特別研究報告

題目

ソースコードの特徴語を用いた
Java ソフトウェア部品の自動分類システム

指導教員
井上 克郎 教授

報告者
仁井谷 竜介

平成 17 年 2 月 17 日

大阪大学 基礎工学部 情報科学科

平成 16 年度 特別研究報告

ソースコードの特徴語を用いた
Java ソフトウェア部品の自動分類システム

仁井谷 竜介

内容梗概

近年のソフトウェアの大規模化と複雑化に伴い，ソフトウェア部品の再利用性の向上が叫ばれている．そのための技術の 1 つとしてソフトウェア検索システムがあり，これまでに様々な手法により実現されている．ソフトウェア部品検索に限らず，検索システムへの入力について考えると，その多くは検索者が入力したクエリとの適合度によって検索するものである．このような検索システムの利点は，適切なクエリを入力すれば検索者の意図通りの結果が得られる，という点にある．逆にクエリを入力しない検索システムとして，カテゴリ検索システムがある．これは，あらかじめ用意されたツリー状のカテゴリから文書を探す形式のもので，その特徴は段階的に対象を絞り込める点である．カテゴリ検索のカテゴリは，その性質上，手作業で維持されることが多く，多大な時間と手間を要する．ソフトウェア部品に適用する場合を考えると，部品に対して適切なカテゴリを人間が選択することは困難であり，作業量は膨大になることから，自動化は不可欠といえる．

本研究ではこのカテゴリ検索に着目し，ソースコードのみを入力とする自動分類手法の提案を行う. 特に，ソースコードの特徴語を用いてソフトウェア部品をツリー状のカテゴリに分類する手法を提案する. ソースコードの特徴語とは「そのソースコードを特徴づける重要な語」のことを指し，1 つのカテゴリを形成するものとする．本手法では特徴語の候補として，ソースコード中に出現する識別子，コメント中の単語といった語を用いる．部品ごとに，これらの全ての語に対して重みを求め，重みの大きい上位の語をその部品の特徴語と決定する．そして，カテゴリ（特徴語）間にツリー状の関係を生成し，カテゴリ検索を実現させる．

提案する分類手法をもとにソースコードを分類するシステムおよび，その分類結果を利用した検索システムを構築した．さらに，分類結果が正しく特徴を表しているかの評価を行ったところ，高い適合率が得られた．この結果は，提案手法によりソフトウェア部品の分類が有効に行えることを示しており，効率的なカテゴリ検索の実現が期待できる．

主な用語

自動分類 (Automatic Categorization)
自動分類 (Automatic Classification)
特徴語 (Feature Word)
カテゴリ木 (Category Tree)
ソフトウェア検索 (Software Retrieve)
ソフトウェア部品 (Software Component)

目 次

1　まえがき

ソフトウェアの大規模化と複雑化に伴い，高品質なソフトウェアを一定期間内に効率良く開発することが重要になってきている．これを実現するために，近年のソフトウェア開発において再利用を用いた開発が頻繁に行われている．再利用とは，既存のソフトウェア部品を同一システム内や他のシステムで利用することを指し，開発期間の短縮や品質向上を期待できるといわれている [1, 2, 3, 4]．ソフトウェアの再利用による効果を最大限に引き出すためには，開発者が開発しようとするソフトウェアに必要な部品に関する知識を持つことが重要である．しかし，知識の共有が満足になされていないために，同種のプログラムが別々の場所で，独立して開発されている事も多い．

一方でインターネットの普及により，SourceForge[5] などのソフトウェアに関する情報を交換するコミュニティが誕生し，大量のプログラムソースコードが容易に入手できるようになった．これらの公開されている大量の部品の中から，開発者が必要としている機能を持つ部品，その機能の使い方を示している部品などの，再利用に有益な情報を提供する検索システムを実現する事で，知識の共有が実現でき，再利用を促進する事ができると考えられる．

これまでに，ソフトウェア部品検索を実現する様々な手法が提案，実現されている．それらは，自然言語を対象とする検索システムを単純にソフトウェア部品に適用したものから，ソフトウェア部品を対象とした検索システムならではの機構を備えているなど，自然言語にはないソフトウェア部品の特徴に対して検索を行うものまである．

ソフトウェア部品検索システムに限らず，検索システムへの入力について考えたとき，その多くが検索者に検索語や検索文といったクエリを入力させ，そのクエリとの適合度の高いものを出力する．クエリを入力させる検索システムの利点は，適切なクエリを入力すれば検索者の意図通りの結果が得られる，という点にある．

クエリを入力しない検索システムとして，カテゴリ検索システムがある．これは，あらかじめ用意されたカテゴリから文書を探す，という検索システムで，その特徴はツリー状に構成されたカテゴリを検索者が順にたどっていくことにより，段階的に対象を絞り込んでいける点である．また，絞り込みの過程で意図しない発見をすることがある．それは，検索者が探そうとしていたものに関連するが，検索者がその存在を知らなかったカテゴリを発見したときなどに起こる．

本研究ではこのカテゴリ検索に着目し，ソフトウェア部品に適用した場合について考える．カテゴリ検索のカテゴリはその性質から，手作業で維持されることが多くそれには多大な時間と手間がかかる．特に，ソフトウェア部品を対象としたときは，追加あるいは更新しようとするソフトウェア部品がどのカテゴリに属するのが適切であるか，を判断するのが非常に困難である．また，対象数が多いため，手作業でそれを判断するには多くの専門知識を

持った人と膨大な時間を要する．さらに，今までにない機能を持つ部品の追加や，カテゴリ内の部品数の増加などが起こった場合に，新しいカテゴリの追加や，巨大なカテゴリの分割といった，カテゴリの再構成・維持作業も必要となる．従って，ソフトウェア部品を対象としたカテゴリ検索には，カテゴリへの分類とカテゴリの維持の自動化は不可欠だといえる．

ソフトウェア部品の性質が記述してあるものとして，ソースコードとドキュメントが存在する．ソフトウェア部品を分類するに当たって，分類のための情報は多い程よいが，プログラム言語の使用に従って記述されているソースコードと異なり，ドキュメントはその記述に規定がなく，どのソフトウェア部品についての記述であるかの対応付けも困難である．従って，分類のための情報はソースコードのみを入力とすることが望ましい．ソースコードは通常は予約語，記号，識別子，コメントといった要素の集合で成り立っている．それを自然言語における文書と単語と考えれば，文書を単語の出現回数のベクトルとみなす自然言語のベクトル空間モデルを用いた分類手法を適用することができる．なお，ソースコードに自然言語を対象としたモデル化を適用した研究により [6]，その有効性は実証されている．

