

Whodunit: Classifying Code as Human Authored or GPT-4 generated- A case study on CodeChef problems

The 21st International Conference on Mining Software Repositories (MSR24)

Oseremen Joy Idialu
j2idialu@uwaterloo.ca
University of Waterloo
Ontario, Canada

Noble Saji Mathews
noblesaji.mathews@uwaterloo.ca
University of Waterloo
Ontario, Canada

Rungroj Maipradit
rungroj.maipradit@uwaterloo.ca
University of Waterloo
Ontario, Canada

Joanne M. Atlee
jmatlee@uwaterloo.ca
University of Waterloo
Ontario, Canada

Meiyappan Nagappan
mei.nagappan@uwaterloo.ca
University of Waterloo
Ontario, Canada

LLMを用いたプログラミング

｜ GPT-4など、LLMを用いたAIアシスタントの登場

例) Github Copilot, ChatGPT

→プログラミングタスクに貢献

アシスト例

- ・コードスニペット生成
- ・バグ修正
- ・リファクタリング
- ・テストケース提案 など



貢献

- ・プログラム品質の向上
- ・労力の軽減
- ・生産性の向上

｜ 初心者の入門用の問題解決のアシストにも

ープログラミング学習の「アシスタント」として活躍

プログラミング教育現場における懸念

学生が、AIで生成したコードを、自分が書いたものとして提出するという不正行為が可能に

—AI生成であることを明示しなかったり、AIに課題を代行させたり、
という不正行為がすでに報告されている

本来、教育者側は、学生がAIアシスタント無しで、どの程度プログラミングできるのかを評価する必要がある

→AIが書いたコードか、人間が書いたコードかを見分ける必要がある(難)

→AIコード検出へ向けた研究へ

コーディングスタイルの違い

AI生成・人間コードには、スタイルの違いが見られる

```
1 def is_palindrome(n):
2     return str(n) == str(n)[::-1]
3
4 def palindromic_numbers_sum(l, r):
5     total = 0
6     for n in range(l, r+1):
7         if is_palindrome(n):
8             total += n
9     return total
10
11 t = int(input())
12
13 for i in range(t):
14     l, r = map(int, input().split())
15     result = palindromic_numbers_sum(l, r)
16     print(result)
```

AI生成：
空行や、補助的な関数の定義が特徴

```
1 for _ in range(int(input())):
2     l, r = map(int, input().split())
3     result = 0
4     for i in range(l, r + 1):
5         if str(i) == str(i)[::-1]:
6             result += i
7     print(result)
```

人間(CodeChefより)：
より短い変数名や関数名が特徴

Code Stylometry特徴量

プログラマー固有のコーディングスタイルの特徴（クセ）

例）インデントや空白，語彙，構文構造 など

→これらの特徴を元に，AI生成コードを検出できるのでは？

目的・アプローチ

目的

人間が書いたコードと、GPT-4が生成したコードを区別することの実現可能性を探る

アプローチ

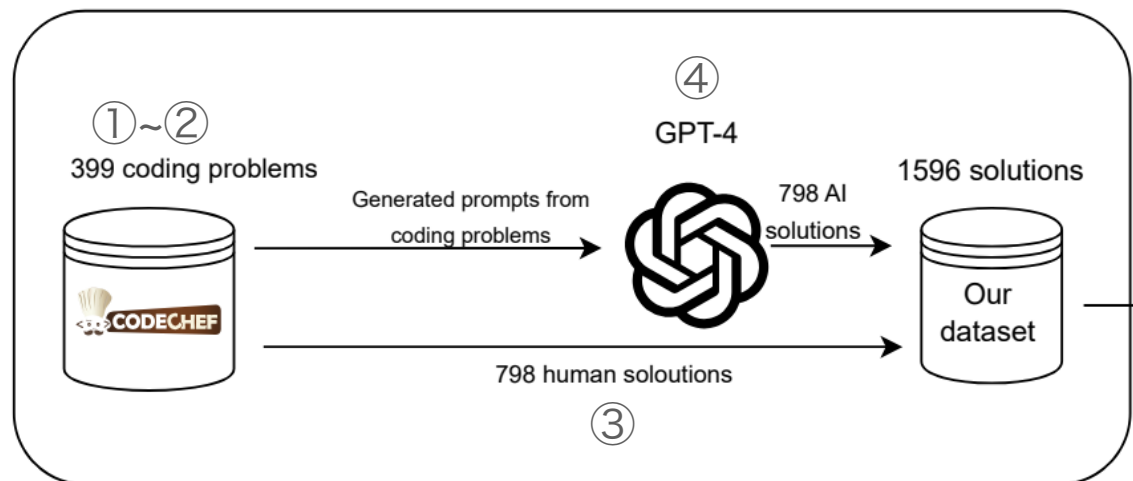
- ・ Code Stylometry
- ・ 機械学習

を用いて、AI生成か、人間かを分類・評価

実験方法 | データ収集

CodeChefより、様々なレベルの問題を収集 (python)

