

博士論文

機械学習に基づくソフトウェア不具合修正作業  
の効率化に関する研究

Improving the efficiency of software defect fixing based on machine learning

九州工業大学大学院  
工学府 工学専攻 電子システム工学コース

平川 凜

Rin Hirakawa

2022年2月

## 摘要

本論文は、オープンソースソフトウェア（OSS : Open Source Software）を用いた大規模ソフトウェア開発の不具合修正作業の効率化を目的として、不具合票の自動分類システムによるバグトリアージの効率化手法、およびログ異常検出によるバグ解析の効率化手法を提案する。

### (1) 不具合票の自動分類システムによるバグトリアージの効率化

OSS を利用したソフトウェア開発では、報告された不具合に対して適切な担当カテゴリへの分類を素早く行うことが求められる。本論文では、不具合報告のコメントを転移学習した BERT モデルによって担当カテゴリへの分類を行う手法を提案した。OSS プロジェクトの不具合報告を対象とした実験の結果、約 87% の精度、約 86% の再現率で不具合を各製品のカテゴリに分類することができた。また、BERT モデルの自己注意層の可視化では、モデルが製品に関連する語彙に対して強い注意を向けて判断を行っていることが示唆された。

### (2) ログ異常検出によるバグ解析の効率化

近年、ソフトウェアシステムの複雑化・大規模化に伴い、実行時に収集されるログデータが膨大になり、その把握が困難になっている。不具合修正作業では、大量のログデータから不具合原因の手がかりとなる部分のみを抽出することで作業時間の大幅な短縮が期待される。本研究では、システムが正常に動作している際のログのパターンを機械学習手法に学習させ、テストデータから異常なパターンのログメッセージを抽出する方法を提案する。

ログメッセージの分散表現に基づく異常検出手法では、提案手法はベースラインを上回る AUC 値を達成し、再現率において約 7 % の改善がみられた。

また、ログの時間パターンに基づく異常検出手法では、提案手法はベースライン手法と比較して、学習データ量の変化にロバストな異常検出を行えることが分かった。空間プーリングによる大規模ログの異常検出では、提案手法は学習データ量が極端に少ない場合には深層教師あり学習手法を上回る精度を持ち、データ量の変化に対しても高い水準の検出精度を維持できることが分かった。

# 目次

第1章 序論 .....	1
1.1 研究背景.....	1
1.1.1 不具合の管理.....	2
1.1.2 不具合修正のプロセス.....	3
1.1.3 不具合修正における課題 .....	4
1.2 関連研究.....	5
1.2.1 バグトリアージの効率化 .....	5
1.2.2 実行ログの異常検出 .....	7
1.3 研究目的.....	23
1.4 論文の構成 .....	24
第2章 深層自然言語処理モデルに基づく不具合票の分類と可視化.....	25
2.1 はじめに.....	25
2.2 学習済み BERT モデルによる不具合票自動分類システム .....	25
2.2.1 提案システムの概要 .....	26
2.2.2 Transformer.....	27
2.2.3 BERT .....	33
2.3 実験.....	37
2.3.1 データセット .....	37
2.3.2 実験方法 .....	37
2.3.3 実験結果 .....	40
2.4 結論.....	44

第3章 分散表現のワンクラス学習に基づくログデータの異常検出.....	45
3.1 はじめに.....	45
3.2 Transformer エンコーダ表現のワンクラス学習.....	45
3.2.1 One-Class Neural Networks .....	46
3.2.1.1 One-Class SVM.....	46
3.2.1.2 OC-NNへの拡張 .....	47
3.2.2 Text Autoencoder .....	47
3.2.3 モデルアーキテクチャ.....	48
3.3 実験.....	49
3.3.1 データセット .....	49
3.3.2 実験方法 .....	50
3.3.3 実験結果 .....	53
3.4 結論.....	54
第4章 時間記憶に基づくログデータの異常検出.....	55
4.1 はじめに.....	55
4.2 提案手法の概要 .....	55
4.3 ログのエンコーディング処理.....	56
4.4 HTM アルゴリズム .....	58
4.4.1 HTM の入力表現 .....	58
4.4.2 HTM のアーキテクチャ .....	59
4.4.3 HTM を用いた異常検出.....	61
4.5 Temporal Pooling (時間記憶) .....	62

4.5.1	概要	62
4.5.2	動作原理	63
4.5.3	学習ルールの定式化	65
4.5.4	異常度の算出	68
4.6	実験	69
4.6.1	データセット	69
4.6.2	実験方法	70
4.6.3	実験結果	71
4.6.4	パラメータ設定の影響	73
4.7	結論	78
第5章 空間プーリングに基づくログデータの異常検出		79
5.1	はじめに	79
5.2	提案手法の概要	79
5.2.1	特徴量変換	80
5.2.2	空間プーリング	81
5.2.3	分類器	85
5.3	実験	86
5.3.1	データセット	86
5.3.2	実験方法	86
5.3.3	実験結果	89
5.3.4	学習データ量の影響	90
5.4	結論	92

第6章 結論 .....	93
謝辞	95
参考文献	96
業績一覧	104

## 図目次

図 1.1 BTS を用いた不具合修正プロセス .....	3
図 1.2 ログ解析の例 .....	8
図 1.3 プログラム実行フローの例 .....	17
図 1.4 決定木の構造.....	20
図 2.1 提案する不具合票自動分類システムのフロー .....	26
図 2.2 Transformer のモデル構造 .....	28
図 2.3 Scaled Dot-Product Attention と Multi-Head Attention .....	30
図 2.4 BERT モデルの事前学習 .....	33
図 2.5 BERT モデルのファインチューニング（分類タスク） .....	34
図 2.6 入力トークンに対する埋め込みの計算方法.....	35
図 2.7 Eclipse におけるカテゴリ分類結果の混同行列.....	41
図 2.8 Apache におけるカテゴリ分類結果の混同行列 .....	41
図 2.9 Apache プロジェクトにおける Self-attention の可視化結果.....	44
図 3.1 提案するログ異常度算出モデルの構造 .....	48
図 4.1 時間記憶によるログ異常検出手法の概要.....	56
図 4.2 ログエントリのテンプレート ID 変換 .....	57
図 4.3 RDSE の python 疑似コード .....	58
図 4.4 HTM ニューロンモデル .....	60
図 4.5 HTM 時間記憶 .....	63
図 4.6 Temporal Pooling の次時刻入力の予測 .....	68
図 4.7 各学習データ量に対する Precision の評価結果(0.1~20%) .....	72
図 4.8 各学習データ量に対する Recall の評価結果(0.1~20%) .....	72
図 4.9 学習データ量ごとの F-measure の評価結果(0.1~20%) .....	73
図 4.10 各モデルサイズ (500~2048cols) における Precision の評価結果 .....	74
図 4.11 各モデルサイズにおける Recall の評価結果 (500~2048cols) .....	74
図 4.12 各モデルサイズでの F-measure の評価結果(500~2048cols) .....	75
図 4.13 各スパース性(0.02~0.22)における Precision の評価結果.....	76
図 4.14 各スパース性(0.02~0.22)における Recall の評価結果 .....	76

図 4.15 各スパース性における F-measure の評価結果 (0.02~0.22) .....	77
図 4.16 各スパース性における Average Precision の評価結果 .....	78
図 5.1 提案手法のパイプライン .....	80
図 5.2 符号化された画像の空間プーリング .....	85
図 5.3 異常検知のベンチマーク結果 .....	89
図 5.4 訓練サブセットを用いたベンチマークの結果 (Precision) .....	90
図 5.5 訓練サブセットを用いたベンチマークの結果 (Recall) .....	91
図 5.6 訓練サブセットを用いたベンチマークの結果 (F-measure) .....	91

## 表目次

表 1.1 不具合票に記載される各情報の詳細 .....	2
表 1.2 BTS を用いた不具合修正プロセス .....	4
表 1.3 代表的なログパーサと詳細 .....	9
表 1.4 loglizer に実装されている機械学習モデルとその詳細 .....	12
表 1.5 Deep-loglizer に実装されている深層学習モデルとその詳細 .....	21
表 2.1 不具合事象の記述の統計情報 (Eclipse) .....	38
表 2.2 不具合事象の記述の統計情報 (Apache) .....	38
表 2.3 Eclipse プロジェクトにおける不具合レポートデータの内訳 .....	38
表 2.4 Apache プロジェクトにおける不具合レポートデータの内訳 .....	39
表 2.5 フайнチューニング時のパラメータ .....	39
表 2.6 Eclipse における分類スコア .....	41
表 2.7 Eclipse プロジェクトにおいてランダムに抽出した誤分類データ .....	42
表 3.1 KERNEL コンポーネントログの正常・異常データ数の内訳 .....	49
表 3.2 順伝播ネットワークの構成 .....	50
表 3.3 提案手法のハイパープラメータ .....	51
表 3.4 Text Autoencoder のハイパープラメータ .....	51
表 3.5 AUC 値の比較 .....	53
表 3.6 提案手法の F-measure .....	53
表 3.7 Text Autoencoder の F-measure .....	53
表 4.1 HDFS データセットの内訳 .....	70
表 4.2 Random Distributed Scalar Encoder のパラメータ .....	70
表 4.3 TM-LAD モデルのハイパープラメータ .....	71
表 5.1 BGL データセットの内訳 .....	86
表 5.2 SPClassifier のパラメータリスト .....	87

# 第1章 序論

## 1.1 研究背景

OSS(Open Source Software)はソースコードが一般に公開されており、商用・非商用を問わず自由に利用・改変・再頒布が可能なため、多くのソフトウェア企業等で利用されている。OSS の利活用領域は幅広く、OS から業務アプリケーション、データベース、ビッグデータ、AI（人工知能）等にも利用されている[1]。Linux システムや汎用ライブラリを中心として、商用製品・サービスにも OSS は積極的に採用されており、今や OSS を用いずに製品・サービスを構築することは困難ともいえる状況である。

OSS を活用することで企業は、以下のようなメリットを享受できる[1]。

- **開発の効率化による開発費の抑制・開発期間短縮**

開発者が必要とする機能が既に OSS に実装されている場合、開発費用の抑制や開発期間の短縮が期待できる。環境の変化の激しいICT関連産業においては、開発を効率化することで非常に大きなアドバンテージが得られる。

- **高い安定性、品質、透明性の確保**

OSS は不特定多数の使用者による改良が重ねられており、ソフトウェアとして高い安定性や品質を有している。多数のユーザによる利用実績の積み重ねもあるため、開発者は安心して使用することができる。また、ソースコードが公開されており、不正なプログラムや脆弱性などを常に確認できる透明性も持ち合わせている。

- **豊富な種類による新たな価値創出**

現在では OSS の種類が非常に豊富になっており、多くの機能が利用可能となっている。AI 等の最新の技術が実装される等、これまでにない新しい価値を創出し、業務を大幅に効率化できる可能性がある。

一方で、以下 2 つの要因による不具合が発生しやすいというデメリットもある。

- OSS の内部に潜在している事象の顕在化
- OSS の適切な利用方法や改変方法における誤り

OSS の開発規模が拡大するにつれてソフトウェアの機能も複雑になり、報告される不具合の数も膨大となる。そのため、不具合修正を行なうための時間は開発プロジェクトに負担を与える要因となっている。

### 1.1.1 不具合の管理

一般的な OSS 開発では、開発者同士が不具合に関する情報を共有するために、Bugzilla[2]、Trac[3]、Redmine[4]などの不具合管理システム(BTS: Bug Tracking System)が利用されている。BTS は、開発者またはユーザから報告された 1 つの不具合に対して 1 つの不具合票を作成し、個々の不具合修正の進捗を追跡するためのシステムである。各不具合票には、不具合の①基本情報、②不具合の詳細情報、③議論情報、④状態遷移管理情報の 4 種類の情報が記録される[5]。それぞれの詳細を表 1.1 に示す。

BTS を用いた不具合修正では、まず OSS に不具合を発見した開発者やユーザが不具合票の「基本情報」と「詳細情報」に不具合に関する情報を記載する。開発者は報告者に対して不具合に関する追加情報の要求、類似・重複する不具合票の提示、修正に関するアドバイスなどをする場合にコメントを書き込む。プロジェクト管理者は、報告された不具合を修正すべきかどうかを審査する。必要な場合には開発者が不具合の修正作業を行い、作業終了後に不具合が解決したことを記録する。また、修正した後には正しく修正されたかどうかをプロジェクト管理者が審査する。

表 1.1 不具合票に記載される各情報の詳細

情報	内容
基本情報	対象プロダクトやバージョン、重要度や優先度など
詳細情報	不具合の内容や不具合を再現するための手順
議論情報	不具合の内容や修正作業に関して行われる議論
状態遷移管理情報	不具合修正処理の状態管理を行うために記録される情報

### 1.1.2 不具合修正のプロセス

不具合修正プロセスは、不具合がプロジェクトに報告されてから正しく修正されたことが確認されるまでの一連の作業である。図 1.1 に BTS を用いた不具合修正プロセスの流れを示す。不具合修正プロセスは、具体的な修正に向けた活動を開始する時期、修正作業が完了し修正内容の妥当性の検証を開始する時期を境に、3 つの区間（修正計画フェーズ、修正フェーズ、検証フェーズ）に分けることができる。表 1.2 に各区間における状態とその詳細を示す。

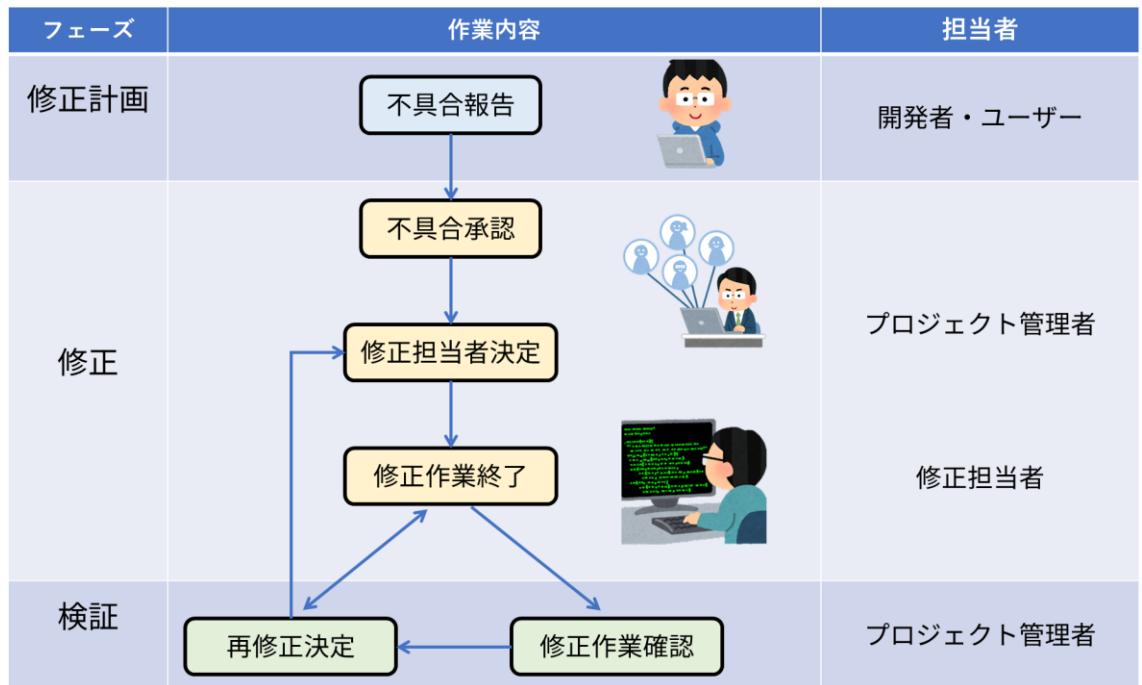


図 1.1 BTS を用いた不具合修正プロセス

表 1.2 BTS を用いた不具合修正プロセス

区間	状態	詳細
修正計画	不具合報告	開発者・ユーザによって発見された不具合が BTS に登録された状態
修正	議論開始	登録された不具合情報に関して議論が開始された状態
	不具合承認	議論の結果、プロジェクト管理者によって修正すべき不具合として承認された状態
	修正担当者決定	プロジェクト管理者によって修正担当者が決定された状態
	修正作業終了	修正担当者によって修正作業が終了した状態
検証	再修正決定	不具合の修正が不十分であり、再度修正が要求された状態
	修正作業確認	プロジェクト管理者によって不具合が正しく修正されたことが確認された状態

### 1.1.3 不具合修正における課題

実際の OSS 開発プロジェクト（Mozilla Firefox、Apache HTTP Server、Eclipse Platform）におけるケーススタディでは、各フェーズにおいて以下のような課題が報告されている[5], [6].

- 修正計画フェーズ

報告者が不具合票に記載する内容と開発者が求める情報が異なる場合、追加情報を求めるため多くの不具合票が議論開始の状態に移行する。

- 修正フェーズ

プロジェクト管理者が適切な修正担当者を決定することが難しく、繰り返し修正担当者が変更されるため時間を要する。また、不具合が複雑なために修正が順調に進まないことがある。

- 検証フェーズ

不具合の修正確認後、再修正の判断が下されるまでに時間がかかり、以前の修正内容を思い出すことが困難となる。これにより、再修正を要する不具合の発見の遅れが修正時間の長期化につながる。

修正に要する時間の長期化は製品のリリース遅延に繋がるため、各フェーズにおいて効率化を行うことで時間の短縮を図る必要がある。

## 1.2 関連研究

本研究では、不具合修正プロセスのなかでも特に修正フェーズにおける作業効率化に焦点をあてる。以下で不具合修正効率化における関連研究について述べる。

### 1.2.1 バグトリアージの効率化

不具合管理システムに報告された各不具合に対して開発者に修正タスクを割り当てるなどをバグトリアージと呼ぶ[7]。人手によるバグトリアージには限界があることが知られており、実際に大規模オープンソースプロジェクトの Eclipse と Mozilla では報告された不具合の 37~44%が他の開発者に再割り当てされている[8]。不具合の再割り当ては修正作業を遅らせるため、バグトリアージを支援するための研究が盛んに行われている。

柏ら[7]は、不具合の割り当てを開発者と不具合の組み合わせ問題としてとらえ、個々の開発者のタスク量に制約条件を課することでタスク割り当てを最適化している。組み合わせ問題はナップサック問題とみなされ、ナップサックをプロジェクト全体の修正能力の上限、アイテムを不具合、重みを不具合の修正時間、利得を不具合に対する開発者の適正（プリファレンス）として考えて 0-1 整数計画法を用いて解が求められる。タスク量を考慮した割り当てを行うことで、ごく一部の優秀な開発者にタスク割り当てが集中することを防ぐことができる。

機械学習に基づく方法では、主に SVM (Support Vector Machine) などの分類アルゴリズムを使用して開発者の推薦を行うことが多い。Anvik[9], [10]らは、不具合票のタイトルと本文フルテキスト中の単語の頻度をもとに特徴ベクトルを作成し、SVM を用いて開発者がどの報告を解決するかを決定している。前処理ではストップワードと非アルファベットのトークンが除去され、作成された特徴ベクトルは文書の長さ、単語の文書内頻度、文書間頻度に基づいて正規化が行われる。

上記の手法では、不具合票のコメントに対して十分な情報を利用できないという問題がある。例えば、割り当てを組み合わせ問題としてとらえる方法では、不具合票のコンポーネントと優先度の情報を使用しているが、不具合事象の内容に関する記述については考慮されていない。また、単語頻度を用いた特徴ベクトルに基づく方法では、前処理でバージョンなどの数値情報が欠落し、語順などの情報も失われる。OSS を利用したソフトウェア開発では複数の OSS が関わるなど複雑な不具合も多く、不具合現

象を記述したコメントからより多くの情報を考慮した特徴量にもとづくバグトリアージ手法が求められる。

### 1.2.2 実行ログの異常検出

製品に OSS を採用することにより機能実装の速度が格段に速くなる一方、システムに不具合が発生した際の解析の難易度が非常に高くなるという問題がある。上記のような不具合の原因を調査する際には、システム実行時のログをエンジニアが目視で確認していく必要がある。

従来のログ解析では”kill”, “exception”などの単純な単語検索や正規表現を使用して不具合に関係するログを調査することが一般的であった[11]。しかしながら、ログデータに致命的でない”exception”などの情報が含まれていることも多い。現代ではシステムの大規模化・複雑化によりログの量が爆発的に増加しており、このような手動のログ検査方法は非現実的になりつつある。上記の背景から、ログデータから不具合に関係する部分のみを抽出する自動ログ異常検出手法の研究が行われている。

#### 1.2.2.1 ログパーサ

ログは、ソフトウェアシステムの開発と保守において重要な役割を果たしている。システムの実行時の詳細な情報をログに記録することで、開発者やサポートエンジニアはシステムの動作を理解し、発生する可能性のある問題を追跡することができる。豊富な情報とログの普及により、利用統計の分析、アプリケーションセキュリティの確保、パフォーマンスの異常の特定、エラー やクラッシュの診断など、さまざまなシステム管理や診断作業が可能である[12]。

その一方で、ログデータをいかに効果的に分析するかが大きな課題となっている。現代のソフトウェアシステムは日常的に大量のログを生成しており、例えば商用クラウドアプリケーションでは 1 時間に約数ギガバイトのデータが生成される。また、ログメッセージは本質的に構造化されていないため、開発者は利便性と柔軟性のためにフリー テキスト形式でシステムイベントを記録する。これらの要因は、ログデータの自動分析の難易度をさらに高めている。最近の研究や産業用ソリューションは、強力なテキスト検索と機械学習ベースの分析機能を提供するために進化してきた。このようなログ分析を可能にするために、フリー テキストの生ログメッセージを構造化されたイベントのストリームに解析するプロセスが必要となる。

## ロギングコード

```
LOG.info("Received block " + block + " of size "
+ block.getNumBytes() + " from " + inAddr);
```

## ログメッセージ

```
2015-10-18 18:05:29,570 INFO dfs.DataNode$PacketResponder: Received
block blk_-562725280853087685 of size 67108864 from /10.251.91.84
```

## 構造化ログ

<b>TIMESTAMP</b>	2015-10-18 18:05:29,570
<b>LEVEL</b>	INFO
<b>COMPONENT</b>	dfs.DataNode\$PacketResponder
<b>EVENT TEMPLATE</b>	Received block <*> of size <*> from /<*>
<b>PARAMETERS</b>	[“blk_-562725280853087685”, “67108864”, “10.251.91.84”]

図 1.2 ログ解析の例

生のログデータはテンプレートとよばれる文字列のあらかじめ決められた位置に変数（パラメータ）を埋め込んだ状態で生成される。この状態ではテンプレートとパラメータを表現する文字列が同質に扱われてしまうため、ログの構造化とよばれる処理によってこれらを分離する必要がある（図 1.2）。ログデータの構造化は一般的にログパーサとよばれるアルゴリズムを用いて行われることが多い。

ログパーサはオフラインとオンラインの 2 つの主要なモードに分類できる。オフラインログパーサはバッチ処理の一種であり、解析する前にすべてのログデータが利用可能である必要がある。これに対して、オンラインログパーサはログメッセージを 1 つずつストリーミングで処理するため、ログがストリームとして収集される場合により実用的である。表 1.3 に、代表的なログパーサとそこで使用されている技術の詳細を示す。

表 1.3 代表的なログパーサと詳細

ログパーサ	使用されている技術	モード
SLCT	頻出パターンマイニング	オフライン
LFA		
LogCluster		
LKE	クラスタリング	オフライン
LogSig		
LogMine		オンライン
SHISO		
LenMa		
AEL	ヒューリスティック	オフライン
IPLoM	反復パーティショニング	オフライン
Drain	解析木	オンライン
Spell	最長共通部分列	オンライン
MoLFI	進化的アルゴリズム	オフライン

これらのログパーサで使用されている技術を、以下で詳しく説明する。

### ● 頻出パターンマイニング

頻出パターンとは、データセットの中で頻繁に発生する項目の集合のことである。同様に、イベントテンプレートはログで頻繁に発生する一定のトークンの集合として見ることができる。したがって、頻出パターンマイニングはログの自動解析を行うための簡単なアプローチである。例として、SLCT[13]、LFA [14]、LogCluster[15]がある。これら 3 つのログパーサはすべてオフライン手法であり、同様の構文解析手順に従う。その手順は、①ログデータを数回のパスで巡回する、②各巡回で頻出アイテムセット(トークン、トークン位置のペアなど)を構築する、③ログメッセージを複数のクラスタにグループ化する、④各クラスタからイベントテンプレートを抽出する、というものである。SLCT は、ログ解析に頻出パターンマイニングを適用した最初の研究である。さらに、LFA はログデータ全体ではなく、各ログメッセージのトークン頻度分布を考慮し、稀なログメッセージを解析する。LogCluster は SLCT を拡張したものであり、トークンの位置の変化にもロバストに対応することができる。

## ● クラスタリング

イベントテンプレートは、ログメッセージのグループの自然なパターンを形成する。この観点から、ログ解析はログメッセージのクラスタリング問題としてモデル化することができる。ログ解析にクラスタリングアルゴリズムを適用した例として、3つのオフライン手法(LKE[16]、LogSig [17]、LogMine[18])と 2 つのオンライン手法(SHISO [19]、LenMa[20])がある。具体的には、LKE はペアワイズしたログメッセージ間の重み付き編集距離に基づく階層的クラスタリングアルゴリズムを採用している。LogSig は、ログメッセージを事前に定義された数のクラスタにクラスタ化するためのメッセージシグネチャベースのアルゴリズムである。LogMine は、ログメッセージを下から上に向かってクラスタにグループ化する階層的なクラスタリング手法でイベントテンプレートを生成することができる。SHISO と LenMa はどちらもオンライン手法で、同様のストリーミング方式でログを解析する。新しく入力される 各ログメッセージについて、パーサはまず、既存のログクラスタの代表的なイベントテンプレートとの類似性を計算する。ログメッセージが正常に一致した場合は既存のクラスタに追加され、そうでない場合は新しいログクラスタが作成される。続いて、対応するイベントテンプレートがそれに応じて更新される。

## ● ヒューリスティック

一般的なテキストデータとは異なり、ログメッセージはいくつかのユニークな特徴を持っている。そのため、いくつかの研究(AEL [21]、IPLoM [22]、Drain[23]など)では、ヒューリスティックに基づいたログ解析手法が提案されている。具体的には、AEL では、定数トークンと変数トークンの出現回数を比較することで、ログメッセージを複数のグループに分離している。IPLoM は反復的なパーティショニング戦略を採用しており、ログメッセージをメッセージの長さ、トークンの位置、マッピング関係によってグループに分割する。Drain はログメッセージを表現するために固定深度木構造を適用し、共通のテンプレートを効率的に抽出する。これらのヒューリスティックはログの特性を利用しておおり、多くの場合非常に優れた性能を発揮する。

### ● その他

上記以外にも、いくつかの方法が存在する。例えば、Spell[24], [25]はストリーム的にログを解析するために 最長共通部分行列アルゴリズムを利用している。Messaoudi ら[26]は、ログ解析を多目的最適化問題としてモデル化し、進化的アルゴリズムを用いて解く MoLFI を提案している。

### 1.2.2.2 古典的機械学習手法によるログ異常検出

機械学習を用いたログデータの異常検出手法は、教師あり学習と教師なし学習の 2 種類がある。またその中には、基本的な機械学習モデルを利用したものや、ログデータのヒューリスティックを活用した独自のアルゴリズム、深層学習技術を利用した手法などが存在する。

Logpai プロジェクト[27]では産業用システムのログに利用可能な機械学習モデルによる異常検出手法が整理されている。これらの手法では統一された入力形式として各ログテンプレートの出現回数をカウントしたイベントカウント行列が採用されている。

Loglizer[11], [28]は自動化された異常検出のための機械学習ベースのログ分析ツールキットであり、7つの基本的な機械学習モデルによるベンチマークを行うことができる。ツールには教師あり異常検出 (LR[29]、Decision Tree[30]、SVM[31]) と教師なし異常検出 (LOF[32]、One-Class SVM[33]、Isolation Forest[34]、PCA[35]、Invariants Mining[36]、LogClustering[37]) が含まれており、統一された入出力形式でログデータの異常検出が可能である（表 1.4）。公開以後、より高精度な手法が数多く提案されているものの、種々の軽量な機械学習モデルをシームレスに試用することができる。

表 1.4 loglizer に実装されている機械学習モデルとその詳細

学習方法	モデル
教師なし	LOF
	One-Class SVM
	Isolation Forest
	PCA
	Invariants Mining
	LogClustering
教師あり	LR
	Decision Tree
	SVM

## ● 特微量変換

logizer では、全ての機械学習モデルに対して統一された特微量抽出ステップが適用される。特微量抽出における入力はログパーサによって生成されたログイベントであり、出力はイベントカウント行列となる[11]。特徴を抽出する際には、まずログデータをグルーピングする必要がある。そのために、固定ウィンドウ、スライディングウィンドウ、セッションウィンドウの 3 つのタイプの窓を使用してログデータを分割する。

### ① 固定ウィンドウ

固定ウィンドウとスライディングウィンドウは、それぞれのログの発生時間を記録するタイムスタンプに基づいている。各固定ウィンドウにはサイズがあり、時間区間または時間の持続時間を意味する。ウィンドウサイズは、1 時間や 1 日などの一定の値である。固定ウィンドウの数は、予め定められたウィンドウサイズに依存する。同一ウィンドウ内で発生したログは、1 つのログシーケンスとみなす。

### ② スライディングウィンドウ

固定窓と異なり、スライディングウィンドウは窓の大きさとステップの大きさ(例: 5 分ごとにスライドする 1 時間の窓) の 2 つの属性で構成されている。一般に、ステップサイズはウィンドウサイズよりも小さいため、異なるウィンドウの重なりが発生する。スライディングウィンドウの数は固定ウィンドウよりも多くなることが多く、主にウィンドウサイズとステップサイズの両方に依存する。また、同じスライディングウィンドウで発生したログは、窓が重複しているため複数のスライドウィンドウで重複することがあるが、ログシーケンスとしてグループ化されている。