自然言語を対象とした分類，検索，データマイニング関連で使われる言葉に「特徴語」というものがある．研究ごとにその定義は異なるが,「文書の特徴を表す語」という共通の意味で使われている．単に「文書中の出現する重要な語」を指す場合と,「その文書中に出現したかに関わらずその文書を特徴づける重要な語」として「文書中に出現する重要な語」と区別して使う場合の 2 つに大きく分けられる．前者は検索対象の語の数を削減するためなど，要約の一種として検索者に提示するために用いられる．後者は前者と同様の目的以外に「文書中に出現しない重要な語の付加」のために用いられる．

本研究では Java ソフトウェア部品の特徴語を,「そのソースコード中に出現したかに関わらずその部品を特徴づける語」として定義する．各部品に対して特徴語を決定し，得られた特徴語を 1 つのカテゴリとみなして各部品を分類する手法を提案する．本手法では分類の入力として，ソースコード中に出現する識別子，コメント中の単語を用いる．部品ごとに，これらの全ての語に対して重みを求め，重みの大きい上位の語をその部品の特徴語と決定する．しかし，このままではカテゴリ同士は互いに無関係で，検索者は得られた分類を検索に利用しようとしても，無数にあるカテゴリから，自分の求めるカテゴリを探さなければいけない，という問題が発生する．このことを解決するために，ツリー状のカテゴリ間の関係および，類似したカテゴリをまとめる関係を生成する．

本手法の分類結果を用いることにより，ソフトウェア部品に対するカテゴリの作成が容易に行える．さらに，提案する分類手法を実現するシステムおよび，検索システムを構築し分類結果の評価を行う．その結果，提案手法を用いたソースコードのみによる分類の有効性を評価する．

以下，2 節で既存のソフトウェア部品検索システムについて述べる．3 節でソースコード

の特徴語を用いてソフトウェア部品を分類する手法について述べる．4 節で提案手法を実現したシステムについて述べる．5 節で，そのシステムの評価実験について述べる．最後に 6 節でまとめと今後の課題について述べる．

2　背景

本節ではソフトウェア部品検索システムについて説明した後，クエリ入力検索とカテゴリ検索による部品検索の概要と，既存の研究について述べる．

2.1　ソフトウェア部品検索システム

部品の検索とは，分類された部品集合から必要な部品を探し出す作業のことである．自然言語文書の検索システムでは検索しようとするトピックのキーワードをクエリに用いて検索するのが一般的であるが，ソフトウェア部品検索システムではその他に部品の特徴をあらわすメトリクスや，記述されている言語，求める機能の抽象表現などをクエリに用いる場合もある．

2.1.1　クエリ入力検索による部品検索

クエリ入力検索システムは，検索者に検索語や検索文といったクエリを入力させ，そのクエリとの適合度の高いものを検索するシステムである．

クエリとの適合度を求める方法は，単純に出現をしたかどうかや，類似する語が出現しているか，検索語の集合と部品が類似するか，などといった様々なものがある．

また，検索結果の順位付けにも様々な手法がある．ソースコード中のどこに適合したかにより重みをつけたり，部品間の利用関係を利用して部品自体に重み付けをするなどといった手法で，有益なものが検索結果の上位にくるようになっている．

クエリを入力させる検索システムの利点は，適切なクエリを入力すれば検索者の意図通りの結果が得られる，という点にある．しかし逆に，適切なクエリを入力しないと，意図したものが検索されなかったり，不要なものの中に意図したものが埋もれてしまう，という問題点がある．

2.1.2　カテゴリ検索による部品検索

クエリを入力しない検索システムとして，ウェブサイト検索などでよく見られるカテゴリ検索がある．カテゴリ検索の利点は，ツリー上のカテゴリに従って段階的に絞り込めることと，文書中に出現する単語に頼らずに検索ができることである．

また，絞り込みの過程で，意図しない発見をすることがある．それは，検索者が探そうとしていたものに関連するが，検索者はその存在を知らないカテゴリを発見したときなどに起こる．検索時の意図には適合しないが，このようなものは大局的には有用である．

カテゴリ検索のカテゴリは内容を把握する必要があることから，手作業で維持されることが多い．その多くは，その検索システムの管理者が与えたカテゴリに従って，管理者自身あるいは登録者がふさわしいカテゴリに分類する，という作業が必要となる．しかしそれには多大な時間と手間がかかる．特にソフトウェア部品を対象にすると，このコストは大きくなることが考えられる．また，管理者が与えた分類の外，あるいは中間といったものは無理矢理既存の分類に当てはめるか，カテゴリの再構成をして分類しなおす必要がある．

2.2 関連研究

我々の研究チームでは，Java のソースコードを対象としたソフトウェア部品検索システム SPARS-J(Software Product Archive，analysis and Retrieval System for Java)[7, 8] の構築を行なっている．SPARS-J は，依存や類似といったソフトウェア部品特有の特性を考慮しながら，大規模なライブラリの構築を行う．検索語とその出現箇所を検索クエリとした全文検索を行い，部品に対する検索語の重みを出現箇所と TF-IDF 法 [9] から求める KR 法 (Keyword Rank 法) と，部品の重みをその利用実績から求める CR 法 (Component Rank 法) の 2 つの手法によって検索結果を順位付けている．これらの順位付けにより，効率の良い全文検索を実現している．

ソフトウェア部品ではなくソフトウェアを対象とした分類手法として，川口の LSA に基づく手法 [6] が挙げられる．これはソフトウェアのソースコードに出現する識別子に対して LSA[10] を適用し，その結果を用いて識別子によるソフトウェアの非排他的クラスタリングを行っている．これにより，前提知識を用いずにソフトウェアの集合を機能やライブラリやアーキテクチャといった，複数の視点による自動分類を実現している．この手法は，ソフトウェアを複数の視点で分類することに重点が置かれているので，カテゴリ間に階層構造を与えることを前提にカテゴリを構成する本稿の手法とは，カテゴリ（この手法ではクラスタ）の持つ意味が異なる．

ソフトウェアシステム間の関連を対象にした研究として，山本らの SMMT[11] が挙げられる．SMMT は二つのソフトウェアシステムの類似度を計測する．SMMT ではコードクローン検出技法 [12] に基づいてソースコード行単位での類似度を測定する．山本らは SMMT を用いて 4.4BSDLite から派生したソフトウェアである FreeBSD，NetBSD，OpenBSD の各バージョン間の類似度を計測し，これらのソフトウェアを適切に分類できることを示している．しかし，SMMT での類似度の定義は二つのソフトウェア全ての行のうち，コードクローンを含む行，および全く同一の行が存在する割合としている．そのため，同じソースから派生したようなソフトウェア間の類似度は適正な値が出力されるが，全く出自が異なるソフトウェア間では類似度の値が極端に低くなる．

鷲崎らの JComponentSearch[13] は分類システムではないが，依存関係に基づく分類システムという見方もできる．これは，独立して再利用可能な部分を 1 つのコンポーネントとしてまとめ，そのまま再利用できる単位でソースコードを検索することを目的する検索システムである．これにより，従来手法では必要だった「得られたソースコードを利用するために依存関係のあるソースコードを検索者が取得しなければいけない」という作業コストを削減している．

