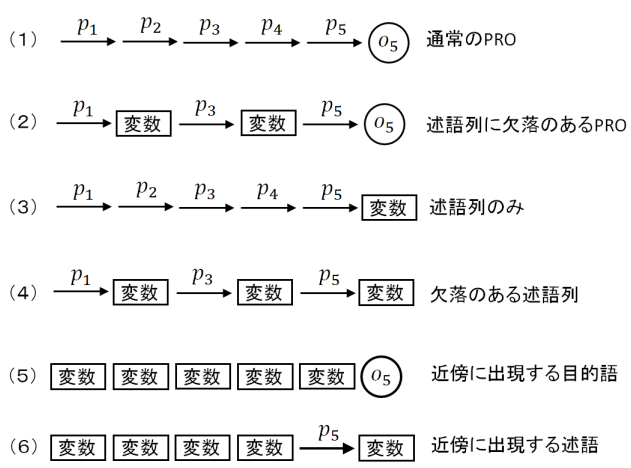

図 3: スキップの例

定理 3.2.1 $F^{var,d}_{skip}(r)$ を構築するための時間計算量は $O(d(2MN)^d)$ である．

一般的に RDF グラフにおいて $M \ll N$ が成立するため，時間計算量の中の N^d が特に高いコストを与える．この問題を解決するために，本節ではまず，全ての要素が変数である Walk パターンに定数を代入してスキップを構成する．さらに，その構成方法を単一のリソースからリソース集合に対する方法へ拡張する．

全ての要素が互いに異なる変数である長さ d の Walk パターンを以下のように定める．

$$wp^d_v = \langle ?x_1, ?x_2, \ldots, ?x_{2d+1}\rangle$$

ここで，$1 \le n \le d$ を満たす wp^n_v の全体集合を WP^d_v とする．また，Walk パターン wp 上の述語の位置および末尾の目的語の位置にとり得る変数集合を $V_{pro}(wp) = \{?x_2, ?x_4, \ldots, ?x_{2|wp|}, ?x_{2|wp|+1}\}$ とする．$V_{pro}(wp)$ を始域とする代入の全体集合を Sub^{wp}_{pro} と表し，リソース r がもつ長さ d 以下のスキップ集合を代入により構築する $F^{sub,d}_{skip}(r)$ を次のように定義する．

$$F^{sub,d}_{skip}(r) = \{\langle r\rangle\} \sqcup \{\theta wp \mid wp \in WP^d_v, \theta \in Sub^{wp}_{pro}, \Theta(\theta wp_r) \neq \emptyset, \{?x_{2|wp|}, ?x_{2|wp|+1}\} \cap Dom(\theta) \neq \emptyset\}$$

ただし，式中の wp_r は Walk パターン wp の先頭の変数 $?x_1$ に r を代入した構造を表す．

この式において，$\Theta(\theta wp_r)$ に着目する．$\theta wp_r = \langle \mathbf{e}, p, o\rangle$ とすると，p と o の組合せの分，同じクエリ $\mathbf{e}$ を何度も解決して，非効率的である．これを改善するために，θwp_r を分解してトリプルパターンごとに再帰的に構成

する $F^{rec,d}_{skip}(r)$ を以下のように定義する．

$$F^{rec,d}_{skip,i}(r) = \{\langle r\rangle, \langle ?x_i, p, ?x_{i+2}\rangle, \langle ?x_i, p, sk\rangle, \langle ?x_i, ?x_{i+1}, sk\rangle \mid (r,p,o) \in G, sk \in F^{rec,d-1}_{skip,i+2}(o)\}$$

ここで，境界条件 $F^{rec,0}_{skip,j}(r) = \{\langle r\rangle\}$ であり，$F^{rec,d}_{skip}(r) = F^{rec,d}_{skip,1}(r)$ とする．この式では，リソース r が $\langle ?x_i, p, o\rangle$ というスキップをもつとき，$\langle ?x_i, p\rangle$ または $\langle ?x_i, ?x_{i+1}\rangle$ に対して o がもつスキップを再帰的に連結している．

$F^{rec,d}_{skip}(r)$ は再帰により無駄なクエリ解決を省く．さらに，$F^{rec,d}_{skip}(r)$ をリソース集合 R がもつスキップ集合 $F^{d}_{skip}(R) = \bigcup_{r\in R} F^{rec,d}_{skip}(r)$ に拡張すれば効率的に構築できる．