### ③ セッションウィンドウ

上記の 2 つのウィンドウタイプと比較して、セッションウィンドウはタイムスタンプの代わりに識別子に基づいている。識別子は、いくつかのログデータの異なる実行パスをマークするために利用される。例えば、block\_id を持つ HDFS[38] のログには、特定のブロックの割り当て、書き込み、複製、削除が記録されている。このように、各セッションウィンドウが一意の識別子を持つように、識別子に応じてログをグループ化することができる。

ウィンドウを用いてログシーケンスを構築した後、イベントカウント行列 $X$ を生成する。各ログシーケンスにおいて、各ログイベントの発生数をカウントしてイベントカウントベクトルを形成する。例えば、イベントカウントベクトルが $[0,0,2,3,0,1,0]$ であれば、このログシーケンスの中でイベント3が2回発生し、イベント4が3回発生したことを意味する。最後に、複数のイベントカウントベクトルは、イベントカウント行列 $X$ となるように構築され、エントリ $X_{i,j}$ は、 $i$ 番目のログシーケンスでイベント $j$ が何回発生したかを記録する。

- 各機械学習モデルの詳細

- (a) 教師なし学習モデル

教師あり学習とは異なり、教師なし学習もまた一般的な機械学習タスクだが、その学習データにはラベルが付けられていない。教師なし手法はラベルがないため、実世界の生産環境での適用性が高い。一般的な教師なし学習手法には、様々なクラスタリング手法、関連ルールマイニング、PCAなどがある。

### ① LogCluster

Lin ら[37]は、オンラインシステムの問題点を特定するために LogCluster と呼ばれるクラスタリングベースの手法を設計している。LogCluster は、知識ベースの初期化段階とオンライン学習段階の 2 つの学習段階を必要とする。

知識ベースの初期化段階では、ログベクトル化、ログクラスタリング、代表ベクトル抽出の 3 つのステップがある。まず、ログシーケンスをイベントカウントベクトルとしてベクトル化し、これを逆文書頻度 (IDF) と正規化によってさらに修正する。第二に、LogCluster は、正常イベント数と異常イベント数のベクトルを別々に凝集型階層クラスタリングによりクラスタリングし、2 組のベクトルクラスタ（正常クラスタと異常クラスタ）を知識ベースとして生成する。最後に、そのセントロイドを計算することで、各クラスタの代表的なベクトルを選択する。

オンライン学習フェーズは、知識ベース初期化フェーズで構築されたクラスタをさらに調整するために使用される。オンライン学習フェーズでは、イベントカウントベクトルを一つずつ知識ベースに追加していく。イベントカウントベクトルが与えられると、それと既存の代表ベクトルとの間の距離が計算される。最小の距離が閾値よりも小さい場合、このイベントカウントベクトルは最も近いクラスタに追加され、このクラスタの代表ベクトルが更新される。そうでなければ、LogCluster はこのイベントカウントベクトルを使用して新しいクラスタを作成する。

知識ベースを構築し、オンライン学習プロセスを完了した後、LogCluster を用いて異常を検出することができる。具体的には、新しいログシーケンスの状態を判断するために、知識ベースの代表ベクトルまでの距離を計算する。最小の距離が閾値よりも大きければ、そのログシーケンスは異常として報告され

る。そうでなければ、最も近いクラスタが正常/異常クラスタであれば、ログシーケンスは正常/異常として報告される。

## ② PCA

主成分分析 (PCA: Principal Component Analysis) は、次元削減を行うために広く用いられている統計手法である。PCA の基本的な考え方は、高次元のデータを、 $k$ を元の次元よりも小さく設定した $k$ 個の主成分で構成される新しい座標系に投影することである。PCA は、高次元データの中で最も分散が大きい成分（軸）を見つけることで、 $k$ 個の主成分を計算する。このように、PCA によって変換された低次元データは、元の高次元データの主要な特徴（例: 2点間の類似度）を保持することができる。

PCA は、Xu ら[39]によってログベースの異常検出に初めて適用された。各ログシーケンスは、イベントカウントベクトルとしてベクトル化される。その後、イベントカウントベクトルの次元間のパターンを求めるために PCA を用いる。PCA を用いると、通常空間 $S_n$ と異常空間 $S_a$ の 2つの部分空間が生成される。 $S_n$ は最初の $k$ 個の主成分によって構成され、 $S_a$ は残りの $(n - k)$ 個の主成分によって構成される。ここで、 $n$ は元の次元である。そして、イベントカウントベクトル $y$ の $S_a$ への投影値 $y_a = (1 - PP^T)y$ を算出する。ここで $P = [v_1, v_2, \dots, v_k]$ は最初の $k$ 個の主成分である。 $y_a$ の長さが閾値よりも大きい場合、対応するイベントカウントベクトルは異常として報告される。具体的には、以下の場合、イベントカウントベクトルは異常とみなされる。

$$SPE \equiv \|y_a\|^2 > Q_a \quad (1.1)$$

ここで、二乗予測誤差 (SPE: Squared Prediction Error) は「長さ」を表し、 $Q_a$ は $(1 - \alpha)$ 信頼度を提供する閾値である。

### ③ Invariants Mining

プログラム不变量とは、様々な入力や負荷があっても、システムが動作している間は常に保持される線形関係のことである。不变量マイニングは、[36]で初めてログベースの異常検出に適用された。同じセッション ID (HDFS のプロック ID など)を持つログは、そのセッションのプログラム実行フローを表すことが多い。簡単なプログラム実行フローを図 1.3 に示す。

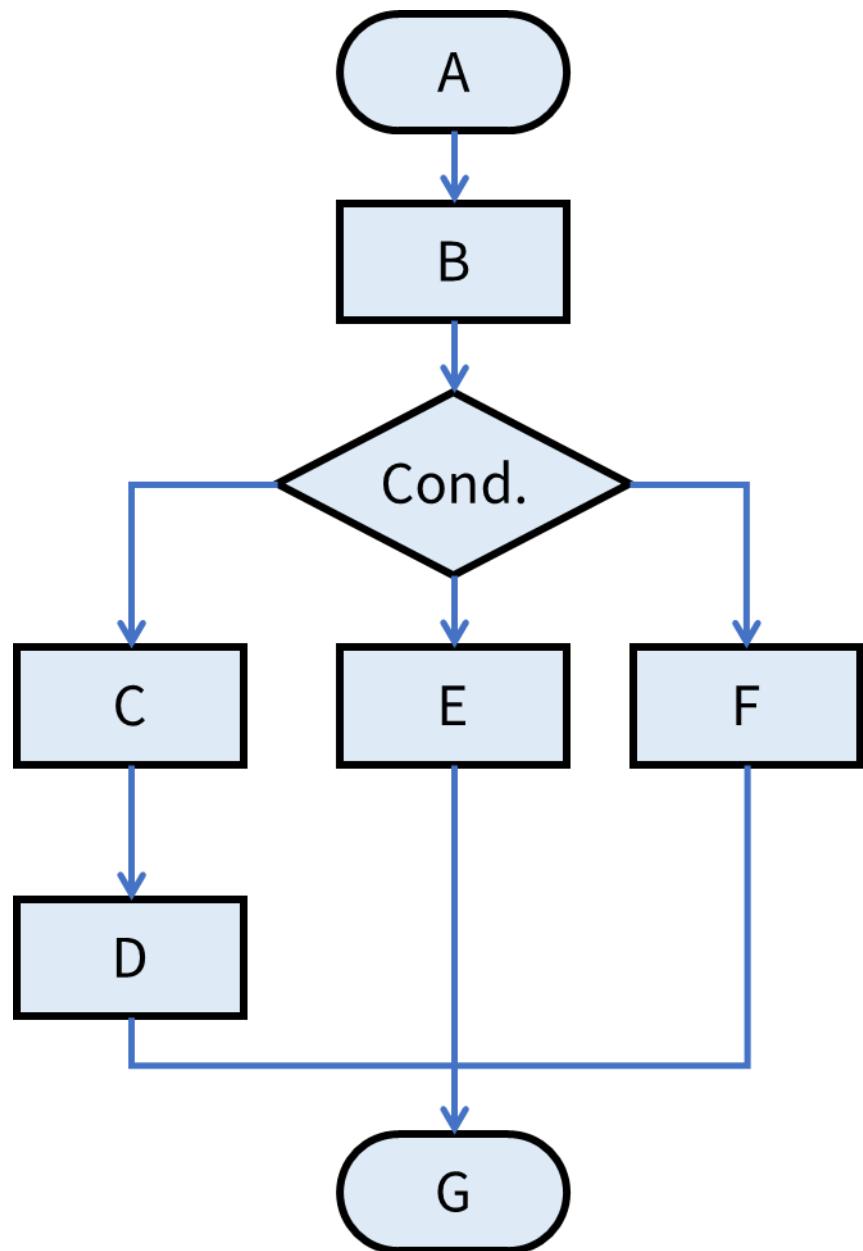


図 1.3 プログラム実行フローの例

この実行フローでは、システムは A から G までの各段階でログメッセージを生成する。システム内に多数のインスタンスが実行されており、それらが図 1.3 のプログラム実行フローに従っていると仮定すると、以下の式が有効となる。

$$\mathbf{n}(\mathbf{A}) = \mathbf{n}(\mathbf{B})$$

$$\mathbf{n}(\mathbf{B}) = \mathbf{n}(\mathbf{C}) + \mathbf{n}(\mathbf{E}) + \mathbf{n}(\mathbf{F})$$

$$\mathbf{n}(\mathbf{C}) = \mathbf{n}(\mathbf{D})$$

$$\mathbf{n}(\mathbf{G}) = \mathbf{n}(\mathbf{D}) + \mathbf{n}(\mathbf{E}) + \mathbf{n}(\mathbf{F}) \quad (1.2)$$

ここで、 $n(*)$ は、対応するイベントタイプ\*に属するログの数を表す。

直感的には、不变量マイニングにより、システムの正常な実行動作を表す複数のログイベント間の線形関係（例： $n(A) = n(B)$ ）を明らかにすることができる。実世界のシステムイベントでは、線形関係が優先される。例えば、通常ファイルは開いた後に閉じなければならない。このように、「ファイルを開く」というフレーズのログと「ファイルを閉じる」というフレーズのログはペアで出現する。インスタンス内のログイベント「オープンファイル」とログイベント「クローズファイル」の数が等しくない場合は、線形関係に反しているため異常と判定される。

不变量（線形関係）を見つけることを目的とした不变量マイニングには、3つのステップが存在する。不变量マイニングの入力は、ログシーケンスから生成されたイベントカウント行列であり、各行はイベントカウントベクトルである。まず、特異値分解を用いて不变空間を推定し、次のステップでマイニングが必要な不变量 $r$ を決定する。この方法では、ブルートフォースサーチアルゴリズムによって不变量を発見する。最後に、マイニングされた各不变量候補は、そのサポートを閾値（例：イベントカウントベクトルの 98% でサポートされている）と比較することによって検証される。このステップは、 $r$ 個の独立した不变量が得られるまで続ける。不变量に基づく異常検出では、新しいログシーケンスが入力されたときに、それが不变量に従うかどうかを確認する。少なくとも 1 つの不变量が壊れている場合、ログシーケンスは異常として報告される。

## (b) 教師あり学習モデル

教師あり学習は、ラベル付けされた訓練データからモデルを導出する機械学習タスクとして定義される。正常・異常状態をラベルで示すラベル付き学習データは、教師あり異常検出の前提条件である。学習データのラベル付けが多いほど、モデルの精度は高くなる。ここでは代表的な教師あり手法である、ロジスティック回帰、決定木、サポートベクターマシン（SVM）の3つを説明する。

### ① ロジスティック回帰

ロジスティック回帰は、分類に広く使われている統計モデルである。あるインスタンスの状態を決定するために、ロジスティック回帰では、考えられるすべての状態（正常または異常）の確率 $p$ を推定する。確率 $p$ は、ラベル付けされた学習データに基づいて構築されたロジスティック関数によって計算される。新しいインスタンスが出現したとき、ロジスティック関数はすべての可能な状態の確率 $p$  ( $0 < p < 1$ )を計算することができる。確率を取得した後、最も確率の高い状態が分類出力となる。

異常を検出するために、各ログシーケンスからイベントカウントベクトルを構築し、各イベントカウントベクトルベクトルとそのラベルを合わせてインスタンスと呼ぶ。まず、学習インスタンスを用いてロジスティック回帰モデルを構築する。モデルを取得した後、テストインスタンス $X$ をロジスティック関数に入力して、その異常の確率 $p$ を計算する。

### ② 決定木

決定木は、各インスタンスの予測状態を示すために枝を用いた木構造図である。決定木は、学習データを用いてトップダウン的に構築される。各ツリーノードは、属性の情報利得によって選択された現在の最良属性を用いて作成される。例えば、図1.4のルートノードは、データセットに20個のインスタンスがあることを示している。ルートノードを分割する際には、イベント2の発生番号を「最良」属性として扱う。このように、学習インスタンス20個全体をこの属性の値に応じて2つのサブセットに分割し、一方は12個のインスタンスを含み、他方は8個のインスタンスから構成される。決定木は、[30]で初めてWebリクエストログシステムの故障診断に適用された。決定木の構築には、イベントカウントベクトルとそのラベルを利用する。新しいインスタンスの状

態を検出するために、トラバースされた各ツリーノードの述語に従って決定木をトラバースする。トラバースの最後には、このインスタンスの状態を反映した葉のいずれかに到達する。

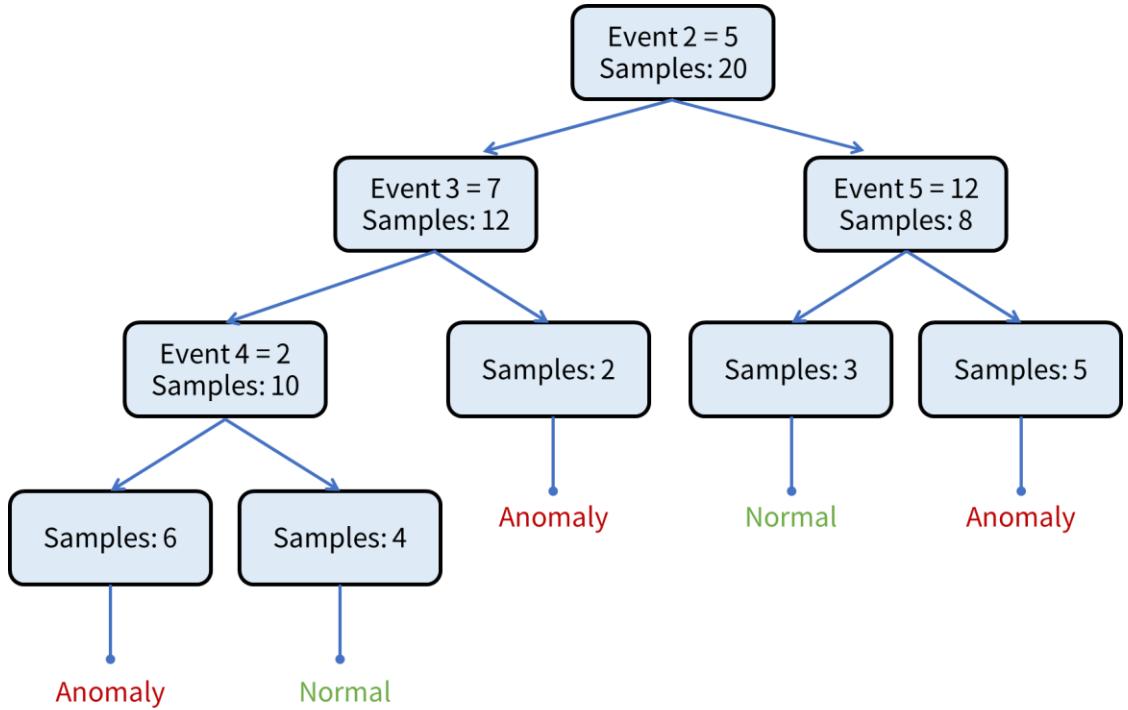


図 1.4 決定木の構造

### ③ サポートベクターマシン

サポートベクターマシン (SVM: Support Vector Machine) は、分類のための教師あり学習手法である。SVM では、高次元空間でインスタンスの様々なクラスを分離するために超平面を構築する。超平面を見つけることは最適化問題であり、異なるクラスでの超平面と最も近いデータ点との距離を最大化する。Liang ら[31]は、故障検出に SVM を採用し、他の手法と比較している。ロジスティック回帰や決定木と同様、学習インスタンスはイベントカウントベクトルとそのラベルである。SVM による異常検出では、新しいインスタンスが超平面の上にある場合は異常として報告され、そうでない場合は正常として扱われる。

### 1.2.2.3 深層学習手法によるログ異常検出

Deep-loglizer[40]は自動化された異常検出のためのディープラーニングベースのログ分析ツールキットであり、近年提案された高精度な異常検出手法の中でも特に注目すべき 6 つのモデルを実装している。表 1.5 Deep-loglizer に実装されている深層学習モデルとその詳細を示す。本研究では、ログベースの異常検出における比較対象として、deep-loglizer に含まれる各種モデルを採用する。以下で、それぞれのモデルの概要と特色を紹介する。

表 1.5 Deep-loglizer に実装されている深層学習モデルとその詳細

学習方法	モデル	アーキテクチャ
教師なし	DeepLog	LSTM
	LogAnomaly	LSTM
	Logsy	Transformer
	Autoencoder	Autoencoder LSTM
教師あり	RobustLog	Attentional BiLSTM (Attn-BLSTM)
	CNNLog	CNN

#### (a) 教師なし学習モデル

教師なし異常検出手法では、システム正常動作時のログデータを学習し、テストデータのパターンが正常時から逸脱した場合にシステムの異常が検出できることを前提としている。また、そのアプローチとして予測ベース、再構成ベースの 2 種類に大別できる。

予測ベースの異常検出手法では、固定長のログシーケンスから次の時刻に現れるログテンプレートのインデックスを予測するように学習を行う。テスト時には、実際に観測されたログが予測されたログの出現確率上位  $k$  個の中に存在しない場合に異常として検出される。DeepLog[41]はログ異常検出に深層学習モデル（LSTM）を適用した最初期のモデルであり、その後提案された予測ベース手法の基礎となっている。LogAnomaly[42]は、ログテンプレートメッセージの意味情報を考慮するために文中の同義語・反意語を考慮した分散表現 template2Vec を特徴量として採用している。本稿では、テンプレートインデックスのみを学習した際の精度を比較するため、ベンチマー

一クリストから除外している。Logsy[43]は、近年自然言語処理において特に注目すべき成果をあげている深層学習モデル Transformer[44]を採用している。Transformer Encoder から得られたログシーケンスの分散表現を用いて、LSTM と同様に次の時刻のログを予測する。

再構成ベースの異常検出手法では、Autoencoder[45]などのモデルを用いて一度次元圧縮された入力特徴量から元の特徴量を再構成するように学習を行う。テスト時には、異常なパターンが入力された場合に MSE などの誤差関数の値が大きくなることを利用して検出を行う。Deep-loglizer の実装では、LSTM を用いてログシーケンスを時間的な特徴量に変換し、全結合層からなる Autoencoder で特徴量を再構成するアプローチが採用されている。

### (b) 教師あり学習モデル

教師あり異常検出手法では、モデルの学習時に明示的にラベルを与え、異常の識別に有用な特徴を自動的に学習させる。

LogRobust[46]は、Attention 付き双方向 LSTM を採用することでログシーケンス中の各テンプレートに異なる重みを割り当てる。DeepLog、LogAnomaly との実装上の違いは、教師データとして次の時刻のテンプレートインデックスを与えるかシーケンスの異常ラベルを与えるかのみである。

CNN を用いた方法[47]では、各ログイベントに対して割り当てた埋め込みを用いてログシーケンスを画像的な特徴量に変換する。特徴量は（埋め込みの次元数×ログシーケンス長）の画像であり、異なるカーネルサイズをもつ複数の畳み込み層で処理される。それらの出力を連結した特徴量をもとに、全結合層によって異常の識別が行われる。

### 1.3 研究目的

OSS を利用したソフトウェアの大規模化により、報告される不具合の件数も増大している。そのため OSS を利用する企業にとって、早期に解決されない不具合は開発サイクルの遅れを生むなど大きな課題となっている。本研究では、不具合修正プロセスにおける修正フェーズにおける作業時間の短縮・効率化を行うシステムの開発を目的とする。

- 不具合票の自動分類システムによるバグトリアージの効率化

OSS を利用したソフトウェアでは、不具合の原因が複雑になりやすくバグトリアージの難易度も高くなる。一方、不具合修正プロセスを円滑に進めるためには、報告された不具合に対して早期に適切な修正担当者（カテゴリ）を割り当てる必要がある。本研究では、増大する不具合報告がプロジェクトに与える負担を解決するため、不具合報告内容の文脈情報をもとに担当カテゴリへの自動割り当てを行うシステムを提案する。

- ログ異常検出によるバグ解析の効率化

OSS を利用したソフトウェア開発の大規模化や不具合の複雑化により、エンジニアがバグ解析で解読するログデータの量が膨大になっている。また、大規模開発におけるバグ解析にはかなりの経験を要するため、このようなログを解読できる人材の不足も OSS を利用する企業の課題となっている。バグ解析では、ソフトウェアが正常に動作しているときのログデータと異なるパターンに着目して不具合の原因を追跡する。そのため、ログデータの中で異常なパターンを示す部分のみを抽出することにより、バグ解析にかかる時間が大幅に短縮できることが期待される。本研究では、システムが正常に動作している際のログのパターンを機械学習手法に学習させ、異常なパターンのログメッセージを抽出する方法を提案する。

## 1.4 論文の構成

本論文における各章の内容を以下に示す。本論文は、「分散表現を用いた分類方法の研究」、「時系列パターンを用いた異常検出方法の研究」の2つに大別でき、「分散表現を用いた分類方法の研究」は第2章～第3章、「時系列パターンを用いた異常検出方法の研究」は第4章～第5章である。

### 第1章 序論

本章では、システム開発における不具合解析の現状を調査し、本研究のテーマ選択の背景について述べる。

### 第2章 深層自然言語処理モデルに基づく不具合票の分類と可視化

本章では、深層自然言語処理モデルの1つであるTransformerに基づくシステム不具合チケットの自動分類手法を提案し、有効性の検証を行う。

### 第3章 分散表現のワンクラス学習に基づくログデータの異常検出

本章では、Transformerによってログメッセージを分散表現へ変換し、分散表現のワンクラス学習によって異常なメッセージを識別する手法を提案する。

### 第4章 時間記憶に基づくログデータの異常検出

本章では、リアルタイム学習が可能なアルゴリズムHTMの時間記憶に基づく異常ログパターンの検出手法を提案する。

### 第5章 空間プーリングに基づくログデータの異常検出

本章では、HTMの空間プーリングに基づく、より軽量な異常ログパターンの検出手法を提案する。

### 第6章 結論

本章では、各章のまとめと実験結果の考察を行い、今後の課題について述べる。

# 第2章 深層自然言語処理モデルに基づく不具合票の分類と可視化

## 2.1 はじめに

序論で述べたように、OSS を使用したソフトウェア開発では不具合管理システム上で不具合修正タスクを適切な担当者へ割り当てる必要がある。従来のシステム開発では、不具合報告の内容から手作業でカテゴリ分類を行い、そのカテゴリ情報をチケットに付加していた。しかし、このような方法は、大規模なシステム開発の場合には多くの人員を必要とし、時間的コストが膨大になるという問題がある。

本研究では、不具合票の記述（メッセージ）をサブワード[48]と呼ばれる単位に分割し分散表現へと変換することにより、未知の単語情報を効果的に扱うことができる不具合票自動分類システムを提案する。また、実際の OSS ソフトウェアのバグレポートデータセットを用いた分類実験を行うことにより提案システムの有効性の検証を行う。

## 2.2 学習済み BERT モデルによる不具合票自動分類システム

提案するシステムは、Redmine などの不具合管理システムで作成された不具合票を入力とする。システムのテスト担当者は、テスト時に発生した不具合事象をコメント付きの不具合票に記載し、不具合管理システムにアップロードする。不具合管理システムに集約されたチケットには、システムに組み込まれたどの OSS が不具合の原因なのかという情報は記載されていない。提案手法では、コメント（テキストデータ）の情報をもとにチケットを深層自然言語処理モデル BERT[49]に基づいて適切な不具合修正タスクのカテゴリへと割り当てる。以下でシステムの詳細について述べる。

### 2.2.1 提案システムの概要

既存の不具合分類手法では、開発者がコメント中で過去に用いた単語の出現頻度に対して SVM などの機械学習アルゴリズムを適用することで、各不具合に対して開発者の割り当てを行う。そのような場合、不具合票中にこれまでに見られなかった新規単語の情報をうまく扱うことが難しく、プロジェクトが進行するにしたがって頻繁にモデルの再学習が必要となる。

本システムでは、コメントを WordPiece アルゴリズム[50]に基づいてサブワードと呼ばれる単位に分割して扱うことで、チケットに記載された不具合事象の記述に未知の単語が含まれている場合にも効果的にカテゴリ割り当てを行うことができる。サブワードに分割されたコメントは Transformer をベースとした大規模自然言語処理モデル BERT によって分散表現へと変換され、文脈的な情報をもとに不具合修正タスクへとカテゴリライズされる（図 2.1）。以下で Transformer、BERT および WordPiece の詳細を述べる。

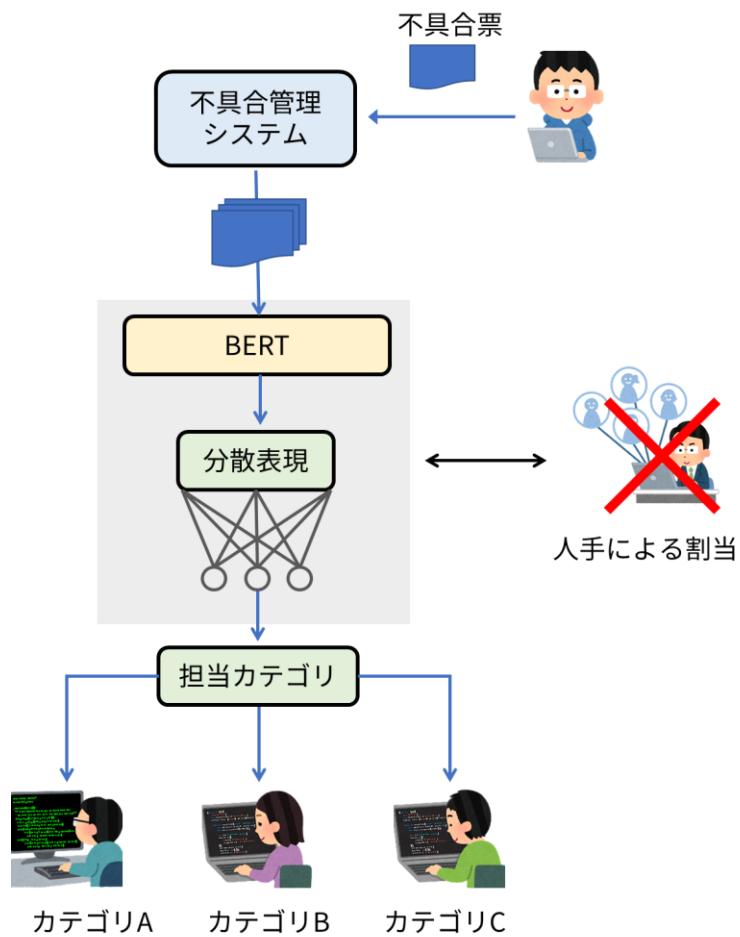


図 2.1 提案する不具合票自動分類システムのフロー

## 2.2.2 Transformer

Transformer は、シーケンス変換モデルの基本構成要素をリカレントニューラルネットワーク(RNN: Recurrent Neural Network)や畳み込みニューラルネットワーク (CNN: Convolutional Neural Network) から Attention Mechanism に置き換えたものである[44]. Attention Mechanism を用いることにより、入力や出力のシーケンス内の距離を気にせずに依存関係をモデリングすることが可能となるメリットが得られる. 以下で Transformer のモデル構造を詳述する.

### 2.2.2.1 エンコーダ・デコーダスタック

ニューラルシーケンス変換モデルはエンコーダ・デコーダ構造をもっており、エンコーダは入力シーケンスのシンボル表現  $(x_1, \dots, x_n)$  を連続表現のシーケンス  $z = (z_1, \dots, z_n)$  にマッピングする.  $z$  が与えられると、デコーダはシンボルの出力シーケンスを 1 要素ずつ生成する. 各ステップにおいてモデルは自己回帰し、前に生成されたシンボルを次の生成時の追加入力として使用する. Transformer は、図 2.2 に示すようにエンコーダ、デコーダの両方にスタックされた自己注意層 (Attention) と位置単位の順伝播ネットワーク(Feed Forward)を使用する. 自己注意層と位置単位順伝播ネットワークについては後節で詳述する.

#### ● エンコーダ

エンコーダは同じ構造の層を積層して構成されており、各層には 2 つのサブレイヤーが存在する. 1 つ目はマルチヘッドアテンション、2 つ目は位置単位の順伝播ネットワークである. 2 つのサブレイヤーの各々に残差接続が設けられており、続いて Layer Normalization が行われる. 残差接続を容易にするため、モデル内の全てのサブレイヤーと埋め込み層では同一次元の出力が生成される.