ソフトウェアを分類してライブラリとして再利用を支援する研究も行われている．Börstler は，コンポーネントの性質や属性からソフトウェアの特徴を抽出し，それに基づいて分類分けを行う手法 FOCS（Feature-Oriented Classification System）[14] を提案した．コンポーネントはいくつかの特徴から構成され，FOCS はそれらの特徴を用いて検索や類似度の測定を行う．

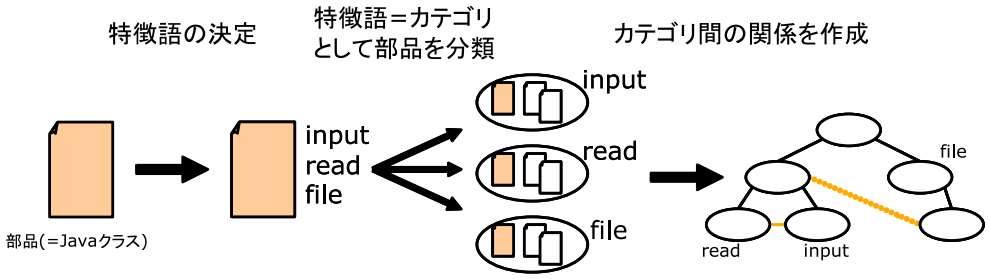
Karlsson らは，重み付けした単語空間や，語間のファジーな関係をもとにキーワードに基づく分類分けの概念 [15] を提案している．

ほかに，ある単一のソフトウェアを何らかの機能的単位へ分割する研究として，LSA を用いた手法 [16]，Self-Organizing-Map を利用する手法 [17]，ファイル構成やファイル名に基づく分類 [18]，コールグラフなどプログラムの構造を使って分割を行う手法 [19, 20, 21] が提案されている．

3　提案手法

本研究では，ソフトウェア部品のカテゴリ検索をするための，部品の分類を自動で行い，得られたカテゴリ間に関係を作成する．

図 1: 提案手法の概要



提案する手法の概要を図 1 に示す．まず，ソースコードを入力として部品ごとに特徴語を決定し，次に，得られた特徴語を 1 つのカテゴリと見なして各部品を分類する．

この段階では，カテゴリは得られた特徴語の数だけ無関係に並んでいるだけなので，できあがった分類を検索に利用しようとしても，無数にあるカテゴリから，自分の求めるカテゴリを探さなければいけない，という問題が発生する．それを解決するために，ツリー状のカテゴリ間の関係および，類似したものをまとめる関係を生成する．

本手法は，ソースコードのみを入力とする，語に対する前提知識などを与えない教師なしの手法である．したがって，ここで各部品に対して重みを決定する語の集合は，全部品のいずれかに出現する語全てである．

教師なしの手法を用いているのは，分類対象をドメインとしたときの語の意味が，辞書上での語の意味と異なることにより，意図しない分類がされることを防ぐためである．また，分類対象に合わせた辞書を作っていては，その作成と維持に手間がかかり，分類の工程が自動化できなくなるためである．

3.1　用語

すでに本稿では何度か使用された語もあるが，改めてここでその本稿における定義，および一般的な説明を記す．

部品・ソフトウェア部品　一般にソフトウェア部品は再利用できるように設計されたソフトウェアの実体とされる [22].

本研究では，Java のクラスを部品として扱う．

特徴語 一般には，自然言語を対象とした分類，検索，データマイニング関連で使われる言葉．研究ごとにその定義は異なるが,「文書の特徴を表す語」という共通の意味で使われている．単に「文書中の出現する重要な語」を指す場合と,「その文書中に出現したかに関わらずその文書を特徴づける重要な語」として「文書中に出現する重要な語」と区別して使う場合の 2 つに大きく分けられる．前者は検索対象の語の数を削減するためや，要約の一種として検索者に見せるために用いられる．後者は前者と同様の目的以外に「文書中に出現しない重要な語の付加」のために用いられる．

本研究では，後者の意味をソースコードに導入した形で用い，その語がソースコードに出現したかに関わらず部品を特徴づける語，と定義する．特徴語の集合は，全部品のソースコードのいずれかに出現する語全ての集合に含まれるものとする．

また，キーワード (keyword) という言葉が，Java の予約語を表すのでそれと明確に区別するためにこの特徴語という表現を用いている．

カテゴリ 本研究では，分類の結果できた部品の集合をカテゴリと定義する．

ある特徴語を持つか持たないかでカテゴリを形成する.「A というカテゴリ」と「A を特徴語として持つカテゴリ」は同値である．

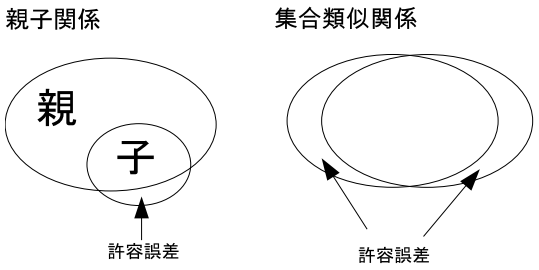
また，部品に対して複数個の特徴語が決定されるので，カテゴリの中の部品は非排他的に分類される．

カテゴリ間の関係 本研究で定義するカテゴリ間の 3 つの関係（親子関係，集合類似関係，類似関係）．

親子関係 カテゴリを部品の集合としてみたときに，ある程度の誤差を許容して集合の包含関係が成り立つとき，その包含する方を親，包含される方を子と定義する．

表 2（左）．

図 2: 親子関係・集合類似関係



集合類似関係 カテゴリを部品の集合としてみたときに，ある程度の誤差を許容して集合の同値関係が成り立つとき，その間に集合類似関係があると定義する．

表 2（右）．

類似関係 親子関係も集合類似関係もないが，そのカテゴリの特徴語同士に高い類似度が見られるものにこの関係があると定義する．

カテゴリ木 カテゴリ間の親子関係に基づく，カテゴリをノードとするツリー．ただし，子の重複を認める．

複合語 1 つ，あるいは複数の語が繋がってできている語．例えば，XMLParser は「XML」と「Parser」の 2 つからなる複合語．

単一語 複合語の最小単位．上の例でいう「XML」と「Parser」．

部分語 複合語を単一語単位で区切ったときの一部．あるいは元の複合語自身．

例えば，ExtensibleMarkupLanguage の部分語は，Extensible，Markup，Language，ExtensibleMarkup，MarkupLanguage，ExtensibleMarkupLanguage．

3.2 特徴語決定

特徴語を決定するために，まず各部品ごとに語に対する重み（単語重み）を求める（図 3）．得られた単語重みの高い語を部品の特徴語とする．ここでは上位 10 語を特徴語とする．

図 3: 単語重み計算の流れ

以降の小々節は，図 3 の各四角に対応する．

3.2.1 語の出現重み

$$WordWeight_{cw} = \sum_{k \in Kind} KindWeight_k \times \log(1 + Count_{cwk})$$

$WordWeight_{cw}$**:** 部品 c に対する語 w の出現重み．

$Kind$**:** 語の種類（表 1）．

$KindWeight_k$: 語の種類に対応する重み.