$F^{d}_{skip}(R)$ の計算を以下に示す．ただし，リソース集合 R に対し，R 上の任意のリソースを主語とする述語の集合を $P(R) = \{p \mid r \in R, (r,p,o) \in G\}$，述語 p に対する目的語の集合を $O_p(R) = \{o \mid r \in R, (r,p,o) \in G\}$ とする．また，任意の述語 $p \in P(R)$ に対して全ての目的語の集合を $O_*(R) = \bigcup_{p\in P(R)} O_p(R)$ と表す．

$$\begin{aligned} F^{d}_{skip,i}(R) = {} & \{\langle r\rangle \mid r \in R\} \sqcup \\ & \{\langle ?x_i, p, ?x_{i+2}\rangle, \langle ?x_i, p, sk\rangle \mid \\ & \quad p \in P(R), sk \in F^{d-1}_{skip,i+2}(O_p(R))\} \sqcup \\ & \{\langle ?x_i, ?x_{i+1}, sk\rangle \mid sk \in F^{d-1}_{skip,i+2}(O_*(R))\} \end{aligned}$$

ここで，境界条件 $F^{0}_{skip,j}(R) = \{\langle r\rangle \mid r \in R\}$ である．また，$F^{d}_{skip}(R) = F^{d}_{skip,1}(R)$ として計算する．

この拡張は，$F^{d}_{skip}(\{r\})$ が r と隣接する特定の o とは関係なく特徴を構成するために，$F^{rec,d}_{skip}(r)$ における「o がもつスキップ」を「$O_p(R)$ または $O_*(R)$ がもつスキップ」に書き換えている．このようにして，各目的語ごとではなく目的語集合ごとに再帰させて，PRO の計算 [11] と同様に探索空間の広がりを抑える．それにより，以下の定理が成立する．

定理 3.2.2 ハッシュにより $O_p(R)$ および $O_*(R)$ の時間計算量を $O(N^2)$ とすると，$F^{d}_{skip}(R)$ 構築の時間計算量は $O(M^dN^2 + d(2M)^dN)$ である．

以上より，スキップ集合の構築のための時間計算量は，単純な手法の $O(d(2MN)^d)$ から，リソース集合に対して再帰的に定義する手法の $O(M^dN^2 + d(2M)^dN) = O(M^d(N^2 + 2^d dN))$ に効率化される．$d(2N)^d$ が $N^2 + 2^d dN$ となることで非常に大きな高速化に繋がる．

4 汎用カーネルとその効率的計算

本章では，スキップを特徴として様々な RDF グラフや学習問題に対して汎用的なスキップカーネルを提案する．まず単純なカーネル関数として，2 つのリソースがもつスキップ集合の共通する要素数を計算し，それを正規化してスキップカーネルを定義する．

4.1 スキップの一致数の効率的計算

本研究では，$F^{d}_{skip}(\{r\})$ をリソース r の特徴集合としてカーネル関数を設計する．ここで，$F^{d}_{skip}(R,R') = F^{d}_{skip}(R) \cap F^{d}_{skip}(R')$ とする．$F^{d}_{skip}(R,R')$ は $F^{d}_{skip}(R)$ と $F^{d}_{skip}(R')$ をそれぞれ構築し，その後，共通のスキップから計算できる．リソース r，r' に対してスキップによる最も単純なカーネル関数は $|F^{d}_{skip}(\{r\},\{r'\})|$ である．本節では，任意のリソース集合 R，R' に対して $|F^{d}_{skip}(R,R')|$ を効率良く構築する方法を検討する．

ここで長さ d のスキップの種類数を考える．それは長さ d の PRO の種類数に，高々各要素を変数化するか否かの組み合わせ数 2^{d+1} をかけた値である．PRO の種類数は d 個の述語列と 1 つの目的語により M^dN となる．したがって，長さ d のスキップの種類数は $O(M^dN \times 2^{d+1}) = O(2(2M)^dN) = O((2M)^dN)$ である．

長さ d のスキップが $F^{d}_{skip}(R)$ に含まれるか否かを $O(d)$ で判定すると，構築済みの $F^{d}_{skip}(R)$ と $F^{d}_{skip}(R')$ から共通集合を求める時間計算量は $O((2M)^dN \times d) = O(d(2M^d)N)$ となる．ここで $F^{d}_{skip}(R)$ を構築する時間計算量は $O(M^dN^2 + d(2M)^dN)$ なので，全体では $O(d(2M)^dN + M^dN^2 + d(2M)^dN) = O(M^dN(N + 2^{d+1}d)) = O(M^dN(N + 2^d d))$ である．

