ベクトル化と機械学習を用いたコメントスメル自動検出手法の精度比較

堂元 優佑　佐伯 昌造　阿萬 裕久　川原 稔

ソースコードの可読性を向上させる上で，コメントは有用な存在になりうる．しかしながら，実際にはコード理解に役に立たないコメントやコードの可読性低下を引き起こすようなコメントも存在しており，そういったコメントは“コメントスメル”と呼ばれている．これまでにコメントスメルを自動検出する手法として，ソースコードとコメントをベクトル化して，機械学習によってコメントスメルの分類を行うものが知られているが，ベクトル化の方法並びに使用する機械学習モデルの組合せには様々なものが考えられるため，それらの定量的な比較は行われていなかった．そこで本論文では，既存手法で行われている（1）ソースコードとコメントのベクトル化のタイミング,（2）ベクトル化に用いる手法,（3）機械学習モデル，それぞれについて複数の代替案を用意し，全部で 70 種類の組合せについて自動検出実験を行っている．その結果,（1）についてはソースコードとコメントを別々にベクトル化した後に結合するとよいことが確認され,（2）と（3）についてはコメントスメルの種類に応じて最適なものが異なることが確認された．

1　はじめに

プログラミングにおいて，コメントの記述というのはソースコードを記述することと同様に重要なことである [2]．コメントをプログラム内に記述するということは，“ソースコードがどのような処理をどういった意図で行っているのか”ということを端的に説明することである．これによりソースコードの可読性を向上させることになりコード理解を容易にする．加えて可読性の向上により，ソースコードの保守や改良を行いやすくなるというメリットも存在する．以上のことからコメントの存在はプログラミングにおいて重要である．しかしながら，コメントが可読性を向上させるのはあくまでも品質の高いコメント，つまりコード理解に役立つようなコメントの場合である．これは品質の低いコメントがソースコードに存在してしまっていると逆にソースコードの可読性を低下させることになるともいえる．品質の高いコメントを書くことがコード理解に重要であるように，品質の低いコメントを書かないようにすることも重要になってくる．

品質の低いコメントとは，言い換えればコード理解に役に立たないコメントやコードの可読性低下を引き起こすようなコメントである．Jabrayilzade ら [3] はそのようなコメントを“コメントスメル”と名付け，11 種類のコメントスメルを提唱している．近年，コメントスメルの検出に関する研究がいくつか行われるようになってきているが [1] [4] [5] [6]，まだ発展途上であり，さまざまな技術の組合せについては十分に検討されていない．本論文では，主要なコメントスメル検出に関する先行研究である Oztas ら [4] の研究に着目し，そこで使用されているソースコードとコメントのベクトル化や機械学習モデルの選択に関して，可能な限り多くの組合せを試すことで，どういった技術や組合せがコメントスメルの検出に有効なのかを検討していく．

本論文の構成は以下の通りである．2 節ではまず，関連研究について説明する．3 節では，コメントスメ

A Comparative Study of Automated Comment Smell Detection Methods Using Vectorization and Machine Learning.

Yusuke Domoto, 愛媛大学大学院理工学研究科, Graudate School of Sc. & Eng., Ehime University.

Shozo Saeki,　Hirohisa Aman, and Minoru Kawahara, 愛媛大学総合情報メディアセンター, Center for Information Technology, Ehime University.

ルについて説明しコメントスメルの自動検出の手法を紹介する．次に 4 節では本研究で行ったコメントスメルの自動検出の実験の結果を報告する．そして 5 節では結果に対する考察を行う．最後に 6 節で本論文のまとめと今後の課題について述べる．

2 関連研究

本節では，コメントスメルに関する関連研究について簡単に紹介する．

Wnag ら [6] は，品質が十分ではないコメントを “suboptimal comments” と呼び，これを特定する際に，コードの変更とは独立して “コメントの変更のみをコミットする” 独立コメント変更（Independent Comment Change:ICC）が代用可能であるかの確認を行っている．その結果，ICC と判断された 3,533 件のうち，3,281 件がコメントの情報の正確性や可読性の向上を目的に行われていることが確認され，ICC が suboptimal comments の特定に有効であることが示されている．

Jabrayilzade ら [3] は，品質の低いコメントを “コメントスメル” と呼び，コードコメントに関する文献調査をもとにコメントスメルを 11 種類に分類している．そして，オープンソースの Java プロジェクト 4 つと Python プロジェクト 4 つから全部で 2,447 件のコメントを収集し，それらがコメントスメルに該当するか否かを手作業で分類している．あわせて，スメルに該当する場合は，どの種類のスメルであるかのラベル付けも行っている．結果として，それらのうちの 1,072 件（約 43.8%）がコメントスメルに該当するものとして報告され，データセットとして公開されている．