$Count_{cwk}$: 部品 c における語 w の k として出現した回数.

出現した語の種類を考慮して重みを与える.「クラス定義に出てくるクラス名は，そのクラスが利用するクラス名や，中で利用する変数名などといったものより，そのクラスに対する重要度が高い」という考えの下に重み付けがなされている.

表 1: 語の種類

クラス定義名	メソッド定義名
インタフェース名	パッケージ名
インポートパッケージ名	呼出しメソッド名
参照フィールド変数名	生成したクラス名
変数の型名	参照する変数名
コメント (/*...*/)	文書コメント (/**...*/)
行末コメント (//...)	文字列リテラル

回数の対数をとっているのは，同じ変数名が何度も使われたときに，クラス名の語の重みの数倍の重みを得てしまう様なことを防ぐためである．確かに，何度も出現する語はその部品に対する重要度が高いと考えられるが，重要度が高くなくても何度も出現する場合がある．例えば，コーディングスタイルやアルゴリズム，ループ展開のような手法が使われているか否かによって，同じ変数名が何度も出現しうる．この場合単純に回数に比例した重みをとると，何度も出現した語の重みが本来一番重要とされるべき語の数倍以上の重みを持ち，最終的な分類結果にも悪影響を与える.

3.2.2 語の記法統一と複合語による重み再計算

$$WordWeight'_{cw'} = \sum_{w \in W} DivisionWeight_{ww'} \times WordWeight_{c'w}$$

$WordWeight'_{cw'}$: 再計算後の部品 c に対する部分語 w' の重み.

W: 処理前の全ての語の集合.

$DivisionWeight_{ww'}$: 語 w に対する部分語 w' の関係の重み.

ここでは，コーディングスタイルなどの違いによる語の表記の違いの統一と，部分語に対する重みの配分計算を同時に扱う．

ソースコード中に出てくる語は自然言語中の語と異なり，そのほとんどがいくつかの語を並べて書いた複合語である．従って単純に出現した語だけを追っていると，その部品が何を扱っているかを見つけにくくなる．例えば，java.util.Vector には elementAt()，setElementAt()，removeElementAt() というメソッドがあるが，複合語であることを考慮しなければ，java.util.Vector の elementAt に対する重みは elementAt() の持つ重みだけになってしまう．

この点を考慮して，setElementAt() と removeElementAt() の分も，elementAt に対する重みを加算することとする．ただし，setElementAt は elementAt と同じ語ではないので，複合語を構成する単一語の個数を考慮した以下の式で元の語から部分語に対する重みを与える．

$$DivisionWeight_{ww'} = \left(\frac{count(w')}{count(w)}\right)^2$$

$count(w)$**:** 語あるいは部分語 w を構成する単一語の数

また，elementAt と element_at では表記が異なるためそのままでは同じ語とみなすことが出来ない．よって，elementAt や element_at といった表記の違う語を Element と At の複合語として同じものとみなし，表記の違いを統一する（大文字小文字の違いは無視する）．

3.2.3 対象語の削減

計算量の軽減のために，分類に影響のない語を削除する．1 つの部品にしか出現しない語は，以降の手法にとって有益な情報ではないので削除対象としている．これにより語数を半数近くに減らせることが期待できる．

3.2.4 利用関係による重み加算

$$WordWeight'_{cw} = WordWeight_{cw} + \sum_{c' \in C, c' \neq c} RelationWeight_{cc'} \times WordWeight_{c'w}$$

$WordWeight'cw$**:** 利用関係による重み加算後の部品 c に対する語 w の重み.

C**:** 全部品の集合.

$RelationWeight_{cc'}$**:** c と c’ の関係の重み.

部品間の利用関係を語の重みに反映する．Java のようなオブジェクト指向言語では，親クラスに実装のほとんどが書いてあり，子クラスにはその差分だけが書いてある，という場合

がよく見られるので，対象のクラスの語を見るだけに比べ，親クラスの語にもある程度の重みを与えて加えた方が，有益な特徴語の決定に結びつく．同様の理由から，メンバ変数として持つクラスなどといった関係のあるクラスの語にも重みを与えている．

なお，部品間の利用関係は SPARS-J を利用して取得しているので，表 2 に従ったものとなっている．

表 2: 利用関係の種類

種類	概要
継承	子クラスが親クラスを利用
抽象クラス実装	実装クラスが抽象クラスを利用
インタフェース実装	実装クラスがインタフェースを利用
変数宣言の型	変数宣言したクラスが変数型を定義するクラスを利用
インスタンス生成	インスタンス化したクラスが生成したクラスを利用
フィールド参照	参照元クラスがフィールドを定義するクラスを利用
メソッド呼出し	呼出し元クラスがメソッドを定義するクラスを利用

3.2.5 LSA

LSA は類似度を求めるための統計的手法で，この LSA を用いることにより，語同士の類似度を求めるためのベクトルを計算する．通常 LSA は「(表 3 の foo と bar の類似度) = (foo と bar の $\cos\theta$)」の様に foo と bar をベクトルとして用いるが，本手法では,「(foo に対する A の重み) = 1.23」の様に，ベクトルの要素を重みとして利用する．

表 3: LSA で得られたベクトル

語＼文書	A	B	C	D
foo	1.23	3.56	5.67	7.89
bar	-4.12	15.2	3.89	0.23

LSA の詳細は付録に記述する．

3.3 カテゴリ間の関係生成

特徴語をカテゴリ，その特徴語を持つ部品をそのカテゴリの要素とみなす．この小節では，カテゴリに親子関係，集合類似関係，類似関係を与える手法について述べる．

3.3.1 包含関係によるカテゴリ木・関係生成

カテゴリは部品の集合なので，それぞれに包含関係が存在する．従って，包含するものを親，包含されるものを子とすると,，ツリー状の関係（ただし子は複数の親を持つことがある）を生成することができる．

また，同じ集合をなすカテゴリを同値の関係とすることができる．

本研究では，包含関係が完全に成り立っていなくても，ある程度の誤差を許容して上記の親子関係と同値関係が成り立ったものとみなして，それぞれ親子関係，集合類似関係を生成する．

3.3.2 特徴語間の類似度による関係生成

特徴語（カテゴリ）同士の類似度は，特徴語決定に用いた行列から求めることができる．その類似度を用いて，類似度が一定以上のカテゴリ同士に類似関係を生成する．

類似度はコサイン尺度で求める．コサイン尺度は以下の式で表される．

$$Similarity_{ab} = \cos\theta = \frac{\vec{a}\cdot\vec{b}}{|\vec{a}||\vec{b}|} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}}$$