#### ● デコーダ

デコーダもまた、エンコーダと同様に同じ構造の層を積層して構成されている. デコーダでは、エンコーダの 2 つのサブレイヤーに加えて 3 つ目のサブレイヤーが挿入される. 3 つ目のサブレイヤーでは、エンコーダスタックの出力に対してマルチヘッドアテンションが適用される. エンコーダと同様、各サブレイヤーでは残差接続に続いて Layer Normalization が行われる.

デコーダスタックの自己注意層では、ある位置が自分よりも後ろの位置に注意する

ことを防ぐためにマスキングが施される。マスキングと出力埋め込みが位置 1 つ分オフセットされていることにより、位置 $i$ の予測では $i$ よりも前の位置の既知の出力のみに依存することができる。

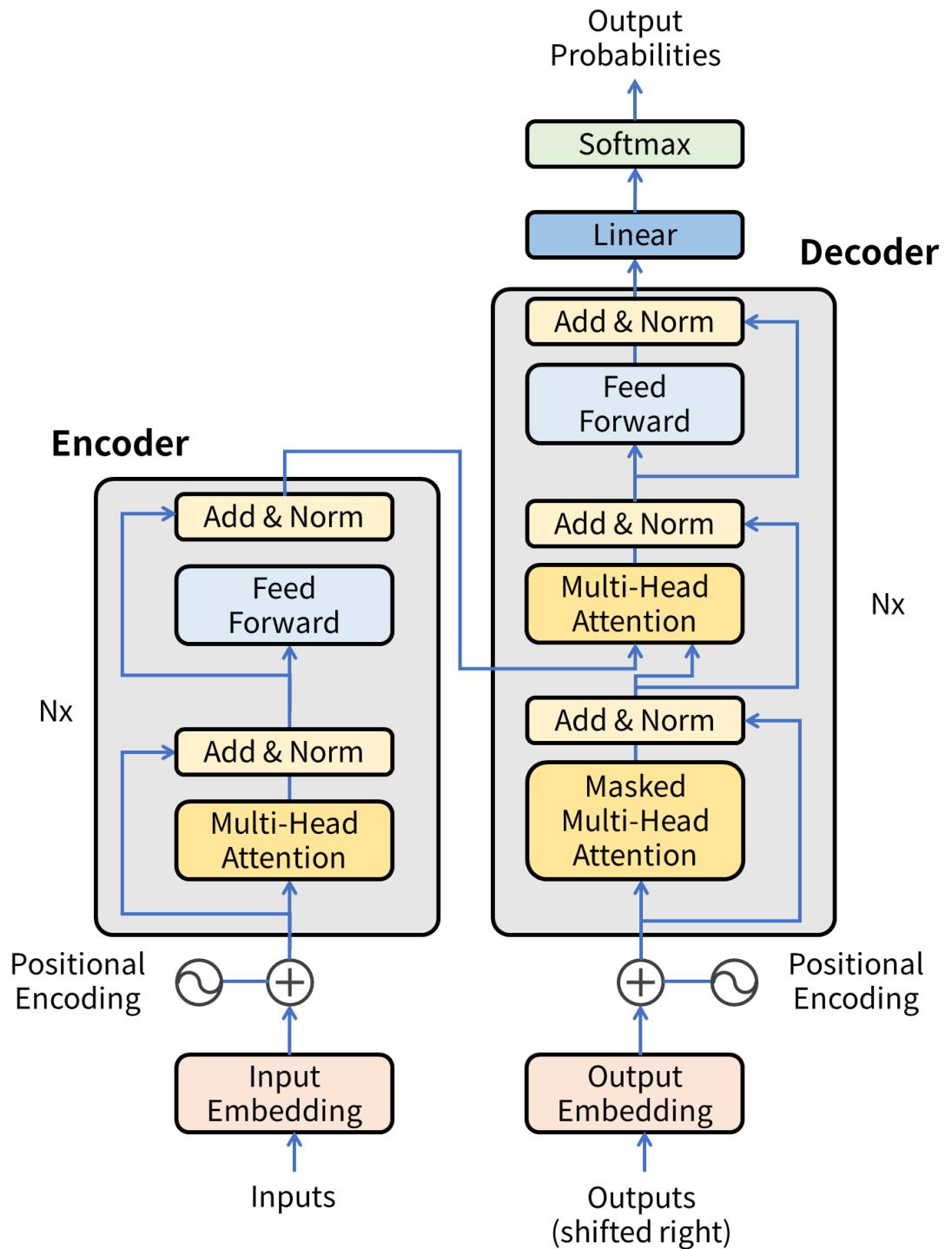


図 2.2 Transformer のモデル構造

### 2.2.2.2 自己注意層

注意関数は Query · Key · Value のペアのセットを出力にマッピングするものであり、Query、Key、Value、出力はすべてベクトルである。出力は Value の加重和として計算され、各 Value に割り当てられた重みは Query と対応する Key の互換性関数によって計算される。

Transformer で使用される自己注意を特に Scaled Dot-Product Attention と呼び、入力は  $d_k$  次元の Query と Key、 $d_v$  次元の Value で構成される。すべての Key と Query の内積を計算し、結果を  $\sqrt{d_k}$  で除算したのち、Value の重みを得るために softmax 関数が適用される。図 2.3 に、Scaled Dot-Product Attention と Multi-Head Attention の構成を示す。

実際には、行列  $Q$  にまとめられた一連のクエリに対して同時に注意関数が計算される。また、Key と Value も行列  $K$  と  $V$  にまとめられ、出力行列は次のように求められる。

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2.1)$$

Dot-Product Attention（乗算型注意）は最もよく使用される注意関数である。Scaled Dot-Product Attention では、 $d_k$  の値が大きい場合に内積の値が大きくなり softmax 関数の勾配が極端に小さくなる効果を打ち消すため、スケーリングファクター  $\frac{1}{\sqrt{d_k}}$  が用いられる。

Multi-Head Attention では、モデルは異なる位置にある異なる表現の部分空間からの情報を共同で注意することができる。単一の Attention Head では、平均化によってこれが阻害される。

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

$$\text{where } \text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \quad (2.2)$$

ここで、射影はパラメータ行列  $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$ 、 $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$ 、 $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ 、 $W_i^O \in \mathbb{R}^{hd_v \times d_{model}}$  である。また、 $h$  は並列して実行される Attention Head の数であり、それぞれの Head では  $d_k = d_v = \frac{d_{model}}{h}$  が使用される。これにより、総計算コストは Single-Head Attention と同様となる。

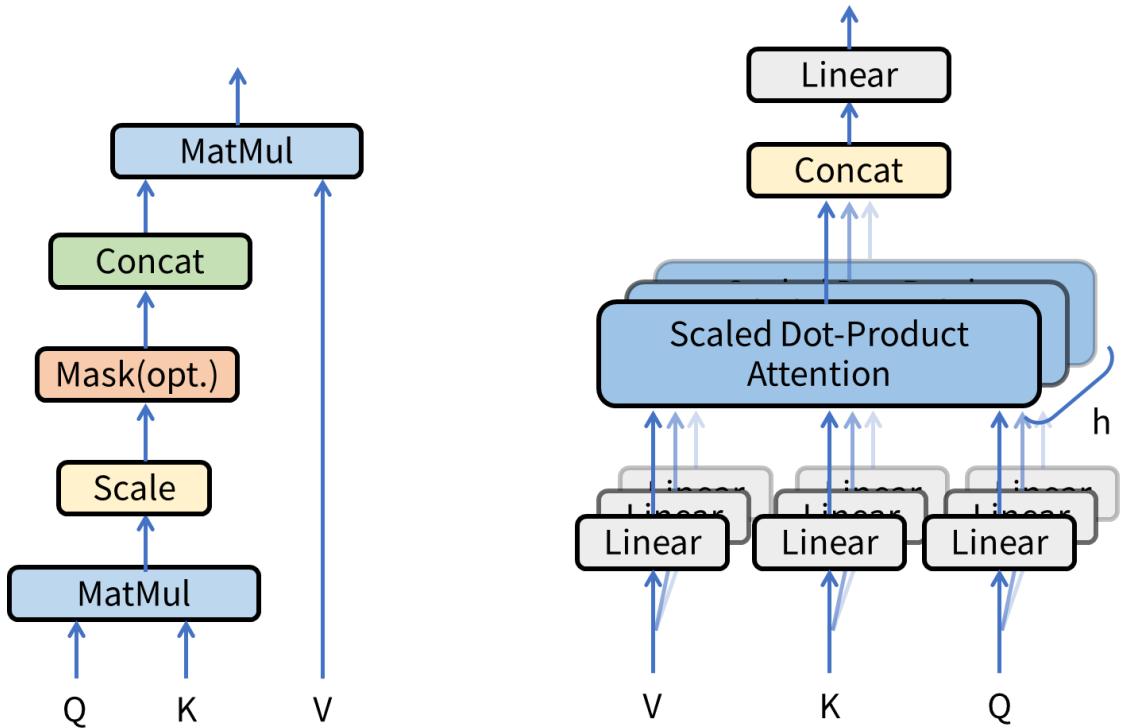


図 2.3 Scaled Dot-Product Attention と Multi-Head Attention

Transformer では、Multi-Head Attention を次の 3 つの異なる用途で使用する。

- エンコーダ-デコーダ間の注意層では、Query は前のデコーダ層から、メモリの Key と Value はエンコーダの出力から送られる。これにより、デコーダのすべての位置が入力シーケンスのすべての位置に注目することができる。
- エンコーダには自己注意層があり、全ての Key・Value・Query がエンコーダの前の層の出力から送られる。エンコーダの各位置は、前の層の全ての位置に対応することができる。
- 同様に、デコーダ内の自己注意層により、各位置がその位置までのすべてのデコーダ内の位置に注意を払うことができる。自己回帰特性を維持するため、デコーダ内の左向きの情報の流れを防ぐ必要がある。Softmax の入力に含まれる不正な接続に対応するすべての値をマスクアウト ( $-\infty$  に設定) することにより、Scaled Dot-Product Attention 内にこの機能を実装する。

### 2.2.2.3 位置単位順伝播ネットワーク

Attention サブレイヤーに加えて、エンコーダとデコーダの各層には全結合のフィードフォワードネットワークが含まれており、このネットワークは各位置において別々に、かつ同一に適用される。これは ReLu 活性化をはさんだ 2 つの線形変換で構成される。

$$FFN(x) = \max(\mathbf{0}, xW_1 + b_1)W_2 + b_2 \quad (2.3)$$

線形変換は異なる位置に対しても同様であるが、層ごとに異なるパラメータを使用する。これは別の表現ではカーネルサイズ 1 の 2 つの畳み込みとなる。入力と出力の次元はいずれも  $d_{model}$ 、隠れ層の次元は  $d_{ff}$  である。

### 2.2.2.4 Embedding 層

他のシーケンス変換モデルと同様に、学習した埋め込みを用いて入力と出力のトークンを  $d_{model}$  次元のベクトルに変換する。また、デコーダの出力を次のトークンの予測確率に変換するため、学習した線形変換と softmax 関数を使用する。エンコーダとデコーダの埋め込み層では同じ重み行列と softmax 前の線形変換を共有し、これらの重みに  $\sqrt{d_{model}}$  を乗算する。

### 2.2.2.5 位置エンコーディング

Transformer には再帰と畳み込みが含まれていないため、モデルがシーケンスの順序を利用するためには相対的もしくは絶対的な位置に関する情報を注入する必要がある。そのため、エンコーダとデコーダの入力埋め込みに位置エンコーディングを追加する。位置エンコーディングは埋め込みと同じ  $d_{model}$  次元であるため、2 つの埋め込みを合計することができる。Transformer では、位置エンコーディングに異なる周波数の sin, cos 関数を使用する。

$$PE(pos, 2i) = \sin\left(pos / 10000^{\frac{2i}{d_{model}}}\right) \quad (2.4)$$

$$PE(pos, 2i + 1) = \cos\left(pos / 10000^{\frac{2i}{d_{model}}}\right) \quad (2.5)$$

ここで $pos$ は位置、 $i$ は次元であり、位置エンコーディングの各次元は正弦波に対応している。波長は $2\pi$ から $10000 - 2\pi$ までの等比数列をなす。

### 2.2.3 BERT

BERT (Bidirectional Encoder Representations from Transformers) [49]は、ラベルなしのテキストデータを用いた事前学習を行い、質問応答や言語推論などの幅広い自然言語タスクへの転移学習が可能な言語表現モデルである。モデルの学習には、事前学習と微調整という2つの段階がある。事前学習では、モデルは異なる事前学習タスクのラベルなしデータに対して訓練される（図2.4）。微調整では、モデルはまず事前学習されたパラメータで初期化され、適用したいタスクのラベル付きデータを使用して全てのパラメータが調整される（図2.5）。

BERT モデルアーキテクチャは多層双方向 Transformer として構成される。本論文では、モデルの構成として  $\text{BERT}_{\text{BASE}}(L = 12, H = 768, A = 12, \text{総パラメータ数}=110\text{M})$  を使用する。ここで  $L$  は Transformer の層の数、 $H$  は入出力ベクトルの次元数、 $A$  は自己注意ヘッドの数である。

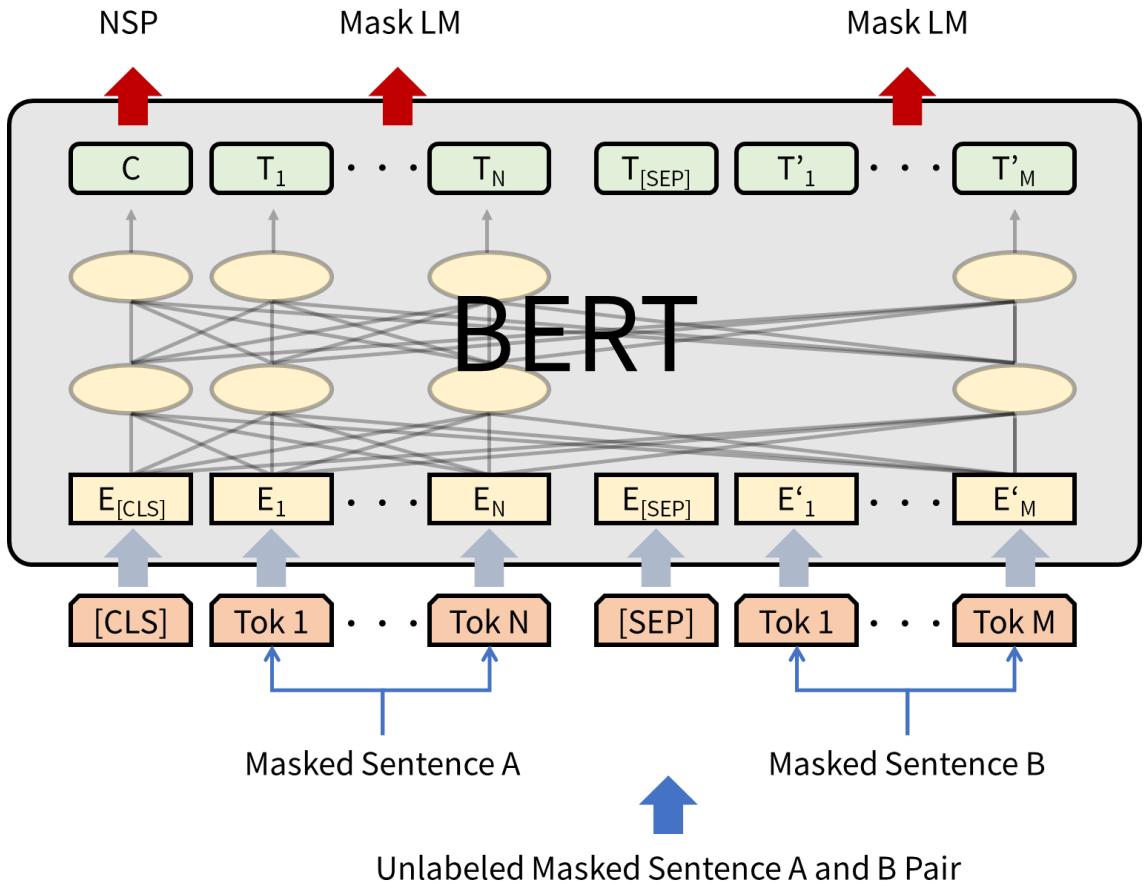


図 2.4 BERT モデルの事前学習

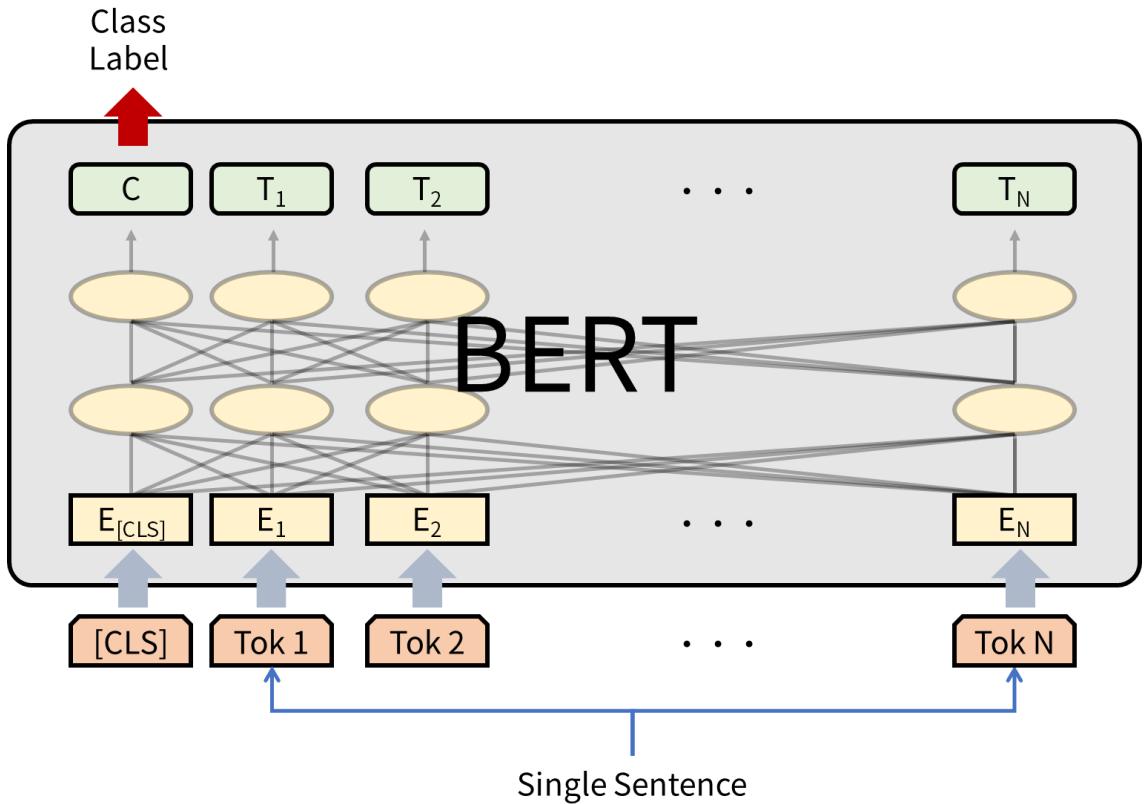


図 2.5 BERT モデルのファインチューニング（分類タスク）

### 2.2.3.1 モデルの入力表現

- WordPiece トークン

BERT モデルの入力として、30,000 トークンの語彙を持つ WordPiece 埋め込みが用いられる。WordPiece はニューラル機械翻訳において希少な単語を処理するために開発されたアプローチであり、語彙にない単語（OOV: Out of Vocabulary）をサブワードと呼ばれる単位に分割する。これは完全なデータ駆動型のアプローチであり、あらゆる文字列に対して決定論的なセグメンテーションを生成することが保証される。

WordPiece モデルは、進化する単語の定義が与えられた際に、学習データの言語モデルの尤度を最大化するデータ駆動型のアプローチを用いて生成される。学習コーパスと必要なトークン数  $D$  が与えられたときの最適化問題は、選択した WordPiece モデルにしたがってセグメンテーションをしたときに得られるコーパスのワードピース数が最小となるような  $D$  個のワードピースを選択することである。

任意の単語を処理するために、まず学習した WordPiece モデルを用いて単語をワードピースに分割する。モデルの学習を行う前に特別な単語の境界記号を追加することで、元の単語列がワードピースのシーケンスから復元できるようにする。

- 特殊トークン

全てのシーケンスの最初のトークンは常に特別なトークン”[CLS]”であり、これに対応するモデルの最終的な隠れ状態が分類タスクのための集約されたシーケンス表現として使用される。文のペアを入力する場合には、特別なトークン”[SEP]”を区切り文字として1つのシーケンスとしてまとめられる。あるトークンに対して、その入力表現は対応するトークン・セグメント・位置埋め込みを合計することで求められる（図 2.6）。セグメント埋め込みは、トークンが区切られた文のどちらに属するかの情報を表すための埋め込みである。

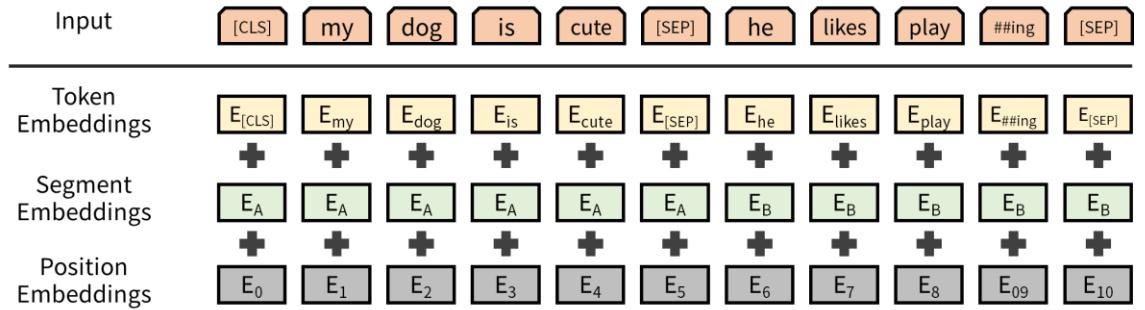


図 2.6 入力トークンに対する埋め込みの計算方法

### 2.2.3.2 モデルの事前学習

BERT の事前学習では、後述する 2 つの教師なし学習タスクを用いる。モデルの事前学習用のデータとしては、BooksCorpus (800M words) と英語版 Wikipedia (2,500M words) が使用される。Wikipedia では、テキストの文章部分のみを抽出し、リスト・テーブル・ヘッダーは無視される。

- **Masked Language Model (MLM)**

深い双方向表現を学習するために、入力トークンの何パーセントかをランダムにマスクし、マスクされたトークンの予測を行う。この手順を Masked Language Model (MLM) と呼び、マスクトークンに対応する最終的な隠れベクトルは、標準的な言語モデルと同様に語彙に対する softmax 関数に与えられる。本論文で使用する事前学習済

みBERTモデルでは、各シーケンスに含まれるすべてのWordPieceトークンのうち15%がランダムにマスクされる。トレーニングデータの生成では、トークンの位置のうち15%をランダムに選んで予測を行う。

$i$ 番目のトークンが選ばれた場合、 $i$ 番目のトークンを80%の確率で”[MASK]”トークンに置き換え、10%の確率でランダムなトークンに置き換え、10%の確率で元のトークンそのままにする。 $i$ 番目のトークンに対応する双方向表現 $T_i$ は、クロスエントロピー損失を用いて元のトークンを予測するために使用される。

### ● Next Sentence Prediction (NSP)

文の関係を理解するモデルを学習するため、任意の単言語コーパスから生成される二値化された次文予測タスクを対象とした事前学習を行う。各事前学習サンプルの文AとBを選択する際、50%の確率でBはAに続く実際の次の文であり、50%の確率でコーパスから選択されたランダムな文となる。文頭の”[CLS]”トークンに対応する表現 $C$ は、次の文の予測に使用される。

#### 2.2.3 モデルのファインチューニング

目的とする言語タスクに向けたモデルのファインチューニングでは、タスク固有の入力と出力をBERTに与え、エンド・ツー・エンドで全てのパラメータが微調整される。テキスト分類に向けたファインチューニングでは、新たな出力層として一層の全結合フィードフォワードネットワークが追加され、エンコーダ最終層の”[CLS]”トークンに対応する出力表現が引き渡される。

## 2.3 実験

提案する自動不具合票分類システムの有効性を検証するため、実際の不具合レポートデータを用いてカテゴリ分類実験を行う。はじめに実験で使用するデータセットの詳細を述べ、つづいて実験の方法と条件について説明する。

### 2.3.1 データセット

カテゴリ分類実験には、Rediscovery Datasets[51]の不具合レポートデータを使用する。データセットには、不具合追跡システムである Bugzilla 上で収集された 3 つの OSS プロジェクト(Apache, Eclipse, KDE)の不具合レポートが含まれている。これらのデータは 1999 年から 2017 年にかけて収集され、約 91 万 4 千件の不具合レポートから構成されている。本論文では、Eclipse と Apache の不具合レポートデータそれぞれにおいてカテゴリ分類実験を行う。

### 2.3.2 実験方法

#### ● データの前処理

不具合レポートには、製品・報告者・不具合事象の記述などの情報が含まれている。不具合が起きている製品のクラスの予測には不具合事象の記述部分のみを用いる。実験に先立って、NA 値や重複したデータは削除される。また、不具合事象の記述における文字数が少なすぎる不具合レポートはモデルが解釈するには短すぎ、製品クラスの学習に悪影響を与えると考えられるため削除される。本実験では、Eclipse では 10 文字、Apache では 20 文字以下の不具合レポートを削除の対象とした。表 2.1 と表 2.2 に、Eclipse と Apache プロジェクトにおける不具合事象の記述の統計情報を示す。また、製品クラスは含まれる不具合レポート数でソートされ、各プロジェクトにおいて上位 5 クラスの不具合レポートのみを実験で使用する。各プロジェクトの不具合レポートデータの内訳を表 2.3 と表 2.4 に示す。

表 2.1 不具合事象の記述の統計情報 (Eclipse)

Statistics	# of characters
Mean	57
Std	22
Min	1
25-percentile	42
50-percentile	55
75-percentile	70
Max	255

表 2.2 不具合事象の記述の統計情報 (Apache)

Statistics	# of characters
Mean	54
Std	23
Min	1
25-percentile	38
50-percentile	51
75-percentile	66
Max	255

表 2.3 Eclipse プロジェクトにおける不具合レポートデータの内訳

Label	Product Class Name	# of data
(0)	Platform	109096
(1)	JDT	54771
(2)	z_Archiv	29737
(3)	BIRT	22655
(4)	CDT	19676

表 2.4 Apache プロジェクトにおける不具合レポートデータの内訳

Label	Product Class Name	# of data
(0)	Apache h	8254
(1)	Ant	5946
(2)	POI	3898
(3)	JMeter	3500
(4)	Tomcat 4	3335

#### ● モデルのファインチューニング

上記の前処理済み不具合レポートデータの一部を用いて、BERT モデルのファインチューニングを行う。はじめに、データ全体を 6 : 2 : 2 の割合で分割し、それぞれをモデルのトレーニング、バリデーション、テスト用のデータとする。続いて、上述のトレーニングデータを用いて BERT モデルのファインチューニングを行う。モデルの学習は、NVIDIA GeForce GTX TITAN Black を 1 機搭載した PC 上で行う。学習に用いるパラメータの詳細を表 2.5 に示す。最後に、ファインチューニングを行った BERT モデルを用いてテストデータを分類し、分類スコアを算出する。

表 2.5 ファインチューニング時のパラメータ

Parameter	Value
Pre-trained Model	bert-base-uncased
Max Sequence Length	128
Train Batch Size	6
Learning Rate	5e-6
Epochs	10
Cyclical Learning Rate Type	warmup cosine hard restarts
Vocab Size	30522

- 分類スコアの算出

精度比較では、学習後の各モデルを用いてテストデータにおける異常検出精度を検証する。各モデルは、受信者操作特性（ROC: Receiver Operating Characteristic）曲線の曲線下面積(AUC: Area Under the ROC Curves)と F-measure の値を使用して分類性能を評価する。AUC 値はモデルの分類性能の閾値に対するロバスト性を、F-measure は検出の正確性と異常検出数のバランスを示す評価指標である。ここで、F-measure は以下のように計算される：

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (2.6)$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (2.7)$$

$$F - \text{measure} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (2.8)$$

ここで、

*TP*: モデルが正確に分類した異常インスタンス

*TN*: モデルが正確に分類した正常インスタンス

*FP*: モデルが誤分類した正常インスタンス

*FN*: モデルが誤分類した異常インスタンス

である。

### 2.3.3 実験結果

図 2.7 と図 2.8 に、Eclipse と Apache プロジェクトにおけるテストデータの分類結果から得られた混同行列を示す。行列の縦軸は真のクラスを、横軸はモデルが予測したクラスを表す。表 2.6 は、Eclipse プロジェクトにおいて各製品クラスのデータ数と混同行列から求めた分類スコアを示している。また、同プロジェクトにおいて 47071 件全てのテストデータを分類するのに要した CPU 時間は 696.5 秒となり、1 件あたり平均して 0.015 秒であった。

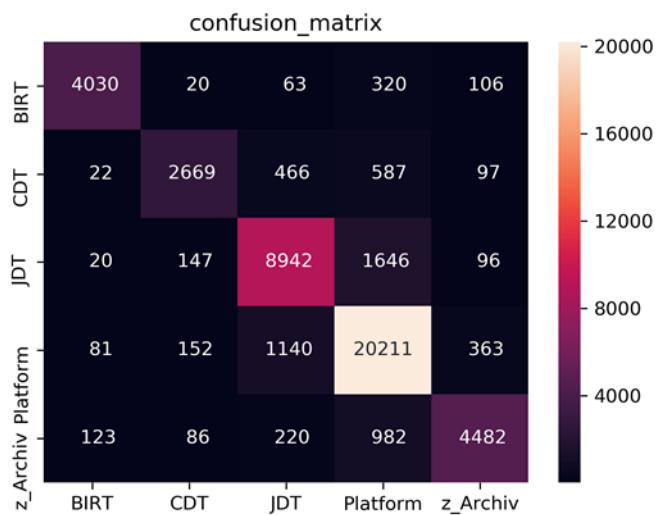


図 2.7 Eclipse におけるカテゴリ分類結果の混同行列

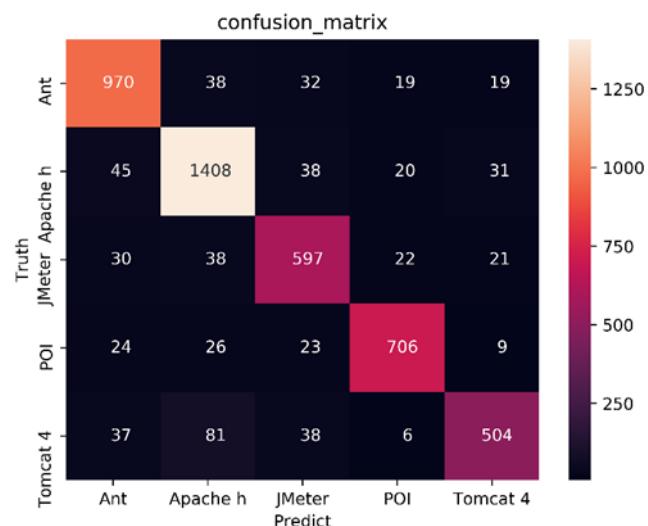


図 2.8 Apache におけるカテゴリ分類結果の混同行列

表 2.6 Eclipse における分類スコア

		Precision	Recall	F-measure	# of data
Product	BIRT	0.94	0.89	0.91	4539
	CDT	0.87	0.69	0.77	3841
	JDT	0.83	0.82	0.82	10851
	Platform	0.85	0.92	0.88	21947
	z_Archiv	0.87	0.76	0.81	5893
Micro Average		0.87	0.86	0.86	47071
Weighted Average		0.86	0.86	0.86	
Accuracy		0.86			

Eclipse プロジェクトの混同行列から、各クラスの誤分類されたデータの多くはデータ数が 5 クラスの中で最も多い JDT と Platform のクラスにカテゴライズされていることが分かる。Apache プロジェクトの混同行列では誤分類のパターンにあまり偏りがみられないことから、クラス間のデータの不均衡が分類の精度に大きな影響を持つと考えられる。

表 2.7 は、Eclipse プロジェクトで誤分類されたテストデータからランダムに抽出された 10 件の不具合レポートの記述を示している。真および誤の列で示されている数は、表 2.3 のラベルの数字と対応している。いくつかのトークン化された記述は、”TestNG”のようなドメイン固有の語彙をサブワードとして処理していることが分かる。

一方で、誤分類された記述において一般的に使用される”unresponsive”のような語彙が意味をなさないサブワードに分割される傾向が確認された。以上の結果から、過剰に分割された語彙が分類に与える影響を改善していく必要があると考えられる。

表 2.7 Eclipse プロジェクトにおいてランダムに抽出した誤分類データ

Label		Tokenized Discription
Truth	Pred	
(1)	(2)	[CLS] format action does not appear for 1st level items in expressions view [SEP] [PAD] [PAD]...
(3)	(2)	[CLS] test ##ng eclipse issue after clicking on run ##as - - > test ##ng test - an exception stack trace is not available . [SEP] [PAD] [PAD] ...
(4)	(3)	[CLS] code does not handle errors in extension definitions [SEP] [PAD] [PAD] ...
(1)	(2)	[CLS] exception in log when working with java projects [SEP] [PAD] [PAD] ...
(4)	(3)	[CLS] problems with editing info on a plug - in [SEP] [PAD] [PAD] ...

