

ARCLIN: Automated API Mention Resolution for Unformatted Texts

Yintong Huo, Yuxin Su, Hongming Zhang, Michael R. Lyu

In Proceedings of the 44th International Conference on Software Engineering, pp.?? - ??

APIとオープンフォーラム

APIはプログラミングにおいて必要不可欠な要素

利点

- 他のライブラリから機能を借りることで開発を効率化

欠点

- APIが数多く存在するため、全てのAPIに精通することが不可能

オープンフォーラムでの知識の共有

- Stack Overflowなどで、開発者は直面している問題の解決方法を議論し情報を収集する

→ API推薦[37], API誤用検知[29][30], 文書拡張[35]などの支援

非構造化テキストのなかからAPIメンションを認識し特定する技術

コンテキストと研究課題 (1)

オープンフォーラムでは誤字や略語が多く見られる

- 専門家でない開発者によって記述される

一般的な単語とAPIの名前の違いを識別できない

- 正規表現を用いたルールベース手法では対応できない

識別できない例 1

Question ID	Sentence	API
#66952125	So far I managed to use view once in my first very simple project...	torch.view()
#59905234	You can use .numpy() to view the internal data...	None

- **view** (メソッド名) : PyTorchのviewメソッド
- **view** (動詞) : 見るを意味するview

研究課題 (2)

識別できない例 2

- コードブロックを使用せずにAPIの情報が記述される

コードブロックを未使用

APIメソッドかどうかの認識とAPIエントリとの照会が必要

コードブロックを使用

APIエントリとの照会のみで済む

PyTorch : How to properly create a list of `nn.Linear()`

Asked 2 years, 11 months ago · Active 2 years, 11 months ago · Viewed 6k times

`torch.nn.Module`

I have created a class that has `nn.Module` as subclass.

17 In my class, I have to create N number of linear transformation, where N is given as class parameters.

I therefore proceed as follow :

```
self.list_1 = []
for i in range(N):
    self.list_1.append(nn.Linear(self.x, 1, bias=mlp_bias))
```

In the forward method, i call these matrices (with `list_1[i]`) and concat the results.

`torch.nn.ModuleList`

You can use `nn.ModuleList` to wrap your list of linear layers as explained [here](#)