Oztas ら [4] は，コメントスメルの自動検出法として，ナイーブベイズや決定木といった機械学習モデルによる検出と LLM（GPT-4）による検出を行っている．その際，前出の Jabrayilzade ら [3] のデータセットに対して，コメントに対応するソースコード断片を追加するかたちでデータセットを拡充し，その上でソースコードとコメントを TF-IDF 法でベクトル化して，機械学習の訓練と予測に用いている．結果として，機械学習モデルの中ではランダムフォレストを用いた検出が最も高い精度を記録し，正解率が 0.67 となっていた．一方，GPT-4 を用いた検出では正解率は 0.55 に留まっており，必ずしも LLM を使う方が優れているとはいえない結果であった．なお，コメントに対応するソースコード断片を与えない元のデータセットを使った場合，GPT-4 での正解率は 0.34 まで低下してしまうことも確認され，それゆえ，上述したデータセットの拡充（対応するソースコード断片の追加）が有効であることが報告されている．したがって，本論文でもこの拡充されたデータセットを用いて実験を行うこととする．

Çağlar ら [1] は，コメントスメルの自動検出精度の向上を目指し，4 種類の LLM（GPT-4o-mini，o3-mini，Codestral-2501，DeepSeek-V3）を 2 種類のプロンプト形式（zero shot, few shot）で使用する実験を行っている．その結果，o3-mini を few shot のかたちで使用するのが最も効果的で，正解率は 0.72 であったと報告されている．さらに，Oztas ら [4] のデータセットに対して，スメルの修正案を追加するかたちで拡充を行い，自動検出のみならずコメントスメルの自動修正も同時に行っている．修正に関しては Codestral-2501 が最も高い精度を記録し，検出と修正それぞれのタスクで有効なモデルが異なることを示している．

3 コメントスメルとその自動検出

本節では，前節で述べたコメントスメルとその自動検出手法についての詳細を解説し，並びに本研究での研究動機について述べる．

3.1 コメントスメル

ソースコードの記述内容の読みやすさは “可読性” と呼ばれ，ソースコードの可読性の向上は保守性を高めることにつながる．適切なソースコードのコメント（以下，コードコメントと呼ぶ）の記述は可読性を向上させ，保守性を高める要素の 1 つとして貢献することが期待される．しかし，その一方で不適切なコードコメントは可読性を低下させてしまう恐れがある．コードスメルが不適切な（改善の余地のある）ソースコードの特徴を示すものであるように，コードコメン

表 1　コメントスメル

スメル名	説明
Misleading	コードの内容と一致していない.
Obvious	コードを見るだけで理解可能な事のみを説明している.
No comment on non-obvious code	コードだけでは意図が分からず説明が必要な箇所にコメントなし.
Commented-out code	不要になったコードを消さずにコメントで残す.
Irrelevant	コードの説明をしていない無関係なもの.
Task	実施する必要がある，または既に完了した作業を書いてある.
Too much information	情報が多すぎる.
Attribution	誰がどのようにしてそのコードを書いたかを説明する.
Beautification	コードを部分ごとに区切る.
Non-local	システム全体，または近い場所にないコードを説明する.
Vague	説明が曖昧でわからない.

トについても同様の概念が提唱されている．これを“コメントスメル”という．例えば，ソースコードの可読性や保守性を低下させてしまうようなコメントやソースコード理解の助けにならないようなコメント等がこれに該当する．

以下で紹介する 11 種類のコメントスメルは Jabrayilzade ら [3] によって提唱されたものである（表 1）．

3.2　コメントスメルの自動検出

上述したように，コメントスメルにはさまざまな種類がある．その検出においては，コメントの内容もさることながら，それと紐付けられたソースコードの内容も加味しながら，そこにコメントスメルがあるといえるのか，さらには，スメル有りという場合にはどの種のスメルに該当するのかを判定しなければならない．この作業はソフトウェアの規模が大きくなるにつれて工数の観点から非常に困難となり，それゆえ自動的な検出が望まれる．

自動検出を行うにあたっては，ソースコードとコメントそれぞれの内容を数値ベクトルといったかたちで定量化し，機械学習によって有無の判断やスメルの判定を行うのが現実的である．これに関してはいくつか先行研究が行われているが [5] [6] [1] [4]，特に新しく，かつ重要なものとして Oztas ら [4] の研究が挙げられる．Oztas らは，コメントスメルの公開データセット [3] に対して複数の機械学習モデルを用いて自動検出実験を行っている．図 1 にそのイメージ図を示す．