(3)	(4)	[CLS] ref ##act ##orin ##g assets packages del ##ete ##s images [SEP] [PAD] [PAD] ...
(0)	(3)	[CLS] game gets stuck on entering the play now option in main menu [SEP] [PAD] [PAD] ...
(4)	(3)	[CLS] getting unix ##pro ##ces ##s . fork ##and ##ex ##ec native error : argument list too long [SEP] [PAD] [PAD] ...
(2)	(3)	[CLS] warnings in n ##200 ##6 ##10 ##28 - 001 ##0 [SEP] [PAD] [PAD] ...
(4)	(3)	[CLS] long running script makes output ##view un ##res ##pon ##sive [SEP] [PAD] [PAD] ...

モデルの分類における判断根拠を分析するため、Apache プロジェクトにおいてモデルが正しく分類を行った不具合レポートをランダムに抽出した。該当するレポートを分類する際にモデルが算出した自己注意の重みを、有向グラフおよび PageRank アルゴリズムを用いて可視化した。実際の可視化において、特に有用と思われるものの抜粋を図 2.9 に示す。図中では BERT モデルの 9 層、ヘッド 4 における自己注意重みを使用している。矢印の太さは注意の大きさを、ノードの大きさは PageRank アルゴリズムによって算出した単語の重要度を表す。可視化の結果から、Apach h 製品に関連する”database”という単語に対して強い注意が払われており、分類においてプロダクト固有の語彙が重視されていることが示唆される。

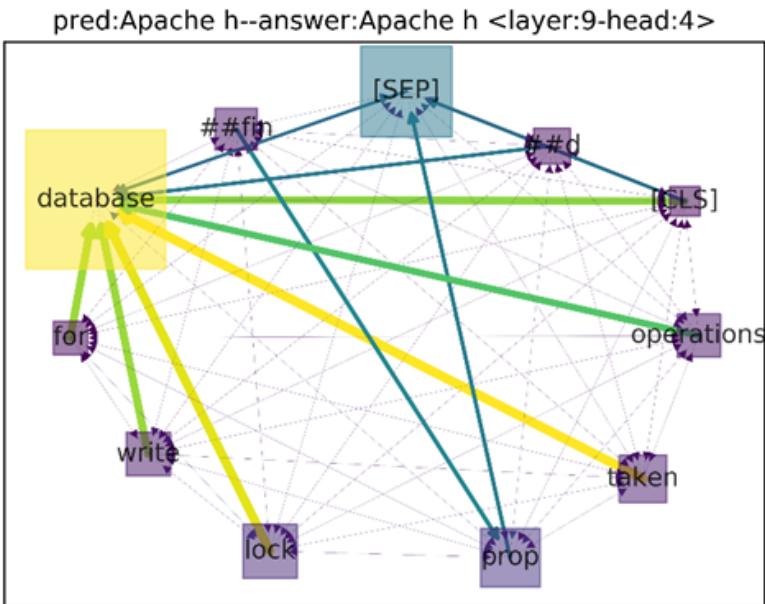


図 2.9 Apache プロジェクトにおける Self-attention の可視化結果

## 2.4 結論

本章では、ファインチューニングした BERT モデルに基づいて不具合票を自動分類するシステムの提案を行った。また、提案手法の有効性を実際の OSS プロジェクトの不具合レポートデータを用いたカテゴリ分類実験によって検証した。結果として、提案手法は約 86% の F-measure を記録し、1 件あたり平均して 0.015 秒と高速な自動不具合票分類を実現できることが分かった。

# 第3章 分散表現のワンクラス学習に基づくログデータの異常検出

## 3.1 はじめに

近年、ソフトウェアシステムの複雑化・大規模化に伴い、実行時に収集されるログデータが膨大になり、その把握が困難になっている。クラウド基盤のように複数のタスクを並行して実行するシステムでは、様々なソフトウェアの出力のログデータがインターリープで生成されることが一般的である。このような場合、不具合の原因を特定する手がかりとなるメッセージを手作業で探すのは非常に困難となる。

不具合に関連するメッセージは、ソフトウェアが正常に動作している場合にあまり見られないものであることが多い。実際の不具合解析では、エンジニアが経験的にそのようなログメッセージに着目し、異常の原因まで遡るという手順がとられる。ログメッセージの中には、テンプレートとしては頻出するが滅多に出現しないパラメータが埋め込まれているものも多い。そのため、単語の出現頻度などに基づいた異常ログの自動抽出では、実際の不具合に無関係なものが多く入り込むという問題がある。

本章では、プロジェクトに蓄積されたテストログデータを用いてワンクラス学習を行い、新規に与えられたログデータがプロジェクトデータの中でどの程度の異常性を持つかを数値化するモデルを提案する。また、オープンなログデータセットを用いて提案手法による異常検出実験を行い、どの程度の効率性で異常なログデータを検出できるかを検証する。ベースライン手法として、テキストデータの異常検出で一般に用いられる Text Autoencoder[52]を使用する。

## 3.2 Transformer エンコーダ表現のワンクラス学習

本章では、提案手法において BERT 分散表現のワンクラス学習を行う際のベースとなっている One-Class Neural Networks[53]について述べる。また、ログデータの異常検出性能の比較実験においてベースラインとして用いる Text Autoencoder の詳細を示す。

### 3.2.1 One-Class Neural Networks

提案手法において特徴表現の異常スコアを算出する方法は、One-Class Neural Networks (OC-NN)に触発されている。One-Class Neural Networks は、One-Class SVM の目的関数をニューラルネットワークに統合することで、隠れ層の表現学習を異常検出向けに最適化する。以下で One-Class SVM の詳細を述べる。

#### 3.2.1.1 One-Class SVM

One-Class SVM(OC-SVM)は、データの確率質量の大部分の周りに滑らかな境界を構築し、異常を検出するために広く使用されている教師なし学習手法である。OC-SVM はサポートベクターマシンの特殊なケースであり、再生核ヒルベルト空間 (RKHS) 内の原点からすべてのデータ点を分離するための超平面を学習し、この超平面から原点までの距離を最大化する。直感的には、OC-SVM では、すべてのデータ点は正にラベル付けされたインスタンスとみなされ、原点は唯一の負にラベル付けされたインスタンスとみなされる。

より具体的には、学習データ  $\mathbf{X}$ 、クラス情報のない集合、および入力空間から特徴空間  $F$ への RKHS 写像関数  $\Phi(\mathbf{X})$  が与えられ、特徴空間  $F$ における超平面または線形決定関数  $f(\mathbf{X}_{n:})$  は、写像ベクトル  $\Phi(\mathbf{X}_{n:}), n : 1, 2, \dots, N$  のうち、できるだけ多くの写像ベクトルを原点から分離するために、 $f(\mathbf{X}_{n:}) = w^T \Phi(\mathbf{X}_{n:}) - r$  として構成される。ここで、 $w$  は超平面に垂直なノルム、 $r$  は超平面のバイアスである。 $w$  と  $r$  を求めるためには、以下の最適化問題を解く必要がある。

$$\min_{w,r} \left\{ \frac{1}{2} \|w\|_2^2 + \frac{1}{\nu} \cdot \frac{1}{N} \sum_{n=1}^N \max(0, r - \langle w, \Phi(\mathbf{X}_{n:}) \rangle) - r \right\} \quad (3.1)$$

ここで  $\nu \in (0,1)$  は、原点からの超平面の距離を最大にすることと、超平面を横切ることが許されるデータ点の数（偽陽性）との間のトレードオフを制御するパラメータである。

### 3.2.1.2 OC-NNへの拡張

OC-NN は、OC-SVM と等価な損失関数を用いたニューラルネットワークのアーキテクチャを設計していると見ることができる。OC-NNを使用することで、教師なし転移学習モデルから得られた特徴量を利用し、特に異常検出のために改良することができるようになる。これにより、正常と異常の境界が非常に非線形である複雑なデータセットにおいて、異常を識別することが可能となる。本章では、転移学習モデルの特徴量として事前学習済み BERT から得られた分散表現を使用する。

異常検出向けに、線形またはシグモイド活性化 $g(\cdot)$ を持つ1つの隠れ層と1つの出力ノードを持つ単純な順伝播ネットワークを設計する。OC-NNの目的関数は次のように定式化できる。

$$\min_{w,V,r} \left\{ \frac{1}{2} \|w\|_2^2 + \frac{1}{2} \|V\|_F^2 + \frac{1}{\nu} \cdot \frac{1}{N} \sum_{n=1}^N \max(0, r - \langle w, g(VX_{n:}) \rangle) - r \right\} \quad (3.2)$$

ここで、 $w$  は隠れ層から出力層までのスカラー出力、 $V$  は入力から隠れ層までの重み行列である。OC-SVMにおけるドット積 $\langle w, \Phi(X_{n:}) \rangle$ をドット積 $\langle w, g(VX_{n:}) \rangle$ に置き換えることで、転移学習モデルから得られた特徴量を活用し、異常検出のための特徴量を洗練させるための追加層を作成することが可能となる。順伝播ネットワークの学習では、 $w$  と  $V$  を標準的な誤差逆伝搬法で、 $r$  は全データを入力した際の決定スコアの  $\nu$  番目の分位数として交互に更新する。

### 3.2.2 Text Autoencoder

本章では、異常検出性能のベースラインとなる手法として Text Autoencoder を使用する。1.2.2.3 節で述べたように、Autoencoder を用いた異常検出では正常データを用いた学習を行い、テスト時に異常データを入力すると再構成誤差が大きくなることを利用して検出を行う。Text Autoencoder では、確率的に文の摂動（単語のシャッフル・欠落・置き換えなど）を行って破壊された入力から元の文を再構成するように学習を行う。エンコーダ・デコーダとして、それぞれ双方向・単方向の LSTM を採用している。本実験では、潜在表現に対して敵対的な学習を行わない、単純な denoising Autoencoder のアーキテクチャを使用する。

### 3.2.3 モデルアーキテクチャ

提案するログデータの異常度算出モデルでは、重みを固定した事前学習済みの BERT エンコーダにログの各行のメッセージを入力し、エンコーダ表現を得る（図 3.1）。エンコーダに入力される際、メッセージは前章で述べた WordPiece トークナイザを用いてサブワード単位に分割される。各トークン化に対して計算されたエンコーダ表現の平均化を行い、得られた分散表現のベクトルをメッセージ全体の特徴表現として使用する。

メッセージの分散表現は 3 層の順伝播ネットワークに引き渡され、最終的に特徴空間における原点からの距離を表すスカラー値へと変換される。本研究では、このスカラ値を異常スコアと呼び、異常なメッセージは各行に対して算出された異常スコアの閾値処理によって検出される。

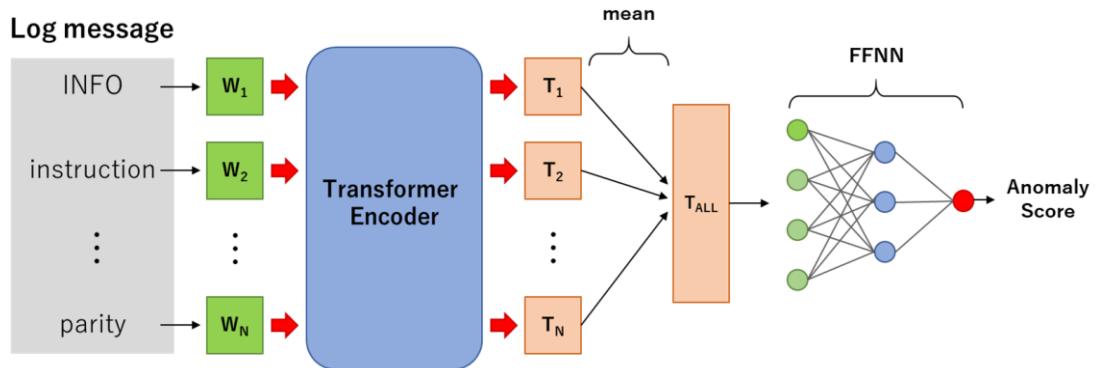


図 3.1 提案するログ異常度算出モデルの構造

### 3.3 実験

提案するログメッセージの異常スコア算出モデルの有効性を確認するため、オープンなログデータセットを用いて異常検出精度の検証を行う。はじめに実験で使用するデータセットの詳細を述べ、つづいて実験の方法と条件について説明する。

#### 3.3.1 データセット

異常検出実験には、カリフォルニア州リバモアにある Lawrence Livermore National Labs (LLNL) の BlueGene/L スーパーコンピュータシステム上で収集された BGL データセット [54] を使用する。ログデータには、タグ付けされたアラート・非アラートメッセージが含まれる。データセットは、AI によるログ解析のためのログデータセット大規模コレクションを提供する Loghub [38], [55] リポジトリで公開されているものである。

#### ● データの前処理

本実験では、BGL データセットから”KERNEL”コンポーネントに関連するメッセージのみを抽出し、提案手法に適したフォーマットへの整形を行う。重複するメッセージは削除され、データ全体を 6 : 2 : 2 の割合で分割し、それぞれをモデルのトレーニング、バリデーション、テスト用のデータとする。データの内訳を表 3.1 に示す。

表 3.1 KERNEL コンポーネントログの正常・異常データ数の内訳

	Training	Test	Validation
Normal	170549	56833	57361
Anomaly	1534	529	508

### 3.3.2 実験方法

提案手法の異常検出精度は、従来のテキスト異常検出手法である Text Autoencoder と比較される。本節では、各手法の実験条件を詳細に述べる。

#### 3.3.2.1 モデルの学習

- 提案手法

本実験では、学習済み BERT<sub>BASE</sub>( $L = 12, H = 768, A = 12$ , 総パラメータ数=110M)のエンコーダ部分を用いてログメッセージを分散表現へと変換する。表 3.2 に、順伝播ネットワークの各層の構成を示す。順伝播ネットワークのワンクラス学習は、以下に示す損失関数に基づいて行われる。

$$L = r + \frac{1}{v} \cdot \frac{1}{N} \sum_{n=1}^N \max(0, \hat{y}_n(w, V) - r) \quad (3.2)$$

ここで、 $w$ と $V$ はそれぞれ順伝播ネットワークの隠れ層と出力層の間の重み、入力層と隠れ層の間の重みを表す。また、 $\hat{y}_n(w, V)$ はシグモイド関数が適用された順伝播ネットワークの最終出力を表す。 $w$ と $V$ の値は、バッチサイズ $N$ のトレーニングデータのミニバッチに対して誤差逆伝搬法を用いて反復的に更新される。 $r$ の値は、エポック終端において順伝播ネットワークの重みを固定した状態で全てのトレーニングデータを入力し、 $\hat{y}_n(w, V)$ の $v$ 番目の分位数として更新される。最適な $v$ の値は、オープンソースのハイパーパラメータ自動最適化フレームワークである Optuna[56]を用いて決定する。表 3.3 に、Optuna を用いて決定したハイパーパラメータの詳細を示す。

表 3.2 順伝播ネットワークの構成

Input (feature dimension)	Hidden layer	Output layer
768	72	1

表 3.3 提案手法のハイパーパラメータ

Parameter	Value	Optimized by Optuna
$v$	2.58e-03	✓
Learning rate	3.19e-05	✓
Epoch	20	✓
Batch size	32	
Max sequence length	128	

### ● Text Autoencoder

異常検出実験における比較対象として、Text Autoencoder を使用する。実装は text-autoencoder リポジトリ [57] とその原論文 [52] を参考とした。提案手法と同様に、WordPiece トークナイザを用いてログメッセージをサブワードに分割してモデルへの入力を行う。また、モデルの学習は一般的な Autoencoder と同様に表 3.1 に示す正常データのみを用いて行う。ログメッセージをモデルに入力した際に得られるクロスエントロピー損失の値を、提案手法における異常スコアに相当する値として異常検出を行う。表 3.4 は、Text Autoencoder モデルを学習する際のパラメータを示している。

表 3.4 Text Autoencoder のハイパーパラメータ

Parameter	Value
Model type	Denoising Auto-Encoder
Learning rate	5e-4
Epoch	2
Batch size	32
Max sequence length	128
Embedding dimension	512
Hidden state dimension	128
Number of layers	1

### 3.3.2.2 精度の比較

精度比較では、学習後の各モデルを用いてテストデータにおける異常検出精度を検証する。各モデルは、受信者操作特性 (ROC: Receiver Operating Characteristic) 曲線の曲線下面積(AUC: Area Under the ROC Curves)と F-measure の値を使用して分類性能を評価する。AUC 値はモデルの分類性能の閾値に対するロバスト性を、F-measure は検出の正確性と異常検出数のバランスを示す評価指標である。ここで、F-measure は以下のように計算される：

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (3.3)$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (3.4)$$

$$F - \text{measure} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (3.5)$$

ここで、

$TP$ : モデルが正確に分類した異常インスタンス

$TN$ : モデルが正確に分類した正常インスタンス

$FP$ : モデルが誤分類した正常インスタンス

$FN$ : モデルが誤分類した異常インスタンス

である。

F-measure を算出する際には、ROC 曲線において  $sensitivity - (1 - specificity)$  の値が最大となる点（カットオフ点）における閾値を使用する。 $sensitivity$  と  $specificity$  は、それぞれ以下の式で定義される。

$$sensitivity = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (3.6)$$

$$specificity = \frac{\text{TN}}{\text{TN} + \text{FP}} \quad (3.7)$$

### 3.3.3 実験結果

表 3.5 に、テストデータを使用して各モデルに対して算出した AUC 値を示す。表 3.6 と表 3.7 は、それぞれ提案手法と Text Autoencoder に対して F-measure とその他のスコアを計算した結果を示す。これらの結果は、いずれの手法においてもモデルが最大のパフォーマンスを示すパラメータを使用したものであり、提案手法は AUC 値と F-measure のいずれにおいても Text Autoencoder を上回る値が得られた。

表 3.5 AUC 値の比較

Proposed Method	LSTM AutoEncoder
0.823	0.786

表 3.6 提案手法の F-measure

		Precision	Recall	F-measure	# of data
Class	Normal	1.00	0.82	0.90	56833
	Anomaly	0.04	0.82	0.08	529
	Accuracy			0.82	
	Macro Avg.	0.52	0.82	0.49	57362
	Weighted Avg.	0.99	0.82	0.89	

表 3.7 Text Autoencoder の F-measure

		Precision	Recall	F-measure	# of data
Class	Normal	1.00	0.76	0.86	56833
	Anomaly	0.03	0.75	0.05	529
	Accuracy			0.76	
	Macro Avg.	0.51	0.75	0.46	57362
	Weighted Avg.	0.99	0.76	0.85	

### 3.4 結論

本章では、教師なし学習によってログデータの各行メッセージに対して異常スコアを算出する深層学習モデルを提案した。BGL ログデータセットを用いた異常検出精度の評価では、Text Autoencoder と比較してより効率的にログデータの異常なメッセージを検出できることが分かった。異常スコアを開発者向けの可視化 GUI ツールにおいて使用することで、大規模なログからより短い時間で不具合に関連するメッセージを見つけることができると期待される。一方で、提案手法はハイパーパラメータである $\nu$ や学習率の値に対して非常に敏感であり、データセットのサイズが変化した際には同じ条件が最適でない可能性があることに注意が必要である。今後の課題として、更新されたデータセットに対して安定した学習を行うための戦略を開発する必要があると考えられる。

## 第4章 時間記憶に基づくログデータの異常検出

### 4.1 はじめに

複雑なシステムの不具合では、実行時のログデータを手がかりとして原因を調査する必要がある。システムが output するログデータの量はますます膨大になっており、不具合に関連するログデータのみを抽出する自動ログ解析手法が求められている。本章では、システム正常動作時と異なるログパターンを自動的に検出する手法 TM-LAD (Temporal Memory based Log Anomaly Detection) を提案する。実験では、loglizer ベンチマーク [28] を用いて 7 つの既存手法と異常検出の性能を比較する。

### 4.2 提案手法の概要

本研究では、構造化されたログシーケンスの時間パターンを学習し、記憶されたパターンとテストケースのパターンのずれをもとに異常度を算出することで異常なログデータの検出を行う。提案手法の概要を図 4.1 に示す。まず、ログメッセージのシーケンスはログパーサと呼ばれるアルゴリズムを用いて構造化されたのち、時間記憶 (TM: Temporal Memory) が理解できる入力形式へと変換される。また、TM に構造化されたログシーケンスを入力する際のデータエンコーディングの方法と時系列パターンの学習については後節で詳しく述べる。モデルの学習と異常度の算出は同時並行で行われ、出力された異常度の数値列に対して予め設定しておいた閾値を用いて異常データの判定を行う。

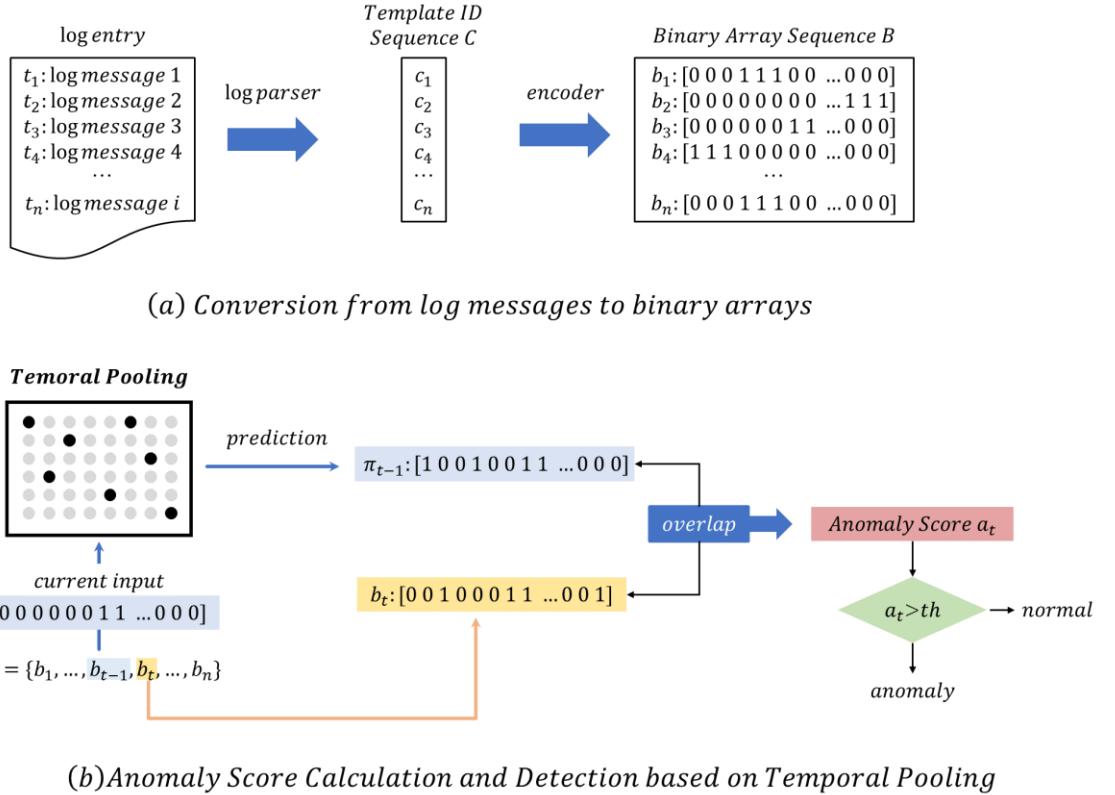


図 4.1 時間記憶によるログ異常検出手法の概要

### 4.3 ログのエンコーディング処理

本研究では、オンライン学習手法である Temporal Memory と相性がよいオンラインログパーサ（例：Drain 等）を想定している。提案手法では構造化されたログデータのテンプレート情報のみを用いて時系列パターンの学習を行う。

ログデータに含まれるテンプレートの辞書を作成し、各テンプレートには整数値の ID が割り振られる（図 4.2）。ここで、辞書には OOV(Out of Vocabulary) と Pad (Padding Token) という値があらかじめ登録されている。OOV は学習データ中に登場しなかったテストデータのあらゆるテンプレートに割り振られ、Pad はテンプレートシーケンスの長さを調整する際のパディングに使用される。ログパーサによって整数テンプレート ID 列に変換されたログシーケンスは、設定された最短の長さに満たない場合、末尾に Pad を追加することでパディングされる。これは TM が予測を行うのに十分な時系列長を確保することが目的である。

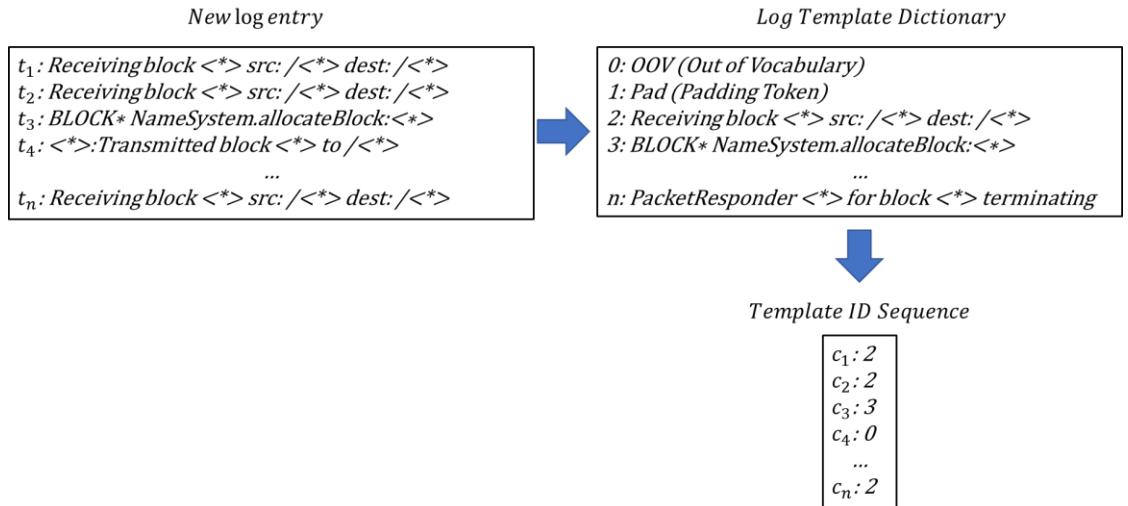


図 4.2 ログエントリのテンプレート ID 変換

Temporal Memoryに入力されるデータは、SDR（固定長のバイナリ配列）に変換される必要がある。ここで、バイナリ配列で値が”1”となる要素をアクティブな要素、”0”となる要素を非アクティブな要素と呼ぶ。SDRは、アクティブな要素が全要素数に占める割合が十分に小さい、すなわち”疎”な状態である必要がある。本研究では、RDSE (Random Distributed Scalar Encoder)[58]と呼ばれるアルゴリズムを用いて整数テンプレート ID を SDR に変換する。ここで、SDR の長さは TM のカラム数と同一の値をとる必要がある。

