

修 士 論 文

計算機トラブルシュートドメインに おける固有表現抽出

Named Entity Extraction in Computer
Troubleshoot Domain

指導教員

近山隆 教授



東京大学 工学系研究科
電気系工学専攻

氏 名 37-086477 栗田 光晴

提 出 日

平成22年2月9日

概 要

一般に人々が計算機を扱う上で何らかのトラブルに遭遇した場合、それに対処するための最も手近な情報源は Web である。もともと計算機を扱う人々によって必要とされる情報であることも手伝い、Web 上にはオープンソースソフトウェアのフォーラムや個人によって書かれたブログ、Wiki など、多種多様な形態でトラブルシュート情報が溢れている。現在これらの中から必要な情報を取得する為の手段たり得るのは、Google をはじめとする検索エンジンであるが、複雑なシステムであればあるほどトラブルシュートには多くの要因が絡み合い、単純に一言のクエリ文字列として検索することが困難な状況も多々ある。

これに対して、近年いくつかの Web 上のトラブルシュート文書に関する研究の例が発表されているが、それらは必ずしもユーザの意図をくみ取ることができないものであったり、また特定の情報源のみにしか適用可能でないような手法を用いているなどの問題がある。

本研究はそのような状況を鑑み、Web 上のトラブルシュート文書からその内容において記述されている情報を可能な限りデータソースの形式に依存しない手法で抽出することで、広範なトラブルシュートに関する情報を構造化して集約することを目的として、自然言語処理における固有表現抽出を用いて情報抽出を行う。

一方固有表現抽出には教師データに含めることができなかった未知の固有表現を抽出することが困難であるという問題が提起されており、その対策としてある程度機械的に作成された辞書を学習の素性に用いる方法が提案されている。

本研究ではそれらの手法を実装した上で、それでもなお問題となる辞書にも含めることができない完全な新語の抽出により精度良く対応することを目的として、大量の文書から機械的に抽出された文脈の学習結果を取り入れることを試みた。

目次

第 1 章 序論	1
1.1 背景と目的	1
1.2 本研究の位置づけ	2
1.3 本論文の構成	2
第 2 章 関連研究	3
2.1 Web 上のトラブルシュート文書の処理	3
2.1.1 Automatic Thread Classification for Linux User Forum Information Access	3
2.1.2 Crawling Bug Tracker for Semantic Bug Search	3
2.2 固有表現抽出	5
2.2.1 固有表現クラスの定義	5
2.2.2 系列ラベリング問題としての固有表現抽出	5
2.2.3 教師あり学習による固有表現抽出	6
2.2.4 固有表現抽出と辞書	6
2.2.5 細分類の固有表現抽出	7
2.2.6 未知固有表現	7
2.3 条件付き確率場 (Conditional Random Fields)	8
2.4 能動学習	8
第 3 章 提案手法	12
3.1 固有表現クラスの定義	12
3.2 能動学習によるタグ付け	12
3.3 辞書による未知語の分類精度の向上	12
3.4 辞書からの文脈の学習結果の利用	13
第 4 章 能動学習を用いた固有表現抽出	14
4.1 データの作成	14
4.1.1 クローリング	14
4.1.2 本文抽出	15
4.1.3 文の切り分け	15
4.1.4 品詞タグ付け、見出し語化	15
4.2 タグ付け	16
4.3 分類器	17

4.3.1	素性	17
4.3.2	能動学習	17
4.3.3	評価	18
4.4	実験	18
4.4.1	教師データ数と精度	18
4.4.2	未知固有表現の抽出精度	18
4.5	考察	20
4.5.1	学習データ数と精度の推移に関して	20
4.5.2	未知固有表現の抽出精度に関して	20
第 5 章	辞書による抽出精度の改善	22
5.1	辞書の作成	22
5.1.1	候補語の収集	22
5.1.2	語彙の拡張	24
5.1.3	一般名詞のフィルタリング	24
5.2	素性への辞書の組み込み	24
5.2.1	Trie 木	25
5.3	実験	26
5.4	考察	27
第 6 章	大量文書からの文脈的特徴の取り込み	29
6.1	未知固有表現の分類	29
6.2	素性としての辞書の意味	29
6.3	辞書に対する文脈情報	30
6.4	辞書を用いた大量の文脈情報の取り込み	30
6.5	実験	30
6.5.1	文脈に特化した分類器	30
6.5.2	固有表現の抽出	31
6.6	考察	32
第 7 章	結論	34
7.1	まとめ	34
7.2	今後の課題	34

目 次

2.1	Mozilla における Bugzilla の運用例	4
2.2	Tran らによるバグ情報記述のためのデータスキーマ	5
2.3	系列ラベリング問題としての固有表現抽出	6
2.4	能動学習のアルゴリズム	9
4.1	テキストデータ取得までの処理フロー	14
4.2	文分割のヒューリスティック	15
4.3	タグ付けのための Web アプリケーション	16
4.4	素性のイメージ図	17
4.5	教師データ数と抽出精度	19
4.6	未知固有表現が増加した場合の抽出精度	20
5.1	辞書の作成	22
5.2	価格.com におけるベンダー一覧	23
5.3	Wikipedia エントリによる英訳	24
5.4	アプリケーション名辞書の一部	25
5.5	辞書を追加した素性のイメージ図	25
5.6	Trie 木の例	26
5.7	教師データ数と抽出精度	26
5.8	未知固有表現が増加した場合の抽出精度	27
6.1	提案手法の概念図	31
6.2	大量の文脈情報による特徴を追加した素性のイメージ図	32
6.3	教師データ数と抽出精度	33
6.4	未知固有表現が増加した場合の抽出精度	33

表 目 次

2.1	Automatic Thread Classification for Linux User Forum Information Access における素性	11
2.2	Automatic Thread Classification の精度	11
2.3	IREX における固有表現クラス	11
4.1	クラスごとの抽出精度	18
4.2	置換して未知語として用いた語	19
5.1	クラスごとの抽出精度	27
6.1	クラスごとの抽出精度	31

第1章 序論

1.1 背景と目的

Web が普及して久しい現在、多くの情報が Web を介して提供されている。なかでも計算機に関するトラブルシューティング情報はもともと Web に触れる機会の多い人々に必要とされることも手伝って数多くの文書として存在しているが、必ずしも適切な情報が容易に手に入るとは限らない。現状の検索エンジンのように単語をいくつか送信するような手法では検索を行おうにもどのような語を送信すればよいか分かりづらい場合も多く、またそれによって求める情報が存在するかどうかのような文書が提示されたとしても、肝心のトラブルの構成要素が自身のものと異なるために有用な情報とならないようなケースは多くの人が経験するところであり、より効率的な情報取得のための方法が必要とされている。

ここで、本研究において対象とする問題を明確にするために、本論文中で「トラブルシューティング」という語によって指し示すものの範囲を明確にしておく。本研究においてはまず実際のトラブルシューティング検索の動向を確認するために、トラブルシューティング専用の検索インタフェースを作成し、ユーザによって実行された検索のログの収集を行った。ユーザには「計算機に関するトラブルシューティング」にのみこのインタフェースを利用するよう呼びかけ、ここに送信されたクエリの内容を確認したところ、ユーザが「計算機に関するトラブルシューティング」と認識しているものは大きく以下の3つに大別できることが分かった。

利用法の検索 ソフトウェアやハードウェアの正規の利用法を調査するために行われた検索。トラブルシューティングを広い意味でとらえればこれも含まれる場合もあるが、一般に求める情報が存在することやその場所に見当がついている状況下で索引代わりに検索しているに過ぎず、後述する2つとは性質が異なる。

プログラミングに関する検索 プログラミングを行う上で生じたコンパイルエラーなどに関する検索。下に述べる「想定外の挙動の検索」に含まれると考えることもできるが、こちらはアプリケーションの作成者としてコードを書き直すことを目的とした検索であり、利用者の立場を前提とした「想定外の挙動の検索」とは異なるので別のものとみなす。

想定外の挙動の検索 ソフトウェアやハードウェアの利用法を知った上で操作している状況下で、それらがとった想定外の振る舞いへの対処法に関する検索。

これらのうち、はじめの2つは問題の原因が特定されており、比較的容易に検索して情報にたどり着くことが可能であるのに対し、最後の1つは検索語の選定などに困難を伴うことが多い。よって、本研究は計算機のトラブルシューティングの中でも、上記の「想定外の挙動の検索」に相当するものを対象とするものとし、以下「トラブルシューティング」という語も同種の問題を指すものとする。

本研究は計算機のトラブルシュートに関わる文書から、それらを分類・解析するために必要となる情報を抽出し、Web上のトラブルシュート情報の構造的なデータベースを作成することによるトラブルシュート情報へのアクセスの改善を目的とする。

1.2 本研究の位置づけ

他の分野、例えば医学生物学においては、論文中のタンパク質名を自動的に抽出し、その相互関係をもとにして文書の検索を行うといった研究が行われている。様々な要素が関わって引き起こされる計算機のトラブルシュートに関しても同様の手法は新たな検索の枠組みとしてユーザの利便性を向上する可能性を持つと考えられるが、医学生物学分野のように特にコーパスが存在する訳でもないためそのような研究例は少ない。本研究は、Web上に散在する計算機トラブルシュートに関わる文書を整理し解析するための基盤として、トラブルシュート文書の構造化されたデータベースの構築を目指している。

1.3 本論文の構成

以降、本論文は次のような構成をとる。

まず、2章で関連研究を示す。ここでは提案手法に関連する研究として、Web上のトラブルシュート文書の処理に関する研究例と固有表現抽出に関する研究例を説明する。次に3章において4章以降で具体的に説明する手法に関する大まかな枠組みを説明する。続いて4章で学習に用いる教師データ数を低減させる能動学習を用いた固有表現抽出の実装と抽出精度に関して述べたのち、5章で先行研究の例をもとに、今回対象とする分野の辞書を作成し、その抽出精度への寄与に関して述べる。6章では5章における辞書の素性としての利用において新たに生じる辞書にも含まれることがない新語の抽出精度の問題に対処するために、表記に依存しない文脈的な特徴を大量の文書から学習した結果の利用に関して述べる。最後に7章において結論と今後の課題を述べる。

第2章 関連研究

2.1 Web上のトラブルシュート文書の処理

計算機のトラブルシュート情報の効率的な処理を目指した研究例としては、Baldwin らによるもの [2] や Tran らによるもの [10] が挙げられる。

2.1.1 Automatic Thread Classification for Linux User Forum Information Access

Baldwin らによる “Automatic Thread Classification for Linux User Forum Information Access” はトラブルシュート文書の「質」に着目した研究である。この研究では、トラブルシュートに関する投稿がなされる Web 上のフォーラムサイトの記事の中には有用な物とそうでないものが混在しているという前提の下で、文書の有用性に関して 3 つの尺度を設定し、それらに関して教師あり学習を行うことで、記事の有用性を推定することを目的としている。ここで、3 つの尺度とはそれぞれ

Task Orientation (タスク指向性) その文書が何らかのトラブルの解決を志向しているか

Completeness (完全性) 最初の書き込みに、第三者がそのトラブルを同定するのに十分な情報が含まれているか

Solvedness (解決度) 問題に対する解決法が提示されているか

である。

250 スレッドに対して、これらの各々の指標に関して人手で 5 段階評価を行ったものを教師データとし、素性としては、表 2.1 のようなものを用いている。

こうして作成された教師データを用い、各指標の値を 2.5 を Break Point として二値化し、Support Vector Machine (SVM) を用いてテストデータの分類を行った結果、表 2.2 のような精度となった。ここで、MC とは対照実験として全てのテストデータを教師データにおける多数派に分類する Majority Classifier による結果である。