$Similarity_{ab}$**:** 特徴語 a，b 間の類似度．

$\vec{a}$**:** 特徴語決定に用いた行列における a の行（あるいは列）．

$\vec{b}$**:** 特徴語決定に用いた行列における b の行（あるいは列）．

4　自動分類システム

本研究で開発したシステムは大きく分けてソースコードの解析部，部品ごとの語に対する重み決定部，特徴語決定部，カテゴリ間の関係決定部，検索部の 5 つに分けられる．

4.1　ソースコード解析部 (SPARS-J)

本研究では Java ソースコードの解析結果を SPARS-J のデータベースから取得している．クラス名やメソッド名といった語の抽出や，各部品間の利用関係を利用する．

説明中の sourcereader，wordvector，bmat は本手法で用いるデータ形式であり 4.6 節で説明する．

4.1.1　SPARS-J から出現単語の情報取得

入力 SPARS-J の source，word の DB

出力 出現単語の情報 (sourcereader)，語・ID テーブル (wordvector)

SPARS-J の source と word の DB を用いることにより，ソースコード中の単語とその単語の種類が得られる．しかし，同じ語の文字列でも出現箇所が異なると異なる ID が振られる．また，部分語も登録されている．従って，本研究の手法には適さないため，部品を形式を変換して以降の手順で利用する．

部品を以下のもの集合とみなす．出現した順序は問わない．

- 出現語 ID（ID と語は全単射・部分語は含まない）
- 出現箇所（語の種類）

4.2　部品ごとの語に対する重み決定部

4.2.1　語の重み

入力 出現単語の情報 (sourcereader)

出力 部品×語 重み行列 (bmat)

部品×語 の重みの行列を生成する．

4.2.2 利用関係による重み加算

入力 SPARS-J の fromrelation の DB，部品×語 重み行列 (bmat)

出力 部品×語 重み行列 (bmat)

SPARS-J の fromrelation の DB から，部品から部品へどのような利用関係があるかを取得できる．

部品同士の関係の重み行列を生成し，入力の重み行列との積を求める．これにより部品 A 上の語は，A と関係のあるすべての部品の語の重みを，その関係の重みに応じただけ加算される．

4.2.3 語の記法統一と複合語による重み再計算

入力 部品×語 重み行列 (bmat)，語・ID テーブル (wordvector)

出力 部品×語 重み行列 (bmat)，語・ID テーブル (wordvector)

語×部分語の重み行列を生成し，入力重み行列との積を求める．これにより，部品×出現単語で扱っていた重み行列が，部品×部分語の重み行列へ変換される．

例えばある部品の出現単語の重みが，WordList 1，WordVector 5 だったとき，部分語の重みは，WordList 1，WordVector 5，Word 1.5，List 0.25，Vector 1.25 となる．

語の ID は部分語の ID へと置き換えられる．

4.2.4 対象語の削減

入力 部品×語 重み行列 (bmat)，語・ID テーブル (wordvector)

出力 部品×語 重み行列 (bmat)，語・ID テーブル (wordvector)

一定値以上の重みを持つ部品が 1 つしかない語を削除する．語の ID は削除された分だけ詰める．

4.2.5 LSA

入力 部品×語 重み行列 (bmat)

出力 部品×語 重み行列 (bmat)

LSA の行列計算を川口の研究に用いられたツールを使用して行う．このツールは内部で SVDPACKC[23] を利用している．

4.3 特徴語決定部

各部品ごとに重み順で語をソートし，上位 10 語をその部品の特徴語とした．

4.4 カテゴリ間の関係生成部

4.4.1 包含関係を用いた関係生成

2 つのカテゴリ A，B があるとして，以下の条件判別により関係を生成する．

1. $|A \cap B| = |A|$ かつ $|A \cap B| = |B| \Rightarrow$ 集合類似関係
2. $|A \cap B| < |A|$ かつ $|A \cap B| = |B| \Rightarrow$ 親子関係（A が親）
3. $|A \cap B| = |A|$ かつ $|A \cap B| < |B| \Rightarrow$ 親子関係（B が親）
4. $|A \cap B| \div |A| \geq 0.8$ かつ $|A \cap B| \div |B| \geq 0.8 \Rightarrow$ 集合類似関係
5. $|B| < |A|$ かつ $|A \cap B| \div |B| > 0.8 \Rightarrow$ 親子関係（A が親）
6. $|A| < |B|$ かつ $|A \cap B| \div |A| > 0.8 \Rightarrow$ 親子関係（B が親）
7. 関係なし

4.4.2 特徴語間の類似度による関係生成

$WordWeight_{cw}^{T} \times WordWeight_{cw}$ で特徴語同士の類似度行列が求められる．

4.5 検索部

分類結果を評価するための検索システム．カテゴリ，部品の検索が行える．

4.5.1 トップページ

検索画面がある．カテゴリ数と部品（クラス）数が表示される．

4.5.2 クラス名・カテゴリ名検索

完全修飾クラス名，カテゴリ名（特徴語）に対して，部分一致の AND 検索を行う．検索結果は名前の短いものが上位に来る．

4.5.3　部品に関する情報表示

指定された部品（クラス）の属するカテゴリの一覧，そのカテゴリに対する重み，SPARS-J におけるその部品の画面を表示する．

4.5.4　カテゴリに関する情報表示

指定されたカテゴリに属する部品一覧，その部品に対する重み，そのカテゴリと関係のあるカテゴリの一覧を表示する．

4.5.5　カテゴリ木表示

全てのカテゴリの親子関係をツリー状に表示する．

4.6　ファイル形式

4.6.1　bmat 形式

行列を表す．拡張子は .bmat
bmat 形式の書式は以下の通り．

行数 size_t

列数 size_t

行列の値 double … （1 行 1 列，1 行 2 列，…，m 行 n 列の値）

バイナリをテキスト表示した例:

```
5 9
1 0 0 1 0 0 0 0 0
1 0 1 0 0 0 0 0 0
1 1 0 0 0 0 0 0 0
0 1 1 0 1 0 0 0 0
0 1 1 2 0 0 0 0 0
```

4.6.2　wordvector 形式

文字列の配列を表す．拡張子は .wv
wordvector 形式の書式は以下の通り．

語数 size_t

語 string, ...

語の長さ size_t

語 char, ...

バイナリをテキスト表示した例:

```
5
3 foo
3 bar
3 baz
3 qux
4 quux
```

4.6.3 sourcereader 形式

SPARS-J のデータベースから得た出現単語に関わる情報を，手法に適した形式に変換したもの.

ID →重複しない語 (size_t, string), ...

ソース source, ...

ソースの語数 size_t

出現した単語 (size_t, int)(= (語の ID, 単語の種類)), ...

4.7 リレーショナルデータベース設計

分類した結果は PostgreSQL に格納する．本小節では，その構造について記す.