この流れについて簡単に説明する．

1. データセットに含まれる各データ（図 2）について，ソースコードとコメントを抽出し，それらを 1 つにつなげたテキストを用意する．
2. 得られたテキスト（ソースコードとコメントをつなげたもの）を TF-IDF によって数値ベクトル化する．
3. 数値ベクトルを入力データ（説明変数）とし，元データにおけるコメントスメルの正解ラベルとともに機械学習モデルに与えて，その訓練（学習）を行う．

正解ラベルには，“Beautification”，“Commented-out code”，“Obvious”，“Task”‘，“Vague”，と

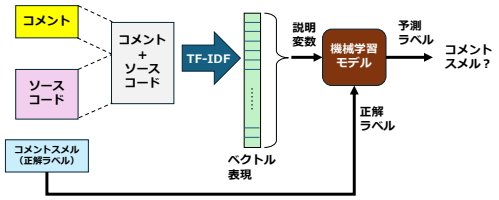

図 1　コメントスメル自動検出のイメージ図

```
1  "comment": "singleton",
2  "label": "Obvious",
3  "code": "    private
       SharedMetricRegistries()
        { /* singleton */ }\n"
```

図 2　コメントスメル検出に用いるデータの例

コメントスメルではないという “Not a smell” の 6 種類を用いている．

4. 学習後のモデルを用いてコメントスメルの有無を未知のデータに対して予測する．なお，その評価にあたっては，7 種類のモデルそれぞれで別々に F1 スコアを用いている．

さらに Oztas らは，GPT-4 を用いたコメントスメル検出も実験している．そこでは，GPT-4 に “熟練のコードレビュー担当者” という役割を割り当て，それぞれのコメントスメルの説明と例を 1 つプロンプトとして渡した後，前述したデータセットに対するスメルの予測を行っている．その際，コメントの文字列のみを渡した場合と，コメントとそれに関連するコードを渡した場合での精度の比較も行っている．結果として，GPT-4 を用いても検出精度としては機械学習モデルと大差ないことが報告されている．

3.3 研究動機

3.2 節で紹介した Oztas ら [4] の研究では，機械学習モデルと GPT-4 といった大規模言語モデル（Large Language Model: LLM）を用いてコメントスメルの自動検出を行っていた．その結果，機械学習モデルと LLM の間で検出精度に大きな違いは見られず，スメルによってはむしろ機械学習モデルの方が優れていた場合も見受けられた．実際のところ，GPT-4 等の LLM をクラウド環境で利用する際には入出力トークン数に応じて費用がかかる従量課金制の契約となる場合も多いことから，積極的に LLM を利用するよりもローカル環境で構築・運用可能な機械学習モデルをうまく活用する方が選択肢としては現実的ではないかと考えられる．

ここで，Oztas らの研究で行われていた機械学習モデルによるコメントスメル検出手法について整理すると，次の 3 つの観点について，それぞれ検討が不十分なのではないかと思われる点の存在に気がついた．

- 観点 1：コメントとソースコードの与え方（コメントとソースコードを別々に与えるか 2 つをつなげて与えるか）
- 観点 2：コメントとソースコードに対するベクトル化手法
- 観点 3：自動検出に用いる機械学習モデル

観点 1（ベクトル化のタイミング）

先行研究では図 1 にも示したように，ソースコードとコメントをいったん統合し（つなげ），それをベクトル化していた．これに対し，ソースコードとコメントを別々にベクトル化して統合するという手法も考えられる．つまり，選択肢としては次の a と b の 2 つが考えられるが，先行研究では a のみが使われていたことになる：

(a) ソースコードとコメントを統合した後にベクトル化する．
(b) ソースコードとコメントをそれぞれベクトル化した後に統合する．

観点 2（ベクトル化手法の選択）

先行研究ではソースコードとコメントのベクトル化手法として TF-IDF が採用されていた．この手法は基本的かつ有用な手法の 1 つではあるが，他にも有用な手法が知られている．具体的には TF-IDF も含めて，以下の A ～ E の 5 つが挙げられる：

(A) TF-IDF
(B) BM25
(C) Word2Vec
(D) CodeT5+
(E) CodeBERT

観点 3（機械学習モデルの選択）

先行研究では，以下に示す 7 種類の機械学習モデルを各種のコメントスメル検出器として構築し，それらの検出精度比較を行っていた：

1. 勾配ブースティング（Gradient Boosting）
2. ランダムフォレスト（Random Forest）
3. 決定木（Decision Tree）
4. サポートベクターマシン（SVM）
5. ロジスティック回帰モデル（Logistic Regression）
6. ナイーブベイズ（Naive Bayse）
7. k-近傍法（KNN）

観点 1 及び 2 については，先行研究では選択肢として a 及び A のみが使われていたことから，それらを別のものに置き換えたとするとコメントスメルに対する検出能力も変わる可能性があり，それゆえこれら 7 種類の機械学習モデルについても再構築と再実