学習は行っているものの、結果として MC とほぼ同じ分類が行われており、今後の改良が望まれる結果であると言える。

トラブルシュートに関する情報の中には有用なものとそうでないものが存在することは事実であるが、実際の利用シーンにおいて有用な文書は検索を行う利用者が置かれた状況によって異なる。Baldwin らの手法では、有用性が検索する主体に依存しない形で定義されているため、これによって取得される情報が個々の利用者にとって有用であることを保証するものではない。我々の研究は利用者各人の状況に合致した文書を提供するためのものであり、これとは異なる。

2.1.2 Crawling Bug Tracker for Semantic Bug Search

Tran らによる “Crawling Bug Tracker for Semantic Bug Search” は、Web 上に散在する Bug Tracking System (BTS) からトラブルシュート情報を収集し、一つのデータベースにおさめた上で、

それらを事例ベース推論のデータとして用いることによって利用者の求める情報を提供することを目的としたものである。

ここで、BTS とは、テスト担当者によるバグ報告から、開発者がそれを確認し修正するまでのプロセスを管理することを目的として作成されているソフトウェアである。オープンソースによる実装も複数なされており、またオープンソースのソフトウェア開発においては広く利用者からのバグ情報を収集し、またその対応状況を提示するために Web 上に公開されている例が散見される。例えば Mozilla では独自に Bugzilla という BTS を開発・運用しており、図 2.1 のような形式で Web において公開されている。



図 2.1: Mozilla における Bugzilla の運用例

Tran らは、このような Web 上に公開されている BTS の中から比較的大規模な 13 サイトを選択し、それらに含まれる情報を収集している。これらの情報は、図 2.1 のようにシステムによってデータベースにから取り出されて定型的に出力されているため、それらに対して手動で抽出ルールを作成して取得することが可能である。

また、取得した情報をストアするために、バグ情報の記述のためのデータスキーマ (図 2.2) を定義し、それに基づいてデータベースを構築している。

これによって収集された情報を、事例ベース推論のためのデータとして用い、ユーザからの検索クエリに対する応答を評価したところ、一般的な検索語のみからなるクエリに対するものと比較して精度の向上が見られたと報告されている。

この研究においては、先述のように収集すべきデータをバグトラッキングシステムに存在するものに限定している。しかしながら、多くの人々が個人の Blog や Wiki などに存在する情報をもとにトラブルシュートに成功した経験を持っているように、バグトラッキングシステム以外の場所にも有益な情報は数多く存在していると考えるのが自然である。

それに対して、ここで用いられている手法は文書を収集する際にソフトウェアや OS の名前といった取得すべき情報がバグトラッキングシステムによって表やリストの形で構造化された状態で出力されていることを前提としており、構造化されておらず自然言語で書かれることが多い Blog や Wiki などの情報源には元来適用することが困難なものである。また、抽出に用いるテンプレートも人手で作成することで成立しているため BTS のテンプレートの変動に対処することができず、収集可能な情報はさらに限定的である。これに対して本研究は、自然言語で書かれた文書から情報を抽出することで、より広範なデータソースから同様のデータベースを作成することを目的としたものである。

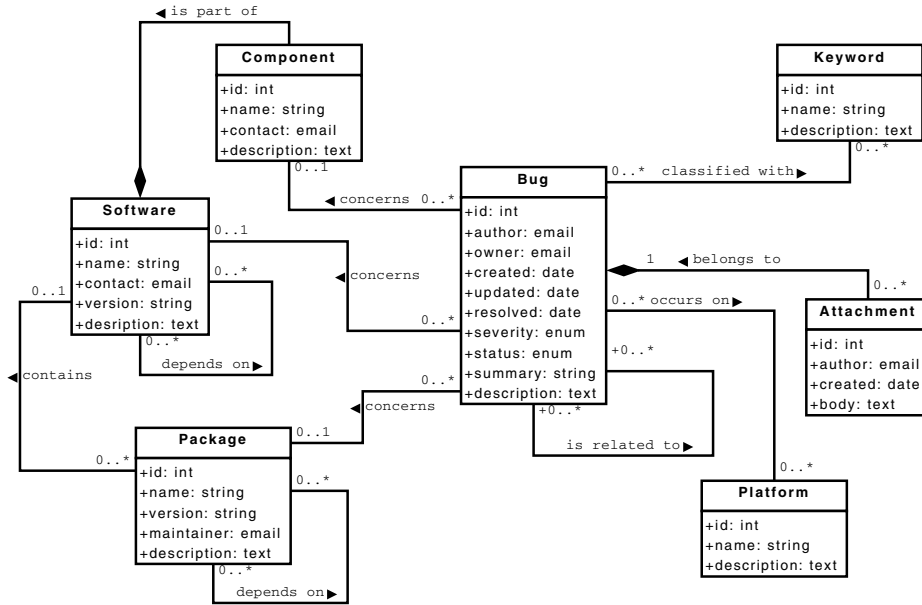


図 2.2: Tran らによるバグ情報記述のためのデータスキーマ

2.2 固有表現抽出

自然言語で書かれた文書から予め定めたクラスに属する情報を抽出するタスクは固有表現抽出として知られており、本研究の目指すところはトラブルシューティング分野に関する固有表現抽出と換言できる。

ここで固有表現とは、文書中における部分単語列のうち、特定のクラスに属する固有名詞句、数詞句を指す。

2.2.1 固有表現クラスの定義


固有表現抽出は、米国防総省の研究機関 DARPA の評価型プロジェクト “Message Understanding Conference (MUC)[9]” に端を発していることから、そこにおいて定義された 7 クラス、もしくはそれをベースに日本で行われた “IREX” における「固有物名」クラスを追加した 8 クラスの分類精度を論じられることが多い。表 2.3 に、IREX における 8 クラスからなる固有表現クラスを示す。

2.2.2 系列ラベリング問題としての固有表現抽出

固有表現抽出は一般に一次元のデータ列に対して適切なラベルを付与する系列ラベリング問題として定式化される。この問題において入力単語の列であり、出力は各単語の固有表現ラベルである。

例えば図 2.3 において、矢印の位置の単語に関して固有表現ラベルを付与する場合、その前後に存在する単語列の表記、品詞、見出し語といった情報を入力とし、ラベルを出力するような分類器を構成することになる。

各固有表現は一語以上の単語によって構成される。そのため、同一のクラスに属する異なる固有表現が 2 つ以上連続して出現する場合、それらの境界が判別できるようにラベルを設計しなければならない。このような問題は一般にチャンク同定問題と呼ばれており、これに対する方策には主に以下の 4 つの方式がある。



	位置	n-2	n-1	n	n+1	n+2
入力	単語	went	to	Washington	yesterday	.
	品詞	VBD	TO	NP	NN	SENT
	見出し語	go	to	Washington	yesterday	.
出力	ラベル	O	O	B-LOC	O	O

図 2.3: 系列ラベリング問題としての固有表現抽出

IOB1 固有表現には I-[クラス名] というタグを付ける。同種の固有表現が連続する場合、後者の先頭語に B-[クラス名] というタグを付ける。固有表現でない語には O というタグを付ける。

IOB2 IOB1 と異なり、いかなる固有表現も先頭語のタグを B-[クラス名] とする。

IOE1 IOB1 と異なり、連続する固有表現のうち前者の末尾語に E-[クラス名] というタグを付ける。

IOE2 IOE1 と異なり、いかなる固有表現も末尾語のタグを E-[クラス名] とする。

2.2.3 教師あり学習による固有表現抽出

一般に固有表現抽出は教師あり学習によって実装される。この場合、教師データとして人手でタグ付けされたデータを与え、それを分類器が学習することで、入力されたデータに対してタグが付与される。

機械学習を用いない最も単純な抽出方法として、予め作成した各固有表現クラスの辞書による単純マッチでも固有表現を抽出することは可能であるが、機械学習を用いた手法には以下のような利点がある。

多義語の処理 人名・地名のいずれとしても出現しうる “Washington” のように、複数のクラスをとり得る語に関しては、辞書でマッチングを図るだけではいずれのクラスに分類すべきか判断がつかない。人間にとって判断するにたる情報が含まれる文においては、何らかの形で文脈を考慮することで適当なクラスを判断することができると考えられるが、当該語句のあらゆる出現に関して妥当な判定ルールを陽に作成することは一般に困難である。一方機械学習の枠組みでは、前後の単語を素性に含めることで文脈を判断材料に含めることができると考えられる。

未知語の処理 辞書によるマッチングでは、各クラスに属する語のうち辞書に含まれていない物を正しく判定することができない。機械学習では、素性設計次第で、人間が未知の語に対してある程度どのような物を指すのか推定できるのと同様に、教師データ中の他の語の出現例から得た知識で新たな語を分類できる可能性がある。

2.2.4 固有表現抽出と辞書

ここまでで固有表現抽出は単純な辞書による抽出が可能な問題ではないことを述べたが、固有表現抽出が辞書と無縁であるわけではない。一般的な固有表現抽出において、辞書は素性の構成要素として用いられる。すなわち、図 2.3 において示した品詞や見出し語といった素性に加え、「当該語がある種の辞書に含まれているか否か」が、二値の素性として利用される。先述の “Washington” の

場合であれば、「人名」の辞書と「地名」の辞書の双方に出現することが学習器の判断材料として利用されることになる。

なお、上述の例は辞書の種別（「人名」及び「地名」）と固有表現クラスが一對一に対応するかのような印象を与える記述となっているが、これらは必ずしも一致している必要はない。たとえば「芸術家の名前」という辞書があればそれは人名の検出精度に貢献する可能性があることは容易に想像でき、また極端な例を出せば一般的な「辞書」のような人間にとって一言で言い表すことのできるようなくくりで作成されたものでなくとも分類精度の向上に寄与する可能性はある。

Web 上の資源をもとにこの辞書を作成して素性に利用したものに、風間らによるものがある [5]。風間らは、Wikipedia における語の出現をもとに、本文の先頭部分から上位語となる語を抜き出すようなパターンを作成している。このパターンによって、「国」や「クリケット選手」といったカテゴリーとその構成要素の辞書が作成され、これが学習に組み込まれることで精度を向上させている。

2.2.5 細分類の固有表現抽出

本研究において対象とする領域では、上述のクラスのうち固有物名の中に含まれるであろうアプリケーションの名前やハードウェアの名前といったものが抽出対象となる。そのため独自に固有表現クラスを定義し、学習のための教師データを与える必要がある。

上述の一般的な 7 ないし 8 クラス以外のクラスを定義した固有表現抽出としては、関根らによる拡張固有表現 [7] がある。これはオントロジーに似た、人間の概念に基づく物の分類の階層構造を定義し、その葉ノードに当たるクラスを固有表現のクラスとするものである。関根らは、こうして取得された固有表現クラスに基づき、On-Demand Information Extraction と呼ばれるシステムを構築している [8]。このシステムは、利用者が検索語として送信した語に関連する文書を既存手法を用いて抽出した上で、その文書中に含まれる固有表現同士の関係の中から他の文書と比較して特徴的なものを抽出し、その関係の構成要素を表として出力するものである。例えば、「企業買収」という語で検索を行った場合、企業買収に関連する文書が集められ、その中で「企業 A が企業 B を金額 C で買収」という A、B、C 三者の関係を発見し、個々の企業買収事例における買収した企業 A、買収された企業 B、買収額 C を 3 カラムからなる表に出力する。