- 表 4: カテゴリ名テーブル
- 表 5: 部品名テーブル
- 表 6: カテゴリの部品一覧テーブル
- 表 7: カテゴリ間の関係テーブル

表 4: category_name

項目	型	説明
category_id	int primary key	カテゴリの ID
name	text	カテゴリ名

表 5: component_name

項目	型	説明
component_id	int primary key	部品の ID
name	text	部品名

表 6: category_component

項目	型	説明
category_id	int	カテゴリの ID
component_id	int	部品の ID
weight	real	部品に対するカテゴリの重み

表 7: category_relation

項目	型	説明
from_category	int	カテゴリの ID
to_category	int	カテゴリの ID
relation_type	int	関係の種類
weight	real	カテゴリ同士の関係重み

5　実験

提案手法を用いて実際に部品の分類を行った. また，その分類結果を検証するために検索システムを構築し分類結果の評価を行った.

5.1　実験方法

実際に分類を行い，どれだけ適正にカテゴリが作られているかを評価する．また，ソースコード中にはない特徴語がどれだけ特徴を表しているかを評価する.

評価はカテゴリに属する部品の適合率，部品が属するカテゴリの適合率，ソースコード中にはない特徴語の適合率の 3 つの値によって行う.

適合率 (precision) とは以下の式で与えられる値で，検索結果の中でどれだけ適合するものがあるかを表す．1 に近いほど良い結果が出たといえる.

$$\frac{|(\text{検索結果}) \cap (\text{適合部品})|}{|(\text{検索結果})|}$$

5.1.1　評価する適合率

カテゴリに属する部品の適合率

$$\frac{|(\text{カテゴリに属する部品}) \cap (\text{カテゴリに適合する部品})|}{|(\text{カテゴリに属する部品})|}$$

部品が属するカテゴリの適合率

$$\frac{|(\text{部品が属するカテゴリ}) \cap (\text{部品に適合するカテゴリ})|}{|(\text{部品が属するカテゴリ})|}$$

ソースコード中にはない特徴語の適合率

$$\frac{|(\text{ソースコード中に } F \text{ がない部品}) \cap (\text{カテゴリ } F \text{ に属する部品}) \cap (F \text{ に適合する部品})|}{|(\text{ソースコード中に } F \text{ がない部品}) \cap (\text{カテゴリ } F \text{ に属する部品})|}$$

F は特徴語

5.1.2　カテゴリ・部品間の適合の条件

カテゴリに部品が適合する条件とは，カテゴリに対応する特徴語が部品のソースコードもしくは，親クラスのソースコードの特徴を表すことである.

ただし，以下のようなものは適合としない.

- そのカテゴリに含まれる部品の集合が同様のものであるが，そのうちのどれかを表すだけの特徴語がそのカテゴリに対応している
  例：Read というカテゴリ中に，FileReader と FileWriter が入っている

- ソースコード中に現れるがあまり特徴を表さない
  例：ループ制御変数 i

- 自身の特徴を表さないような子クラスの特徴語
  例：FileWriter の子クラスが FileURLWriter で FileWriter の特徴語に URL が入っている

5.1.3 実験対象

ロボットシステム（Robocode のロボット）の部品 245 クラス（35 システム）を分類対象とした．

Robocode を分類対象とした理由は，主に，

- 小規模のソースコード集合を作ることができる
- 出現する単語にある程度偏りが見られる
- 作者が独立して複数いるソースコード集合を作ることができる

の 4 つである．

大規模なソースコード集合に対する実験は現実的ではないため，小規模のソースコード集合を作れるものを対象とした．

小規模のソースコード集合を複数のドメインで構成すると，ドメインの異なるソースコード間に共通の語が少ない可能性がある．この場合，その集合を適切に分類できたとしても，ドメインごとに分類しただけになる可能性が高い．従って，本実験では 1 つのドメインのソースコード集合を対象とした．

ソースコードの作者が一人の場合や，組織立ってソースコードが開発されている場合は，命名規則やコーディングスタイルが統一されている場合が多い．それらが統一されていると，ソースコード中に出現する語の使われ方がある程度限られたり，複合語を作るときの語順が統一されていたりする．しかし，実際ソフトウェア部品の検索をするときは，必ずしも統一されたソースコード集合から部品の検索を行うとは限らないので，この実験では作者が独立して複数いるソースコード集合を対象とした．

5.2 結果

図 4 は各カテゴリに属している部品の適合率で，縦軸が適合率を表し，各縦棒が 1 つのカテゴリに対応する．横軸はカテゴリを適合率でソートして並べている．横軸の数はカテゴリの数を表し，横軸に平行な点線は適合率の平均を表している．

図 5 は各部品が属しているカテゴリの適合率で，各カテゴリに属している部品の適合率の図 4 とは，部品とカテゴリの関係が逆転している．従って，各縦棒が 1 つの部品に対応し，横軸は部品を適合率でソートして並べている．横軸の数は部品の数.

図 4: 各カテゴリに属している部品の適合率（左）

図 5: 各部品が属しているカテゴリの適合率（右）

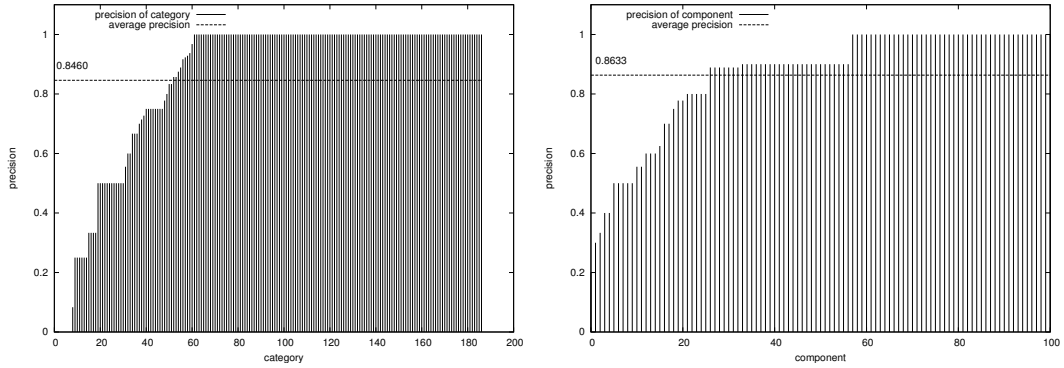


図 6 はソースコード中にはない特徴語の適合率で，縦軸，横軸については図 4 と同様である.

図 6: ソースコード中にはない特徴語の適合率

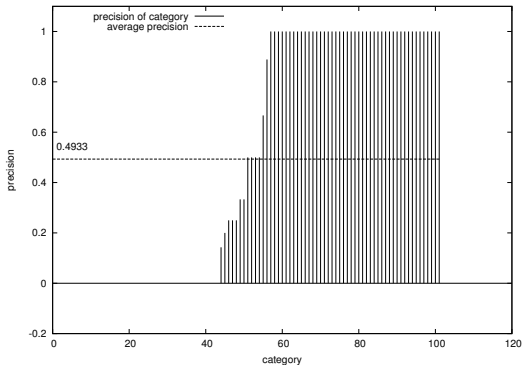


表 8 は上記の図にある適合率の平均の一覧.