表 2　手法・モデルの組合せ

		機械学習モデル								
		1	2	3	4	5	6	7		
ベクトル化手法	A	**a-A-1**	**a-A-2**	**a-A-3**	**a-A-4**	**a-A-5**	**a-A-6**	**a-A-7**	a	ベクトル化のタイミング
		b-A-1	b-A-2	b-A-3	b-A-4	b-A-5	b-A-6	b-A-7	b	
	B	a-B-1	a-B-2	a-B-3	a-B-4	a-B-5	a-B-6	a-B-7	a	ベクトル化のタイミング
		b-B-1	b-B-2	b-B-3	b-B-4	b-B-5	b-B-6	b-B-7	b	
	C	a-C-1	a-C-2	a-C-3	a-C-4	a-C-5	a-C-6	a-C-7	a	ベクトル化のタイミング
		b-C-1	b-C-2	b-C-3	b-C-4	b-C-5	b-C-6	b-C-7	b	
	B	a-D-1	a-D-2	a-D-3	a-D-4	a-D-5	a-D-6	a-D-7	a	ベクトル化のタイミング
		b-D-1	b-D-2	b-D-3	b-D-4	b-D-5	b-D-6	b-D-7	b	
	B	a-E-1	a-E-2	a-E-3	a-E-4	a-E-5	a-E-6	a-E-7	a	ベクトル化のタイミング
		b-E-1	b-E-2	b-E-3	b-E-4	b-E-5	b-E-6	b-E-7	b	

験が必要になると考えられる．

手法・モデルの組合せ

上述した 3 つの観点について，すべての選択肢を組み合わせると表 2 に示す 70 種類が考えられることになる．便宜上，それぞれの組合せを “a-C-7” といった表記で表すことにする．例えば，“a-C-7” はソースコードとコメントを統合したものを Word2Vec によりベクトル化し，モデルとして k-近傍法を使用したものに該当する．

先行研究では，表 2 に示す 70 種類のうち，太字で強調された 7 種類のみが実験されていたことになる．つまり，残りの 63 種類についてはどのような検出精度となるかは明らかになっていない．そこで，これらについても実験を行う価値があるのではないかと考えたのが本研究の動機である．実験の内容と結果については次章で報告することとする．

4　コメントスメル自動検出実験

我々は，先行研究におけるコメントスメル自動検出法に対し，前節で紹介したソースコードとコメントのベクトル化手法や機械学習モデルとの組合せに着目して行った 70 種類の検出精度に関する比較実験を行った．本節ではその手順並びに実験結果について報告する．

4.1　目的

本実験では，前章で述べた Oztas らの先行研究 [4] に対する疑問点に基づき，以下の研究課題（Research Question: RQ）に対して回答することを目的とする．

RQ: 機械学習を用いたコメントスメルの自動検出において以下の 3 つの観点ではどういった組合せが有用か？

観点 1: コメントとソースコードの与え方（コメントとソースコードを別々に与えるか 2 つをつなげて与えるか）

観点 2: コメントとソースコードに対するベクトル化手法

観点 3: 自動検出に用いる機械学習モデル

この RQ は，先行研究で提案されていた機械学習によるコメントスメル自動検出法に対し，そこで使用するモデルや入力データ形式の違いが，検出精度にどのように影響するのかを確認しようというものである．先行研究では，入力としてコメントとソースコードをまとめてベクトル化したものを使用していたが，そこでの入力形式に工夫を施すことでより高い精度でコメントスメル検出を行うことができるのではないかと筆者は考えた．さらには入力形式が変わることで，有効な機械学習モデルも先行研究のものとは変わる可能性がある．それゆえ，新たな組合せについても再度実験を行う価値があると考えた．

4.2　対象

本実験では，Oztas らの先行研究 [4] と同じデータセットを使用する．データセットは JSON 形式のファイルとして公開されている．図 3 に実際のデータセットの一部を示す．2 行目から 6 行目や 7 行目から 11 行目のように波括弧で囲まれている範囲がコメントとコメントが該当するスメル，並びに当該コメントが紐

```
1  [
2  {
3    "comment": "TODO Tracked in https://github.com/ankidroid/Anki-Android
       /issues/5019",
4    "label": "Task",
5    "code": "NA"
6  },
7  {
8    "comment": "Only enable AnkiDroid notifications unrelated to due
       reminders",
9    "label": "Not a smell",
10   "code": "    public static final int PENDING_NOTIFICATIONS_ONLY =
       1000000;\n"
11 }, (以下省略) ...
```