RDSE ではまず、入力された値を解像度と呼ばれる幅でバケット単位に分割し、各バケットに対して一意のエンコードを行う。バケットのインデックスをハッシュ関数によって変換し、SDR の長さで剩余をもとめることによって SDR で”1”となる要素の位置を求める。そして、アクティブな要素が指定した数に達するまでオフセットをインデックスに加えながら以上の操作を繰り返す。RDSEのエンコード処理を python で実装した疑似コードを図 4.3 に示す。実験では、ハッシュ関数として MurmurHash3[59]を使用する。

```

def random_distributed_scalar_encoder(input, encode_size, resolution, active_bits):
    """
    Args:
        input (float): RDSE で変換する入力値
        encode_size (int): SDR の長さ
        resolution (int): バケットの幅 (解像度)
        active_bits (int): SDR で値が”1”となる要素数
    Returns:
        SDR (ndarray): 変換後の SDR を表す配列
    """
    SDR = numpy.zeros(encode_size) # ゼロ行列を初期化
    index = input / resolution # バケットのインデックスを求める

    for offset in range(active_bits): # 指定要素数に達するまで以下をループ
        hash_buffer = index + offset # インデックスにオフセットを加える
        bucket = Hash(hash_buffer) # ハッシュ関数でインデックスを変換
        bucket = bucket % encode_size # “1”にする要素のインデックスを求める
        SDR[bucket] = 1

    return SDR

```

図 4.3 RDSE の python 疑似コード

#### 4.4 HTM アルゴリズム

本研究では、時系列パターンベースのログ異常検出における新しいアプローチとして、階層的時間記憶 (HTM: Hierarchical Temporal Memory) アルゴリズム[60]に基づく方法を検討する。HTM は、大脳新皮質の多くの構造的・アルゴリズム的特性をモデル化した理論的な枠組みである。HTM ネットワークは何千ものシナプスを持つ非線形のアクティブな樹状突起を組み込んだニューロンモデルを用いて、時間的なシーケンスをオンラインで連続的に学習し過去に見たシーケンスから予測を生成することができる。いくつかの実世界のリアルタイム・ストリーム・データセットに適用され、HTM ネットワークは異常検知やシーケンス予測タスクにおいて優れた性能を発揮することが示されている[61]。

##### 4.4.1 HTM の入力表現

HTM の入力表現として、疎分散表現 (SDR: Sparse Distributed Representation)[62], [63]が採用されている。SDR は、大きな表現容量を維持しながら、ほとんど干渉することなく異なるアイテムの同時表現を可能にする。また、意味が直接表現にエンコードされているという点で、テキストの ASCII のような標準的なコン

ピュータ表現とは全く異なる。SDR は、大部分が 0 で少数が 1 である大きなビットの配列で構成されている。各ビットは何らかの意味を持っているので、2 つの SDR で”1”ビットが数個以上重なっている場合、その 2 つの SDR は似たような意味を持っていることになる。SDR に変換できるデータであれば、HTM システムを使って幅広い用途に使用することができる[64]。そのため、HTM システムを使用する際の最初のステップは、エンコーダを使用してデータソースを SDR に変換することである。

エンコーダは、データのフォーマットを HTM システムに供給できる SDR に変換する。エンコーダは、ある入力値に対してどの出力ビットを 1 にし、どの出力ビットを 0 にするかを、データの重要な意味上の特徴を捉えられるように決定する役割を担っている。似たような入力値は、非常にオーバーラップした SDR を生成することが期待される。

HTM の空間プーリング (SP: Spatial Pooler) [65]は、感覚入力のストリームを連続的に SDR にエンコードする HTM ネットワークの重要な構成要素である。HTM の空間プーリングは、SDR を用いた下流の計算をサポートする一連の計算特性を達成するように設計されている。これは、①類似した入力を類似した出力にマッピングすることで、入力空間のトポロジーを保存すること、②入力ストリームの統計的变化に継続的に適応すること、③固定のスペース表現を形成すること、④ノイズに頑健であること、⑤フォールトトレラントであること、などの特性である。HTM に不可欠な要素である SP の出力は、エンド・ツー・エンドの HTM システムの性能向上に貢献する。本研究では、画像様特徴量に変換されたログデータの時間的なパターンを SP に基づいて学習することを試みる。

#### 4.4.2 HTM のアーキテクチャ

エンド・ツー・エンドの HTM システムでは、SP は連続的なオンライン方式で入力パターンを SDR に変換する。HTM の時間記憶 (TM: Temporal Memory) [66]は、これらの SDR の時間的シーケンスを学習し、将来の入力に対する予測を行う。HTM ネットワークの 1 つの層は、それぞれがセルの集合を持つミニカラムの集合として構成されている。HTM ニューロンモデルには、新皮質の錐体細胞の樹状突起の特性が組み込まれており、HTM ニューロン上の近位樹状突起セグメントと遠位樹状突起セグメントは異なる機能を持っている（図 4.4）。近位樹状突起で検出されたパターンは活動電

位につながり、ニューロンの古典的な受容野を規定する。ニューロンの遠位シナプスで認識されたパターンは、直接活動電位を起こさずに細胞を脱分極させることで予測として機能する。

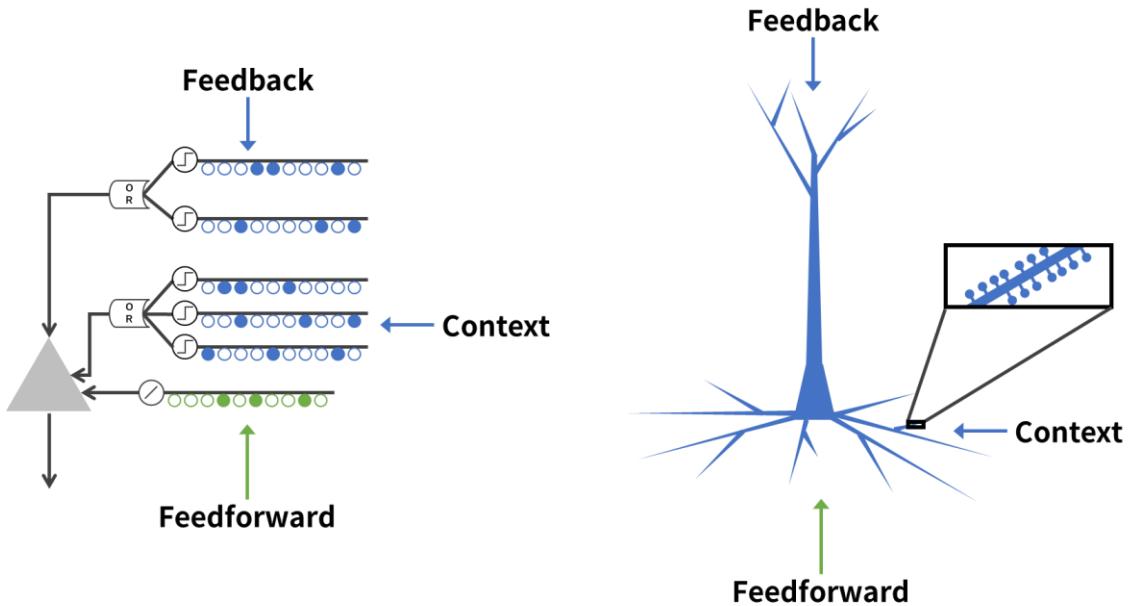


図 4.4 HTM ニューロンモデル

HTM 理論では、ミニカラム内の異なるセルがフィードフォワード入力を異なる時間的文脈で表現する。SP は近位樹状突起セグメントにおけるシナプスの成長をモデル化する。ミニカラムの細胞は同じフィードフォワードの受容野を共有しており、SP はこの共通の受容野が入力からどのように学習されるかをモデル化する。SP の出力は、フィードフォワード入力に応じたミニカラムの活性化を表す。TM は、セルの遠位樹状突起セグメントをモデル化し、時間的な文脈に応じて異なるセルの集合を活性化することで SDR の遷移を学習する。TM の出力は、すべてのミニカラムにまたがる個々のセルの活性化を表す。

時間記憶と空間プーリングの詳細は、それぞれ 4.5 節と 5 章において詳述する。

#### 4.4.3 HTM を用いた異常検出

ストリーミングデータでは、システムの統計量は時間の経過とともに変化する可能性があり、これは「概念ドリフト」として知られている問題である。このような場合、モデルは教師なしの自動化された方法で「通常」の新しい定義に適応しなければならない。HTM アルゴリズムは、ストリーミングデータを教師なしつつオンラインで学習できるため、リアルタイムデータの異常検出に非常に適している。

HTM を用いてシステムの異常を検出する研究が、これまでにいくつか行われている。回避型マルウェア検出に関する研究[67]では、EFLAGS レジスタのシーケンスパターンを HTM で学習し、実行ファイルのコーディングシーケンスを予測している。また、システムログストリームの異常検知に関する研究[68]では、メッセージ内の単語を疎分散表現シーケンスに変換し、シーケンスの最後に出力される予測を用いてログメッセージを異常カテゴリに分類している。

本論文では、上記の研究に比べて計算量が少ない入力特徴量を用いてログパターンを学習できる手法を提案し、大規模なオープンデータセットを用いて異常検知の精度を評価する。

## 4.5 Temporal Pooling (時間記憶)

### 4.5.1 概要

HTM ニューロンは、皮質ニューロンと樹状突起の機能に関する非線形シナプス統合を実装している。ネットワーク内の各ニューロンは、単一の樹状突起セグメントを含む近位ゾーンと、独立した樹状突起セグメントの集合を含む遠位ゾーンの 2 つの別々のゾーンを含んでいる。各セグメントはシナプスの集合を維持しており、シナプスの接続先はゾーンによって異なる。近位シナプスは層へのフィードフォワード入力を表し、遠位シナプスは層内の横方向の接続と高次の領域からのフィードバック接続を表している。本章では、单層の時間記憶を使用するため、フィードバック接続は無視している。

各遠位樹状突起セグメントは、層内の他のニューロンからの側方シナプス接続の集合を含む。同時にアクティブになっている接続の数が閾値を超えると、セグメントはアクティブになる。アクティブセグメントは細胞を発火させるのではなく、代わりに細胞を脱分極状態にさせる。このようにして、各セグメントは特定の時間的文脈を検出し、その文脈に基づいて予測を行う。各ニューロンは、アクティブ状態、予測状態、非アクティブ状態の 3 つの内部状態のいずれかをとる。ニューロンの出力は常にバイナリであり、アクティブかどうかを表す。

#### 4.5.2 動作原理

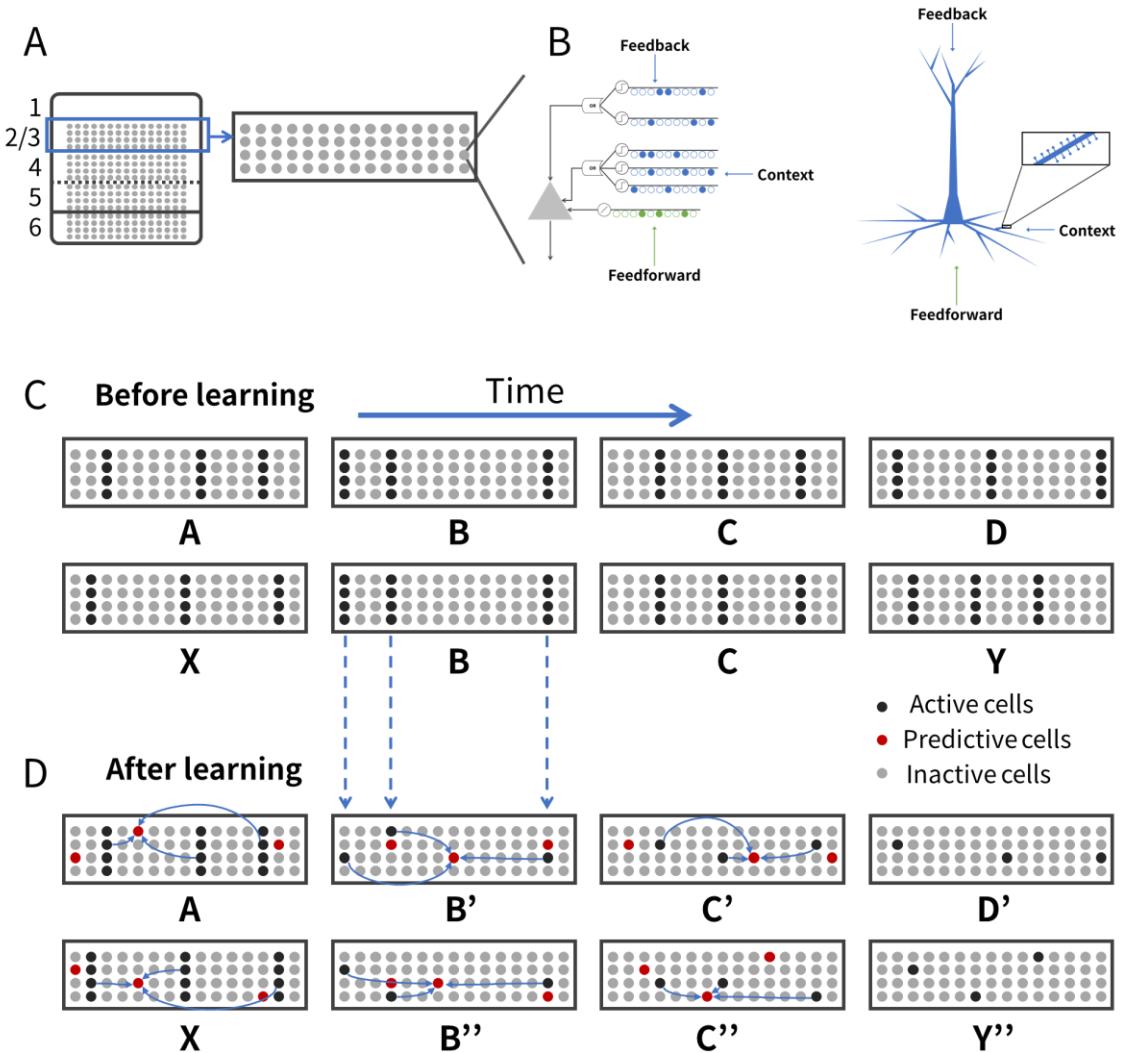


図 4.5 HTM 時間記憶

図 4.5 は HTM の時間記憶の概要と動作原理を示している。

(A) 大脳皮質は 6 つの細胞層で構成されている。各細胞層は、複数の細胞を含むミニカラムの集合からなり、各ミニカラムは複数の細胞を含む。

(B) HTM ニューロン（左）は、錐体ニューロン（右）の樹状突起の異なる部分に對応する 3 つの異なる樹状突起統合帯を持っている。HTM ニューロンは、樹状突起と NMDA スパイクをシナプスの集合を持つ同時検出器の配列としてモデル化されている。遠位樹状突起上の一連のシナプスの共活性化は、NMDA スパイクを引き起こし、ソーマを脱分極させる（予測状態）。

(C, D) 共有部分系列を持つ高次マルコフ配列 (ABCD 対 XBCY) の学習。各シーケ

ンス要素は、カラム間の阻害により、疎なミニカラムの集合を呼び出す。

(C) シーケンスを学習する前に、ミニカラム内のすべてのセルがアクティブになる。

(D) 学習後、側方接続を介して脱分極した細胞はより速く活性化し、同じカラム内の他の細胞の発火をカラム内阻害によって阻止する。モデルは2つの同時表現を維持する。1つは現在のフィードフォワード入力を表すミニカラムレベルで、もう1つは入力のコンテキストを表す個々のセルレベルである。異なる細胞は、2つの配列 ( $C'$  と  $C''$ ) で  $C$  に反応するので、 $D$  または  $Y$  のいずれかの正しい高次予測を呼び出すことができる。

HTM ネットワークは、カラムの集合に編成された HTM ニューロンの層で構成されている。ネットワークは、2つの別々のスペース表現を組み合わせて高次のシーケンスを表現する。常に、現在のフィードフォワード入力と前のシーケンス文脈の両方が、疎分散表現 (SDR) を用いて同時に表現される。

第一の表現はカラムレベルである。カラム間抑制機構を介して、各入力要素は任意の時点でのカラムの疎分散活性化として符号化される。常に、最もアクティブなフィードフォワード入力を受け取る上位 2% のカラムがアクティブになる。

第二の表現は、これらのカラム内の個々の細胞のレベルである。任意の時点で、アクティブなカラムの細胞のサブセットは、現在のパターンの学習された時間的文脈に関する情報を表す。これらの細胞は、順番に同じネットワーク内の他の細胞への横方向の投影を介して、今後の入力の予測につながる。細胞の予測状態は、カラム内での抑制を制御する。カラムが予測された細胞を含み、後に十分なフィードフォワード入力を受けた場合、これらの予測された細胞は活性化し、そのカラム内の他の細胞を阻害する。予測された状態にセルがなかった場合、カラム内のすべてのセルがアクティブになる。

#### 4.5.3 学習ルールの定式化

1カラムあたり  $M$  個のニューロンを持つ  $N$  カラムのネットワークを考える。ここでは、時間ステップ  $t$  における活性化状態を  $M \times N$  の二値行列  $\mathbf{A}_t$  で表す。ここで、 $a_{ij}^t$  は  $j$  番目のカラムの  $i$  番目のセルの活性化状態である。同様に、 $M \times N$  のバイナリ行列  $\mathbf{\Pi}^t$  は時刻  $t$  で予測状態にあるセルを表し、ここで  $\pi_{ij}^t$  は  $j$  列目のセルの予測状態である。各シナプスをスカラーの永続値でモデル化し、その永続値が接続閾値以上であればシナプスは接続されているとみなす。 $j$  列目のセルの  $d$  番目のセグメントの永続性を表すために、 $M \times N$  の行列  $\mathbf{D}_{ij}^d$  を使用する。シナプス永続値行列は 0 と 1 の間に拘束されている。接続されているシナプスのみを表すために、2 値行列  $\tilde{\mathbf{D}}_{ij}^d$  を使用する。

ネットワークは、各セグメントが層内のセルのランダムに選択された部分集合に対する潜在的なシナプス（非ゼロの永続値を持つ）の集合を含むように初期化することができる。シミュレーションを高速化するために、すべてのセグメントとすべてのセルにまたがるシナプスの完全な集合を明示的に初期化する代わりに、実行時に貪欲にセグメントを作成する。

神経細胞の予測状態は次のように処理される。樹状突起セグメントが十分な入力を受けると活動的になり、その後すぐにスパイクを起こすことなく細胞体を脱分極させる。数学的には、時間ステップ  $t$  における予測状態は以下のように計算される。

$$\pi_{ij}^t = \begin{cases} 1 & \text{if } \exists_d \|\tilde{\mathbf{D}}_{ij}^d \circ \mathbf{A}^T\|_1 > \theta \\ 0 & \text{otherwise} \end{cases} \quad (4.1)$$

しきい値  $\theta$  はセグメント活性化しきい値を表し、 $\circ$  は要素ごとの乗算を表す。遠位シナプスは、同じ層の以前に活動していた細胞からの入力を受け取るので、過去の入力の文脈情報が含まれており、将来の入力を正確に予測するために使用することができる。

常に、カラム間抑制プロセスは、現在のフィードフォワード入力パターンに最もよくマッチするカラムの疎な集合を選択する。各カラムの活性な近位シナプスの数を計算し、最も多くのシナプス入力を受けるカラムの上位 2% を活性化する。この集合を  $\mathbf{W}_t$  と呼ぶ。

原理的には、近位シナプスも空間競争的学習規則に従って学習中に連続的に適応することができる。予測状態（脱分極状態）にあるニューロンは、同じフィードフォワード入力を受けて他のニューロンよりも競合的に優位になる。

具体的には、脱分極した細胞は、その後に十分なフィードフォワード入力を受けると、他の非脱分極した細胞よりも速く発火する。発火を早くすることで、同じカラム内の隣接する細胞がカラム内抑制で活性化するのを防ぐ。各細胞の活性状態は以下のように計算される。

$$a_{ij}^t = \begin{cases} 1 & \text{if } j \in W^t \text{ and } \pi_{ij}^{t-1} = 1 \\ 1 & \text{if } j \in W^t \text{ and } \sum_i^{ij} \pi_{ij}^{t-1} = 0 \\ 0 & \text{otherwise} \end{cases} \quad (4.2)$$

最初の条件式は、前の時間ステップの間に予測状態にあった場合に、勝者カラムのセルがアクティブになることを表している。勝利したカラムのセルのいずれも予測状態にない場合、第2条件のように、そのカラムのすべてのセルがアクティブになる。

時間記憶モデルの側方接続は、ヘブ学習則のようなルールを使って学習される。具体的には、細胞が脱分極してその後活性化した場合、脱分極の原因となった樹状突起セグメントを強化する。アクティブなカラムのセルが予測されない場合、最もアクティブなセグメントを持つセルを選択し、そのセグメントを強化する。樹状突起セグメントの強化には、非アクティブなシナプスの永続値を小さい値 $p^-$ だけ減らし、アクティブなシナプスの永続値を大きい値 $p^+$ だけ増やすことが含まれる。

$$\Delta D_{ij}^d = p^+ D_{ij}^d \circ A^{t-1} - p^- D_{ij}^d \circ (1 - A^{t-1}) \quad (4.3)$$

$D_{ij}^d$ は、 $D_{ij}^d$ の正のエントリのみを含む2値行列である。すなわち、

$$D_{ij}^d = \begin{cases} 1 & \text{if } D_{ij}^d > 0 \\ 0 & \text{otherwise} \end{cases} \quad (4.4)$$

また、活性化しなかった細胞の活性セグメントに非常に小さな減衰を適用する。

$$\Delta D_{ij}^b = p^{--} \dot{D}_{ij}^d \text{ where } a_{ij}^t = \mathbf{0} \text{ and } \|\tilde{D}_{ij}^d \circ A^{t-1}\|_1 > \theta \quad (4.5)$$

ここで、 $p^{--} \ll p^-$ である。

#### 4.5.4 異常度の算出

予測状態のセルを含むカラムは次の時刻にアクティブになる入力の位置を示しており、これがすなわち次の時刻の入力の予測値のバイナリ表現となる（図4.6）。

ここで、現在の入力を  $b_t$ 、 $b_t$  の予測値を  $\pi_{t-1}$  とした場合に異常度  $s_t$  を以下の式で定義する：

$$a_t = 1 - \frac{\pi_{t-1} \cdot b_t}{|b_t|} \quad (4.6)$$

ここで上式の  $\pi_{t-1} \cdot b_t$  の部分は、入力の予測値と実際の入力のバイナリ表現のオーバーラップを、 $|b_t|$  は  $b_t$  のうち値が”1”となるビット数を表している。すなわち、式全体としては予測の不一致度、言い換えれば予測のずれの度合いを示している。テストデータを用いた異常検出では、学習データから得られた異常度列に対してもっとも良好な精度を与える閾値を使用して閾値判定を行う。

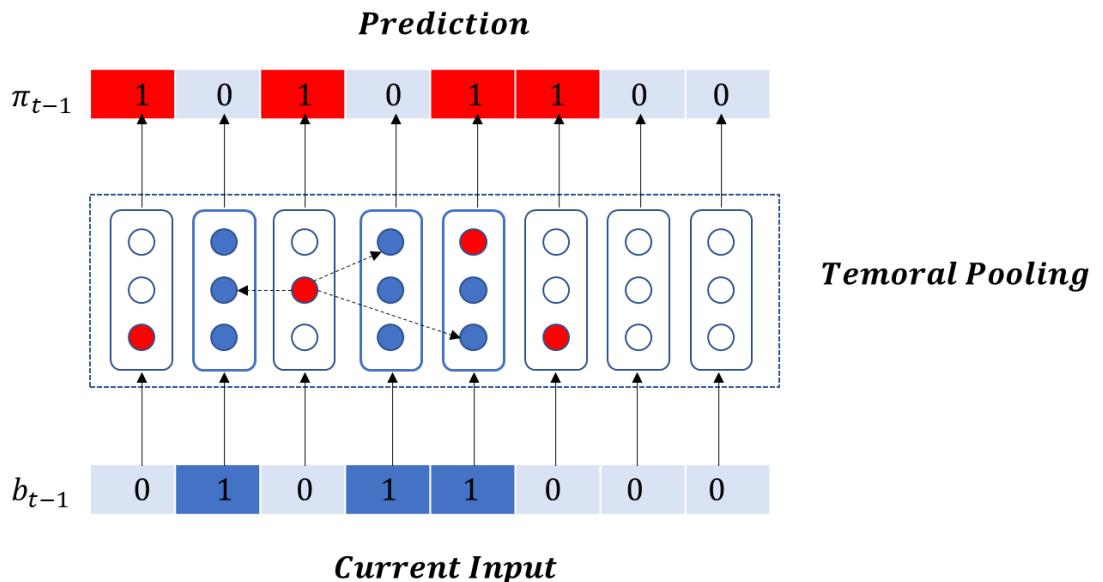


図 4.6 Temporal Pooling の次時刻入力の予測

HTM アルゴリズムで時系列データの認識を行う際には、Temporal Memory 層の前に Spatial Pooler 層によって現在の入力に対してアクティブとなるカラムを計算する。Spatial Pooler を用いる目的としては、類似した入力を類似した出力にマッピングすることで、ノイズにロバストな学習を行うことがあげられる。ただし、入力のバイナリ配列をアクティブなカラムの位置としてとらえることで、TM 単体で時系列学習を行うこともできる。提案手法では、整数のテンプレート ID シーケンスを RDSE によってある程度疎なバイナリ表現に変換しており、Spatial Pooler を省略した場合にも十分な精度が得られることが予備実験で確認された。そこで、よりアルゴリズムの計算量を削減する目的で Temporal Memory のみを用いた時系列パターン記憶を採用する。

## 4.6 実験

TM-LAD の性能を評価するため、大規模なオープンログデータセットを用いて異常検出実験を行う。また、logizer を用いて 7 つの先行手法と性能比較を行う。はじめに実験で使用するデータセットの詳細を述べ、つづいて実験の方法と条件について説明する。

### 4.6.1 データセット

本研究では、loghub リポジトリで公開されているオープンな実世界のログデータである HDFS ログデータセット[35]を使用する。HDFS ログデータセットは Amazon EC2 ノード上で実行された Hadoop ベースの Map-reduce ジョブから生成されたログで、専門家によってラベル付けされている。実験では学習データの冒頭 0.1%~20%を使用してモデルの学習を行い、データの残りの部分を使用して異常検出精度を評価する。表 4.1 にデータセットに含まれる正常・異常インスタンスの内訳を示す。HDFS のラベルはセッションごとに付与されているため、実験の前段階としてログデータをセッション ID ごとに振り分けたスライスに分割する操作（セッションウィンドウの作成）が行われる。提案手法による異常なセッションの識別は、各セッションに含まれるログシーケンスに対して出力した異常度列の平均値を用いて閾値処理によって行う。

#### 4.6.2 実験方法

教師あり学習モデルである LR, Decision Tree, SVM は正常・異常ラベルを使用して学習が行われる。教師なし手法のうち PCA、Invariants Miner、Isolation Forest は学習データのラベルを使用せずにモデルの学習が行われるが、LogClustering と TM-LAD は学習データのうち正常なデータのみを用いる。RDSE エンコーダのパラメータと各モデルのパラメータを表 4.2、表 4.3 に示す。ベンチマークに含まれる 7 つの手法のパラメータは初期状態から一切の変更を加えず、提案手法のハイパーパラメータは予備実験の結果をもとに最も有望と判断されたものを使用する。また、TM-LAD モデルの実装は htm.core ライブライ [69] を用いて行う。

表 4.1 HDFS データセットの内訳

Category	# of instances
Total	575061
Anomaly	16838
Normal	558223

表 4.2 Random Distributed Scalar Encoder のパラメータ

Parameter	value
resolution	0.310
size	1600
category	true
sparsity	0.22

表 4.3 TM-LAD モデルのハイパーパラメータ

Parameter	value
Minimum token length	10
columnDimensions	0.31
cellsPerColumn	8
activationThreshold	11
initialPermanence	0.22
connectedPermanence	0.14
minThreshold	8
maxNewSynapseCount	18
permanenceIncrement	0.081
permanenceDecrement	0.080
predictedSegmentDecrement	0.0
maxSegmentsPerCell	161
maxSynapsesPerSegment	48

#### 4.6.3 実験結果

図 4.7～図 4.9 は、トレーニングに使用するデータの割合を 0.2、0.1、0.01、0.001 (20%、10%、1%、0.1%) の間で変化させた場合の各モデルのメトリクスを示している。ここで、図中の割合は、トレーニングとテストに使用したデータの分割比率を示している（例えば、20/80 は、トレーニングに 20%，テストに 80% のデータを使用したこと示している）。また、提案手法は図中で TM-LAD (1600-0.22) と表されており、RDSE のパラメータとして size=1600、sparsity=0.22 であることを示している。提案手法では、学習データ量の変化に対して Precision, Recall ともに安定しており、総合的な精度を表す F-measure の値がベンチマークの他の手法よりも安定していることがわかる。LR, SVM, 決定木などの教師あり学習法は、非常に高い Precision が得られる一方で、学習データ量が非常に少なくなると Recall の値が急激に減少する (rate : 0.1/99.9)。教師なし学習法である LogClustering は精度が非常に高く安定しているが、Recall の値が他の手法よりも低いレベルで一定しているため、ベンチマークでは F-measure が低くなっている。したがって、多数の誤報が許容できない場合には LogClustering を使用し、多少の誤報は含まれるが、より多くの異常を検出したい場合