-CodeChef：有名な競技プロプラットフォーム。様々な難易度の問題。



①各難易度レベル(11段階)から人気の高い100問ずつを取得(1100問)

②2020年提出の、正答が2パターン以上あるもののみに絞り込み(399問)

③399問に対し、人間の回答を2パターンずつ収集(798回答)

④399問に対し、GPT-4の回答を2パターンずつ生成(798回答)

実験方法 | 分類器学習

収集した回答データから、特徴量を抽出

-特徴量は、いろんな先行研究からもってきた

例)

特徴量	概要
emptyLinesDensity	空白行の割合
ASTNodeTypeAvgDep	ASTノードの平均の深さ
keywordsDensity	特定のキーワードの出現頻度
cyclomatic Complexity	循環的複雑度

こういうのを、
31の基本特徴量と、
いろんな派生含めて、
合計140の特徴量を利用

抽出した特徴量とXGBoostで分類モデルを構築・学習

- ・ XGBoost：機械学習手法. 勾配ブースティングの一種
- ・ 10分割交差検証, accuracy, precision, F1, AUC-ROCで評価
- ・ RQに応じて追加の分析を行う

RQ1・2

RQ1.Code stylometryで、人間コードとGPT-4コードをどの程度区別できるか？

—AI生成コードの検出について、提案手法の有効性を評価

RQ2.人間コードとGPT-4コードの区別において、Gameableでない特徴量の影響力はどの程度か？

Gameableな特徴量：
簡単に変更できるスタイル
主に空白やスペースの使用など



- ・ AIが生成したコードに空行を追加したり
 - ・ スペースの調整をしたり
- することで、AI生成であることを隠蔽しようとする可能性がある

→こういった特徴量を除外しても、分類性能は保たれるか？

RQ1・2 | 分類性能

	Our Approach		Baseline		
	All	Non-Gameable	Naive	n-grams + L n = 2	n = 3
Accuracy	0.91	0.89	-	0.86	0.88
Precision	0.91	0.89	0.5	0.86	0.87
Recall	0.91	0.89	0.5	0.88	0.88
F1-score	0.91	0.89	0.5	0.87	0.88
AUC-ROC	0.91	0.89	-	0.86	0.88

ベースライン

XGBoostを用いた既存手法
特徴量には、コード中のトークンの出現頻度を利用
トークンをn-gramで取得

要するに、コードの表層的な情報を
特徴量とした先行研究

Code Stylometry使用により、分類性能が向上

Gameable特徴量を除外した場合、わずかに精度が低下したが、十分高い精度が得られた

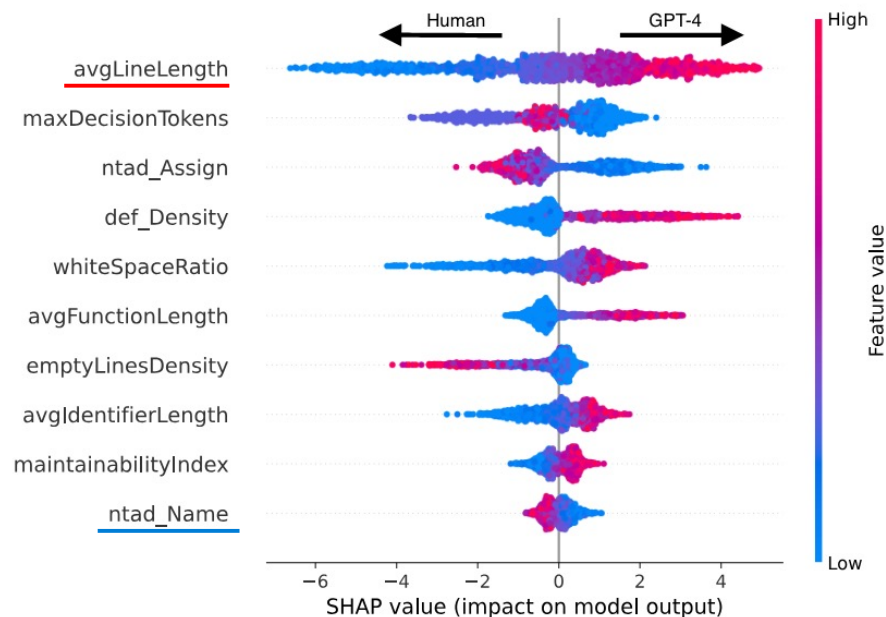
RQ1 | 特徴量の重要度

SHAP (Shapley Additive Explanations)