拡張固有表現のような既存の固有表現クラスの細分類を目指したもの以外の固有表現クラスの定義としては、医学生物学分野において固有表現抽出が盛んに行われている [4]。医学生物学分野においては、抽出すべき対象となる固有表現クラスとして protein、DNA、cell line、cell type、lipid、other names が定義されている。

2.2.6 未知固有表現

一般的に、固有表現抽出の精度を論じる場合、テストデータ中の抽出対象となる語が教師データに出現したか否かは特に考慮されない。福島らは、固有表現抽出における機械学習の長所の一つである「未知語の扱い」に着目し、未知固有表現の抽出に取り組んだ [13]。この研究においては、作成のための人手のコストによって数量が制限される教師データに対して、先述のように辞書を用いることで学習データには含まれていない語の抽出精度の向上を試みている。福島らの研究の主眼はこの辞書をいかに効率的に作成するかに置かれており、この研究では大量にストックされた Web 上の文書リソースの中から共通したパターンの中に出現する語を集めた複数の辞書を作成し、素性に利用している。

本研究が抽出対象とするアプリケーション名やハードウェア名はコンスタントに新語が出現するクラスであるため、未知語の抽出精度の向上は重要な課題である。

2.3 条件付き確率場 (Conditional Random Fields)

機械学習を用いた固有表現抽出には、Support Vector Machine を用いたもの [3] や条件付き確率場 (Conditional Random Fields; CRF) を用いたものなどがある。本節では一般に現状最も高精度に固有表現を抽出することが可能とされる CRF について述べる。

CRF では、入力列 \mathbf{x} に対する出力列 \mathbf{y} の条件付き確率 $P(\mathbf{y}|\mathbf{x}; \boldsymbol{\lambda})$ を以下のように定義する。

$$P(\mathbf{y}|\mathbf{x}; \boldsymbol{\lambda}) = \frac{1}{Z(\mathbf{x})} \exp \left(\sum_{i=1 \dots |\mathbf{y}|} \sum_k \lambda_k f_k(\mathbf{x}, y_{i-1}, y_i, i) \right) \quad (2.1)$$

ここで、 $Z(\mathbf{x})$ は $P(\mathbf{y}|\mathbf{x}; \boldsymbol{\lambda})$ が確率として成立するための正規化項であり、

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \exp \left(\sum_{i=1 \dots |\mathbf{y}|} \sum_k \lambda_k f_k(\mathbf{x}, y_{i-1}, y_i, i) \right) \quad (2.2)$$

である。

$f_k(\mathbf{x}, y_{i-1}, y_i, i)$ は素性関数と呼ばれ、入力列とラベルの関係に基づき値が決定される関数である。素性関数は上記の形式で表現される関数であれば内部的にはどのような処理が行われても問題ないが、一般的にはトークン部分のみを引数とする関数とタグの値の組み合わせに応じて 0, 1 の二値を出力するような関数が用いられる。この関数の設計によって分類器が学習すべきモデルが決定されることになる。例えば

$$f_k = \begin{cases} 1 & \text{if } (x_i \text{ が大文字で始まる}) \cap (y_i = \text{"B-LOC"}) \\ 0 & \text{otherwise} \end{cases} \quad (2.3)$$

という素性関数を定義すると、先頭文字列のキャピタライズの有無に着目し地名クラス先頭語であるか否かを判定するモデルとなることが期待される。

λ_k は各素性関数の重みを表す。CRF における学習とは、教師データとして与えられた N 個の入力列-ラベル列対 $(\mathbf{x}_n, \mathbf{y}_n)$ ($n \in \{1 \dots N\}$) に関して尤度

$$\prod_{n=1}^N P(\mathbf{y}_n|\mathbf{x}_n; \boldsymbol{\lambda}) \quad (2.4)$$

を最大とするような $\boldsymbol{\lambda}$ を求めることにほかならない。

学習したモデルをもとに実際に分類を行う際には、求められた $\boldsymbol{\lambda}$ を用いて、入力列 \mathbf{x} に対して条件付き確率 $P(\mathbf{y}|\mathbf{x}; \boldsymbol{\lambda})$ が最大となるラベル列、つまり

$$\mathbf{y}^* = \underset{\mathbf{y}}{\operatorname{argmax}} P(\mathbf{y}|\mathbf{x}; \boldsymbol{\lambda}) \quad (2.5)$$

を求める。このときラベル列の候補数は入力列の長さの指数関数オーダーとなるが、動的計画法の一種である Viterbi アルゴリズムによって効率的に求めることが可能である。また y_i が各値をとる周辺確率¹も forward-backward アルゴリズムを用いて求められる。

2.4 能動学習

一般に教師あり学習では、予め一定のデータを選択したのち人手でタグ付けを行い、それを学習器に与えて分類器を作成する。ここで、教師とするデータは全データ集合からランダムに抽出するなどして用意されることが多いが、当然ながら学習の結果は与えられるデータによって変動する。

¹周辺確率とは、事象 A_i ($i \in \{1 \dots n\}$) に対する事象 B の条件付き生起確率 $P(B|A_i)$ が与えられた場合の事象 B の生起確立の総和 $\sum_{i=1}^n P(B|A_i) P(A_i)$ を指す。

能動学習はこの点に着目し、ある程度学習が進んだ時点でその学習結果を踏まえ、その後に利用する教師データの選別を行い、トータル教師データ数を少なく抑えることを目指す手法である。一般に教師データの作成は人間が行うしかなく機械学習において最も高コストな作業であるため、この手間を削減することは非常に有意義である。

```

 $B \leftarrow$  各ラウンドで追加するサンプル数
 $L \leftarrow$  ラベル付けされたサンプル集合
 $P \leftarrow$  ラベル付けされていないサンプル集合
 $U_M \leftarrow$  効用関数
repeat
  モデル  $M$  を  $L$  を用いて学習
  for  $p_i \in P$  do
     $u_i \leftarrow U_M(p_i)$ 
  end for
  効用  $u_i$  の上位から  $B$  件のサンプルを選択する
   $B$  件のサンプルを人間に問い合わせる
  新たにラベルが付与された物を  $P$  から  $L$  に移動する
until 学習が収束する

```

図 2.4: 能動学習のアルゴリズム

教師データとして用いるデータは、何らかの形でそのデータにラベルが付与された場合に学習器に対してどれほど情報を与えうるか (Informativeness) を評価される必要がある。これは分類器の出力形式によって様々な取り方が考えられるが、例えば線形分類器では分離超平面からの距離、確率によるモデルを用いる分類器では出力されたラベルの確率値の低いものといった基準を用いて選択される。CRF を用いた系列ラベリング問題における能動学習での Informativeness に関しては、Settles らの研究が詳しい [1]。

Settles らが示した既存の Informativeness には以下のようなものがある。(以下、式中の記号は断りなき場合 CRF の式におけるものに準ずる。)

Least Confidence

$$\Phi^{LC}(\mathbf{x}) = 1 - P(\mathbf{y}^* | \mathbf{x}; \theta) \quad (2.6)$$

これは、最尤と判断されるラベル列の条件付き確率が低いもの、すなわち分類器にとって最も確信度の低いものを選択する指標である。

Margin

$$\Phi^M(\mathbf{x}) = -(P(\mathbf{y}_1^* | \mathbf{x}; \theta) - P(\mathbf{y}_2^* | \mathbf{x}; \theta)) \quad (2.7)$$

ここで、 \mathbf{y}_n^* は n 番目に尤度の高いラベル列を表す。

これは、最も尤度の高いラベル列と、その次に尤度の高いラベル列の尤度差が小さいもの、すなわち分類器が大差を付けて判別できなかったものを選択する指標である。

Token Entropy

$$\Phi^{TE}(\mathbf{x}) = -\frac{1}{T} \sum_{t=1}^T \sum_{m=1}^M P_\theta(y_t = m) \log P_\theta(y_t = m) \quad (2.8)$$

ここで、 M はラベル数、 T はトークン数、 $P_{\theta}(y_t = m)$ はモデル θ において第 t トークンのラベルが m である周辺確率を表す。

これは、各トークンごとにラベルの情報量を求め、全トークンのラベルの情報量の平均値の高いものを選択する指標である。

Total Token Entropy

$$\Phi^{TTE}(\mathbf{x}) = T \times \Phi^{TE}(\mathbf{x}) \quad (2.9)$$

これは、情報の総量の観点から、長いトークンほど多くの情報を持つことを考慮して、全トークンのラベルの情報量の総和が高いものを選択する指標である。

Sequence Entropy

$$\Phi^{SE}(\mathbf{x}) = - \sum_{\hat{\mathbf{y}}} P(\hat{\mathbf{y}}|\mathbf{x}; \theta) \log P(\hat{\mathbf{y}}|\mathbf{x}; \theta) \quad (2.10)$$

ここで、 $\hat{\mathbf{y}}$ は入力 \mathbf{x} が取り得る全てのラベル列の集合を表す。

これはラベル列のエントロピーが大きいものを選択する指標である。

N-best Sequence Entropy

$$\Phi^{NSE}(\mathbf{x}) = - \sum_{\hat{\mathbf{y}} \in N} P(\hat{\mathbf{y}}|\mathbf{x}; \theta) \log P(\hat{\mathbf{y}}|\mathbf{x}; \theta) \quad (2.11)$$

ここで、 N は尤度の降順で第 N 番目までのラベル列を表す。

これは Sequence Entropy と同様であるが、値の算出を N 位までで打ち切ることで計算量を削減することを意図した指標である。

Feature Description	Feature Type
Number of posts	Integer
Number of Linux distribution mentions (e.g. redhat)	Integer
Number of beginner keywords mentioned (e.g. noob)	Integer
Number of emoticons detected (e.g. :-))	Integer
Number of version numbers mentioned (e.g. version 5.1)	Integer
Number of URLs detected (e.g. www.ubuntu.org)	Integer
Proportion of words relative to full thread	Real
Proportion of sentences relative to full thread	Real
Proportion of question sentences in set	Real
Proportion of exclamation sentences in set	Real
Proportion of simple declarative sentences in set	Real
Proportion of code sentences in set	Real
Proportion of other sentences in set	Real
Average sentence length Average word length	Real
Proportion of sentences to first question	Real
Proportion of thread posts to first class post	Real
Proportion of thread posts to last class post	Real

表 2.1: Automatic Thread Classification for Linux User Forum Information Access における素性

Method	Task Orientation	Completeness	Solvedness
MC	0.816	0.948	0.765
SVM	0.816	0.948	0.779

表 2.2: Automatic Thread Classification の精度

固有表現クラス	例
ARTIFACT (固有物名)	ノーベル文学賞
DATE (日付表現)	五月五日
LOCATION (地名)	日本、韓国
MONEY (金額表現)	2000 万ドル
ORGANIZATION (組織名)	社会党
PERCENT (割合表現)	二〇%、三割
PERSON (人名)	村山富市
TIME (時間表現)	午前五時

表 2.3: IREX における固有表現クラス

第3章 提案手法

本研究では、文書自体の形式による制約を受けず、様々なデータソースに散在するトラブルシューティング情報から機械的にそこで述べられているトラブルに関わる要素を抜き出し、トラブルシューティングに関するデータベースを作成するための手法として、トラブルシューティング文書における固有表現抽出を行う。

3.1 固有表現クラスの定義

Tran らによるデータスキーマ (図 2.2) を参考にトラブルシューティングに関わる固有表現のクラスを定義する。Tran らのデータスキーマはバグトラッキングシステムに登録されている情報を前提として設計されているものであるため、キーワードや添付ファイルといった投稿にまつわるデータが組み込まれている。また、ソフトウェアに焦点が絞られるバグトラッキングと異なり、一般のトラブルシューティングではハードウェアも重要な構成要素となる。