表 8: 適合率の平均

	適合率の平均
図 4:各カテゴリに属している部品	0.8460
図 5:各部品が属しているカテゴリ	0.8633
図 6:ソースコード中にはない特徴語	0.4933

5.3 考察

各カテゴリに属している部品，各部品が属しているカテゴリの適合率は，0.85 前後の高い値を示している．従って，有効な分類ができているといえる．

しかし，カテゴリに属している部品の中で，適合率が 0 のカテゴリがいくつか発見された．それらは，Javadoc タグ (@param,@author など) や，HTML タグ (br,li など)，前置詞，助詞，代名詞といったもので，ドキュメントコメント中に繰り返し出現していた．this は代名詞以外にも，Java の予約語として使われているが，他の識別子と区別をしなかったため，いくつかの部品では特徴語となっていた．

特徴語のカテゴリに含まれる部品が，全てソースコード中にその特徴語を持っていた場合，分母が 0 になり適合率が計算できない．このようなカテゴリの，全カテゴリに対する割合は 0.5677 であり，多くを占める．これより，特徴語の多くはソースコード中に含まれていることがわかる．

また，ソースコード中にはない特徴語の適合率は，0.5 近くと部品対カテゴリの適合率に対し，低い値となった．

この原因は，特徴語が 10 個固定である点にあると考えられる．複雑な部品では特徴語の数は 10 個では足りず，高い適合率を得られたものの，特徴語のほとんど，あるいは全てがソースコード中に出現していた．逆に，単純な部品では特徴語の数は 10 個では多すぎ，その中にいくつもの無関係な特徴語が含まれたために，適合率は低い値となっていると考えられる．

6　まとめ

本研究では，ソフトウェア部品に対して特徴語を決定し，それを用いてカテゴリ木を構成する分類手法の提案と実装を行った．また実験を行い，本手法を用いることにより対象部品群に対する詳細な知識がなくとも自動的に分類が行えることを示した．

今後の課題としては，まず，評価実験により明らかになった問題を解決するために，現在10 個に固定してある部品ごとの特徴語の数について適切な数を調査し，その数の取得を自動化すること，および不適切な特徴語の排除方法の考案が挙げられる．また，評価用ではない通常使用のためのユーザインターフェースの開発，SPARS-J との連携，利用関係の重みといったパラメータのよりよい決定方法の模索や，より大規模な実験，そして実行速度およびスケーラビリティの向上が挙げられる．

謝辞

本研究において，常に適切な御指導および御助言を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上克郎 教授に深く感謝致します.

本研究において，常に適切な御指導および御助言を頂きました同 楠本真二 助教授に深く感謝致します.

本研究において，常に適切な御指導および御助言を頂きました同 松下誠 助手に深く感謝致します.

本研究において，常に適切な御指導および御助言を頂きました立命館大学情報理工学部情報システム学科 山本哲男 講師に深く感謝致します.

本研究において，常に適切な御指導および御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 特任研究員 横森励士氏に深く感謝致します.

最後に，その他様々な御指導，御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 ソフトウェア工学講座 井上研究室の皆様に深く感謝いたします.

参考文献

[1] V. R. Basili, G. Caldiera, F. McGarry, R. Pajerski, G. Page and S. Waligora: “The software engineering laboratory - an operational software experience”, *Proceedings of 14th International Conference on Software Engineering*, pp. 370-381, Melbourne, Australia, 1992.

[2] C. Braun: “Reuse”, *Encyclopedia of Software Engineering*, ed. J.J. Marciniak, Vol. 2, pp. 1055-1069, 1994.

[3] S. Isoda: “Experience report on a software reuse project: Its structure, activities, and statistical results”, *Proceedings of 14th International Conference on Software Engineering*, pp. 320-326, Melbourne, Australia, 1992.

[4] B. Keepence and M. Mannion: “Using patterns to model variability in product families”, *IEEE Software*, Vol. 16, No. 4, pp. 102-108, 1999.

[5] SourceForge: “http://sourceforge.net/”.

[6] 川口: “潜在的意味解析法 LSA に基づくソフトウェアシステム分類法の提案”, 2003.

[7] 横森, 梅森, 西, 山本, 松下, 楠本, 井上: Java ソフトウェア部品検索システム SPARS-J, 電子情報通信学会論文誌, D-I, Vol. J87-D-I, No. 12, pp. 1060-1068, 2004.

[8] Katsuro Inoue, Reishi Yokomori, Tetsuo Yamamoto, Makoto Matsushita, Shinji Kusumoto: “Ranking Significance of Software Components Based on Use Relations”, Transactions on Software Engineering, to be published March 2005.

[9] 北, 津田, 獅々堀: “情報検索アルゴリズム”, 共立出版, 2002.

[10] Landauer, T. K., Foltz, P. W. and Laham, D.: “An Introduction to latent Semantic Analysis”, Discourse Processes, Vol. 25, pp. 259-284, 1998.

[11] 山本, 松下, 神谷, 井上: “ソフトウェアシステムの類似度とその計測ツール smmt” 電子情報通信学会論文誌 D-I, Vol. J85-D-I, No. 6, pp. 503-511, 2002. 6.

[12] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue: “CCFinder: A Multi-Linguistic Token-based Code Clone Detection System for Large Scale Source Code”. *IEEE Transactions. Software Engineering*, Vol. 28, No. 7, pp. 654-670, 2002.

[13] 鷲崎, 深澤: “オブジェクト指向プログラムのためのコンポーネント抽出型検索システム”, 情報処理学会オブジェクト指向シンポジウム 2003 論文集, pp. 77-84, 2003. 8.

[14] Börstler, J: “Feature Oriented Classification for Software Reuse”, In *Proceedings of the 7th International Conference on Software Engineering and Knowledge Engineering*, SEKE ’95, KSI Knowledge Systems Institute, Skokie, IL, pp. 204-211, 1995.

[15] Karlsson, E.-A: “Software Reuse - A Holistic Approach”, Wiley, Chichester, West Sussex, UK, 1995.

[16] J. I. Maletic and A. Marcus: “Using latent semantic analysis to identify similarities in source code to support program understanding”, In *12th IEEE International Conference on Tools with Artificial Intelligence*, pp. 46-53, November 2000.

[17] Alvin Chan and Tim Spracklen: “Discoverying common features in software code using self-organising maps”. In *International Symposium on Computational Intelligence*, Kosice, Slovakia, August 2000.

[18] Nicolas Anquetil and Timothy Lethbridge: “Extracting concepts from file names; a new file clustering criterion”, In *International Conference on Software Engineering*, pp. 84-93, April 1998.