図 3　コメントスメル検出のデータセット例

付けされているソースコードとなっている 2 行目から 6 行目のデータでは code の部分が “NA” となっているが，これはコメントに対応するようなソースコードが見当たらなかったことを表している．また 7 行目から 11 行目のデータでは label が “Not a smell” となっているが，これは当該コメントはコメントスメルではないことを意味している．このような形式でデータセットには 2211 件のコメントデータが格納されていて，そのうち，“Not a smell” は 1253 件となっていた．一方，コメントスメルの中で最も多いのは “Obvious” で，672 件だった．

4.3　手順

本実験では，以下の手順でコメントスメル検出モデルの構築と検出精度比較を行った．

(1) データセットからコメントとソースコード，スメルのラベルを取得
データセットからコメントとそれに対応するソースコード，そしてコメントがどのコメントスメルなのかというラベルを取得する．

(2) サンプルの少ないスメルのデータを除外
取得したデータの中でラベル付けされたものの総数が 30 未満となるコメントスメルを除外する．ここで 30 件未満のデータを除外する目的としては，データ数が少ないものに関しては機械学習のモデルが十分に学習を行うことが出来ないこと，並びに先行研究においても統計的な問題が発生する可能性を考慮して 30 件未満のコメントスメルを除外していたことから，本研究でも同様に 30 件未満のコメントスメルは対象外とした．そのため，本研究では “Beautification”, “Commented-out code”, “Obvious”, “Task”‘, ‘Vague”, 及び “Not a smell” の 6 種類を対象として扱うこととした．

(3) データを訓練用とテスト用に分ける
機械学習を行う準備として，データを訓練用のものとテスト用のものに分ける．テストに使用するデータ量は全体の 2 割で，残りの 8 割のデータを学習に使用する．

(4) モデルによる学習と予測
表 2 に示す 70 種類の組合せについて，それぞれ機械学習モデルの学習とそれによるコメントスメルの予測を行う．

(5) モデルの評価と比較
予測として得られた 70 種類の結果に基づいて，RQ の観点 1 から 3 までの有効性の評価を行い，どの項目が特に影響が強いのかを調べる．それぞれの観点による影響が統計的に有意であるかどうかを調べるために，はじめに分散分析を行い，その後，観点内の項目ごとで “対応のある” t 検定（以下，単に t 検定と呼ぶ）とウィルコクソンの符号付順位検定（以下，単にウィルコクソン検定と呼ぶ）を行う．分散分析では，観点内での精度に差はないという帰無仮説を立て，t 検定とウィルコクソン検定では，観点内の対象とする 2 間での精度の差に影響はないという帰無仮説を立てる．有意水準は 5%とし，帰無仮説が棄却されるかどうかでもって，どの観点の影響が強

いのかを判断する．なお，t 検定とウィルコクソン検定に関しては，観点内で複数回検定を行うことから多重検定となってしまうため，有意水準は 5%をそれぞれの検定回数で割ったものを用いる．つまり，観点 2 の場合は 10 回検定を行うので 0.5%，観点 3 の場合は 21 回検定を行うので 0.238%とする．

4.4 結果

前述した 70 種類のコメントスメル自動検出の結果を示す．

まず，表 3〜表 8 に各コメントスメルに対する検出精度（F1 スコア）を示す．なお，各表では最も高い精度を太字で強調してある．

コメントスメルごとに最も高い結果となった組合せは異なるものの，どれもコメントとソースコードを別々にベクトル化した場合である b に該当していた．その他の結果についても，同じベクトル化手法では a と比較して b の方が高い精度を記録していることが見てとれる．

5 考察

本節では，前節で示した実験結果について，RQ の 3 つの観点から考察を述べる．

考察を行うため，それぞれの観点でスメルごとに有意差があるかどうか検定を行ったので，それを踏まえつつ議論する．

5.1 観点 1

観点 1 では，分散分析によってすべてのコメントスメルにおいて a と b の間で精度に差があることを確認でき，b の方が a と比較して平均値，中央値ともに高い結果となった．このような結果になった要因として，ソースコードとコメントを別々にベクトル化することで，それぞれの記述の特徴をより的確に表現できるようになったことが大きいと考えられる．例えば Beautification や Task といったコメントスメルでは，そのコメントスメル特有のものが含まれる傾向にある．Beautification では変数宣言や関数を区切る際に “+++++++” のような記号で区切ることが多く，Task ではコメントの始まりが “TODO” や “FIXME” といった特定の文字列になることが多い．別々にベクトル化したことで，このようなスメル固有の記述内容がより的確に表現されたのではないかと考える．また，コメントとソースコードを別々にベクトル化した場合，統合してベクトル化したものよりベクトルの次元数は多くなる．それにより検出精度が向上したとも考えられる．

5.2 観点 2

観点 2 では，コメントスメルごとに異なる結果となった．まず，Beautification 及び Vague で有意差があった組合せはいずれも Word2Vec が含まれているということが確認された．また，Obvious に関しても Word2Vec が有意差有りの組合せに含まれていた．このことから，Beautification と Vague，Obvious に限っていえば，Word2Vec 以外のベクトル化手法がスメル検出に有効であるといえる．