以上を踏まえ、本研究では一般の計算機トラブルシューティングに関する固有表現クラスとして、以下の4クラスを定義する。

アプリケーション名 OS の上で動作する特定機能を持ったソフトウェア

プラットフォーム名 上記のアプリケーションの動作環境を提供する OS

ハードウェア名 コンピュータやそのパーツの品名および型番

ベンダ名 ソフトウェアおよびハードウェアの製造者

3.2 能動学習によるタグ付け

関連研究の多くは固有表現抽出の精度の比較のために用意されたデータセットを利用している。しかしながら本研究の対象とする領域は未だ一般的に固有表現抽出の対象としては認識されておらず、そのような既成の教師データは存在しない。そこで本研究では自ら教師データを作成するために大量の文書に対してタグ付けを行う必要があるが、ここに能動学習を用いることで少数の教師データによる効率的な学習を行う。

3.3 辞書による未知語の分類精度の向上

2章において関連研究を示したように、固有表現抽出では教師データに出現していない未知固有表現の抽出精度がその他の語に比して低迷するという問題が存在する。これに対し、福島らの研究例をもとに、Web 上のテキストからヒューリスティックを用いて各クラスの語の出現例を抽出し、それらをもとに作成した辞書を素性に組み込む。

3.4 辞書からの文脈の学習結果の利用

辞書を素性に組み込むことで、教師データには存在しないながらも辞書には含まれているような語を抽出することが容易となるが、辞書作成時点においては存在していないような未知語の抽出は困難になる。辞書と同様にある程度機械的に作成した文脈情報による判断基準を学習させることで、未知の語のクラスをその文脈から推定できる分類器を目指す。

第4章 能動学習を用いた固有表現抽出

4.1 データの作成

今回教師データおよびテストデータとして用いるテキストは、Web 上のトラブルシューティング情報が掲載されているサイトから作成する。以下、図 4.1 に沿って実装について述べる。

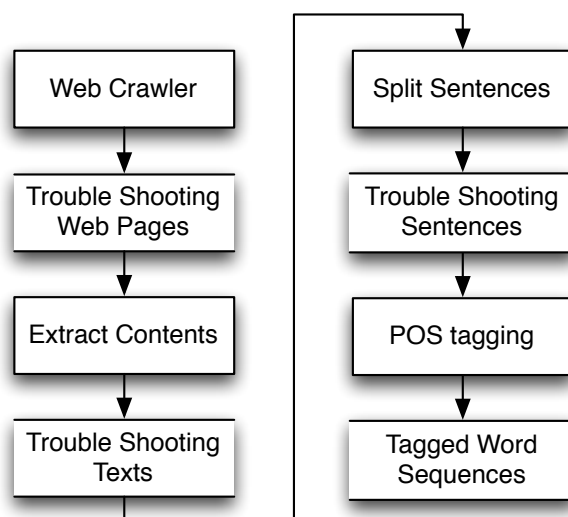


図 4.1: テキストデータ取得までの処理フロー

4.1.1 クローリング

まず、教師データ及びテストデータのもととなる HTML ドキュメントを取得するために、簡易的なクローリングを行う。このクローラはひたすらリンクをたどる一般的な検索エンジンのクローラとは異なり対象とするフォーラムサイトに特化しており、URL パターンに対してたどるべきアンカー要素を予め指定しておくことで、必要なページのみを取得するように作成している。

ここで、アンカー要素の指定には XPath を用いた。HTML 文書は Document Object Model (DOM) という木構造で扱う事が可能であり、DOM ノードの相対的な位置関係をこの木構造上のパスで表現するためのデータ形式が XPath である。

今回データソースとして用いたフォーラムサイトは“Linux Forums¹”である。このサイトには各種の Linux ディストリビューションに関するフォーラムが存在し、トラブルシュートに関する情報も多く扱われている。

¹<http://www.linuxforums.org/>

4.1.2 本文抽出

取得した HTML には記事本文とは別に、フォーラム内の各所へのリンクや広告などが含まれている。この研究の興味はあくまで利用者によって投稿された記事の内容にあるので、これらは不要なノイズであるため、本文のみを抽出する必要がある。不特定多数の Web ページからの本文抽出は機械的に行うことが困難であるが、今回は上述の通り対象とするページを事前に定めているため、予め HTML の構造が分かっている。ここでもやはり XPath によって対象となるテキストの DOM 要素を指定することで、目的のテキストのみを抽出している。

4.1.3 文の切り分け

最終的に学習に用いるデータは一文ごとの単語列となる。また、次の工程である品詞タグ付けに用いるためには、入力するテキストは文ごとに切り分けられていなければならない。今回対象としている文書は英語で書かれているため、基本的に文の終端は “.”、“!”、“?” のいずれかである。しかしながら、“!” および “?” はほぼ文末のみに現れるものの、“.” に関しては数値や略称などにも出現し必ずしも文の終端のみに出現するわけではないため、単純な文字列分割による手法は不適当である。今回は Weiss らの著書 “Text Mining[11]” に掲載されているヒューリスティック (図 4.2) を用いて文を分割する。Weiss らによれば、このアルゴリズムは 90%以上の精度で正しく文の分割を行うことが可能である。

1. 全ての “?” と “!” は終端である
2. ” もしくは ’ に続くピリオドは終端である
3. 直後の文字が空白でないピリオドは終端でない
4. “)”, “}”, “]” に続くピリオドは終端である
5. 「大文字で始まり」「5 文字以下からなる」トークンに付いていて、次のトークンの先頭が大文字で始まるピリオドは終端でない
6. 途中にピリオドを含むトークンの末尾のピリオドは終端ではない
7. 小文字で始まるトークンに付いており、直後のトークンが大文字で始まるピリオドは終端である
8. 2 文字以下からなるトークンの末尾のピリオドは終端ではない
9. 直後のトークンが “\$”, “(”, “{”, “[” で始まるピリオドは終端である
10. 上記いずれにも当てはまらないピリオドは終端でない

図 4.2: 文分割のヒューリスティック

4.1.4 品詞タグ付け、見出し語化

テキストを文に切り分けたのち、品詞タグ付けと見出し語化を行う。見出し語化とは、名詞であれば単数形、動詞であれば原型といった、辞書に載っているような基本形を取得するタスクである。

今回、このタスクには、品詞タグ付けと見出し語化を一度に行う “Tree Tagger[6]” を用いた。

4.2 タグ付け

分類器の精度を評価するためにはテストデータを用意する必要がある。今回のタスクはテキストデータを直接編集するような方法で対処できるようなデータ量ではないため、独自にアプリケーションを作成してタグ付けを行った。このアプリケーションでは、前節までの工程を経たデータがサーバ上のデータベースに格納され、ユーザは Web 上のインタフェースを介してグラフィカルにタグ付けを行う事が可能である。また、必要なデータが増加した場合に備え、複数のユーザによって実行した際のトレーサビリティを確保するため、アカウント作成によるユーザ管理を行っている。

アプリケーションの外観を図 4.3 に示す。

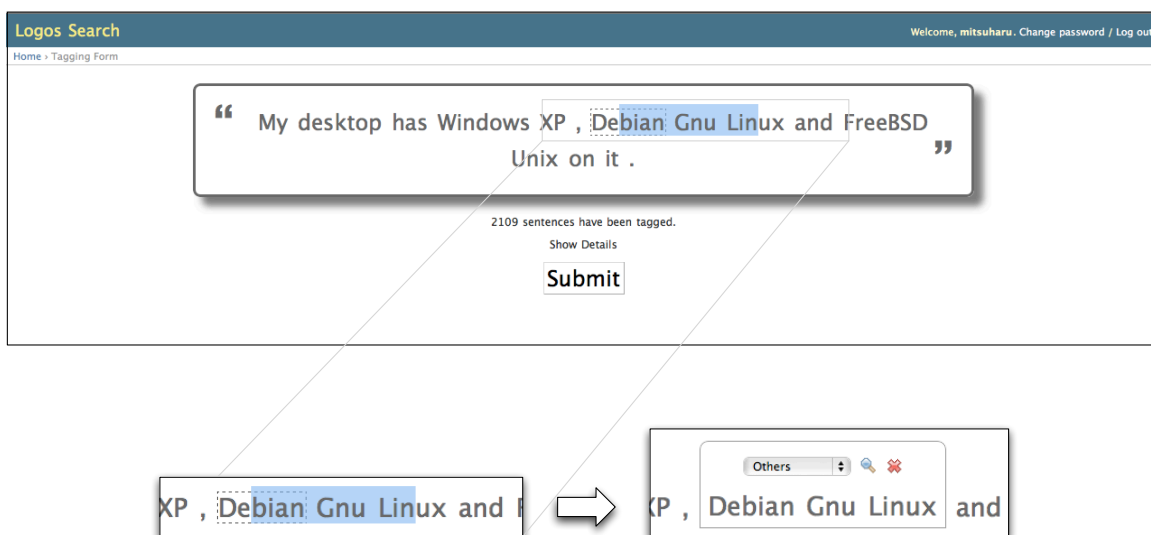


図 4.3: タグ付けのための Web アプリケーション

このアプリケーションに Web ブラウザでアクセスすると、前節までの工程を経てデータベースに格納されているデータからランダムに選択された 1 センテンスが提示される。ユーザはそのセンテンスにおいてタグを付与すべき単語列の始点と終点をマウスによるドラッグで選択する。選択が終わると同時に、選択範囲に含まれた単語が一つのブロックとなり、そこにタグを選択するためのドロップダウンリストが表示される。ここで、ユーザはリストの中から適切なラベルを選択する。ラベルが選択されると、内部ではタグ付けされた範囲に応じて 2 章において述べた IOB2 形式でラベルが設定されるようになっている。そのためユーザ側では語句内の位置によるラベルの種類の違い (B-[クラス名], I-[クラス名]) を意識する必要はない。

タグ付けを行うべき語句全てに対してラベルを選択したのち、Submit をクリックするとタグ情報がサーバに送信され、タグ付けを行ったユーザのアカウントや日時とともにデータベースに保存される。

このときユーザに対しては、次にタグ付けを行うべきセンテンスがランダムに選択され、再度提示される。

今回はこの工程を筆者自身が実行し、テストデータとして 1000 センテンスにタグ付けを行った。

4.3 分類器

4.3.1 素性

素性としてはラベル付け対象とその前後 2 単語に関して、以下のような特徴を用いた。


表記 文中に出現したその語の文字列そのもの。全て小文字に変換している。

品詞 その語の品詞。Penn Treebank による定義に基づく。

見出し語 その語の辞書に掲載されるような原形。books であれば book、など。これも表記同様全て小文字としている。辞書に掲載されていない語に関しては “<unknown>” となる。

キャピタライズの有無 その語の先頭 1 文字がアルファベットの大文字であれば 1、そうでなければ 0。

アルファベット以外の文字の有無 その語の中にアルファベット以外の文字 (数字や記号) があれば 1、そうでなければ 0。



入力	位置	n-2	n-1	n	n+1	n+2
	単語	to	uninstall	Editra	in	Xubuntu
	品詞	TO	NP	NP	IN	NP
	見出し語	to	<unknown>	<unknown>	in	<unknown>
	キャピタライズ	0	0	1	0	1
	文字種	0	0	0	0	0
出力	ラベル	O	O	BA	O	BP