単純な方法では $F^{d}_{skip}(R)$ と $F^{d}_{skip}(R')$ を構築して保持するため，空間計算量は $O(|F^{d}_{skip}(R)|) = O((2M)^dN)$ となる．これに対して R と R' の共通するスキップ数をカウントするだけならば，$F^{d}_{skip}(R)$ や $F^{d}_{skip}(R')$，スキップの共通集合 $F^{d}_{skip}(R,R')$ を構築せずに済む．以下の方法では，逐次的なカウントが可能なように共通のスキップを直接探索する．

$$\begin{aligned} F^{d}_{skip,i}(R,R') = {} & \{\langle r\rangle \mid r \in R \cap R'\} \sqcup \\ & \{\langle ?x_i, p, ?x_{i+2}\rangle, \langle ?x_i, p, sk\rangle \mid p \in P(R) \cap P(R'), \\ & \qquad sk \in F^{d-1}_{skip,i+2}(O_p(R), O_p(R'))\} \sqcup \\ & \{\langle ?x_i, ?x_2, sk\rangle \mid sk \in F^{d-1}_{skip,i+2}(O_*(R), O_*(R'))\} \end{aligned}$$

ここで，境界条件 $F^{0}_{skip,j}(R,R') = \{\langle r\rangle \mid r \in R \cap R'\}$，$F^{d}_{skip}(R,R') = F^{d}_{skip,1}(R,R')$ とする．この式では，$F^{d-1}_{skip,i+2}(\cdot,\cdot)$ で R の要素 r やその述語 p に続く共通のスキップ sk を再帰的に全て探索していく．以上により，$|F^{d}_{skip}(R,R')|$ を次のように効率的に計算する．

$$\begin{aligned} |F^{d}_{skip}(R,R')| = {} & |R \cap R'| + |P(R) \cap P(R')| \\ & + \sum_{p \in P(R)\cap P(R')} |F^{d-1}_{skip}(O_p(R), O_p(R'))| \\ & + |F^{d-1}_{skip}(O_*(R), O_*(R'))| \end{aligned}$$

ここで，境界条件 $F^0_{skip}(R,R') = |R \cap R'|$ である．第1〜4項はそれぞれ，$F^d_{skip}(R,R')$ の $\langle r \rangle$，$\langle ?x_i, p, ?x_{i+2} \rangle$，$\langle ?x_i, p, sk \rangle$，$\langle ?x_i, ?x_{i+1}, sk \rangle$ に対応する．

このスキップの一致数 $|F^d_{skip}(R,R')|$ の計算は $F^d_{skip}(R)$ と再帰の構造が変わらず，時間計算量は定理 3.2.2 の左項と同じ $O(M^d N^2)$ である．また，空間計算量は，以下のように効率化される．

定理 4.1.1 リソース集合 R，R' がもつ共通するスキップの直接探索による $|F^d_{skip}(R,R')|$ の計算に対する空間計算量は $O(dN)$ である．

以上より，リソース集合 R，R' がもつスキップの一致数の計算における空間計算量は，単純な手法の $O((2M)^d N)$ から，共通のスキップを直接探索する方法の $O(dN)$ に減少する．

4.2 スキップカーネル

本節では，$|F^d_{skip}(R,R')|$ の各再帰ステップにおいてスキップ集合の一致率を正規化して統合し，全体のカーネル値を正規化する．

まず，リソース集合 R，R' がもつ述語集合，目的語集合を探索する．$r \in R$，$r' \in R'$ について，$r = r'$ とすると，r，r' がもつ述語と目的語は同じなので，$R \cap R'$ は再帰的に探索する必要がない．したがって，$|F^d_{skip}(R,R')|$ で探索されている $P(R)$，$P(R')$，$O_p(R)$，$O_p(R')$，$O_*(R)$，$O_*(R')$ の R，R' を，それぞれ，$X = R \setminus R'$，$X' = R' \setminus R$ に置き換える．これを **XOR 制約**と呼ぶ．