次にその他のコメントスメルについて見ていくと，Commented-out code では，TF-IDF と CodeBERT 間では有意差が見られなかったが，TF-IDF 及び CodeBERT でその他のベクトル化手法との間には有意差が見られ，これら 2 つのベクトル化手法が有効であったといえる．Not a smell も，Commented-out code と同様の傾向となっていた．一方，異なる箇所として CodeBERT と CodeT5+ の間には有意差は見られないものの，Word2Vec と CodeT5+ の間では有意差が確認された．よって，TF-IDF，CodeT5+，及び CodeBERT が Not a smell の検出には有効な手法といえる．CodeT5+ と CodeBERT に関しては，事前に大量のソースコードを学習していたことで，ソースコードとコメントの特徴をより的確に捉えることができ，“コメントスメルではない” という判定に有効であったと考えられる．一方，TF-IDF はコメントスメルになるような “特定の単語を含まない” という観点から検出が可能であったものと思われる．

Task に関しては TF-IDF と BM25 間でのウィルコクソン検定を除いて，すべてで有意差があった．Task において，TF-IDF が最も有効であった理由として，Task というコメントスメルは “TODO” 等のアノ

表 3　Beautification の結果

	1	2	3	4	5	6	7	
A	0.83	0.88	0.85	0.89	0.93	0.35	0.70	a
	0.87	0.86	0.87	0.91	0.88	0.56	0.70	b
B	0.89	0.92	0.86	0.69	0.84	0.58	0.73	a
	0.89	0.93	0.90	0.71	0.91	0.56	0.90	b
C	0.66	0.63	0.57	0.40	0.41	0.39	0.56	a
	0.85	0.80	0.70	0.78	0.80	0.65	0.73	b
D	0.80	0.80	0.30	0.91	0.86	0.62	0.83	a
	0.86	0.89	0.67	**0.95**	0.93	0.65	0.91	b
E	0.86	0.86	0.81	0.75	0.89	0.71	0.88	a
	0.82	0.90	0.78	0.71	0.93	0.66	0.92	b

表 4　Commented-out code の結果

	1	2	3	4	5	6	7	
A	0.57	0.54	0.47	0.36	0.51	0.38	0.33	a
	0.60	0.56	0.43	0.47	0.65	0.51	0.48	b
B	0.31	0.39	0.26	0.22	0.51	0.74	0.21	a
	0.61	0.50	0.47	0.15	0.57	0.32	0.23	b
C	0.27	0.33	0.29	0.19	0.16	0.15	0.25	a
	0.43	0.45	0.38	0.21	0.18	0.14	0.20	b
D	0.37	0.36	0.24	0.32	0.22	0.25	0.30	a
	0.38	0.37	0.15	0.37	0.47	0.30	0.29	b
E	0.46	0.47	0.38	0.24	0.46	0.20	0.29	a
	0.73	**0.79**	0.56	0.69	0.73	0.55	0.47	b

表 5　Not a smell の結果

	1	2	3	4	5	6	7	
A	0.72	0.75	0.71	0.74	0.73	0.69	0.49	a
	0.75	0.77	0.72	0.79	0.79	0.76	0.44	b
B	0.76	0.76	0.68	0.68	0.74	0.46	0.51	a
	0.78	0.78	0.70	0.66	0.76	0.54	0.52	b
C	0.64	0.69	0.62	0.40	0.39	0.40	0.60	a
	0.71	0.73	0.65	0.62	0.59	0.69	0.57	b
D	0.71	0.72	0.63	0.74	0.64	0.68	0.57	a
	0.75	0.77	0.60	**0.80**	0.73	0.71	0.53	b
E	0.73	0.75	0.66	0.68	0.70	0.73	0.56	a
	0.78	**0.80**	0.66	0.68	0.79	0.63	0.55	b

表 6　Obvious の結果

	1	2	3	4	5	6	7	
A	0.54	0.59	0.53	0.48	0.59	0.55	0.53	a
	0.58	0.62	0.54	0.47	0.63	0.51	0.56	b
B	0.59	0.58	0.50	0.51	0.54	0.30	0.52	a
	0.60	0.59	0.50	0.49	0.55	0.51	0.52	b
C	0.48	0.49	0.45	0.16	0.25	0.13	0.46	a
	0.52	0.48	0.44	0.32	0.33	0.23	0.43	b
D	0.55	0.53	0.42	0.58	0.52	0.49	0.54	a
	0.59	0.54	0.41	0.62	0.62	0.52	0.58	b
E	0.58	0.58	0.48	0.37	0.60	0.24	0.53	a
	0.65	0.66	0.50	0.52	**0.68**	0.49	0.60	b