図 4.4: 素性のイメージ図

CRF の実装としては、工藤らによる CRF++[12] を用いた。

4.3.2 能動学習

2 章で述べたように、能動学習を行うに当たっては各データの Informativeness を定義しなければならない。

今回の実験においては、取得されたテキストの中にコマンドラインが混入することが避けがたい。コマンドラインには一般の文に現れるような句読点が出現しないため複数行にわたる文字列がひとかたまりの文として切り分けられることになる。そのため一部の文においては長大なターミナルの出力が文の一部として入り込み、非常に長い単語列を形成しているものが存在する。この場合、ラベル列全体の条件付き確率をもとにして計算を行う指標では、長大な単語列において条件付き確率が低く算出されるため、結果としてコマンドラインが混入した長大なセンテンスばかりが問い合わせられる傾向が見られた。そのため、今回の実装では Term Entropy を Informativeness として用いることとする。

タグ付けには、テストデータの場合と同様に上述の Web アプリケーションを利用したが、一部能動学習のために変更を加えている。ここでは、データが処理されるごとにランダムに次のデータが

選択されるのではなく、各データに優先順位がひも付けられ、これに従ってラベル付けするデータが決定される。

アプリケーションの具体的な動作は以下になる。はじめに、取得された全 205440 センテンスからランダムに 1000 センテンスに対して優先順位が設定され、ユーザはこれらに対してラベル付けを行う。優先順位が設定されており、なおかつラベル付けが済んでいないデータがなくなった時点で、アプリケーションは学習フェーズに入る。

学習フェーズの動作は基本的に Web 上のアプリケーションとは独立して実装されており、これまでに作成された教師データを用いてモデルを作成し、それを用いてまだラベル付けのなされていないデータに対して分類を行う。分類した結果から、Term Entropy の降順にデータをソートし、その上位 100 件に新たに優先順位を付与する。この間 Web アプリケーションには、学習過程のログファイルが表示されている。

学習フェーズが完了するとシステムはラベル付けフェーズに戻り、ユーザは新たなデータにラベル付けを行う。

テストデータと同様に、これも筆者自身で 2000 センテンスに対してタグ付けを行った。

4.3.3 評価

評価には、一般的な機械学習の評価指標である Precision、Recall、および F 値を用いる。Precision はシステムが正例と判断したデータのうち実際に正例であったものの割合、Recall は全テストデータ中に存在する正例のうちシステムが正例と判定できたものの割合である。

固有表現抽出は系列に対して付けられたラベルに意味があるため、今回の正誤判定は単語単位ではなく語句単位で行っている。すなわち、分類器によって出力されたクラスと正解のクラスが一致しており、なおかつその始点と終点が一致した場合のみ正解であるとする。

4.4 実験

4.4.1 教師データ数と精度

タグ付けデータ数が 1000 センテンスから 2000 センテンスまでの各段階で作成された分類器でテストデータ 1000 センテンスを分類した結果が図 4.5 である。

図 4.5 から、教師データが増加するにつれて抽出精度が向上していることがわかる。

最も精度の高い教師データ 2000 件の時の抽出精度をクラスごとに調べたものが表 4.1 である。

表 4.1: クラスごとの抽出精度

クラス	Precision	Recall	F 値
ベンダ	0.74	0.57	0.65
ハードウェア	0.35	0.28	0.31
プラットフォーム	0.76	0.61	0.67
アプリケーション	0.59	0.50	0.54

4.4.2 未知固有表現の抽出精度

2 章で述べた通り、固有表現抽出における機械学習は多義性の解消と未知語の処理にその意義があり、また本研究においては頻繁に未知語が出現するクラスを対象とすることから特に未知の固有表

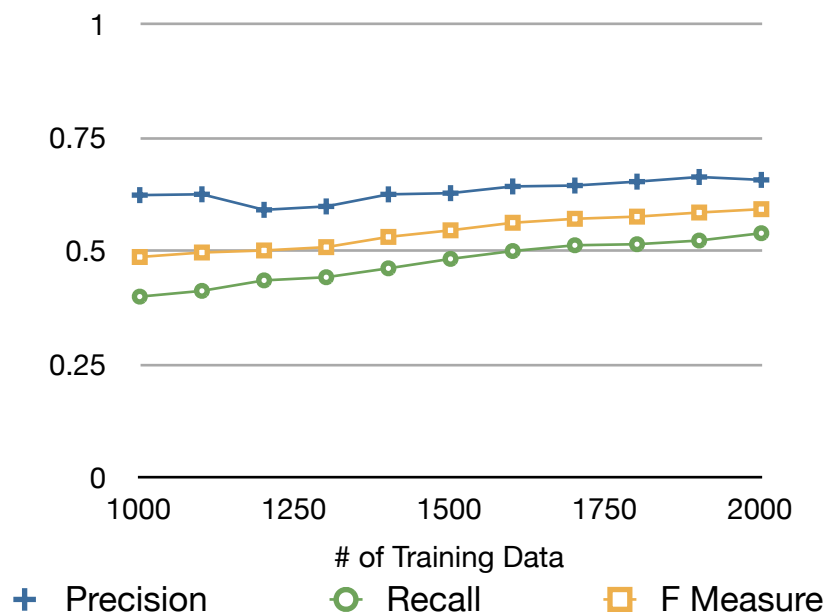


図 4.5: 教師データ数と抽出精度

現の抽出可能性は重要な問題である。

そこで、未知語が出現した場合を想定して、テストデータの中から各クラスのいくつかの語を全て “foobarbaz” という教師データに出現しない語に置換した上で同様の抽出実験を行った。置換した語を表 4.2 に示す。

クラス	単語
ベンダ	intel
	nvidia
ハードウェア	thinkpad
	vaio
プラットフォーム	debian
	ubuntu
アプリケーション	apache
	gnome

表 4.2: 置換して未知語として用いた語

ここで、今回学習に用いている素性のうち表記は一致するか否かのみを見るものであるため、文字列の類似度などは影響を与えないことに留意されたい。

こうして作成されたテストデータに対する抽出精度を示したものが図 4.6 である。

ここで、F 値が最大となる教師データ 2000 件での分類結果を見ると、置換する前と比較して F 値において 0.59 から 0.47 と、大きく落ち込んでいることが分かる。この内訳としては、Precision は 0.66 から 0.58 と 0.08 の低下であったのに対し、Recall は 0.58 から 0.39 と 0.19 も低下しており、未知固有表現の多くを抽出しのがしていることが読み取れる。

置換した語は 248 語であり、うち 161 語にはもともと正しくラベル付けがなされていたが、置換後に正しくラベル付けされたものは 19 語であった。

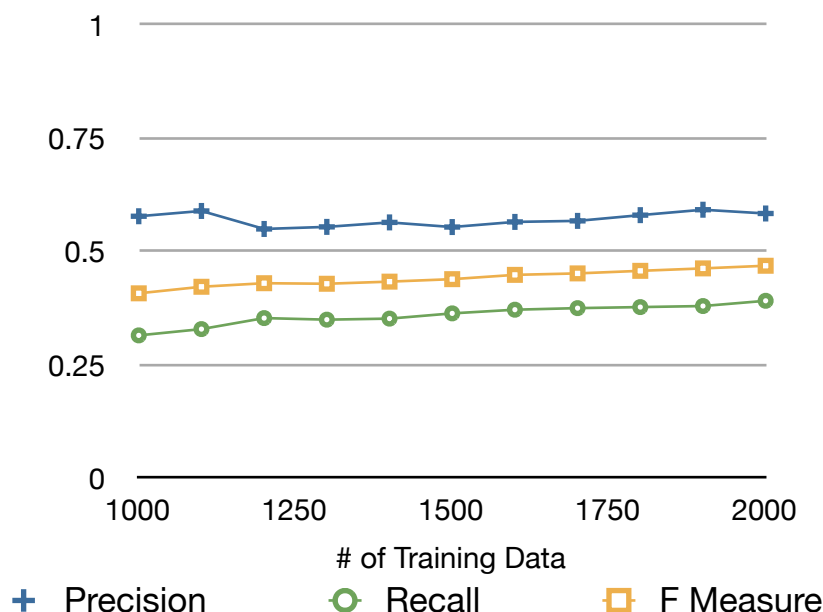


図 4.6: 未知固有表現が増加した場合の抽出精度

4.5 考察

4.5.1 学習データ数と精度の推移に関して

図 4.5 から、今回実験を行った 2000 件までの能動学習の範囲では Precision はおおよそ収束しているように読み取れるが、Recall はまだ改善の余地が残されているように思われる。

今回抽出対象とした語句は、IREX における日本語固有表現抽出のタスクにおける固有表現クラスの定義の中で言えば、ベンダ名が『組織名』、それ以外の全てが『固有物名』クラスに属するものとなる。IREX における固有表現抽出の研究例では、固有物名クラスの語句の抽出精度はおおよそ 0.5 程度となっており、それよりも高い精度で抽出が可能であったことがわかる。

今回抽出対象とする文書は英語であり、英語での固有表現抽出には日本語での固有表現抽出に対して固有表現の先頭がキャピタライズされるなどの傾向があるため抽出精度は高く出やすい面があるが、IREX では教師データとして 10,000 文の利用を許していることを鑑みると、2000 件という少量の教師データとしては効率的な学習が行われているものと考えられる。

4.5.2 未知固有表現の抽出精度に関して

いくつかの語を未知の固有表現に置換した実験結果から、この素性では学習データに出現していないデータに対しての抽出精度が非常に低いことが確認された。また、表 4.1 のクラスごとの分類精度を見ると、特にハードウェアの Recall が低くとどまっている。

この現象に関しては、ベンダやプラットフォームといったクラスでは該当する表現が取り得るバリエーションが比較的少ないのに対し、ハードウェア名は出現する語のバリエーションの多いクラスであるために教師データに含まれていない語が多く出現し、それらの抽出が困難であるために抽出精度が低迷してしまっているものと思われる。

ここで確認された教師データに出現しない未知の固有表現の抽出精度の問題に対し、次章では福島らの研究の例をもとに、教師データにおいて未知であった語の Recall の向上に有効であったとさ

4.5. 考察

れる、Web 上のテキストから作成した辞書での出現の有無を素性に組み込むこと手法によって、抽出精度の改善を図る。

第5章 辞書による抽出精度の改善

2章で述べたように、辞書は固有表現抽出において素性の一つとしれ取り込まれ、分類精度の向上に寄与する。本章では、各固有表現クラスに関する辞書を作成し、それらを素性に組み込むことで抽出精度の向上を図る。

5.1 辞書の作成

福島らや風間らの研究では、人手で作成したきれいな辞書でなくとも、整合性のあるものであれば素性として利用する範囲では有用であることが示されている。そこで、今回設定した各固有表現クラスに関する大まかな辞書を Web 上の文書から作成して、学習に取り込む。