には TM-LAD を使用することが有効であると考えられる。

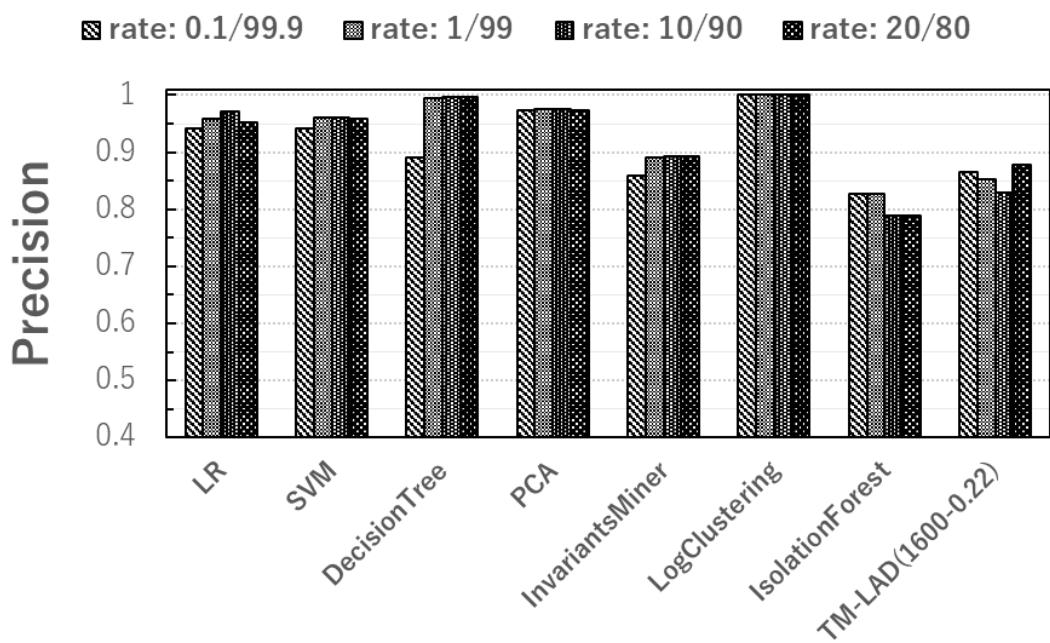


図 4.7 各学習データ量に対する Precision の評価結果(0.1~20%)

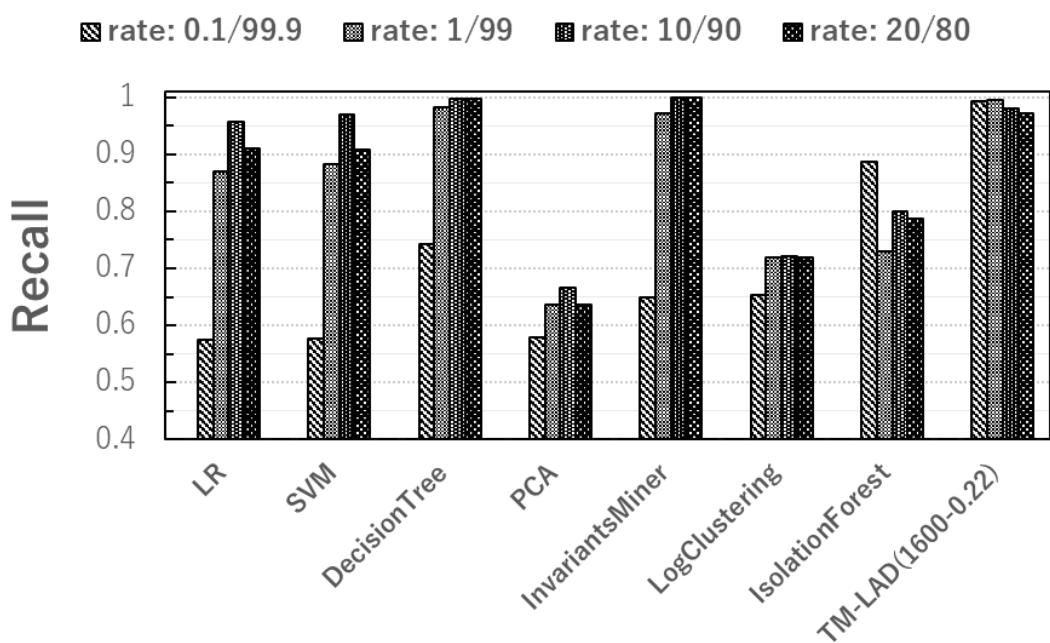


図 4.8 各学習データ量に対する Recall の評価結果(0.1~20%)

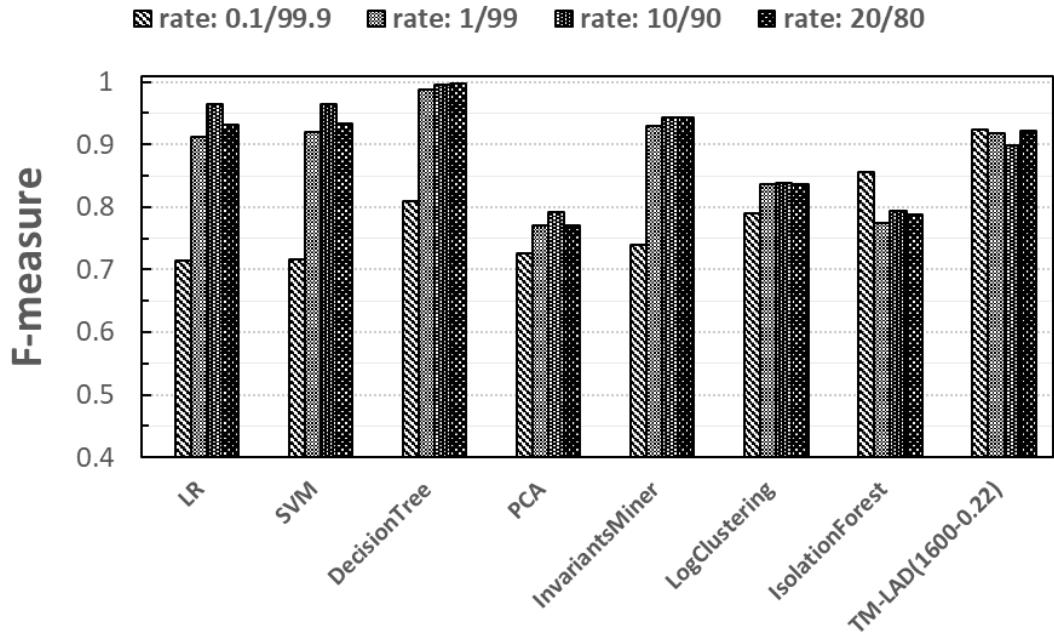


図 4.9 学習データ量ごとの F-measure の評価結果(0.1~20%)

#### 4.6.4 パラメータ設定の影響

HTM アルゴリズムでは、設定すべきハイパープラメータが多岐にわたるため、ユーザが使いたいデータに合わせて最適化することが重要な課題となる。ここでは、異常検知の指標に寄与すると考えられるモデルサイズと入力データのスパース性について、その効果を検証する。

##### 4.6.4.1 モデルサイズ

今回の実験では、入力データのスパース性を 0.15 に固定し、前節と同様に 500, 800, 1600, 2048 の 4 種類のモデルサイズ（カラム数）で異常検知性能を評価した。精度評価の結果を図 4.10～図 4.12 に示す。カラム数と精度には明確な相関関係はないが、カラム数に関わらず、学習データ量が多い場合には精度が向上する傾向にある。また、適切なモデルサイズ（1600 cols）を選択することで、学習データ量が多いほど Recall が向上する可能性が示唆された。F-measure は、学習データ量やモデルサイズによって変化するものの、0.8~0.9 程度に落ち着くことがわかった。

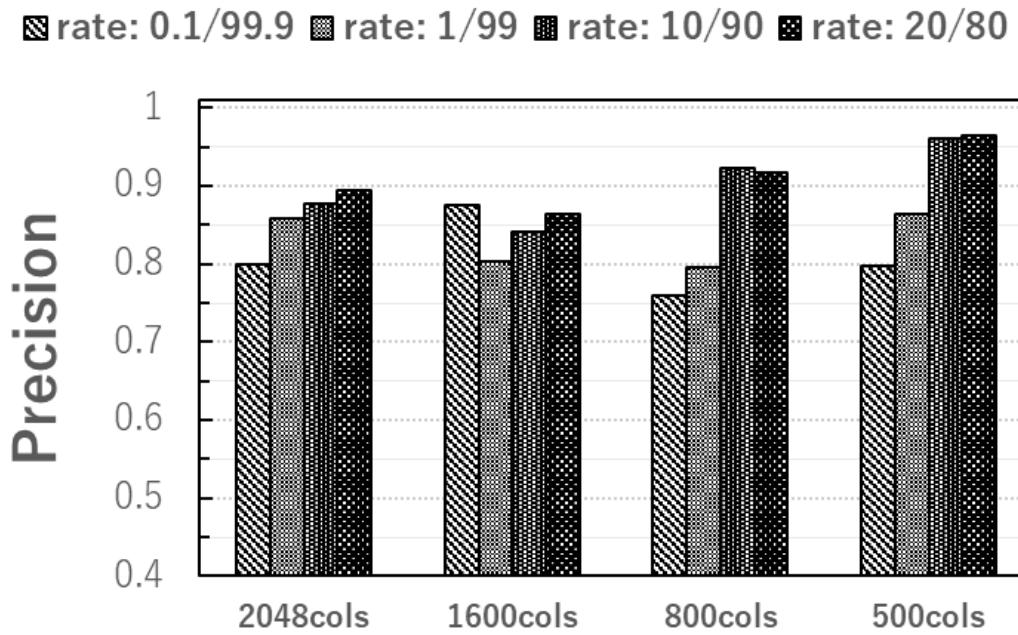


図 4.10 各モデルサイズ (500～2048cols) における Precision の評価結果

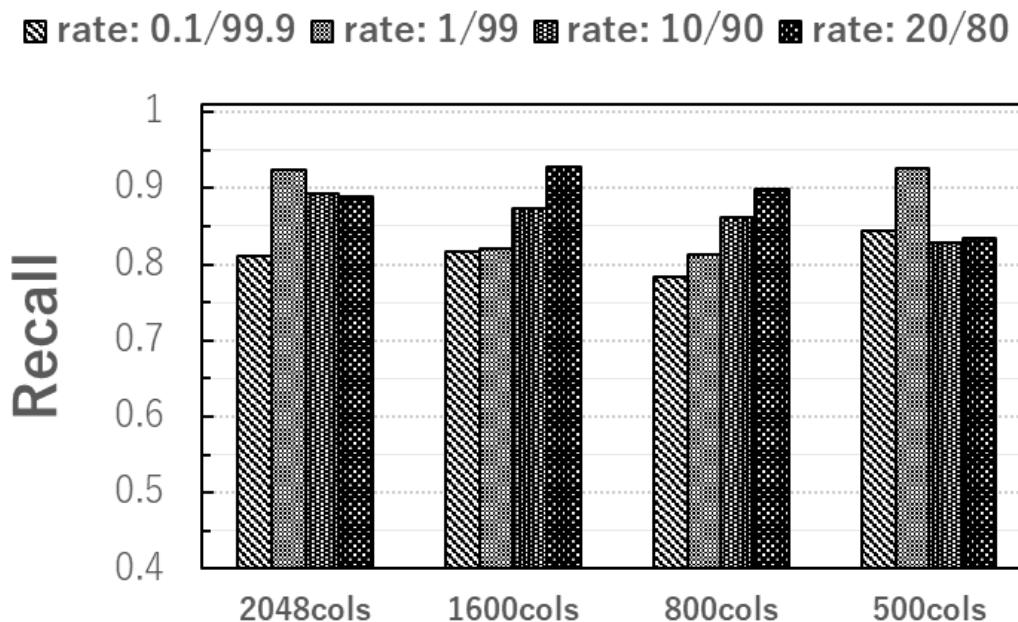


図 4.11 各モデルサイズにおける Recall の評価結果 (500～2048cols)

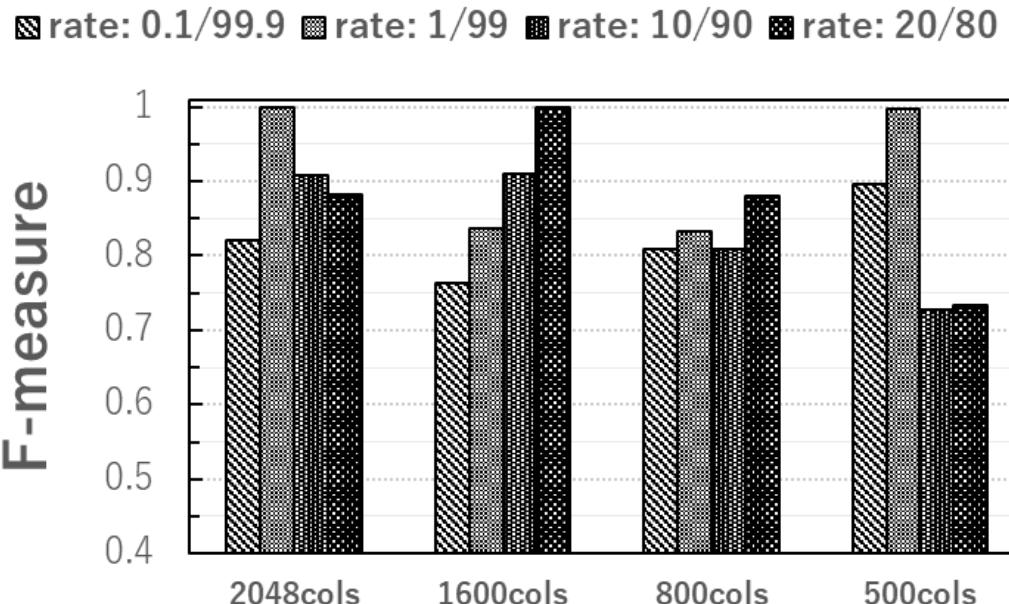


図 4.12 各モデルサイズでの F-measure の評価結果(500~2048cols)

#### 4.6.4.2 入力データのスパース性

提案手法では、HTM アルゴリズムの Temporal Pooling 層のみを使用するため、入力データのスパース性が学習に与える影響を制御する必要がある。本実験では、カラム数を 1600 に固定し、前節と同様に 0.02、0.11、0.15、0.22 の 4 種類のスパース性に対して異常検知性能を評価した。その評価結果を図 4.13～図 4.15 に示す。Precision に関しては、スパース性との明確な相関は見られなかったが、0.8~0.9 程度で落ち着くことが分かった。Recall に関しては、スパース性の値が小さい場合 (0.02, 0.11)，学習量の増加に伴って指標が減少するが、スパース性の値が一定以上の場合 (0.15, 0.22)，指標が増加するか、一定のレベルで安定することがわかった。この結果から、F-measure はスパース性の増加とともに増加することが示唆される。

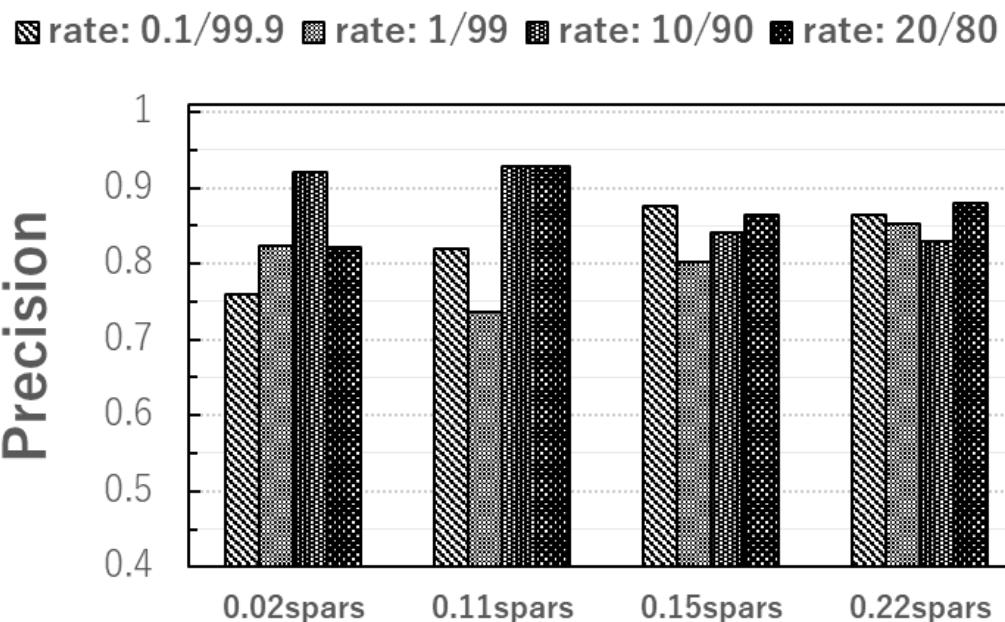


図 4.13 各スパース性(0.02~0.22)における Precision の評価結果

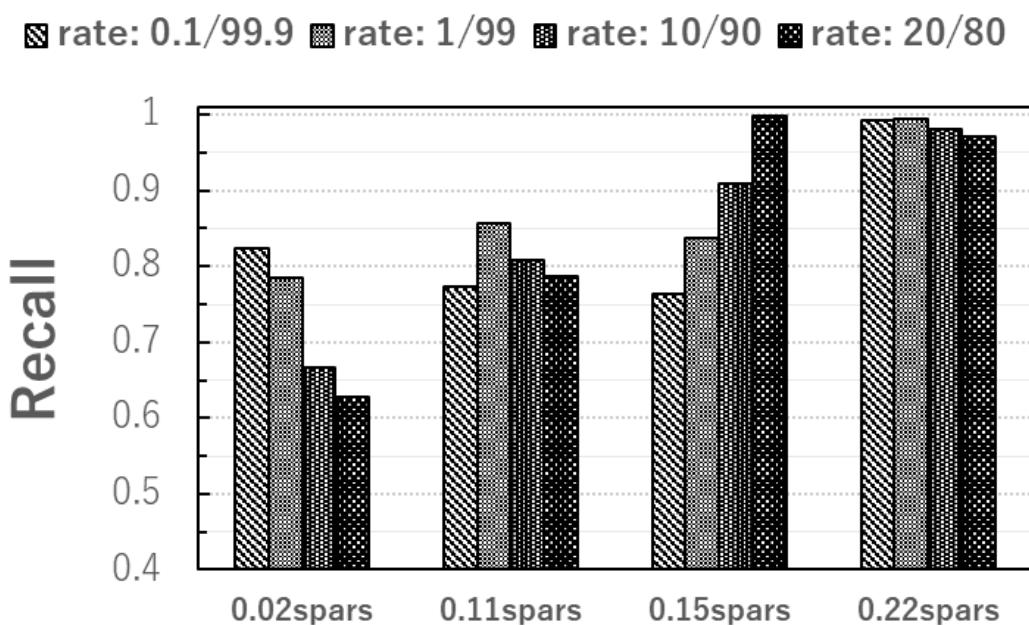


図 4.14 各スパース性(0.02~0.22)における Recall の評価結果

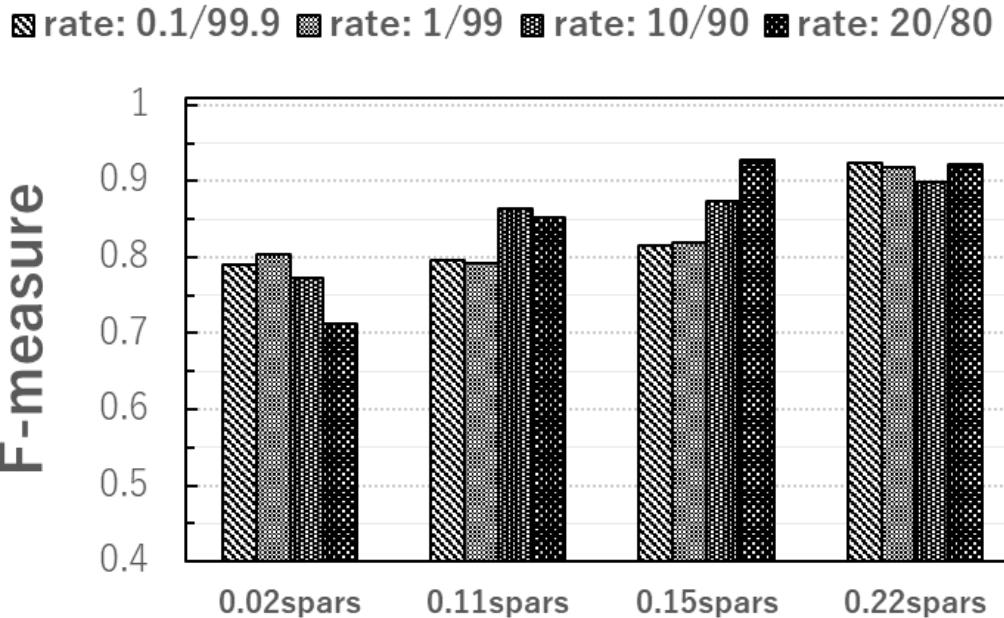


図 4.15 各スパース性における F-measure の評価結果 (0.02~0.22)

また、閾値の変化に対する検出精度の感度を検証するために、各スパースについて AP (Average Precision) を評価した。ここで、Average Precision は以下のように定義される。

$$AP = \int_0^1 p(r) df \quad (4.7)$$

ここで  $p(r)$  は Recall の値を  $r$  としたときの Precision の値を与える関数である。AP の値が 1 に近いほど、しきい値の変動に対して精度が安定していると解釈することができる。

図 4.16 は、Average Precision の評価結果である。その結果、各学習量の AP の値は、スパース性が高いほど高くなる傾向にあることが確認された。以上の結果から、計算量を考慮せずに評価指標ができるだけ高めるためには、モデルサイズとスパース性を大きく設定することが必要であると考えられる。実際には、総処理時間が許容範囲内に収まるパラメータのうち、モデルサイズとスパース性の値が最も大きいパラメータを使用することになる。

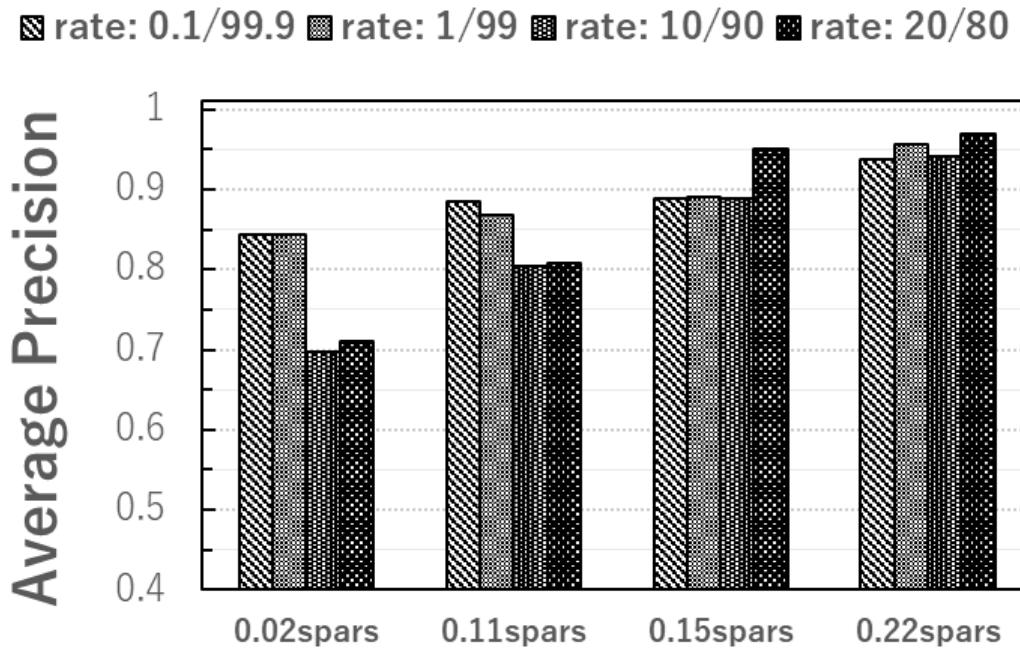


図 4.16 各スパース性における Average Precision の評価結果

#### 4.7 結論

本稿では、時間記憶に基づいて、正常なデータとは異なるパターンのログデータを自動的に抽出する手法である TM-LAD を提案する。提案手法の異常検出では、時間記憶による予測のずれを考慮して、ログデータの全メッセージに対して異常スコアを算出する。閾値処理によって検出された異常データは、エンジニアが障害の原因を調査する際に、前後のデータの異常スコアと関連づけて解釈することができる。

評価実験では、オープンな大規模ログデータである HDFS を用いたベンチマークにより、提案手法がデータ量の変化に対して安定して異常を検出できることを示した。今回の研究により、日々の開発作業に追われているエンジニアが、より少ない労力で解析を自動化できる一つの選択肢を示すことができたと考えている。今後は、ログデータの種類に応じてパラメータを自動的に最適化する手法の開発や、並列実装による処理時間の短縮などを予定している。

# 第5章 空間プーリングに基づくログデータの異常検出

## 5.1 はじめに

現代のソフトウェアのログを有効活用するために、ログの自動解析に関する研究が盛んに行われている。近年、ログ異常検知の分野では深層学習を用いた手法が数多く提案されており、産業用ログで大きな成果を上げている。一方で、これらの手法は、計算機的に複雑なモデルの学習や、よく整備された大規模なログデータセットを前提としていることが多い。そのため、学術分野では技術が発展しているものの、高性能な計算資源を得るためのコストや、ラベリングなどのデータ準備にかかる人的コストが実装のハードルとなっている。

本章では、CPU 環境での学習に適した、疎な特徴と内部表現を用いたモデルを提案する。提案手法は、1 つの空間プーリング層と分類用の単層ニューラルネットワークで構成されており、画像様の特徴に変換されたログデータから異常なパターンを識別する。スーパーコンピュータの大規模なログデータセットを用いたベンチマーク結果から、本手法は最先端の深層学習を用いた手法と競合する精度を達成でき、データ量が少なくとも高い精度を維持できることがわかった。

## 5.2 提案手法の概要

本節では、我々が提案するログベースの異常検知手法である SPClassifier について詳細に説明する。提案手法のパイプラインを図 5.1 に示す。まず、システムから収集された生のログデータは、ログパーサによって構造化されたログメッセージに変換され、抽出されたログ・イベント・テンプレートには一意のインデックスが割り当てられる。次に、ログメッセージのシーケンスは、テンプレートのインデックスを使用して、整数の時系列データに変換される。さらに、固定長の時間窓を一定時間ごとにまたぐようにして整数値の配列を切り出し（スライディング・パーティション）、各パーティションに画像のような特徴へのエンコードを施す（図 5.1 (b)）。符号化された画像は、空間プーリングの後フラット化された出力表現として单層のニューラルネットワーク

に渡され、異常を識別する。以下では、特徴変換・空間プーリング・分類の詳細な処理について説明する。

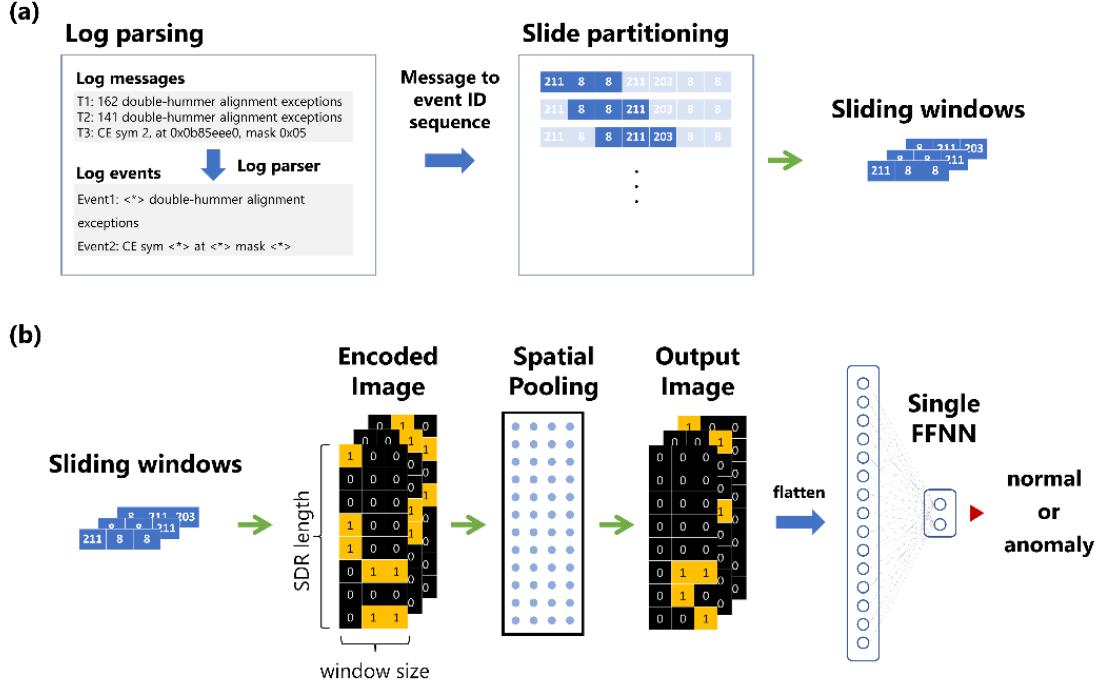


図 5.1 提案手法のパイプライン

(a) 生のログメッセージから整数シーケンスへの変換は、ログイベントに割り当てられたインデックスを用いて行われる。変換されたシーケンスにスライディング・パーティションが適用され、データは固定長の時間窓に分割される。(b) 各時間窓に含まれる整数値は、疎分散表現に変換され、ログシーケンスの時間方向に積層された画像としてエンコードされる。符号化された画像は、空間プーリング層で処理された後、入力と同じフォーマットの SDR として出力される。出力された SDR はフラット化され、単層のニューラルネットワークによってデータのラベルが予測される。