表 7　Task の結果

	1	2	3	4	5	6	7	
A	0.67	0.51	0.61	0.40	0.54	0.22	0.36	a
	0.77	0.58	0.74	0.59	0.68	0.37	0.32	b
B	0.60	0.57	0.40	0.35	0.56	0.38	0.43	a
	0.69	0.60	0.62	0.21	0.58	0.26	0.45	b
C	0.26	0.24	0.20	0.19	0.19	0.13	0.24	a
	0.43	0.46	0.33	0.48	0.36	0.00	0.30	b
D	0.36	0.33	0.19	0.39	0.37	0.39	0.29	a
	0.56	0.63	0.30	0.64	0.61	0.00	0.40	b
E	0.48	0.47	0.24	0.46	0.49	0.21	0.36	a
	0.65	0.62	0.43	0.46	0.70	0.05	0.38	b

表 8　Vague の結果

	1	2	3	4	5	6	7	
A	0.17	0.20	0.15	0.16	0.14	0.17	0.10	a
	0.20	0.20	0.20	0.23	0.20	0.23	0.10	b
B	0.18	0.19	0.13	0.07	0.22	0.06	0.15	a
	0.23	**0.27**	0.13	0.10	0.25	0.14	0.11	b
C	0.13	0.16	0.10	0.07	0.06	0.05	0.11	a
	0.12	0.17	0.11	0.11	0.08	0.05	0.13	b
D	0.18	0.19	0.10	0.15	0.13	0.13	0.09	a
	0.21	0.20	0.15	0.19	0.14	0.15	0.12	b
E	0.21	0.19	0.11	0.10	0.15	0.13	0.11	a
	0.23	0.23	0.06	0.11	0.24	0.13	0.14	b

テーションコメントが含まれることが多い．TF-IDF というベクトル化手法は単語の出現回数と珍しさを数値で表現していることから，Task ではアノテーションコメントに該当するベクトルの成分が大きい数字になりやすく，Task 以外のコメントスメルではほとんどの場合でゼロになると考えられる．その結果，他のベクトル化手法以上に Task の特徴をベクトルで表現でき，高い検出精度を記録したものと考えられる．

以上をまとめると，Word2Vec 以外の手法が有効であり，その中でも TF-IDF が全体的に高い検出能力を持っていたことが分かる．Word2Vec は次元数が少なかったこともあり他の手法と比べると検出が困難であったと推察される．単純な方法でベクトル化を行っている TF-IDF が高い精度を記録したのは，コメントスメルごとにそれ固有の単語や記号が使われやすいという特徴をとらえるのに適した手法であったと考えられる．一方，実験結果の節でも示したように，コメントスメルによっては TF-IDF 以外の手法が最も高いスコアを記録していた例もあるため，一概に TF-IDF が適しているとは言い難い．

5.3 観点 3

観点 3 も観点 2 と同様にコメントスメルごとに異なる結果となった．まず，Beautification に関してはナイーブベイズと他のモデルとの間で有意な差が見られ，勾配ブースティングやランダムフォレスト，ロジスティック回帰モデルは決定木やサポートベクターマシンとの間で有意差が確認された．このことから，勾配ブースティング，ランダムフォレスト，ロジスティック回帰モデルの 3 つの機械学習モデルが Beatutification の検出には有効であるといえる．これは Beautification に限らず，全体的な傾向でもあった．特に Commented-out code では顕著であり，これら 3 つのモデルと残り 4 つのモデルとの間では，ほとんどの場合で（決定木とロジスティック回帰モデルの間を除くすべての場合で）有意差が見られた．

総合的に見て，勾配ブースティング，ランダムフォレスト，ロジスティック回帰モデルの 3 つがスメル検出精度の点で優れている可能性が見て取れ，全体の傾向を見るとランダムフォレストが最も有望なモデルあると思われる．ただし，これはあくまでも統計的有意差の有無を調べただけであるため，そこに現実的なレベルの差があることを保証するものではない．

最後に，検出精度とは別の視点として，モデルの構築と予測に要する時間にも着目する．検出精度の高かった 3 つのモデルについて，その他の条件を揃えた a-A-1, a-A-2, 及び a-A-5 のモデル構築と予測にかかった実時間を測定したところ，表 9 に示す結果が得られた．

表 9 から分かるように，ロジスティック回帰モデルが最も速く，約 1 分で処理を完了できていた．次いで速かったのはランダムフォレストであり，勾配ブースティングは他に比べて約 8 ～ 12 倍の時間を要していた．なお，ここでは TF-IDF を用いた測定であったが，試しに BM25 や CodeBERT も用いて実験してみたところ，数時間以上を要することもあった．このことから，勾配ブースティングは検出精度としては優れているといえるが，処理時間という観点では必ずしも良いモデルであるとは言い難い結果となった．