XOR 制約により，$|F^d_{skip}(R,R')|$ の再帰の各ステップは $R \cap R'$ の評価と X，X' の評価に分けられる．ここで，集合 S，S' の一致率は Dice 係数 $\frac{|S \cap S'|}{ave(S,S')}$（ただし，$ave(S,S') = \frac{|S|+|S'|}{2}$）による，$ave(S,S')$ を最大一致数としたときの S と S' の一致数の割合とする．$R \cap R'$ は一致率 $\frac{|R \cap R'|}{ave(R,R')}$ を計算し，R，R' の各要素には $\frac{1}{ave(R,R')}$ の重みが与えられる．X と X' の評価は再帰的に述語集合あるいは目的語集合を探索して計算する．カーネル値 $K_x(X,X')$ を X と X' の類似度（0 以上 1 以下）とすれば，X，X' の評価の最大値は $\frac{ave(X,X')}{ave(R,R')}$ だから，$K_x(X,X')\frac{ave(X,X')}{ave(R,R')}$ として評価できる．ここで，$ave(X,X')K_x(X,X')$ は X と X' の最大一致数と類似度の積であり，X と X' の仮想的な一致数である．故に，再帰ステップは $\frac{(R \text{ と } R' \text{の一致数}) + \lambda \cdot (X \text{ と } X' \text{の仮想的な一致数})}{(R \text{ と } R' \text{の最大一致数})}$ の形で計算できる．ここで割引係数 λ（$0 < \lambda \leq 1$）は X と X' の一致数の精度を制御する．

以上の正規化を組み込み，リソース集合 R，R' に対する長さ d 以下のスキップを特徴とした**スキップカーネル** $K^d_{skip}(R,R')$ を以下に定義する．

$$K^d_{skip}(R,R') = \frac{|R \cap R'| + \lambda \cdot ave(X,X')K^d_x(X,X')}{ave(R,R')}$$

ここで，互いに素なリソース集合 X，X' に対する長さ d 以下のスキップを特徴とした類似度 $K^d_x(X,X')$ は以下のように表される．

$$\begin{aligned}
K^0_x(X,X') &= 0 \\
K^d_x(X,X') &= \frac{1}{3}\sum_{i=1}^{3} k^d_i(X,X') \\
k^d_1(X,X') &= \frac{|P(X) \cap P(X')|}{ave(P(X),P(X'))} \\
k^d_2(X,X') & \\
&= \frac{\sum_{p \in P(X) \cap P(X')} K^{d-1}_{skip}(O_p(X),O_p(X'))}{|P(X) \cap P(X')|} \\
k^d_3(X,X') &= K^{d-1}_{skip}(O(X),O(X'))
\end{aligned}$$

以上において，$K^d_x(X,X')$ 内の $k^d_1(X,X')$〜$k^d_3(X,X')$ は次の値を計算する．

$k^d_1(X,X')$: X と X' の述語集合の一致率

$k^d_2(X,X')$: 各述語に対する目的語集合の平均類似度

$k^d_3(X,X')$: 隣接する目的語集合の類似度

これらは，X，X' からの探索方法として，前節で述べた $|F^d_{skip}(R,R')|$ の第 2〜4 項に対応する．それぞれ，述語集合の探索，述語を固定したときの目的語集合の探索，述語を固定しないときの目的語集合の探索を行う．

5 カーネル関数の性能比較実験

本章ではスキップカーネル，既存カーネル（PRO，Walk，Path，Full Subtree，Partial Subtree カーネル），スキップカーネルから性能比較用に PRO の成分を除いたホップカーネル（$K^d_x(X,X') = \frac{k^d_1(X,X') + k^d_3(X,X')}{2}$ となる）の 7 種類のカーネル関数の比較を行う．ホップカーネルは図 3 における（5），（6）の特徴に対応し，スキップカーネルの汎用性の評価に使用する．実験に使用する RDF グラフは，Wikidata[1]，日本語 DBpedia[2]，YAGO[3]，LiveJournal[4] の FOAF データである．ここでは以下に示す 10 種類の実験を行う．