辞書の作成は、図 5.1 に示すように、

1. 候補語の収集
2. 語彙の拡張
3. 一般名詞のフィルタリング

の3つのステップを経て行う。

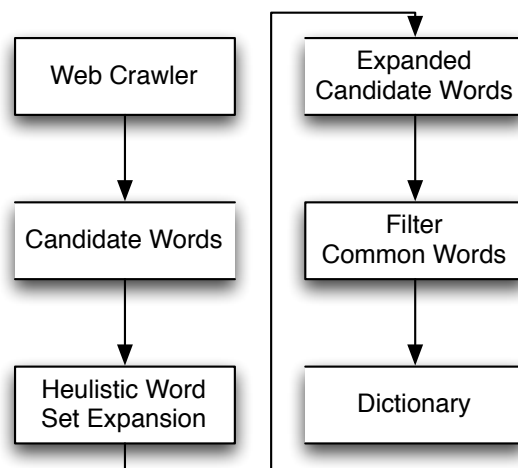


図 5.1: 辞書の作成

5.1.1 候補語の収集

各クラスの語や、それを一部として含む語を収集する。今回用いた情報源は以下のようなものである。

価格.com¹ ベンダ名、ハードウェア名およびプラットフォーム名の候補語を収集した。価格.com における「パソコン」カテゴリの商品一覧は、さらにその下位分類に機器の種別（「パソコン本体」、「プリンタ」等）を持ち、そのさらに下位にベンダごとの分類がある（図 5.2）。この位置にリスト表示されている文字列をクロールによって取得することで、コンピュータ関連製品のベンダの名称が得られる。

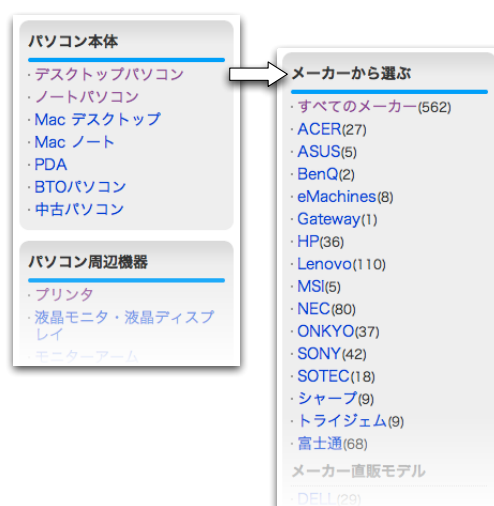


図 5.2: 価格.com におけるベンダ一覧

ここには日本語で表示されるベンダも多く存在する一方、今回対象とするサイトは英語で書かれたものであるため、日本語で書かれたものに関しては英語表記を取得する必要がある。今回は、Wikipedia²のエントリを利用してこれを行った。Wikipedia では、他言語版にも記載があるエントリにはそれぞれの言語版の記事へのアンカーが表示される。これを利用して、Ascii コード外の文字を含む名称を Wikipedia 日本語版において検索し、記事が存在しなおかつ英語版へのアンカーが発見された場合に、そのリンク先の記事のタイトルを英訳として取得した。

ハードウェア名に関しては「パソコン」カテゴリのうちハードウェアに該当するサブカテゴリ、プラットフォーム名に関しては「OS ソフト」サブカテゴリから、商品リストをクロールして候補語を取得した。

Debian Package List³ アプリケーション名の候補語には Debian のパッケージリストを用いた。ここには Debian GNU Linux のパッケージ管理システム “apt” において提供されている全てのパッケージの名前が列挙されている。

Wikipedia 日本語版⁴ プラットフォーム名に関しては、価格.com にリストアップされているものだけではオープンソースソフトウェアの多くが取得できないため、Wikipedia 日本語版の “Linux ディストリビューション” の記事から名称を取得した。

¹<http://kakaku.com/>

²<http://www.wikipedia.org/>

³<http://packages.debian.org/stable/allpackages?format=tst.gz>

⁴<http://ja.wikipedia.org/>



図 5.3: Wikipedia エントリによる英訳

5.1.2 語彙の拡張

上述の候補語は当該クラスの固有表現を含んでいるものの、一般的に文章の中では表記されない文字列を含んでいるものも多い。例えば CPU ベンダの “Intel” は、上述の過程で “Intel Corporation” として取得されるが、実際の文書の中では “Corporation” まで書かれることはまれである。また、アプリケーション名として Debian のパッケージを用いているが、例えばバージョン番号などは省略されることも多い。そこで、これらの省略される可能性のある部分を除いたパターンを作成し、候補語のリストに追加する。ベンダ名の場合、“Inc.” や “Corporation” がこれに該当し、アプリケーション名では数値や記号の出現位置で切断し、先頭から順につないで部分文字列に拡張する。また、スペースで区切られているものに関しては全てスペースで分割している。これにより、例えば “Intel Corporation” は (“Intel”) という 1 語による表現と (“Intel”, “Corporation”) の 2 語からなる表現に拡張され、“apache 2.2” は “apache”、“(“apache”, “2”)”、“(“apache”, “2.2”)” の 3 つのパターンに拡張される。

5.1.3 一般名詞のフィルタリング

語彙の拡張の過程では、各固有名詞の部分文字列が切り出されるため、非常に一般的であるために固有表現の辞書として不適切な語が含まれている可能性が高い。例えばアプリケーション名の中には “audio-tools” から拡張された (“audio”) というパターンが含まれているが、これは明らかに固有表現としての性質を失っており不適切である。

このような語が最終的に利用する辞書に含まれることがないよう、一般的な英英辞典に掲載されている語を除く処理を行った。ここでは、“Longman English Dictionary” のオンラインサービスである “Longman English Dictionary Online⁵” を使い、検索クエリとして各語を送信し、エントリが発見されたものを削除することでこれを実現した。

最終的に作成された辞書のうち、アプリケーション名の辞書の一部を図 5.4 に示す。

5.2 素性への辞書の組み込み


辞書は素性の一部として組み込む。辞書の組み込みを経た素性の全体像は図のようになる。

⁵<http://www.ldoceonline.com/>

5.2. 素性への辞書の組み込み

exim
exim,4
exim,4,base
exim,4,config
exim,4,config,2
exim,4,daemon
exim,4,daemon-heavy

図 5.4: アプリケーション名辞書の一部



入力	位置	n-2	n-1	n	n+1	n+2
	単語	to	uninstall	Editra	in	Xubuntu
	品詞	TO	NP	NP	IN	NP
	見出し語	to	<unknown>	<unknown>	in	<unknown>
	キャピタライズ	0	0	1	0	1
	文字種	0	0	0	0	0
	辞書(V)	O	O	O	O	O
	辞書(H)	O	O	O	O	O
	辞書(P)	O	O	O	O	BP
	辞書(A)	O	O	BA	O	BP
出力	ラベル	O	O	BA	O	BP

図 5.5: 辞書を追加した素性のイメージ図

ここで、辞書によるマッチングは最左最長一致を用いる。つまり、先頭から順に辞書の先頭語に一致する語を見つけ、先頭語を同じくする語が辞書に複数存在した場合、語数が長いものを選択する。これを実現するために、内部のデータ構造としては Trie 木を用いて実装を行った。

5.2.1 Trie 木

Trie 木は文字列をキーとした辞書を表現する際に用いられることの多いデータ構造である。図 5.6 は“AB”、“ABC”、“D”という3つの文字列にそれぞれ100、500、20という値が割り当てられている辞書を表す Trie 木の例である。各々のノードはそれぞれ相異なる文字列と対応しており、ある親ノードに対する子ノードは、親ノードが対応している文字列の末尾に1文字を付加した文字列に対応する。ルートノードは空文字列に対応する。そのため Trie 木においては値を持つノードはツリー中の一部のノードのみであり、キーとして入力された文字列の途中に対応するノードは値を持たない。各ノードは自身が表す文字列に繋がりうる1文字をキー、その1文字が付加された文字列に対応する子ノードを値とする辞書を持っている。

Trie 木は長さ N のキーが $O(N)$ で探索可能であり、またキーの重複部分が共有される分、文字列そのものをキーにするデータ構造と比較して空間計算量が軽減される。反面、キーの一覧を取得す

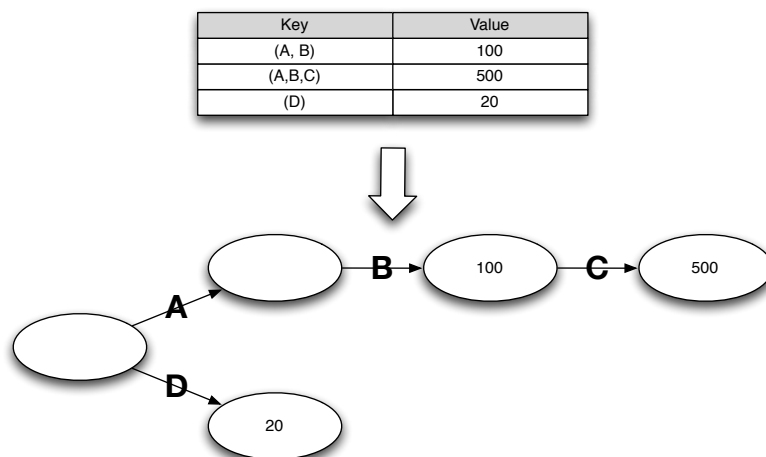


図 5.6: Trie 木の例

るためにはツリー全体を走査しなければならないといった不都合はあるものの、一度作成した後は値を取り出すだけで済むような辞書の実装としては優れたデータ構造である。

ここまで、キーは文字列であるという前提で Trie 木に関して述べてきたが、同値関係が定義されたデータの列であれば何でも Trie のキーとして扱うことが可能であり、今回の実装では単語列をキーとして固有表現辞書の引き当てに利用した。

5.3 実験

辞書を素性に組み込んだ場合の教師データ数に対する抽出精度の推移を図 5.7 に示す。

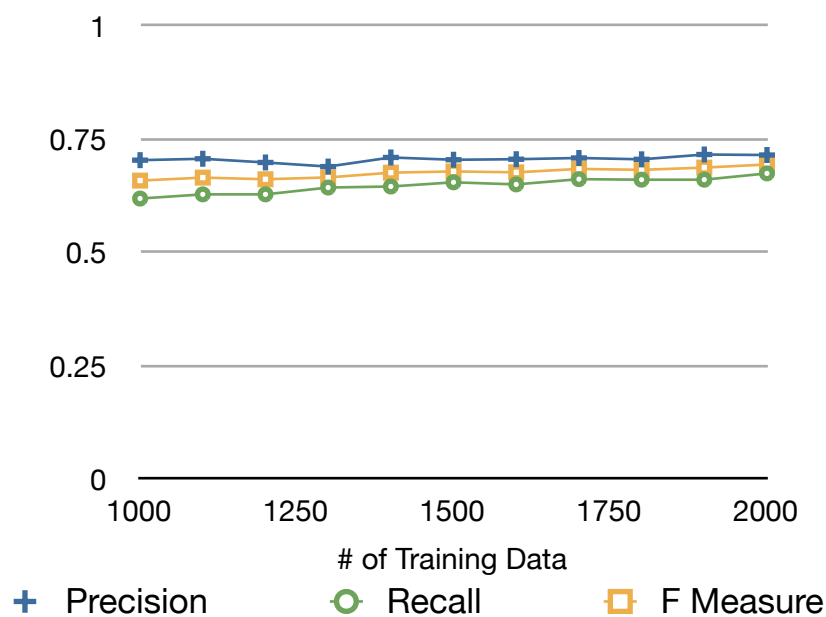


図 5.7: 教師データ数と抽出精度

4 章における実験結果と比較すると、教師データ 1000 件の段階ですでに辞書を用いない場合の教

教師データ 2000 件における結果を Precision、Recall とともに上回っていることがわかる。また、教師データが 2000 件になっても精度の上昇はあまり大きくないことから、この段階でおおよそ学習は収束していると考えられる。

クラスごとの抽出精度は表 6.1 のようになった。

表 5.1: クラスごとの抽出精度

クラス	Precision	Recall	F 値
ベンダ	0.74	0.66	0.69
ハードウェア	0.40	0.34	0.36
プラットフォーム	0.78	0.75	0.76
アプリケーション	0.69	0.65	0.67