表 9　モデルごとの構築・予測時間

モデル	構築・予測時間
勾配ブースティング	755.73 秒
ランダムフォレスト	95.62 秒
ロジスティック回帰モデル	62.73 秒

RQ への回答

以上より RQ に対して以下の通り回答する：

RQ: 機械学習を用いたコメントスメルの自動検出において以下の 3 つの観点ではどういった組合せが有用か？

観点 1 の視点で見ると，ソースコードとコメントを別々にベクトル化を行うことで検出能力が向上することが期待される．観点 2 や観点 3 では，一概にどの方法が良いと断言するのは難しいが，ベクトル化手法では TF-IDF，BM25，CodeT5+，及び CodeBERT が有効であるといえる．そして，機械学習モデルでは，勾配ブースティング，ランダムフォレスト，並びにロジスティック回帰モデルが有効であるが，処理時間も考慮するとランダムフォレストが最も有望なモデルあると思われる．

6　まとめと今後の課題

本論文では，ソースコード理解の助けにならない等，必ずしも有用でないコメント，即ち “コメントスメル” の存在に注目し，その自動検出を扱った有望な先行研究の 1 つである Oztas ら [4] の研究に着目した．そこでは，ソースコードとコメントの内容をベクトル化し，機械学習モデルを用いることでコメントスメルの自動検出を行っていた．しかしながら，そこでのベクトル化のタイミング，ベクトル化の手法，並びに使用する機械学習モデルにはさまざまな組合せが考えられ，先行研究ではその一部しか試されていなかった．

そこで，より多様な組合せについても実験を行い，検出精度の比較・検討を行った．具体的には，全部で 70 種類の組合せを考え，先行研究と同じデータセットに対して比較実験を行った．結果として，次の知見が得られた：

- 先行研究ではソースコードとコメントをいったん 1 つの長い文字列にまとめ，それをベクトル化して機械学習モデルの入力データとしていたが，本実験の結果，先にソースコードとコメントを別々にベクトル化する方がスメル検出にはより有効であることが確認された．
- ベクトル化の手法に関しては，特定の手法が優れていると断言するのは難しく，TF-IDF，BM25，CodeT5+，及び CodeBERT のいずれかを使用するのが有効であることが分かった．
- 判定に使用する機械学習モデルとしては，勾配ブースティング，ランダムフォレスト，並びにロジスティック回帰モデルが有効であるが，処理時間も考慮するとランダムフォレストが最も有望なモデルあることが確認された．

今後の課題として，さらなる手法の組合せについて検討することが挙げられる．具体的には，今回の実験ではソースコードに対するベクトル化手法とコメントに対するベクトル化手法を同じものに揃えるかたちとしていたが，ソースコードがプログラミング言語で記述されるのに対し，コメントは自然言語で書かれるのが一般的であるため，同じベクトル化手法を適用するのが妥当かどうか定かでない．それゆえ，そういった観点についても実験を通じて検討していく必要がある．

謝辞 本研究は JSPS 科研費 25K15059，25K15062，26K15121 の助成を受けたものです．

参 考 文 献

[1] Çağlar, H. K., Çağlar, S., and Tüzün, E.: Automated Inline Comment Smell Detection and Repair with Large Language Models, *Proc. 40th IEEE/ACM International Conference on Automated Software Engineering*, November 2025, pp. 1008–1019.

[2] de Souza, S. C. B., Anquetil, N., and de Oliveira, K. M.: A study of the documentation essential to software maintenance, *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting & Designing for Pervasive Information*, New York, NY, USA, Association for Computing Machinery, September 2005, pp. 68–75.

[3] Jabrayilzade, E., Yurtoğlu, A., and Tüzün, E.: Taxonomy of inline code comment smells, *Empirical Softw. Engg.*, Vol. 29, No. 3(2024), pp. 58:1–58:53.

[4] Oztas, I., Torun, U. B., and Tüzün, E.: Towards Automated Detection of Inline Code Comment Smells, *Proceedings of the 29th International Conference on Evaluation and Assessment in Software Engineering*, New York, NY, USA, Association for Computing Machinery, June 2025, pp. 1127–1137.

[5] Tan, L., Yuan, D., Krishna, G., and Zhou, Y.: /* iComment: Bugs or Bad Comments? */, *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*, October 2007, pp. 145–158.

[6] Wang, C., He, H., Pal, U., Marinov, D., and Zhou, M.: Suboptimal Comments in Java Projects: From IndependentCommentChanges to Commenting Practices, *ACM Transactions on Software Engineering and Methodology*, Vol. 32, March 2023, pp. 1–33.