ゲーム理論に基づいた、機械学習モデルの予測を説明するための手法

協力ゲームで、各プレイヤーの貢献度を測るためのもの？

→どの特徴量が、どれくらい分類に寄与しているかを調査

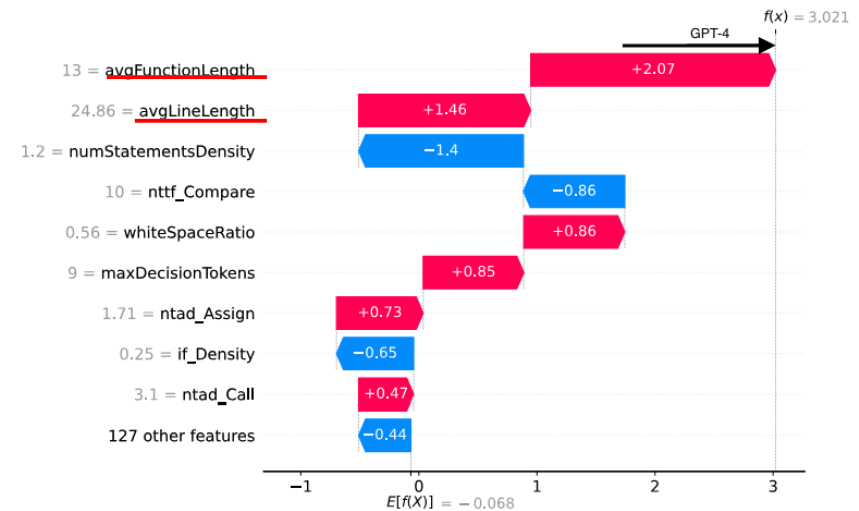


- **avgLineLength**が、分類に大きく寄与
→コードの平均行長について、
人間は短く、AIは長い傾向にある
- **ntad_Name**は、比較的寄与度は小さい
→ASTにおける識別子の登場位置について、
人間は深く、AIは浅い位置に出現する
傾向がある

偽陽性のコード例

```
1 def getCount(h,m,i) :
2     h=int(h)
3     m=int(m)
4     h1=0
5     lst=[11,22,33,44,55,66,77,88,99]
6     while(h1<h) :
7         for m1 in range(0,m) :
8             if h1<10 :
9                 if(m1<10 and h1==m1) : count[i]+=1
10                if (m1 in lst and m1%10==h1) : count[i]
11                ]+=1
12            else :
13                if(m1 in lst and h1==m1) : count[i]+=1
14                if(h1 in lst and h1%10==m1) : count[i]+=1
15            h1+=1
16 t=int(input())
17 count=[0]*t
18 for i in range(0,t) :
19     h,m=input().split()
20     getCount(h,m,i)
21 for i in count : print(i)
```

人間が書いたコード
(AIが書いたと誤分類)



そのSHAP図

偽陽性の例では、平均行長や平均関数長の値が高い
→長い行を持つコードは、偽陽性として誤分類されやすい

RQ3・4 | 軽くまとめ

RQ3： **正解の回答**のみで学習した場合の性能

- ・ 元々の回答には、不正解の回答も含まれているらしい
→ 正解の回答のみに絞って学習した場合の性能は？
- ・ 結果： ほぼ同等の性能
→ コードの正確性ではなく、コーディングスタイルの違いに基づいて分類していることがわかる

RQ4： **問題の難易度**による分類性能の違い

- ・ 問題には様々な難易度がある
→ 難易度別(Easy, Medium, Hard)に学習した場合の性能は？
- ・ 結果： 難易度が上がるにつれて性能は僅かに低下
→ 差は小さく、提案手法は難易度に関わらず利用可能

まとめ・所感

まとめ

コードスタイルの特徴であるCode Stylometryを用いると,

- ・ 人間が書いたコード
- ・ AIが生成したコード

を分類できる可能性を示唆

所感

- ・ プロンプト次第では誤魔化せたりするのか？
- ・ CodeBERTなどで得られるコードの分散表現を用いた分類はどうなんだろう？