表 4.1 の結果と比較すると全てのクラスにおいて結果が改善しており、上昇幅の大きい順にアプリケーション (0.13)、プラットフォーム (0.09)、ハードウェア (0.05)、ベンダ (0.04) であった。

次に、4 章の場合と同様に、いくつかの文字列を “foobarbaz” に置換して、未知固有表現を増加させた場合の精度を図 5.8 に示す。

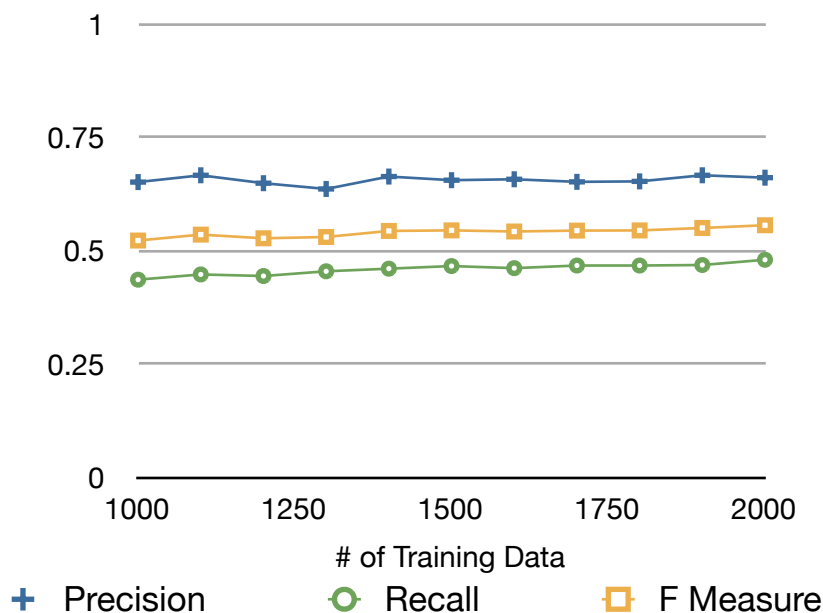


図 5.8: 未知固有表現が増加した場合の抽出精度

4 章の結果と比較して、やはりこれも精度が大幅に上昇している。しかしながら、4 章の場合、2000 件の教師データを用いた場合の F 値は、置換前と置換後の差が 0.12 であったのに対し、ここでのデータでは減少幅が 0.14 に増加している。また、置換された 248 語のうち、正しく抽出できたものは 11 語であった。

5.4 考察

辞書を素性に組み込むことで、全体としての抽出精度は大きく向上している。最も上昇幅が大きかったアプリケーション名に関しては、辞書のデータソースとして用いたものが Linux のパッケー

ジリストであったことが、対象としている文書に対して特に効果的な素性として働いたものと思われる。

一方 4 章において特に精度が低迷していたハードウェア名に関してはあまり大きな改善を得ることができなかった。

これとともに、人工的に与えた未知の固有表現の抽出について見てみると、248 語のうち 11 語と 4 章の 19 語から 4 割以上減少している。

これは、辞書を素性として用いたことによって、未知固有表現の中でも特に辞書にすら出現しない語に対して、誤分類を行う傾向が強まったことが示されていると考えられる。

次章では、辞書素性が加わることによるこの新規の固有表現抽出精度の減少に焦点を当てて、新たな素性を提案しその評価を行う。

第6章 大量文書からの文脈的特徴の取り込み

5章において行った実験から、辞書による素性は本研究が対象とするドメインにおいても非常に有効であることが示された。

しかしながら、教師データのみならず辞書にすら含まれていない語に関しては、抽出精度は向上するどころかむしろ悪化していくことが確認された。本章ではこの点に焦点をあてる。

6.1 未知固有表現の分類

2章で示したように、福島らの研究において未知固有表現という概念が導入され、その抽出精度が論じられたが、実際には未知固有表現には、さらに「学習の時点で存在していたが教師データからこぼれ落ちた表現」と「そもそも学習の時点では存在していなかった表現」の2種が存在することに注目する必要がある。

福島らはこれに対して辞書を用いることで抽出精度の改善を図っているが、5章において示したように、この手法は辞書がその語を含んでいない場合には有効ではなく、むしろ「辞書に載っていない」ことがネガティブな条件として反映され、新語は積極的に排除される傾向にある。つまり、この手法は、未知固有表現はあくまで「教師データにとって」未知であり、辞書の作成時点ではそれを取り得ることが可能であるという前提においてのみ有効なものであるといえる。

しかしながら、本研究が対象とする計算機のトラブルシューティングに関する文書においては、この前提条件には問題がある。

計算機のトラブルシューティングに関する文書に出現する固有表現は、人名や地名と異なりその性質上コンスタントに新語が出現するクラスを含んでいる。そのため、既存手法では頻繁に辞書を更新し、そのたびに学習をし直す必要が生じるが、近年のOSですら半年から一年程度でリリースされるものが存在する状況を鑑みると、一度作成したモデルが有効な期間は非常に短くなるものと考えられる。

また、一般にユーザが何らかのトラブルを経験するのはハードウェアやソフトウェアにおいて何らかの変更があった場合であると考えられ、それは新たな固有表現の出現と機を同じくする可能性が高い。こういった場合に素早く対処できなければ、ユーザにとっての効用は非常に限定的なものになってしまう。よって、本研究の目的に照らして、新語の抽出精度の低下は可能な限り回避すべき問題であると言える。

6.2 素性としての辞書の意味

本節では素性の一つとして組み込まれた場合の辞書の寄与について考察する。

機械学習の観点から見ると辞書は単なる一素性でしかないが、本研究や2章で述べた福島ら、風間らなどの研究で用いられている手法を考えると、Web上の文書をはじめとする大量の文書からの辞書の作成は、そこに存在する文書群から一定のパターンに基づいて荒く固有表現抽出を行った結果を再度学習器に利用させているものと捉えることができる。

この観点で見ると、辞書の素性は単純に分類対象の語の表記のみに着目して分類を行う分類器であり、もともと多義性や文脈を考慮することができない学習器である辞書の出力をその他の情報を

共に利用することができる新たな分類器に入力することで、辞書そのものを用いるよりも高精度な分類を可能としている。

つまり、辞書は図 2.3 の中で、分類対象の語の表記と見出し語のみを素性とした分類器に近い。

そのため、辞書を用いることで、教師データに存在していないものに関してもこの素性の情報がある程度反映されるという意味で辞書素性は有効に働くわけであるが、一方表記以外の情報、つまり先ほど言及した図 2.3 において、 n 語目の表記以外の要素は完全に無視しており、一部の語の抽出精度に悪影響を与えている。

6.3 辞書に対する文脈情報

辞書は教師データからこぼれ落ちたデータのうち、分類対象の表記のみを半機械的に組み込むという点で貢献するものであるが、一般的な分類器の学習に用いられる素性のうち表記のみを対象にしたものであり、局所的な特徴にとどまっているため、新語のように辞書に掲載されていない語に対しては当然ながらこの特徴は有効に機能しない。

一方で、人間は既知のカテゴリの新語を発見した場合、それらが既知のクラスの語であることをある程度推定することが可能であり、この判断に用いられるのは対象の表記ではなく前後の文脈情報であると考えられる。

局所的な特徴を大量に投入する辞書に対して、ある程度機械的なものであっても文脈的な特徴を大量に投入することができれば、辞書によってそこに含まれる語の抽出精度が改善するのと同様に当該クラスの文脈による判断が向上することが期待される。

しかしながら、これまでの学習の素性としてこれを直接組み込むことは不可能である。本研究では、二段階の学習を行うことでこれを実現する。

6.4 辞書を用いた大量の文脈情報の取り込み

本手法の概念図を図 6.1 に示す。

本手法では、これまで用いてきた学習器の前段階の素性の作成において、新たに学習器を導入する。

この学習器によって大量の機械的に生成された文脈的特徴を学習し、その出力を素性の一つとして用いて最終的な分類を行う。

この学習器は、図 2.3 に示した素性のなかから表記および見出し語を無視して学習を行う。これにより、この学習器は表記を全く考慮せず、純粹に単語の前後の文脈のみに特化した分類器となる。

この学習器の教師データは 4 章において取得した大量のテキストデータに対して辞書を用いて作成する。

これによって作成された分類器は、それ自体が一つの固有表現抽出機であり、入力されたトークンの各々に対して固有表現ラベルのいずれかを出力する。

最終的な分類器は、そうして出力されたラベルを一つの素性として判断材料に用いる。

6.5 実験

6.5.1 文脈に特化した分類器

センテンスに対する辞書の適用は、5 章の場合と同様の方法を用いた。ただしここでは複数の辞書から一つのラベルを取得するため、ラベル同士が競合する可能性がある。今回は、重複するラベルに関してはアプリケーション名、プラットフォーム名、ハードウェア名、ベンダ名の順に優先されるようになっている。

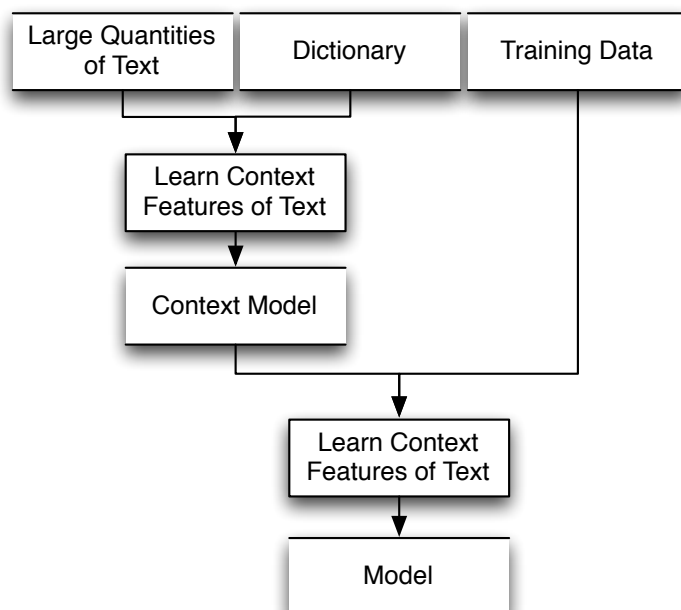


図 6.1: 提案手法の概念図

文脈を学習するための大量のテキストとしては、能動学習の際に取得したテキストデータの中から学習対象とならずに残った 203331 センテンスを用いた。これらに上述の通り辞書を適用したのち、辞書によって付与されたラベルを解として機械学習を行う。これによって各クラスの語に特徴的な文脈をある程度学習することができるものと思われる。

6.5.2 固有表現の抽出

次に、この分類器によって出力されるラベルを本来の分類器に素性として組み込む。最終的に素性は図 6.2 のようになった。

この素性を用いて学習・分類を行った結果が図 6.3 である。

教師データ 2000 件の時に F 値にして 0.70 となり、5 章における結果と比較して 0.01 の向上が見られた。


また、各クラスごとの抽出精度は表 6.1 のようであった。

表 6.1: クラスごとの抽出精度

クラス	Precision	Recall	F 値
ベンダ	0.71	0.63	0.66
ハードウェア	0.43	0.36	0.40
プラットフォーム	0.80	0.77	0.78
アプリケーション	0.68	0.65	0.66