### 5.2.1 特徴量変換

提案手法におけるログシーケンスの特徴変換では、各ログイベントに割り当てられた特徴量を時間方向に積み重ね、(特徴量の次元数×ログシーケンス長) の画像のような特徴量を生成する。ここで、各ログイベントに割り当てられた特徴量は、SDR と呼ばれる固定次元のバイナリ配列である。SDR は、配列中のある割合のビットが 1 (オ

ン) になり、残りのビットが 0 (オフ) になる、疎なビット配列である。オンになるビットは、テンプレートインデックスのハッシュ値に基づいて計算される。各ログイベントに対する SDR は、4 章と同様に RDSE によって変換される。

### 5.2.2 空間プーリング

空間プーリング(SP: Spatial Pooling または Spatial Pooler)は、HTM ニューロンモデルを用いて入力データの空間的特徴を処理する層である。4 章では、時間記憶 (TM) を用いてログの異常を検出する手法を提案した。しかし、時間記憶に基づく方法では、インターリープされたログが非常に長い場合に時間的な情報を効率的に学習することができない。本章では、スライディングウィンドウを用いた画像様特徴を用いて、空間プーリング層のみでの時間パターン学習を実現する。

SP は、隣接するミニカラム間の局所的な抑制をモデル化しており、k-wins-take-all 計算を実装している。常に、最もアクティブな入力を持つミニカラムのごく一部だけがアクティブとなる。アクティブなセルへのフィードフォワード接続は、各時間ステップでヘブの学習則に従って変更される。恒常的な興奮制御メカニズムは、よりゆっくりとした時間スケールで動作する。このメカニズムは、十分に活動していないミニカラムの相対的な興奮性を高めることから「ブースティング」と呼ばれる。ブースティングは、接続が十分でないニューロンがアクティブになり、入力の表現に参加するよう促す。

空間プーリング層は、与えられた入力に対して似たような反応をするニューロンの集合体であるカラムの集合体として構成される。各 SP ミニカラムは、入力ニューロンの集合にシナプス接続を形成する。入力ニューロンは入力空間にトポロジー的に配置されていると仮定する。 $j$  番目の入力ニューロンの位置を  $x_j$  とし、その活性化状態をバイナリ変数  $z_j$  で表す。入力空間の次元数はアプリケーションによって異なる。例えば、入力が画像の場合に入力空間は 2 次元となり、入力がスカラー値の場合に入力空間は 1 次元となる。さまざまなデータタイプに対応するために、さまざまなエンコーダが用意されている[64]。また、出力ニューロンは異なる空間にトポロジー的に配置されており、 $i$  番目の SP ミニカラムの位置を  $y_i$  とする。

$i$ 番目の SP ミニカラムのシナプスは  $x_i^c$  を中心とする入力空間のハイパーキューブに位置し、エッジ長は  $\gamma$  である。各 SP ミニカラムは、この領域の入力の一部に潜在的な接続を持つ。シナプスの永続性が接続閾値以上である場合にのみシナプスが接続されるので、これらを「潜在的な」接続と呼ぶ。 $i$  番目のカラムの潜在的な入力接続の集合は以下のように初期化される。

$$\Pi_i = \{j \mid I(x_j; x_i^c, \gamma) \text{ and } Z_{ij} < p\} \quad (5.1)$$

$I(x_j; x_i^c, \gamma)$  は、 $x_j$  が入力空間の  $x_i^c$  を中心とした辺の長さが  $\gamma$  であるハイパーキューブ内にある場合に 1 を返す指示関数である。 $Z_{ij} \sim U(0, 1)$  は [0,1] に一様に分布する乱数で、 $p$  はハイパーキューブ内の入力のうち、潜在的な接続をもつ割合である。潜在的接続は一度初期化され、学習中は固定される。

各シナプスをスカラー値の永続値でモデル化し、その永続値が接続閾値以上であればシナプスは接続されていると考える。SP の性能は接続閾値パラメータの影響を受けない。接続されたシナプスの集合を 2 値の行列  $W$  で表す。

$$W_{ij} = \begin{cases} 1 & \text{if } D_{ij} \geq \theta_c \\ 0 & \text{otherwise} \end{cases} \quad (5.2)$$

ここで、 $D_{ij}$  は  $j$  番目の入力から  $i$  番目の SP ミニカラムへのシナプス永続性を与える。シナプス永続値は 0 と 1 の間のスカラー値であり、潜在的なシナプスに対して 0 と 1 の間の一様分布に従って独立同分布するように初期化される。

$$D_{ij} = \begin{cases} U(0, 1) & \text{if } j \in \Pi_i \\ 0 & \text{otherwise} \end{cases} \quad (5.3)$$

隣接する SP ミニカラムは、局所的な阻害メカニズムを介して互いに阻害し合う。ここでは、 $i$  番目の SP ミニカラム  $y_i$  の近傍性を次のように定義する。

$$N_i = \{j \mid \|y_i - y_j\| < \phi, \quad j \neq i\} \quad (5.4)$$

局所抑制は隣接するミニカラム間で行われるため、パラメータ $\phi$ は抑制半径を制御する。局所抑制は、入力空間にトポロジーがある場合、すなわち、隣接する入力ニューロンが入力空間の類似した部分領域からの情報を表す場合に重要となる。抑制半径は、局所抑制が入力空間の同じ領域からの入力を持つミニカラムに影響を与えるように動的に調整される。つまり、平均的な受容野の大きさが大きくなれば、 $\phi$ は大きくなる。

具体的には、 $\phi$ はすべての SP ミニカラムの平均連結入力スパンと、入力ごとのミニカラムの数との積によって決定される。SP 入力とミニカラムが同じ次元であれば、最初は $\phi = \gamma$ である。実際には、カテゴリ情報のような自然なトポロジーを持たない入力空間も扱う。この場合、入力の自然な順序付けではなく、グローバルな抑制を実装するために無限に大きな $\phi$ を使用する。

入力パターン $\mathbf{z}$ が与えられたとき、SP のミニカラムの活性化は、まず各ミニカラムへのフィードフォワード入力（オーバーラップ値 $o_i$ ）を計算することによって決定される。

$$o_i = b_i \sum_j w_{ij} z_j \quad (5.5)$$

ここで $b_i$ は個々のカラムの興奮性を制御する正のブースト係数である。

SP のミニカラムは、フィードフォワード入力が刺激閾値 $\theta_{stim}$ 以上で、かつ近傍の上位 $s$ パーセントに位置する場合にアクティブになる。

$$a_i = \begin{cases} 1 & \text{if } o_i \geq Z(V_i, 100 - s) \text{ and } o_i \geq \theta_{stim} \\ 0 & \text{otherwise} \end{cases} \quad (5.6)$$

通常、十分な入力がないミニカラムがアクティブになるのを防ぐために、 $\theta_{stim}$ を小さな正の数に設定する。ここで、 $Z(\mathbf{X}, p)$ は、区間 $[0, 100]$ 内のパーセンテージ $p$ に対するデータベクトル $X$ の値のパーセンタイルを返すパーセンタイル関数である。 $V_i$ は、 $i$ 番目のミニカラムの隣接するすべてのミニカラムのオーバーラップ値である。

$$\mathbf{V}_i = \{\mathbf{o}_j \mid j \in N_i\} \quad (5.7)$$

$s$ は、目標とする活性化密度である（通常、 $s = 2\%$ を使用する）。活性化ルール（式 5.6-5.7）は、ローカルな近傍領域内の k-wins-take-all 計算を実装している。

SP ミニカラムのフィードフォワード接続はヘブの法則を用いて学習される。シナプス永続値を  $p^+$  増加させることでアクティブな入力接続を強化し、シナプス永続値を  $p^-$  減少させることで非アクティブな接続にペナルティを与える。シナプス永続値は 0 と 1 の境界で切り取られる。

ブースト係数を更新するために、各ミニカラムの最近の活性度をその隣のミニカラムの最近の活性度と比較する。各ミニカラムの過去  $T$  回の入力における時間平均の活性化レベルを次のように計算する。

$$\bar{a}_i(t) = \frac{(T - 1) * \bar{a}_i(t - 1) + a_i(t)}{T} \quad (5.8)$$

ここで  $a_i(t)$  は時間  $t$  における  $i$  番目のミニカラムの現在の活動である。 $T$  はブースト係数の更新速度を制御する。アクティビティはまばらなので、活性化レベルの意味のある推定値を得るには多くのステップが必要となる。

ミニカラムの近傍での最近の活動は、次のように計算される。

$$\langle \bar{a}_i(t) \rangle = \frac{1}{|N_i|} \sum_{j \in N_i} \bar{a}_j(t) \quad (5.9)$$

最後に、ブースト係数  $b_i$  は、 $a_i(t)$  と  $\langle \bar{a}_i(t) \rangle$  の差に基づいて更新される。

$$b_i = e^{-\beta(a_i(t) - \langle \bar{a}_i(t) \rangle)} \quad (5.10)$$

ここで、 $\beta$  は適応効果の強さを制御する正のパラメータである。このメカニズムは、平均発火率が十分に低いミニカラムのゲインを増加させることで、ミニカラムの効率的な使用を促す。

上記のように計算された空間プーリング層全体のカラムの状態は、入力と同様に、2 つの値（オンとオフ）からなる SDR として得られる（図 5.2）。

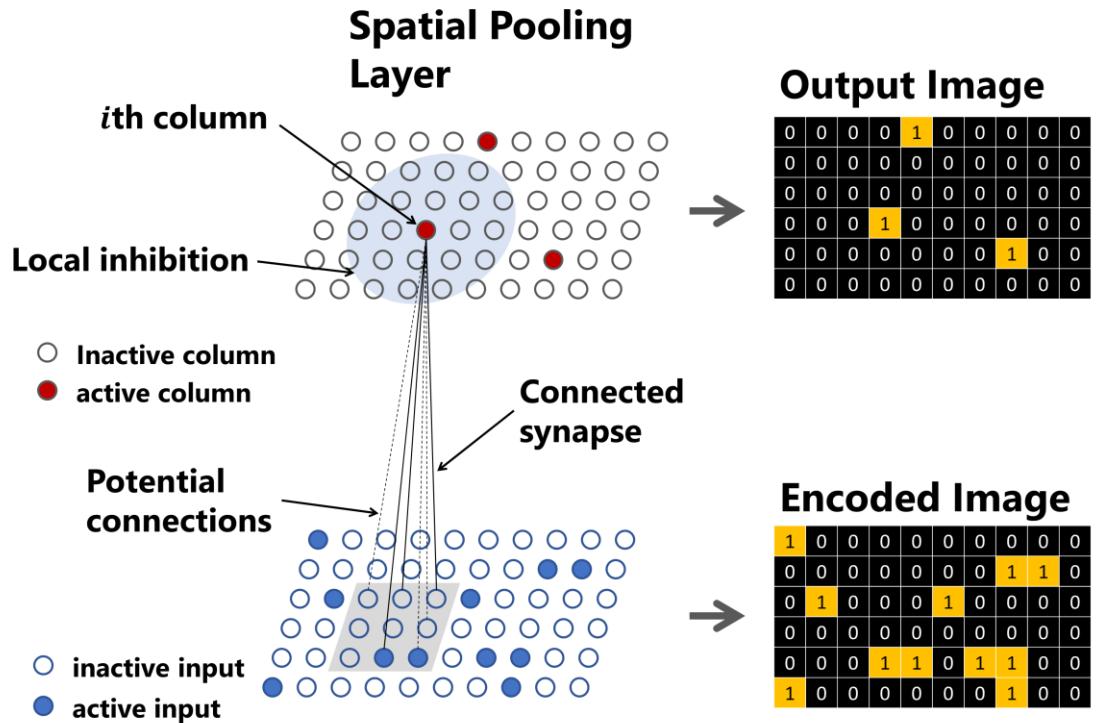


図 5.2 符号化された画像の空間プーリング

### 5.2.3 分類器

単層ニューラルネットワーク分類器は、空間プーリング層の出力であるフラット化された SDR の 2 値配列を入力とし、異常または正常のラベルを予測する。ネットワーク出力の活性化関数には、softmax 関数を採用した。ネットワークの重みは、学習時に与えられたラベルの情報をもとに、以下の式で更新される。

$$\mathbf{w}_{ij} \leftarrow \mathbf{w}_{ij} + \alpha \times \sum_{i=0}^{c-1} \left\{ \frac{1}{c} - \text{softmax} \left( \sum_{i=0}^{c-1} \mathbf{w}_{ij} \mathbf{z}_j \right) \right\} \quad (5.11)$$

ここで、 $\mathbf{w}_{ij}$  は、平坦化された入力  $\mathbf{z}_j$  の  $j$  番目の値と、ニューラルネットワークの  $i$  番目の出力ノードとの間の重みである。 $c$  はカテゴリの数であり、異常検出には正常カテゴリと異常カテゴリが使用されるため、 $c = 2$  となる。 $\alpha$  は、学習の速度を制御する係数である。

### 5.3 実験

提案手法の性能を評価するために、deep-loglizer[70]に実装されている 5 つの深層学習手法に対してベンチマークを行い、その結果を報告する。はじめに実験で使用するデータセットの詳細を述べ、つづいて実験の方法と条件について説明する。

#### 5.3.1 データセット

本実験では、3 章すでに述べた BGL データセットを用いてログデータの異常検出実験を行う。このデータセットを使用するのは、BGL データが冒頭で述べたように様々なコンポーネントのログがインターリープされた状態で提供されているためである。表 5.1 に、BGL データセットに含まれる正常データと異常データの内訳を示す。

表 5.1 BGL データセットの内訳

Parameter	Train	Test
#messages	4045666	664206
#anomalies	298962	48770
#windows	4040575	662938
#templates	340	21

#### 5.3.2 実験方法

まず、BGL データセットをタイムスタンプ情報に基づいて、1 時間ごとの 6 つのセッションに分割する。セッションはランダムにシャッフルされ、そのうちの 80%がトレーニングデータ、残りの 20%がテストデータとして使用される。トレーニングデータとテストデータの両方に、ウィンドウサイズが 10、ストライドが 1 のパーティションを適用し、シャッフルを行う。窓の異常ラベルは、その窓に少なくとも 1 つの異常なメッセージが含まれていれば 1、そうでなければ 0 が割り当てられる。各ベンチマーク手法は、上記の訓練データを用いて 1 エポックの訓練を行った後、テストデータのラベルを予測する。予測結果から得られた TP, FP, FN の測定値に基づいて、Precision, Recall, F-measure の各スコアを算出し、各手法の性能を比較する。ランダム性の影響を考慮して、すべての実験を各手法で 5 回ずつ行い、その平均値をスコアとした。

`deep-loglizer` の各モデルのパラメータは、学習エポック数を除いてデフォルトの状態に保たれている。表 5.2 は、提案手法における特徴変換と空間プーリング層のパラメータを示したものである。すべての実験は、Intel(R) Core (TM) i9-10900X CPU @ 3.70GHz, 64GB RAM を搭載したマシン上で行われ、モデルの学習とテストはすべて CPU 上で行われた。`deep-loglizer` では、ユーザは、テンプレートインデックス（シーケンシャル）と、メッセージに含まれる単語情報に基づいて計算される tf-idf ベクトル（セマンティック）の 2 種類の入力特徴を選択することができる。上記の実験環境では、RAM (64GB) の使用量が上限を超えていたため、セマンティクス特徴量での学習は不可能であり、本実験ではシーケンシャル特徴量のみでベンチマークを行うこととした。

表 5.2 SPClassifier のパラメータリスト

Parameter	Value	Description
Window size	10	スライディングウィンドウの長さ
Stride	1	スライディング・パーティションのステップ数
Input SDR length	500	单一のテンプレートインデックスに対する SDR 変換の次元数
Input SDR sparsity	0.15	SDR 変換後の全次元において 1 の値を持つビットの割合
Input Dimensions	(500,10)	SDR を積層して生成した符号化画像の形状
Column Dimension	(830, 15)	空間プーリング層のカラムを 2 次元的に配置した形状
potentialRadius	7	1 つのカラムが潜在的な接続を持つ入力範囲を決定する値
potentialPct	0.1	ハイパーキューブ内で潜在的な接続を持つ入力の割合
globalInhibition	True	カラムのアクティブな状態を決定する際に、すべてのカラムを近傍として考慮するかどうか
localAreaDensity	0.2	近傍間でアクティブになることができるカラムの割合
stimulusThreshold	6	カラムがアクティブになるために必要なシナプス接続の最小数

synPermActiveInc	0.14	学習ステップごとにアクティブなシナプスの永続値を増加させる量
synPermInactiveDec	0.02	アクティブでないシナプスの永続値が学習ステップごとに減少する量
synPermConnected	0.5	シナプスが接続されているとみなされる最小の永続値
dutyCyclePeriod	1402	各カラムがアクティブになる頻度に基づいて ブースト係数を更新する際に考慮される時間
minPctOverlapDutyCycles	0.2	ステップの長さ カラムが <code>stimulusThreshold</code> 以上のアクティブな入力を行う頻度の下限値
boostStrength	7	ブーストファクターの適応効果の強さを制御するパラメータ
wrapAround	False	入力とカラムの間のマッピングにおいて、入力の最初の次元と最後の次元が隣接しているとみなされるかどうか

提案手法における特徴変換、空間プーリング層、分類層の実装は、HTM アルゴリズムのコミュニティフォークである `htm.core` ライブラリ[69]を用いている。また、テンプレートインデックスに変換されたログデータのスライディング・パーティションは、`deep-loglizer` の特徴抽出機能を用いて実現している。

ベンチマークの性能評価指標として、異常検知で最も広く使われている `Precision`、`Recall`、`F-measure` を使用する。`Precision` は異常なウィンドウを正しく予測した割合、`Recall` は全ての異常なウィンドウを異常だと検出した割合、`F-measure` は `Precision` と `Recall` の調和平均として計算される。各指標の定義は 3 章と同様である。

### 5.3.3 実験結果

図 5.3 は、BGL データセットを用いた異常検知性能のベンチマーク結果である。この結果から、提案手法である SPClassifier は、教師あり学習手法である Attn-BLSTM と CNN に次いで、3 番目に良い F-measure を達成できることがわかる。また、教師なし学習法は、教師あり学習法ほどの精度は得られないことがわかる。

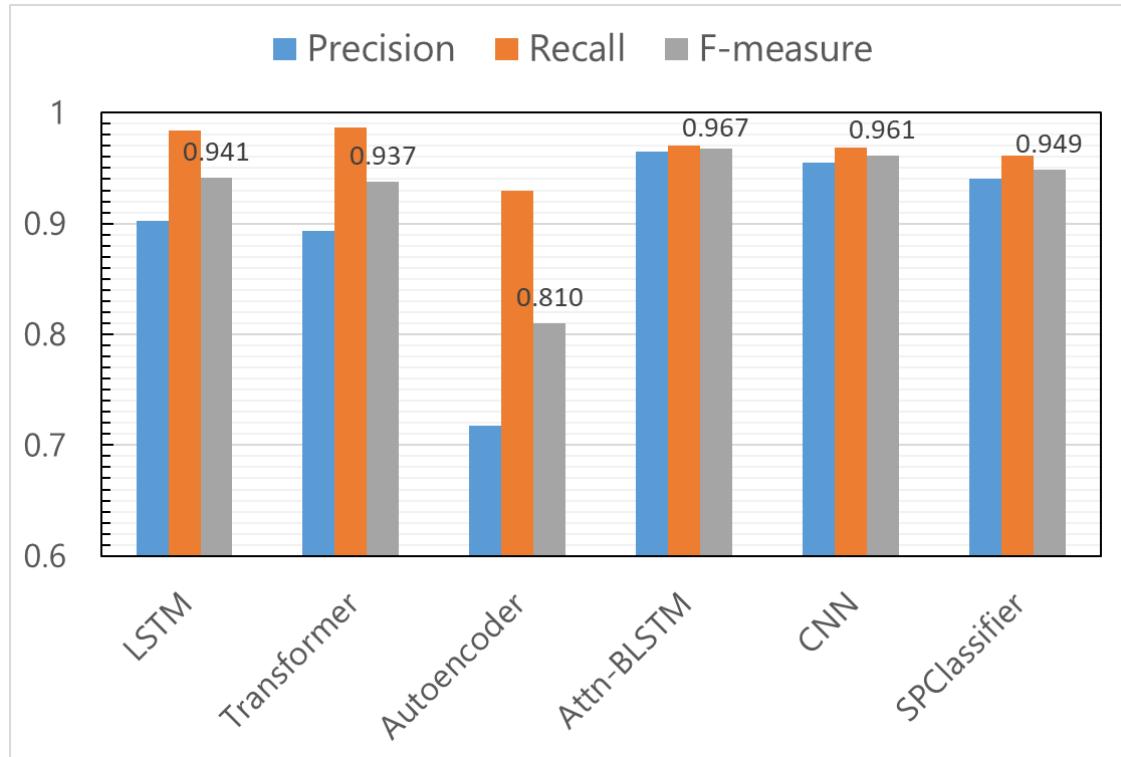


図 5.3 異常検知のベンチマーク結果

#### 5.3.4 学習データ量の影響

データ量が異常検出性能に及ぼす影響を調べるために、追加実験を行う。分割率を10%から100%の間で10%刻みで変化させて、学習データのサブセットを作成する。サブセットを用いて各モデルを学習し、4.3節と同じ量のテストデータを用いて異常検知ベンチマークを行う。

図5.4～図5.6は、各学習サブセットに対する追加実験の結果を示したものである。教師付き学習手法であるAttn-BLSTM, CNN, SPClassifierは、学習データの30%以下のサブセットを用いても、F-measureとprecisionの値を大きく低下させないことがわかった。また、提案手法は、サブセットが10%と極端に少ない場合には深層学習手法を上回るが、データ量が増えると他の教師付き手法の精度が大きくなることがわかった。

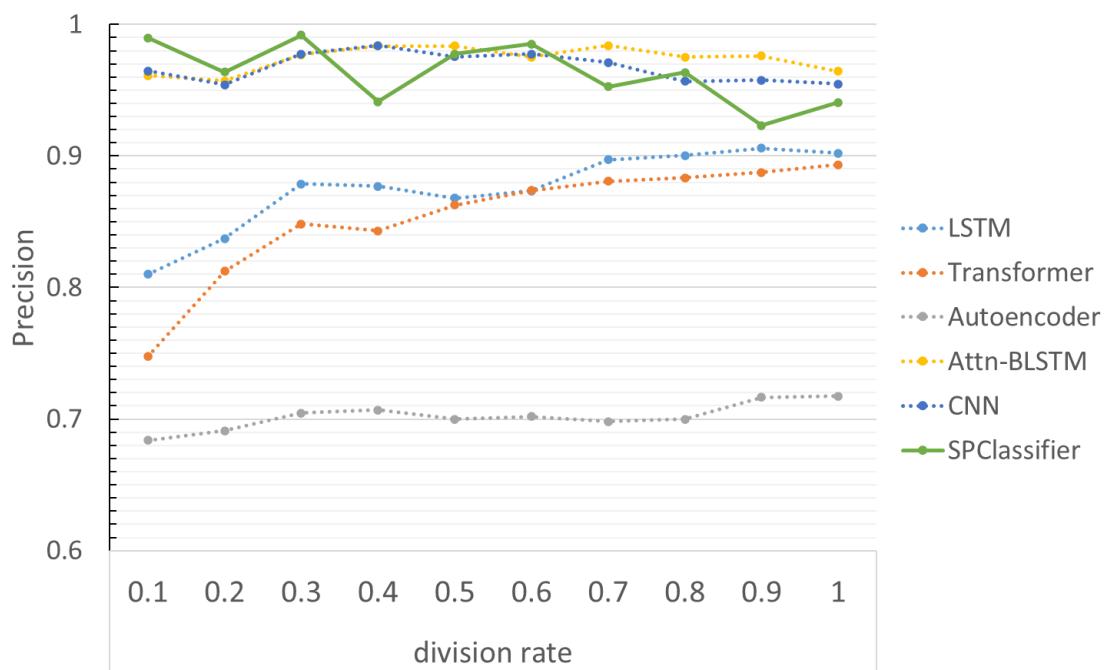


図5.4 訓練サブセットを用いたベンチマークの結果(Precision)

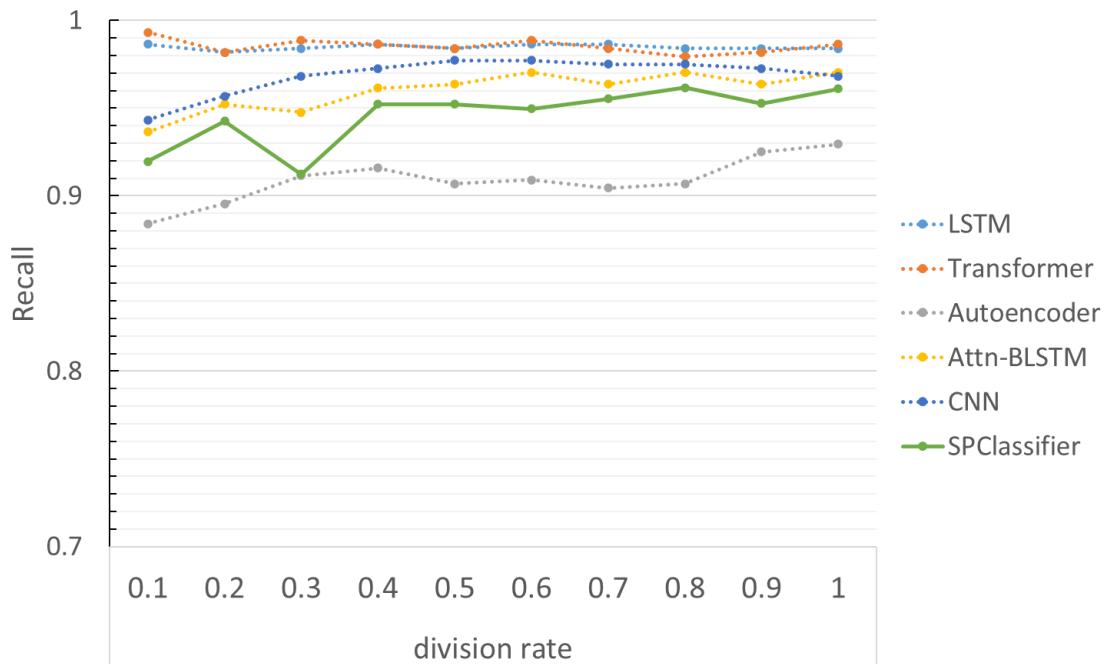


図 5.5 訓練サブセットを用いたベンチマークの結果(Recall)

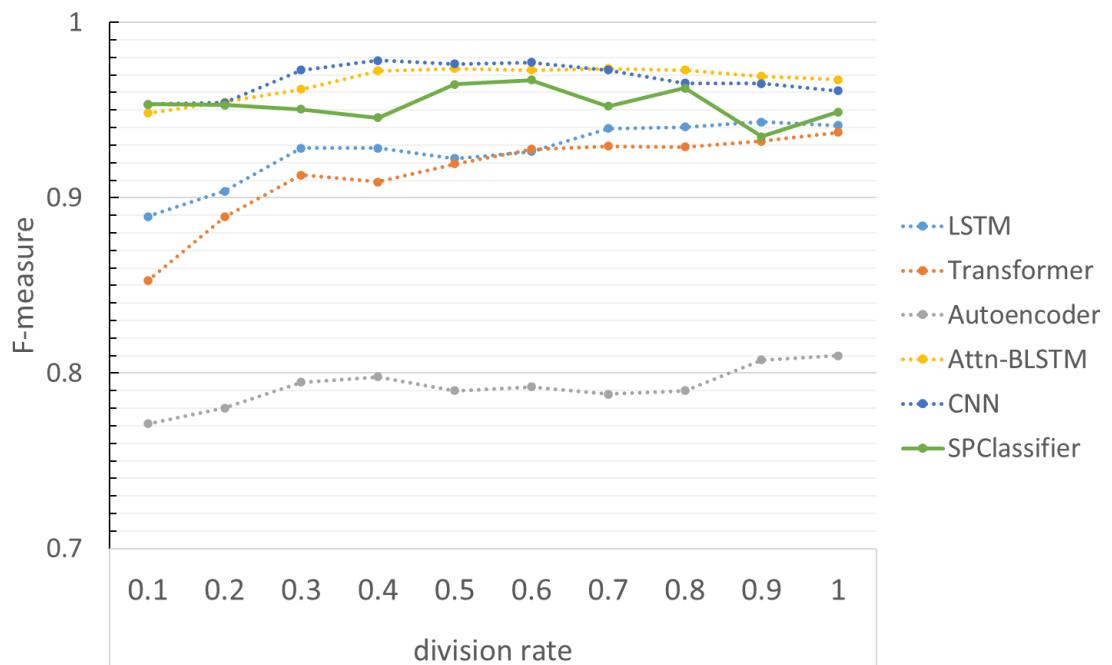


図 5.6 訓練サブセットを用いたベンチマークの結果(F-measure)

## 5.4 結論

BGL データセットでのベンチマークを通じて、提案手法である SPClassifier は、教師付き深層学習手法に匹敵するログ異常検知精度を実現できることがわかった。また、学習データのサブセットを用いた実験では、データ量が極端に少ない場合でも、提案手法の精度はこれらの手法と同等であることが分かった。一方で、サブセットのデータ量が増えると、深層学習法に比べてスコア値が変動する傾向が見られた。これは、空間プーリング層のシナプスが、サブセット内の新しいパターンの出現に適応しすぎてしまったためと考えられる。よりロバストな検出のためには、ブースト係数の更新周期やシナプスの永続値の量などのパラメータを詳細に調べる必要があると考えられる。

一方で、空間プーリングはデータを逐次処理する必要があるため、バッチ処理が可能な深層学習に比べて学習時間が長くなるという問題がある。今後は、並列化の導入や空間プーリングのより効率的で高速な実装により、CPU 環境での学習時間の改善を図る予定である[71]。

## 第6章 結論

本論文は、OSS を利用したソフトウェア開発における不具合修正作業の効率化に向けて、不具合報告を適切な担当カテゴリに自動分類するシステム、および分散表現・時間パターンに基づくログデータの異常検出手法を提案した。

第 2 章では、不具合修正プロセスにおけるバグトリアージの効率化を目的として、不具合報告の内容をもとに不具合票を正しい製品のカテゴリに自動分類することを試みた。Eclipse プロジェクトにおける不具合報告データを対象に分類実験を行った結果、約 87% の精度、約 86% の再現率で不具合票を各製品のカテゴリに分類することができた。また、BERT モデルの自己注意層の可視化では、モデルが製品に関連する語彙に対して強い注意を向けて判断を行っていることが示唆された。誤分類された不具合報告では、一般的な語彙がサブワードに分割されている傾向が確認され、トークナイザの辞書における語彙数を最適化する必要があると考えられる。

第 3 章では、教師なし学習によってログデータの各行メッセージに対して異常スコアを算出する深層学習モデルを提案した。また、スーパーコンピュータ Blue Gene/L のログデータを対象として、提案モデルによるワンクラス学習がベースライン手法と比較してどの程度効率的に異常なログメッセージを検出できるかを評価した。結果として、提案手法はベースラインを上回る AUC 値を達成し、再現率において約 7 % の改善がみられた。一方、提案手法はハイパーパラメータの変化に敏感であるため、データセットの更新にロバストな学習戦略を作成する必要がある。

第 4 章では、システム正常動作時のログデータの時間パターンを記憶し、新規に与えられたログの異常度を算出する手法を提案した。そして、分散ファイルシステム HDFS のログデータを対象として、loglizer ツールキットを用いて異常検出ベンチマークを行った。結果として、提案手法はベースライン手法と比較して、学習データ量の変化にロバストな異常検出を行えることが分かった。また、モデルサイズとスパース性が精度に与える影響を分析したところ、それらの値を処理時間とのトレードオフを

考慮して最大化する必要があることが分かった.