[19] G. A. Di Lucca, A. R. Fasolino, F. Pace, P. Tramontana, and U. De Carlini: “Comprehending web applications by a clustering based approach”, In *Proc. of 10th International Workshop on Program Comprehension*, pp. 261-270, Paris, France, June 2002.

[20] Kunrong Chen and Václav Rajlich: “Case study of feature location using dependency graph”, In *8th International Workshop on Program Comprehension* , pp. 231-239, Limerick, Ireland, June 2000.

[21] 椴山, 山本, 阿草: “Fungram に基づくプログラムパターンとその応用”, 電子情報通信学会ソフトウェアサイエンス研究会, Vol. 97, No. 29, pp. 31-38, 1997. 8.

[22] C. Braun: “NATO Standard for the Development of Reusable Software Components”, *NATO Communications and Information Systems Agency*, 1992.

[23] M. W. Berry, T. Do, G. W. O’Brien, V. Krishna, and S. Varadhan: “SVDPACKC (Version 1.0) User’s Guide”, Technical Report CS-93-194, University of Tennessee, Knoxville, TN, April 1993.

付録

1. Latent Semantic Analysis

    (a) ベクトル空間モデル

    (b) 特異値分解

1 Latent Semantic Analysis

ここでは, 提案手法が利用している LSA, および LSA が前提としているベクトル空間モデルについて, その概要を述べる.

1.1 ベクトル空間モデル

一般に文書に含まれる単語間の関連や類似度を計算し, 統計学的にある種の傾向を見出すために, それぞれの単語を何らかの数学的モデル上に変換することが必要となる. ベクトル空間モデルでは, 対象となる文書内に含まれる単語とその頻度を元にして行列を作成し, その行列から単語や文書に対応するベクトルを定義する.

分類対象となる文書の集合 $D = \{d_1, \ldots, d_m\}$ として, d_i に含まれる単語の集合を $W(d_i)$ とする. このとき分類対象全体の語彙 W は以下の式で表される.

$$W = \bigcup_{i=1}^{m} W(d_i)$$

そして, c_{ij} を文書 d_i に単語 $w_j \in W$ が出現した回数として, 文書 d_i に対応する文書ベクトル $\vec{d_i}$ を次の式で定義する. (ただし $|W| = n$)

$$\vec{d_i} = (c_{i1}, c_{i2}, \ldots, c_{in})$$

このベクトルを全ての文書について計算すると, 以下のような $m \times n$ の行列 A が得られる.

$$A = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{bmatrix}$$

このようにして得られた行列 A を共起行列と呼ぶ.

このとき, 行列 A の第 j 列に着目すると, 単語 w_j が各文書に含まれている頻度を並べたベクトルとなる. これを単語 w_j の単語ベクトル $\vec{w_j}$ とする.

このような方法で文書と単語のモデル化を行う. 上記で定義した文書ベクトル $\vec{d}$ や単語ベクトル $\vec{w}$ を用いて文書同士, もしくは単語同士の類似度を計算することができる. ベクトルからの類似度の計算方法としては, 一般に $\cos\theta$ の値が用いられる. 二つのベクトル $\vec{a} = (a_1, a_2, \ldots, a_n)$ と $\vec{b} = (b_1, b_2, \ldots, b_n)$ 間の $\cos\theta$ は以下の式によって求められる.

$$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|}$$

$$= \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}}$$

角度が 0 に近い, すなわち $\cos\theta$ が 1 に近いほど類似度が高く, 逆に -1 に近いほど類似度が低い.

1.2 特異値分解

ベクトル空間モデルにおいて, 分類対象の文書やそこに含まれる単語をベクトルに変換する手法はすでに述べた. しかし, 上述の手法で定義されるベクトル空間はかなり疎 (sparse) であり, そのままの状態では必ずしも適正な類似度が算出できるとは限らない.

例えば文書間の類似度を計算した場合, それらの文書に全く同じ単語が含まれていない限り類似度は 0 である. そのため同一の単語はなくとも類似語を数多く含んだ文書間の類似度が適正なものとならない. また, 直接の関連はなくとも, 間接的に関連づいている文書もこのベクトル空間モデルでは適切に扱うことができない.

LSA では共起行列 A に対して特異値分解 (Singular Value Decomposition:SVD) を利用することにより, この問題の解決を図っている. SVD を行うと $m \times n$ 行列 A を次のような行列 U, S, V という 3 つの行列に一意に分解することができる. (ただし, $l = min(m, n), U : m \times l, S : l \times l, V : n \times l$)

$$A = USV^T$$

$$\begin{aligned} A &= USV^T \\ &= \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1l} \\ u_{21} & u_{22} & \cdots & u_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ u_{m1} & u_{m2} & \cdots & u_{ml} \end{pmatrix} \begin{pmatrix} s_1 & & & 0 \\ & s_2 & & \\ & & \ddots & \\ 0 & & & s_l \end{pmatrix} \begin{pmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ v_{l1} & v_{l2} & \cdots & v_{ln} \end{pmatrix} \end{aligned}$$

ここで, U は左特異行列 (left singular matrix) と呼ばれる $m \times l$ の正規直行行列 ($UU^T = I$, ただし I は単位行列), S は特異値行列 (singular value matrix) と呼ばれる $l \times l$ の対角行列, V は右特異行列 (right singular matrix) と呼ばれる $l \times n$ の正規直行行列 ($VV^T = I$) である. S の対角要素に並ぶ値は特異値と呼ばれ, s_1 が最も大きく, 順に小さくなっていって s_l が最も小さな値となる.

こうして特異値分解によって分解された行列には,「上位 r 個の特異値のみを使って USV^T を掛け合わせた結果は, 元の行列 A の rank r における最小二乗誤差になる」という性質がある. すなわち, $\hat{U} : m \times r, \hat{S} : r \times r, \hat{V} : n \times r$

$$
\begin{aligned}
\hat{U} &= \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1r} \\ u_{21} & u_{22} & \cdots & u_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ u_{m1} & u_{m2} & \cdots & u_{mr} \end{pmatrix} \\
\hat{S} &= \begin{pmatrix} s_1 & & & 0 \\ & s_2 & & \\ & & \ddots & \\ 0 & & & s_r \end{pmatrix} \\
\hat{V}^T &= \begin{pmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ v_{r1} & v_{r2} & \cdots & v_{rn} \end{pmatrix}
\end{aligned}
$$

としたとき, $\hat{A} = \hat{U}\hat{S}\hat{V}^T$ は A の rank r における最小二乗誤差である.

このように, 誤差を最小に保ったまま行列の rank をあえて削減することによって, より関連の強い単語ベクトルが同一次元に縮退され, 類似した値に近似されることが期待できる. そのため, 従来のベクトル空間モデルでは適正な類似度が得られなかった, 間接的に関連のある単語間についても高い類似度を得ることができる.