次に、再び一部の語を未知語に置換して、抽出精度を測定した。その結果を 6.4 に示す。

ここで、置換された 248 語のうち、抽出が可能であったのは 14 件であった。



	位置	n-2	n-1	n	n+1	n+2
入力	単語	to	uninstall	Editra	in	Xubuntu
	品詞	TO	NP	NP	IN	NP
	見出し語	to	<unknown>	<unknown>	in	<unknown>
	キャピタライズ	0	0	1	0	1
	文字種	0	0	0	0	0
	辞書(V)	0	0	0	0	0
	辞書(H)	0	0	0	0	0
	辞書(P)	0	0	0	0	BP
	辞書(A)	0	0	BA	0	0
	文脈のみ	0	0	0	0	BP
出力	ラベル	0	0	BA	0	BP

図 6.2: 大量の文脈情報による特徴を追加した素性のイメージ図

6.6 考察

まず、基本的な抽出精度に関しては5章における辞書を用いた手法と比してF値で0.01の改善が見られた。

未知固有表現が増加した場合に関しては、F値で0.01の低下が見られている。しかしながら、未知固有表現の中でも新規の児湯表現の抽出に関しては、248語のうち14件となり、辞書を素性に含めなかった場合には及ばないものの、辞書を加えただけの結果よりは持ち直していると言えるため、この精度低下は未知語が出現したことによってその周囲の語の分類が影響を受けたものであると考えられる。

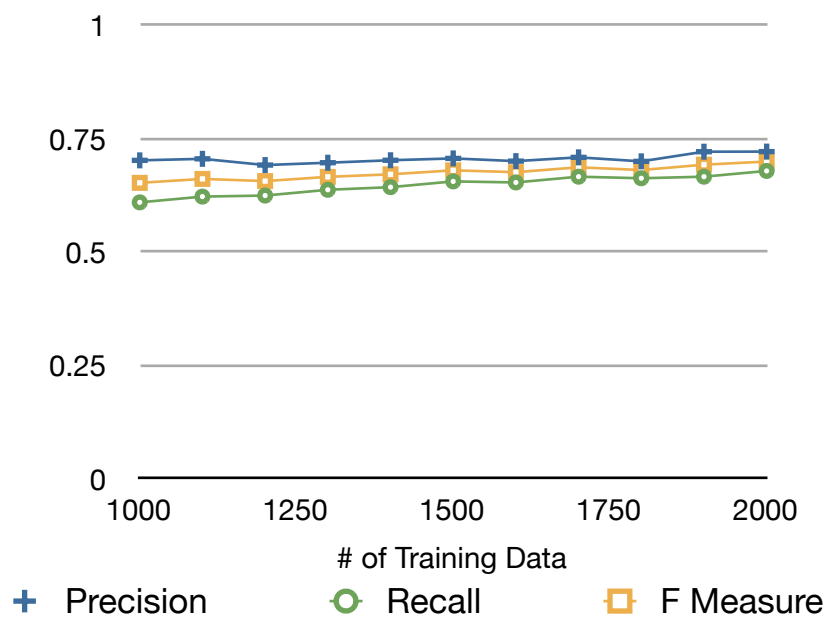


図 6.3: 教師データ数と抽出精度

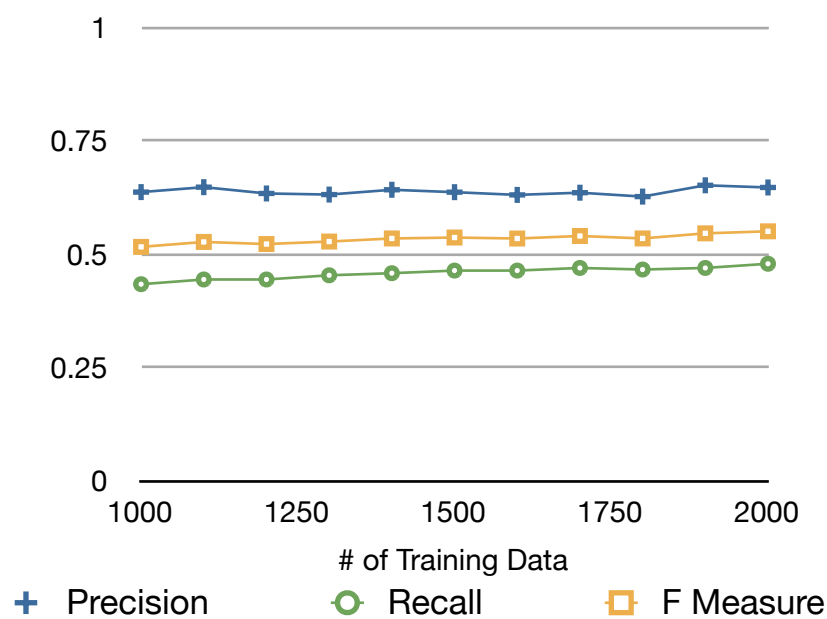


図 6.4: 未知固有表現が増加した場合の抽出精度

第7章 結論

7.1 まとめ

本研究では、計算機に関するトラブルシュート情報を構造化したデータへ変換するための手段としての計算機トラブルシュートドメインにおける固有表現抽出に関する手法を述べた。

能動学習の利用 これまでに教師データの存在しない分野に関する固有表現抽出を行うにあたり、能動学習を用いて教師データ数を抑えて学習を行った。IREX における日本語固有表現抽出タスクでの固有物名の抽出精度がおおよそ 0.5 程度であるのに対し、それを上回る精度を IREX での教師データ数より少ないデータで達成した。

Web から作成した辞書の利用 先行研究の例をもとに、Web 上のデータ資源からある程度のヒューリスティックを用いて大量の辞書データを作成し、それを組成として用いることで抽出精度を大きく改善した。一方、この手法において、辞書にすら出現しない新規の固有表現の抽出精度が悪影響を受けること確認した。

大量の文脈的特徴の取り込み 上述の辞書にすら出現しない語の抽出精度の問題に対処することを目的として、大量の文脈情報を学習の枠組みに導入することを提案した。これによって、辞書が教師データに出現しない表記の抽出を容易にするのと同様に、文脈による判断材料を増加させることで、わずかながら辞書にも出現しない語の抽出精度が向上することが確認された。これによって、新語が日常的に生成される計算機トラブルシュートに関する固有表現の抽出精度が向上することが期待できる。

7.2 今後の課題

今後の課題として、次のような点が挙げられる。

他のデータソースに対する適用 今回の実験はテストデータとしては全て “Linux Forums¹” から取得したテキストデータを用いて行った。本文と関係のないテキストは排除しており、用いているのは純粋に人手で書かれた部分であるためさほど大きな影響はないと思われるが、このフォーラムに特徴的な記述の傾向があった場合、他の文書に対しては抽出精度が低下するおそれがあるため、異なるソースから取得したテキストに対して実験を行い精度を評価する必要がある。

運用のためのシステム作成 本研究において自然言語で書かれたトラブルシュートに関する文書からそのトラブルの構成要素を抽出することが可能となったが、実際にこれを利用するには Web 上のトラブルシュートに関わる文書を収集した上で本手法を適用し、抽出された情報をデータベースに格納する必要がある。

トラブルシュートに関わる文書とそれ以外の文書が混在する文書集合からトラブルシュートに関わる文書を抽出する手法としては、筆者は過去に文書中の構文的な特徴をもとにトラブルシュートらしさを評価する手法を提案しており、それと組み合わせることで Web 上の様々なデータソースから一括して情報抽出を行うことが可能となると考えられる。

¹<http://www.linuxforums.org/>

トラブルシュートに関する Relation Extraction 他の分野における自然言語処理の利用例を見ると、例えば医学生物学分野ではタンパク質名を同定した後それらの関係性を調べ、特定のタンパク質と相互作用する他のタンパク質を検索するといったことが研究対象となっている。トラブルシュートもまた各構成要素がどのように関連を持つかが重要な情報となり得る分野であり、同様の方向で本研究の成果を活用できる可能性がある。

References

- [1] *An analysis of active learning strategies for sequence labeling tasks*, Morristown, NJ, USA, 2008. Association for Computational Linguistics.
- [2] T. Baldwin, D. Martinez, and R.B. Penman. Automatic thread classification for linux user forum information access. In *Proceedings of the Twelfth Australasian Document Computing Symposium (ADCS 2007)*, pp. 72–9, 2007.
- [3] H. Isozaki and H. Kazawa. Efficient support vector classifiers for named entity recognition. In *Proceedings of COLING*, pp. 390–396, 2002.
- [4] Jun’ichi Kazama, Takaki Makino, Yoshihiro Ohta, and Jun’ichi Tsujii. Tuning support vector machines for biomedical named entity recognition. In *Proceedings of the ACL-02 workshop on Natural language processing in the biomedical domain*, pp. 1–8, Morristown, NJ, USA, 2002. Association for Computational Linguistics.
- [5] Jun’ichi Kazama and Kentaro Torisawa. Exploiting wikipedia as external knowledge for named entity recognition. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pp. 698–707, Prague, Czech Republic, June 2007. Association for Computational Linguistics.
- [6] H. Schmid. Probabilistic part-of-speech tagging using decision trees. In *Proceedings of International Conference on New Methods in Language Processing*, Vol. 12. Manchester, UK, 1994.
- [7] S. Sekine, K. Sudo, and C. Nobata. Extended named entity hierarchy. In *Proceedings of the LREC-2002 Conference*, pp. 1818–1824. Citeseer, 2002.
- [8] Satoshi Sekine. On-demand information extraction. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pp. 731–738, Morristown, NJ, USA, 2006. Association for Computational Linguistics.
- [9] Beth M. Sundheim. Overview of the third message understanding evaluation and conference. In *MUC3 ’91: Proceedings of the 3rd conference on Message understanding*, pp. 3–16, Morristown, NJ, USA, 1991. Association for Computational Linguistics.
- [10] H.M. Tran, G. Chulkov, and J. Schonwalder. Crawling bug tracker for semantic bug search. In *Proceedings of the 19th IFIP/IEEE international workshop on Distributed Systems: Operations and Management: Managing Large-Scale Service Deployment*, pp. 55–68. Springer, 2008.
- [11] Sholom M. Weiss, Nitin Indurkha, Tong Zhang, and Fred Damerau. *Text Mining: Predictive Methods for Analyzing Unstructured Information*. Springer, 1 edition, 10 2004.
- [12] 工藤拓. Crf++: Yet another crf toolkit. <http://crfpp.sourceforge.net/>.

- [13] 福島健一, 鍛冶伸裕, 喜連川優. 日本語固有表現抽出における超大規模ウェブテキストの利用. 電子情報通信学会第 19 回データ工学ワークショップ/第 6 回日本データベース学会年次大会 (DEWS2008), 2008.

謝辞

本研究を進めるにあたり、大変多くの方にお世話になりました。

近山隆教授には、論文作成やプレゼンテーションなど要所での的確なアドバイスをいただきました。

田浦健次朗准教授には、研究の方針に関して熱心にご指導いただきました。特に進学直後研究の行方に戸惑う頃には一方ならぬお世話になりました。

現在喜連川研究室所属の横山大作さん、ポスドクの柴田剛志さん、現在辻井研究室所属の三輪誠さんには、ミーティングなどにおいてより身近な立場で様々なご助言を頂き、研究を進めることができました。

先輩である斎藤秀雄さん、弘中健さん、安井雅章さんには、研究を進めるに当たって必要な技術を教えていただき、また良き相談相手ともなっていただきました。特に安井さんはあるときは頼れる先輩として、またあるときはともに修了を目指す同士として、その存在に何度助けられたか分かりません。

ともに修士の2年間、あるいは学部からの3年間を過ごした研究室の同期のみなさん、また後輩のみなさんには公私ともに様々にお世話になりました。

ここに、心より感謝の意を表します。