第 5 章では、HTM アルゴリズムの空間ブーリング処理を用いて、教師あり学習によって異常なログシーケンスを検出する手法を提案した. そして、スーパーコンピュータ Blue Gene/L のログデータを対象として、deep-loglizer ツールキットを用いて異常検出ベンチマークを行った. 結果として、提案手法は学習データ量が極端に少ない場合には深層教師あり学習手法を上回る精度を持ち、データ量の変化に対しても高い水準で検出精度を維持できることが分かった. 一方、提案手法はバッチ処理の適用ができないために処理時間が深層学習手法よりも長くなるため、並列化や実装の効率化を行う必要がある.

OSS を利用したソフトウェア開発は、多機能性や頻繁なリリースが求められる現代の製品にとって必要不可欠なものとなってきている. OSS を製品に採用する企業は今後も増えていくと予想され、その利用に伴って発生する不具合をいかに早期に解決できるかが重要となる. 本論文の成果は、OSS を利用したソフトウェア開発の不具合修正作業の効率化ツールや経験の浅いエンジニアが不具合に対処する際のサポートツールとして活用されることが期待される. また、疎な学習モデルに基づく分類方法をさらに効率化することで、深層学習手法と同等の精度をより少ない計算性能の環境で実現する必要があり、これは今後の課題である.

## 謝辞

本研究を進め学位論文をまとめるに当たり、多くの支援とご指導を承りました。

指導教員である中藤良久教授には深く感謝しています。研究活動を通して、大変多忙な中、数々のご指導、ご鞭撻を賜りましたこと、ここに厚く御礼申し上げます。有難うございました。

本論文に対して貴重なご意見を頂きました、九州工業大学 芹川 聖一 教授、神谷 亨 教授に感謝申し上げます。芹川先生には、学会等でも貴重な御助言をして頂きました。神谷先生には、暖かい言葉を掛けて頂き、本論文執筆の大きな励みとなりました。有難うございました。

折りに触れて御指導、貴重なご助言を頂きました 中司 賢一 准教授、 河野 英昭 准教授に心より感謝致します。国際会議、国際交流へ向けた御指導、大変お世話になりました。中司先生には論文内容に関して丁寧なご指導をいただきました。有難うございました。

本研究を円滑に行う上で研究に関連する書籍、器具の調達等で本当にお世話になりました、原田 勝也 技術職員に心より感謝致します。有難うございました。

中藤研究室の皆様には、本研究に関する数多くの御討論や御意見にご協力を頂きました。ここに深く感謝致します。

パナソニックシステムデザイン株式会社の皆様には、本研究におけるデータ提供や現場の課題についてのご意見、資金面における援助ならびに長期インターンシップの受け入れをして頂きました。ここに厚く御礼申し上げます。有難うございました。

父、母には、博士後期課程への進学を快く認めていただき、博士取得のために必要な経済的、精神的援助をして頂きました。心より感謝致します。有難うございました。

最後に、大学・大学院在学中に渡って暖かく見守っていただいた親族、友人の皆様に心から感謝致します。

## 参考文献

- [1] 経済産業省 商務情報政策局 サイバーセキュリティ課. “OSSの利活用及びそのセキュリティ確保に向けた管理手法に関する事例集,” (online), available from <[https://www.meti.go.jp/policy/netsecurity/wg1/ossjirei\\_20210421.pdf](https://www.meti.go.jp/policy/netsecurity/wg1/ossjirei_20210421.pdf)> Apr. 21, 2021.
- [2] “Home :: Bugzilla :: bugzilla.org.” <https://www.bugzilla.org/> (accessed Dec. 31, 2021).
- [3] “The Trac Project.” <https://trac.edgewall.org/> (accessed Dec. 31, 2021).
- [4] “Overview - Redmine.” <https://www.redmine.org/> (accessed Dec. 31, 2021).
- [5] 伊原彰紀, “OSSの不具合修正プロセス効率化のためのコミッター推薦と不具合修正時期の予測,” 2012. Accessed: Jan. 07, 2022. [Online]. Available: <https://ci.nii.ac.jp/naid/500000561309.bib>
- [6] 伊原彰紀, 大平雅雄, and 松本健一, “OSS 開発における不具合修正プロセスの現状と課題：不具合修正時間の短縮化へ向けた分析,” 情報社会学会誌, vol. 6, pp. 5–16, Mar. 2012, [Online]. Available: <http://www.redmine.org/>
- [7] 柏祐太郎, 大平雅雄, 阿萬裕久, and 亀井靖高, “大規模OSS開発における不具合修正時間の短縮化を目的としたバグトリアージ手法,” 情報処理学会論文誌ジャーナル (Web), vol. 56, pp. 669–681, Feb. 2015.
- [8] G. Jeong, S. Kim, and T. Zimmermann, “Improving bug triage with bug tossing graphs,” in *ESEC/FSE '09: Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, Aug. 2009, pp. 111–120. doi: 10.1145/1595696.1595715.
- [9] J. Anvik, L. Hiew, and G. C. Murphy, “Who should fix this bug?,” in *Proceedings of the 28th international conference on Software engineering*, May 2006, pp. 361–370. doi: 10.1145/1134285.1134336.
- [10] J. Anvik and G. C. Murphy, “Reducing the effort of bug report triage,” *ACM Transactions on Software Engineering and Methodology*, vol. 20, no. 3, pp. 1–35, Aug. 2011, doi: 10.1145/2000791.2000794.

- [11] S. He, J. Zhu, P. He, and M. R. Lyu, “Experience Report: System Log Analysis for Anomaly Detection,” in *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, Dec. 2016, pp. 207–218. doi: 10.1109/ISSRE.2016.21.
- [12] J. Zhu *et al.*, “Tools and Benchmarks for Automated Log Parsing,” in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, May 2019, pp. 121–130. doi: 10.1109/ICSE-SEIP.2019.00021.
- [13] R. Vaarandi, “A data clustering algorithm for mining patterns from event logs,” in *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003) (IEEE Cat. No.03EX764)*, 2003, pp. 119–126. doi: 10.1109/IPOM.2003.1251233.
- [14] M. Nagappan and M. A. Vouk, “Abstracting log lines to log event types for mining software system logs,” May 2010. doi: 10.1109/MSR.2010.5463281.
- [15] R. Vaarandi and M. Pihelgas, “LogCluster - A data clustering and pattern mining algorithm for event logs,” in *2015 11th International Conference on Network and Service Management (CNSM)*, Nov. 2015, pp. 1–7. doi: 10.1109/CNSM.2015.7367331.
- [16] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, “Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis,” 2009.
- [17] L. Tang, T. Li, and C.-S. Perng, “LogSig: generating system events from raw textual logs,” in *Proceedings of the 20th ACM international conference on Information and knowledge management - CIKM ’11*, 2011, p. 785. doi: 10.1145/2063576.2063690.
- [18] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen, “LogMine: Fast pattern recognition for log analytics,” in *International Conference on Information and Knowledge Management, Proceedings*, Oct. 2016, vol. 24-28-October-2016, pp. 1573–1582. doi: 10.1145/2983323.2983358.
- [19] M. Mizutani, “Incremental mining of system log format,” in *Proceedings - IEEE 10th International Conference on Services Computing, SCC 2013*, 2013, pp. 595–

602. doi: 10.1109/SCC.2013.73.
- [20] K. Shima, “Length Matters: Clustering System Log Messages using Length of Words,” Nov. 2016, [Online]. Available: <http://arxiv.org/abs/1611.03213>
- [21] M. J. Zhen, A. E. Hassan, P. Flora, and G. Hamann, “Abstracting execution logs to execution events for enterprise applications,” in *Proceedings - International Conference on Quality Software*, 2008, pp. 181–186. doi: 10.1109/QSIC.2008.50.
- [22] A. A. O. Makanju, A. N. Zincir-Heywood, and E. E. Milios, “Clustering event logs using iterative partitioning,” in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*, 2009, p. 1255. doi: 10.1145/1557019.1557154.
- [23] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, “Drain: An Online Log Parsing Approach with Fixed Depth Tree,” in *Proceedings - 2017 IEEE 24th International Conference on Web Services, ICWS 2017*, Sep. 2017, pp. 33–40. doi: 10.1109/ICWS.2017.13.
- [24] M. Du and F. Li, “Spell: Streaming Parsing of System Event Logs,” in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, Dec. 2016, pp. 859–864. doi: 10.1109/ICDM.2016.0103.
- [25] M. Du and F. Li, “Spell: Online Streaming Parsing of Large Unstructured System Logs,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 11, pp. 2213–2227, Nov. 2019, doi: 10.1109/TKDE.2018.2875442.
- [26] S. Messaoudi, A. Panichella, D. Bianculli, L. Briand, and R. Sasnauskas, “A search-based approach for accurate identification of log message formats,” May 2018. doi: 10.1145/3196321.3196340.
- [27] “LogPAI. Log Analytics Powered by AI · GitHub.” <https://github.com/logpai> (accessed Jun. 04, 2021).
- [28] “GitHub - logpai/loglizer: A log analysis toolkit for automated anomaly detection [ISSRE'16].” <https://github.com/logpai/loglizer> (accessed Jun. 05, 2021).
- [29] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, “Fingerprinting the datacenter,” in *Proceedings of the 5th European conference on Computer systems - EuroSys '10*, 2010, p. 111. doi: 10.1145/1755913.1755926.

- [30] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer, “Failure diagnosis using decision trees,” in *International Conference on Autonomic Computing, 2004. Proceedings.*, pp. 36–43. doi: 10.1109/ICAC.2004.1301345.
- [31] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, “Failure Prediction in IBM BlueGene/L Event Logs,” in *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, Oct. 2007, pp. 583–588. doi: 10.1109/ICDM.2007.46.
- [32] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, “LOF: Identifying Density-Based Local Outliers.”
- [33] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, “Estimating the support of a high-dimensional distribution,” *Neural Computation*, vol. 13, no. 7, pp. 1443–1471, Jul. 2001, doi: 10.1162/089976601750264965.
- [34] F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation Forest,” in *2008 Eighth IEEE International Conference on Data Mining*, Dec. 2008, pp. 413–422. doi: 10.1109/ICDM.2008.17.
- [35] W. Xu, L. Huang, A. Fox, D. A. Patterson, and M. Jordan, “Large-Scale System Problems Detection by Mining Console Logs,” 2009. [Online]. Available: <http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/>
- [36] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, “Mining Invariants from Console Logs for System Problem Detection,” in *USENIX Annual Technical Conference (ATC)*, 2010, pp. 231–244.
- [37] Q. Lin, H. Zhang, J. G. Lou, Y. Zhang, and X. Chen, “Log clustering based problem identification for online service systems,” in *Proceedings - International Conference on Software Engineering*, May 2016, pp. 102–111. doi: 10.1145/2889160.2889232.
- [38] S. He, J. Zhu, P. He, and M. R. Lyu, “Loghub: A Large Collection of System Log Datasets towards Automated Log Analytics,” Aug. 2020, [Online]. Available: <http://arxiv.org/abs/2008.06448>
- [39] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, “Detecting Large-Scale System Problems by Mining Console Logs.”

- [40] Z. Chen, J. Liu, W. Gu, Y. Su, and M. R. Lyu, “Experience Report: Deep Learning-based System Log Analysis for Anomaly Detection; Experience Report: Deep Learning-based System Log Analysis for Anomaly Detection,” 2021, doi: 10.1145/1122445.1122456.
- [41] M. Du, F. Li, G. Zheng, and V. Srikumar, “DeepLog: Anomaly detection and diagnosis from system logs through deep learning,” in *Proceedings of the ACM Conference on Computer and Communications Security*, Oct. 2017, pp. 1285–1298. doi: 10.1145/3133956.3134015.
- [42] W. Meng *et al.*, “LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs,” in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, Aug. 2019, pp. 4739–4745. doi: 10.24963/ijcai.2019/658.
- [43] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, “Self-Attentive Classification-Based Anomaly Detection in Unstructured Logs,” in *2020 IEEE International Conference on Data Mining (ICDM)*, Nov. 2020, pp. 1196–1201. doi: 10.1109/ICDM50108.2020.00148.
- [44] A. Vaswani *et al.*, “Attention Is All You Need,” Jun. 2017, [Online]. Available: <http://arxiv.org/abs/1706.03762>
- [45] A. Farzad and T. A. Gulliver, “Unsupervised log message anomaly detection,” *ICT Express*, vol. 6, no. 3, pp. 229–237, Sep. 2020, doi: 10.1016/j.icte.2020.06.003.
- [46] X. Zhang *et al.*, “Robust log-based anomaly detection on unstable log data,” in *ESEC/FSE 2019 - Proceedings of the 2019 27th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Aug. 2019, pp. 807–817. doi: 10.1145/3338906.3338931.
- [47] S. Lu, X. Wei, Y. Li, and L. Wang, “Detecting Anomaly in Big Data System Logs Using Convolutional Neural Network,” in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology*

- Congress(DASC/PiCom/DataCom/CyberSciTech)*, Aug. 2018, pp. 151–158. doi: 10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00037.
- [48] R. Sennrich, B. Haddow, and A. Birch, “Neural Machine Translation of Rare Words with Subword Units,” Aug. 2015, [Online]. Available: <http://arxiv.org/abs/1508.07909>
  - [49] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” Oct. 2018, [Online]. Available: <http://arxiv.org/abs/1810.04805>
  - [50] Y. Wu *et al.*, “Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation.”
  - [51] M. Sadat, A. B. Bener, and A. v. Miranskyy, “Rediscovery Datasets: Connecting Duplicate Reports of Apache, Eclipse, and KDE,” Mar. 2017, doi: 10.5281/ZENODO.400614.
  - [52] T. Shen, J. Mueller, R. Barzilay, and T. Jaakkola, “Educating Text Autoencoders: Latent Representation Guidance via Denoising,” May 2019, [Online]. Available: <http://arxiv.org/abs/1905.12777>
  - [53] R. Chalapathy, A. K. Menon, and S. Chawla, “Anomaly Detection using One-Class Neural Networks,” Feb. 2018, [Online]. Available: <http://arxiv.org/abs/1802.06360>
  - [54] A. Oliner and J. Stearley, “What Supercomputers Say: A Study of Five System Logs,” in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN’07)*, Jun. 2007, pp. 575–584. doi: 10.1109/DSN.2007.103.
  - [55] Anonymous, “Loghub,” Sep. 2021, doi: 10.5281/ZENODO.3227177.
  - [56] “Optuna - 株式会社Preferred Networks.” <https://www.preferred.jp/ja/projects/optuna/> (accessed Nov. 06, 2021).
  - [57] “GitHub - shentianxiao/text-autoencoders.” <https://github.com/shentianxiao/text-autoencoders> (accessed Nov. 06, 2021).
  - [58] “Random Distributed Scalar Encoder.” <http://fergalbyrne.github.io/rdse.html> (accessed Jun. 05, 2021).

- [59] “GitHub - aappleby/smhasher: Automatically exported from code.google.com/p/smhasher.” <https://github.com/aappleby/smhasher> (accessed Jun. 07, 2021).
- [60] “HIERARCHICAL TEMPORAL MEMORY including HTM Cortical Learning Algorithms”, Accessed: Nov. 06, 2021. [Online]. Available: <http://www.numenta.com/software-overview/licensing.php> 翻訳株式会社アルト一  
ク 2011/1/25 出典:[http://www.numenta.com/htm-overview/education/HTM\\_CorticalLearningAlgorithms.pdf](http://www.numenta.com/htm-overview/education/HTM_CorticalLearningAlgorithms.pdf) 脚注はすべて訳者による注釈である。本書の改訂版を確認するには<http://ai.altalk.com> 参照。
- [61] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, “Unsupervised real-time anomaly detection for streaming data,” *Neurocomputing*, vol. 262, pp. 134–147, Nov. 2017, doi: 10.1016/j.neucom.2017.04.070.
- [62] S. Ahmad and J. Hawkins, “Properties of Sparse Distributed Representations and their Application to Hierarchical Temporal Memory,” Mar. 2015, Accessed: Jun. 05, 2021. [Online]. Available: <http://arxiv.org/abs/1503.07469>
- [63] S. Ahmad and J. Hawkins, “How do neurons operate on sparse distributed representations? A mathematical theory of sparsity, neurons and active dendrites,” Jan. 2016, Accessed: Jun. 05, 2021. [Online]. Available: <http://arxiv.org/abs/1601.00720>
- [64] S. Purdy, “Encoding Data for HTM Systems,” Feb. 2016, [Online]. Available: <http://arxiv.org/abs/1602.05925>
- [65] Y. Cui, S. Ahmad, and J. Hawkins, “The HTM spatial pooler—a neocortical algorithm for online sparse distributed coding,” *Frontiers in Computational Neuroscience*, vol. 11, Nov. 2017, doi: 10.3389/fncom.2017.00111.
- [66] Y. Cui, S. Ahmad, and J. Hawkins, “Continuous online sequence learning with an unsupervised neural network model,” *Neural Computation*, vol. 28, no. 11, pp. 2474–2504, Nov. 2016, doi: 10.1162/NECO\_a\_00893.
- [67] J. Barnett, “A Hierarchical Temporal Memory Sequence Classifier for Streaming Data,” 2020. [https://nsuworks.nova.edu/gscis\\_etd](https://nsuworks.nova.edu/gscis_etd) (accessed Oct. 13, 2021).
- [68] G. Rydholm, “Using Neurobiological Frameworks for Anomaly Detection in

- System Log Streams,” 2018. <http://kth.diva-portal.org/smash/get/diva2:1295390/FULLTEXT01.pdf> (accessed Oct. 13, 2021).
- [69] “GitHub - htm-community/htm.core: Actively developed Hierarchical Temporal Memory (HTM) community fork (continuation) of NuPIC. Implementation for C++ and Python.” <https://github.com/htm-community/htm.core> (accessed Jun. 06, 2021).
- [70] “logpai/deep-loglizer.” <https://github.com/logpai/deep-loglizer> (accessed Sep. 19, 2021).
- [71] L. Li, T. Zou, T. Cai, D. Niu, and Y. Zhu, “A Fast Spatial Pool Learning Algorithm of Hierarchical Temporal Memory Based on Minicolumn’s Self-Nomination,” *Computational Intelligence and Neuroscience*, vol. 2021, 2021, doi: 10.1155/2021/6680833.

## 業績一覧

<学術雑誌等に発表した論文>

- [1] R. Hirakawa, H. Uchida, A. Nakano, K. Tominaga, and Y. Nakatoh, “Large scale log anomaly detection via spatial pooling,” *Cognitive Robotics*, vol. 1, pp. 188–196, Oct. 2021, doi: 10.1016/j.cogr.2021.10.001. [査読有]
- [2] R. Hirakawa, H. Uchida, A. Nakano, K. Tominaga, and Y. Nakatoh, “Anomaly detection on software log based on Temporal Memory,” *Computers & Electrical Engineering*, vol. 95, p. 107433, Oct. 2021, doi: 10.1016/j.compeleceng.2021.107433. [査読有]
- [3] 川原 俊介, 平川 凜, 中藤 良久. “音声スペクトルの時間変化強調による明瞭性改善方法の検討”, 産業応用工学会論文誌, 産業応用工学会, Vol. 6, No.1, pp.51–54, 2018. [査読有]

<国際会議における発表>

- [1] R. Hirakawa, K. Tominaga, and Y. Nakatoh, “Software Log Anomaly Detection Method Using HTM Algorithm,” in *Advances in Intelligent Systems and Computing*, vol. 1334, Springer, 2021, pp. 71–79. doi: 10.1007/978-981-33-6981-8\_6. [査読有]
- [2] D. Joumori, R. Hirakawa, H. Kawano, and K. Nakashi, “Sitting Posture Assessment Method for Back Pain Prevention System,” in *Advances in Intelligent Systems and Computing*, vol. 1253 AISC, 2021, pp. 351–355. doi: 10.1007/978-3-030-55307-4\_53. [査読有]
- [3] K. Nishikawa, R. Hirakawa, H. Kawano, K. Nakashi, and Y. Nakatoh, “Effective Speech Features for Distinguishing Mild Dementia Patients from Healthy Person,” in *Advances in Intelligent Systems and Computing*, vol. 1253 AISC, 2021, pp. 356–361. doi: 10.1007/978-3-030-55307-4\_54. [査読有]
- [4] Y. Sakaguchi, R. Hirakawa, H. Kawano, K. Nakashi, and Y. Nakatoh, “Speaker Verification Method Using HTM for Security System,” in *Advances in Intelligent Systems and Computing*, vol. 1253 AISC, 2021, pp. 160–165. doi: 10.1007/978-3-030-55307-4\_25. [査読有]
- [5] A. Kuwahara, R. Hirakawa, H. Kawano, K. Nakashi, and Y. Nakatoh, “Blink

- Detection Using Image Processing to Predict Eye Fatigue,” in *Advances in Intelligent Systems and Computing*, vol. 1253 AISC, 2021, pp. 362–368. doi: 10.1007/978-3-030-55307-4\_55. [査読有]
- [6] R. Hirakawa, K. Tominaga, and Y. Nakatoh, “Study on Software Log Anomaly Detection System with Unsupervised Learning Algorithm,” in *Advances in Intelligent Systems and Computing*, vol. 1152 AISC, 2020, pp. 122–128. doi: 10.1007/978-3-030-44267-5\_18. [査読有]
- [7] R. Hirakawa, K. Tominaga, and Y. Nakatoh, “Software Log Anomaly Detection Through One Class Clustering of Transformer Encoder Representation,” in *Communications in Computer and Information Science*, vol. 1224 CCIS, 2020, pp. 655–661. doi: 10.1007/978-3-030-50726-8\_85. [査読有]
- [8] R. Hirakawa, K. Tominaga, and Y. Nakatoh, “Study on Automatic Defect Report Classification System with Self Attention Visualization,” in *2020 IEEE International Conference on Consumer Electronics (ICCE)*, Jan. 2020, vol. 2020-Janua, pp. 1–2. doi: 10.1109/ICCE46568.2020.9043062. [査読有]
- [9] R. Hirakawa, R. Hirakawa, H. Kawano, K. Nakashi, and Y. Nakatoh, “Programming Assistant System Using Improved Flowchart for the Visually Impaired,” in *Advances in Intelligent Systems and Computing*, vol. 1018, 2020, pp. 496–501. doi: 10.1007/978-3-030-25629-6\_77. [査読有]
- [10] R. Komatsu, R. Hirakawa, H. Kawano, K. Nakashi, and Y. Nakatoh, “Study on Mistype Correction Support Using Attention in Japanese Input,” in *2020 IEEE International Conference on Consumer Electronics (ICCE)*, Jan. 2020, vol. 2020-Janua, pp. 1–2. doi: 10.1109/ICCE46568.2020.9043138. [査読有]
- [11] E. Urabe, R. Hirakawa, H. Kawano, K. Nakashi, and Y. Nakatoh, “Electrolarynx System Using Voice Conversion Based on WaveRNN,” in *2020 IEEE International Conference on Consumer Electronics (ICCE)*, Jan. 2020, vol. 2020-Janua, pp. 1–2. doi: 10.1109/ICCE46568.2020.9043135. [査読有]
- [12] R. Hirakawa, H. Kawano, K. Nakashi, and Y. Nakatoh, “Study on Watching System for Door-to-door Sales Detection and Risk Determination,” in *Proceedings of the 7th ACIS International Conference on Applied Computing and Information Technology*,

May 2019, pp. 1–6. doi: 10.1145/3325291.3325391. [査読有]

- [13] R. Hirakawa and Y. Nakatoh, “Study on Door-to-door Sales Detection Method in Watching System for Seniors living alone,” in 2019 IEEE International Conference on Consumer Electronics (ICCE), Jan. 2019, pp. 1–2.  
doi: 10.1109/ICCE.2019.8662118. [査読有]
- [14] T. Okuno, R. Hirakawa, H. Kawano, K. Nakashi, and Y. Nakatoh, “Discussion on differences in frequency band required for horizontal sound localization by age,” in *Proceedings of the 7th ACIS International Conference on Applied Computing and Information Technology*, May 2019, pp. 1–6. doi: 10.1145/3325291.3325395. [査読有]
- [15] E. Urabe, R. Hirakawa, H. Kawano, K. Nakashi, and Y. Nakatoh, “Enhancement of Electrolarynx speech based on WaveRNN,” in *Proceedings of the 7th ACIS International Conference on Applied Computing and Information Technology*, May 2019, pp. 1–6. doi: 10.1145/3325291.3325396. [査読有]
- [16] Y. Mihara, R. Hirakawa, H. Kawano, K. Nakashi, and Y. Nakatoh, “Study on evaluating risks of routes damages at Earthquakes for Evacuation Guidance System,” in Proceedings of the 7th ACIS International Conference on Applied Computing and Information Technology, May 2019, pp. 1–6.  
doi: 10.1145/3325291.3325390. [査読有]
- [17] Y. Kinoshita, R. Hirakawa, H. Kawano, K. Nakashi, and Y. Nakatoh, “Examination of Acoustic Features for discriminating between Real and Loudspeaker speeches,” in *Proceedings of the 7th ACIS International Conference on Applied Computing and Information Technology*, May 2019, pp. 1–6. doi: 10.1145/3325291.3325394. [査読有]
- [18] C. Inoshita, R. Hirakawa, H. Kawano, K. Nakashi, and Y. Nakatoh, “In load collapse prevention system: Examination of the relationship between truck vibration and load inclination,” in Proceedings of the 7th ACIS International Conference on Applied Computing and Information Technology, May 2019, pp. 1–6.  
doi: 10.1145/3325291.3325393. [査読有]
- [19] T. Hirayama, R. Hirakawa, H. Kawano, K. Nakashi, and Y. Nakatoh, “Study on method of veering correction by vibration transmission in walking support for visually impaired people,” in Proceedings of the 7th ACIS International Conference

- on Applied Computing and Information Technology, May 2019, pp. 1–6.  
doi: 10.1145/3325291.3325392. [査読有]
- [20] Rin Hirakawa, Yoshihisa Nakatoh, “Study on Visit Sales Detection using Similarity of Paragraph Vector”, Proceedings of the 6th IIAE International Conference on Intelligent Systems and Image Processing 2018, pp.39-42, January. 2018[査読有]

<国内会議における発表>

- [1] 平川凜, 中藤良久. HTM アルゴリズムを用いたリアルタイムログ異常検知方法の検討, 2019 年電子情報通信学会ソサイエティ大会, 大阪, 2019 年 9 月.
- [2] 平川凜, 中藤良久, “Doc2Vec を用いた訪問販売検知方法の研究”, 情報科学技術フォーラム講演論文集, Vol.17th 第 2 分冊, pp.171-172, September. 2018
- [3] 川原俊介, 平川凜, 中藤良久, “音声スペクトルの時間変化強調による明瞭性改善効果の検討”, 日本音響学会研究発表会講演論文集(CD-ROM) Vol.2018 春季 Page.ROMBUNNO.2, pp.49, January. 2018
- [4] 平川凜, 中藤良久, “音声強調を用いた雑音下での音声認識性能改善方法の研究”, 日本音響学会九州支部第 12 回学生のための研究発表会, 福岡, 日本, 12 月, 2017