Wikidata リソースに対する

[1] Wikidata. https://www.wikidata.org/

[2] DBpedia Japanese. http://ja.dbpedia.org/

[3] YAGO. https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/

[4] LiveJournal. http://www.livejournal.com/

実験１ 男女の分類

実験２ 海と湖の分類

実験３ 純利益による企業の分類

日本語 DBpedia リソースに対する

実験４ 山と川の分類

実験５ 科学者と芸術家の分類

実験６ 興行収入による映画の分類

YAGO リソースに対する

実験７ 男女の分類

実験８ GDP による国の分類

実験９ 人口密度による場所の分類

LiveJournal の FOAF リソースに対する

実験１０ 年代による人の 4 クラス分類

各分類問題に対して，各カーネルを適用した SVM を用いて分類を行い 10 分割交差検証により正解率を算出する．各カーネルにおいて，対象のリソースからグラフ上を探索するの深さ depth は全て 2 とする．Walk，Path カーネルの特徴の最大長 size は 2 とする．PRO カーネルの情報利得率のボーダー border は 0.0 から 0.5 まで 0.1 刻みで設定する．割引係数 discount について，Partial Subtree カーネルは 0.0001，0.001，0.01，0.1 に設定する．その他のカーネルは 0.1 から 1.0 まで 0.1 刻みで設定する．ただし，実験１０の PRO カーネルは，計算時間が膨大なので割引係数を 0.5 に固定する．

表 1 は各カーネルの実験に対する正解率の最高値を表す．全体的にスキップ，ホップ，PRO カーネルは Walk，Path，Full Subtree，Partial Subtree カーネルよりも正解率が高かった．PRO カーネルの正解率は，実験１，３，８で 1 位，実験 9 では 2 位，実験２，４～７では 3 位であった．しかし，実験１０では Partial Subtree カーネルに負けて 4 位であった．ホップカーネルの正解率は，実験４，５，１０で 1 位，実験２，６，７では 2 位，実験１，３，８，９では 3 位であった．特に実験 3 では Full Subtree カーネルと同率の 3 位であった．スキップカーネルの正解率は，実験２，４，６，７，９で 1 位，その他の実験でも 2 位であった．

表 2 は，表 1 の結果に対する計算時間（秒）を表す．全体的に，単純な設計である Full Subtree，Partial Subtree カーネルの計算時間が短かった．また，ホップ，Walk，Path カーネルはほぼ同等の計算時間であった．PRO カーネルは情報利得率による特徴の削減を行うため，正解率が最高値となるときの border の設定により，計算時間が短い場合と長い場合がある．スキップカーネルは全体的に計算時間が長いが，他のカーネルの高々数倍程度であり，十分現実的な時間で計算できる．

表 3 は，各カーネルの平均順位，順位の標準偏差，最高順位，最低順位を表す．スキップカーネルの平均順位は 1.5 位，順位に対する標準偏差は 0.5270 であり，これは何れも全カーネルの中で最良である．また，最高順位はスキップ，ホップ，PRO カーネルが 1 位で最良である．最低順位はスキップカーネルが 2 位で最良である．よってスキップカーネルはここで示した全ての指標で最良の値を示しており，ぶれなく正解率が高いと言える．したがって，スキップカーネルは汎用的なカーネル関数と考えられる．

計算時間について，スキップカーネルは全体的に他のカーネルよりも計算時間が長いが，定数個のリソースペアに対する時間計算量については PRO カーネルと等しい．実際に実験 4 ではほぼ同じ計算時間であり，実験１０ではスキップカーネルの方が 6 倍ほど短い．したがって，RDF グラフや学習問題が大規模になったとしても，PRO カーネルと同程度のスケーラビリティを保つと考えられる．

6 まとめ

本研究では，RDF グラフにおけるリソースの完全な特徴集合から段階的に制限を与えてスキップを新たに提案した．このスキップは，目的語が末尾にしか存在できない Walk パターンであり，PRO を拡張したリソースの特徴である．また，スキップの特徴集合 $F^d_{skip}(r)$ がクエリ解決の考え方とリソース集合ごとの再帰を用いて効率的に計算できることを示した．このような特徴集合を基に，その共通集合 $F^d_{skip}(R, R')$ を直接探索し，その要素数 $|F^d_{skip}(R, R')|$ を効率的に計算する方法を示した．この計算を正規化して，多様な RDF グラフや分類問題に対し汎用的に扱えるスキップカーネルを設計した．このスキップカーネルと既存のカーネル（PRO，Walk，Path，Full Subtree，Partial Subtree カーネル）に比較用のホップカーネルを加えた 7 種類のカーネルの性能を，4 種類の RDF グラフ，10 種類の実験により比較した．この結果，スキップカーネルは実験全体を通して高い正解率を示した．また，計算時間は他のカーネルに劣るものの，定数個のリソースペアに対する時間計算量は PRO カーネルと等しいことを示した．

参考文献

[1] 赤穂昭太郎. カーネル正準相関分析. 2000 年情報論的学習理論ワークショップ (July 2000), 2000.

[2] Michael Collins and Nigel Duffy. Convolution kernels for natural language. In *Advances in neu-*

表 1: 各実験における各カーネルの正解率

カーネル	実験 1	実験 2	実験 3	実験 4	実験 5	実験 6	実験 7	実験 8	実験 9	実験 1 0
skip	0.88	**0.98**	0.82	**1**	0.99	**0.99**	**0.735**	0.93	**0.935**	0.5817
hop	0.755	0.97	0.81	**1**	**0.995**	0.975	0.71	0.92	0.9	**0.6033**
pro	**0.945**	0.84	**0.83**	0.94	0.97	0.965	0.695	**0.98**	0.93	0.5367
walk	0.625	0.73	0.79	0.83	0.77	0.875	0.605	0.85	0.895	0.5183
path	0.625	0.73	0.8	0.825	0.785	0.865	0.605	0.86	0.895	0.5133
full	0.63	0.77	0.81	0.915	0.865	0.92	0.62	0.83	0.895	0.515
partial	0.595	0.73	0.79	0.92	0.895	0.925	0.605	0.82	0.89	0.5417

表 2: 各実験における各カーネルの計算時間（秒）

カーネル	実験 1	実験 2	実験 3	実験 4	実験 5	実験 6	実験 7	実験 8	実験 9	実験 1 0
skip	80.73	13.85	20.90	4.75	8.09	44.60	5.77	43.76	47.39	2805.76
hop	20.89	4.52	5.02	2.30	4.33	11.04	2.35	12.60	13.49	693.97
pro	12.69	6.57	4.44	4.58	6.05	22.75	1.36	6.76	24.30	17896.84
walk	30.12	5.10	6.97	2.42	4.34	14.65	1.71	19.88	14.85	586.24
path	31.12	5.20	7.02	2.79	4.49	14.98	1.58	22.27	16.05	570.95
full	**3.83**	**0.33**	**0.72**	0.43	0.40	**1.89**	0.32	6.56	5.70	44.66
partial	4.07	0.43	**0.72**	**0.32**	**0.34**	2.24	**0.21**	**6.48**	**5.67**	**44.45**

表 3: 正解率に基づく順位

カーネル	平均順位	標準偏差	最高順位	最低順位
skip	**1.5**	**0.5270**	**1**	**2**
hop	2.1	0.8756	**1**	3
pro	2.4	1.0750	**1**	4
walk	5.4	0.8433	4	7
path	5.5	1.1785	4	7
full	4.6	0.9661	3	6
partial	5.2	1.4757	3	7

ral information processing systems, pp. 625–632, 2001.

[3] 福水健次. カーネル法による因果ネットワークの学習 (非線形科学と統計科学の対話, 研究会報告). 物性研究, Vol. 91, No. 2, pp. 168–171, 2008.

[4] Mark Girolami. Mercer kernel-based clustering in feature space. *Neural Networks, IEEE Transactions on*, Vol. 13, No. 3, pp. 780–784, 2002.

[5] David Haussler. Convolution kernels on discrete structures. Technical report, Citeseer, 1999.

[6] 鹿島久嗣, 坂本比呂志, 小柳光生. 木構造データに対するカーネル関数の設計と解析. 人工知能学会論文誌, Vol. 21, No. 1, 2006.

[7] 兼岩憲. RDF と RDF スキーマの推論. 人工知能学会論文誌, Vol. 26, No. 5, pp. 473–481, 2011.

[8] 申吉浩, 久保山哲二. Haussler 畳み込みカーネルの一般化と応用 マッピングカーネル. 人工知能学会論文誌, Vol. 24, No. 2, pp. 263–271, 2009.

[9] U. Lösch, S. Bloehdorn, and A. Rettinger. Graph kernels for rdf data. In *The Semantic Web: Research and Applications*, pp. 134–148. Springer, 2012.

[10] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, Vol. 10, No. 5, pp. 1299–1319, 1998.

[11] 荒井大地, 兼岩憲. RDF グラフの冗長な特徴表現に対するカーネル関数とその高速計算. 人工知能学会論文誌, Vol. 32, No. 1, pp. B–G34_1, 2017.

[12] 藤原浩司, 兼岩憲. 大規模 rdf グラフのための効率的なクエリ解決. 人工知能学会論文誌, Vol. 29, No. 4, pp. 364–374, 2014.