22 `self.list_1 = nn.ModuleList(self.list_1)`

Share Improve this answer Follow

answered May 22 '18 at

phi 5,978 3 15

Add a comment

目的

従来手法の課題に対応したAPIの認識と特定をおこなう手法の提案

課題

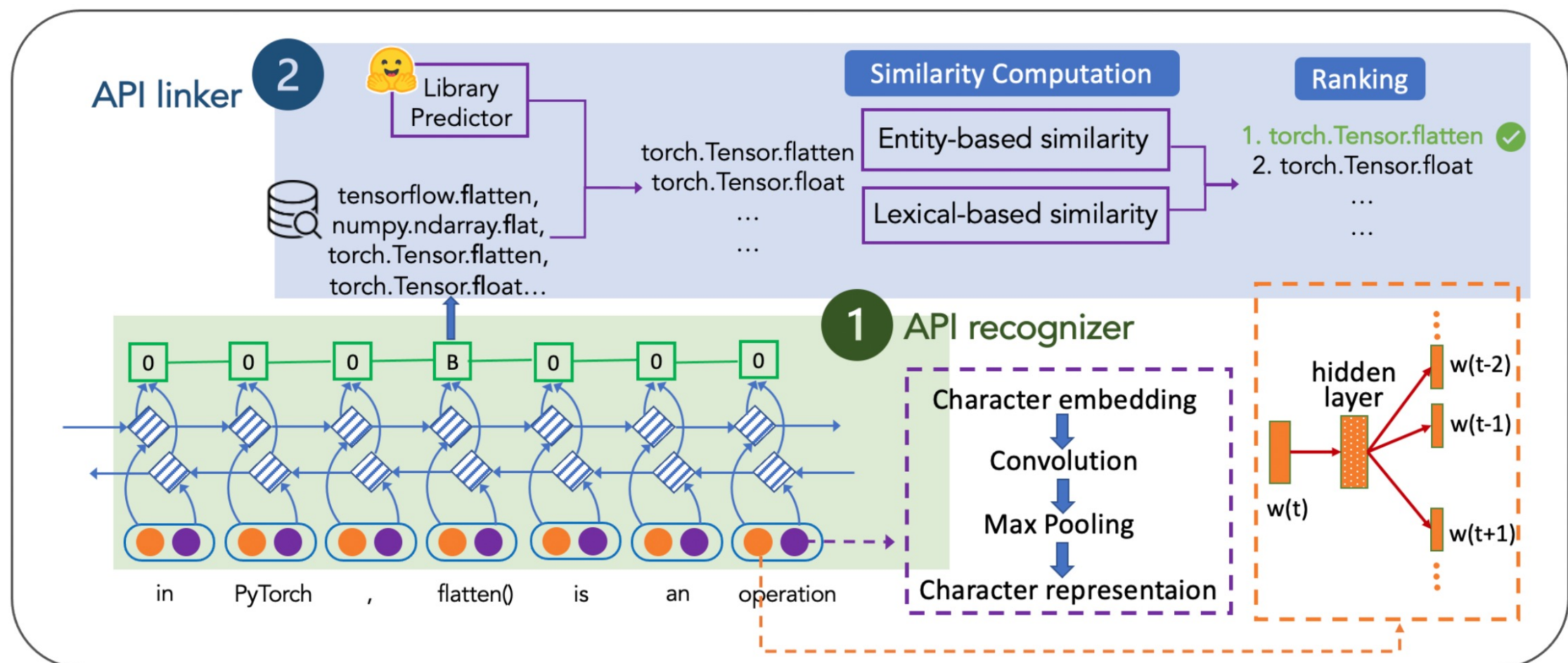
- 単語の曖昧性や略語を考慮した識別
- コードブロックを使用していないAPI記述の識別

ARCLIN

| API Recognition and Contextual Linking

② API linker

特定されたトークンと類似度が高いAPIエントリをランキング



① API recognizer

文章をBi-LSTMに入力し，CRF（Conditional Random Field）を用いて出力値からAPIを言及しているトークンを特定

API recognizer (1)

正解トークンの自動ラベリング

- 手動で正解ラベルを付与するにはコストがかかりすぎるため、以下の方法で自動でラベリング
 - トークンがエントリされているAPIの識別子と完全一致
 - 最後に“()”を含むトークン
 - “.”を含むトークン

Data Augmentation

- 上記の方法で特定したAPIを言及しているトークンを変形しデータを増やす
 - x.viewをviewに置換させて学習させる など

※ 実際に全てのAPIの識別子が記述されることは多くないため

API recognizer (2)

Context Encoder (Bi-LSTM)

入力するベクトル

- 単語レベルの埋め込み表現 (Skip-gram^[25]) ^{[15][18][38]}
- 文字レベルの埋め込み表現 (CNN) ^[22]

Tag Decoder (CRF)

- 隣接する単語の依存関係を考慮しながらAPIを言及しているトークンかどうかの確率値をスコアリング
- 最も確率値が高いトークンを採用しAPI linkerに送信

API linker (1)

Candidate Selection

- 全比較は時間的コストを浪費するため以下の処理を実施
 - 入力されたトークンの最後のエンティティを抽出
 - 抽出されたエンティティの最初の2文字を抽出
 - 抽出された2文字と一致するAPIエントリを候補に選択

Library Predictor

- 文章を入力し, 関係するライブラリを予測
 - 異なるライブラリでも, 同じ名前のAPIを保有する場合があるため

API linker (2)

Lexical Similarity

- M : API recognizerで特定したAPIを言及しているトークン
- E_i : Candidate SelectionでリストアップしたAPIエントリ候補
- $sim(w, E_i)$: FastTextで算出したベクトルのコサイン類似度

$$Sim_L(M \rightarrow E_i) = \frac{\sum_{w \in M} sim(w, E_i) * IDF_{token}(w)}{\sum_{w \in M} IDF_{token}(w)}, \quad IDF(w) = \log\left(\frac{\#Documents_Number}{\#Document_with_w + 1}\right)$$

$$Sim_L(M, E_i) = \frac{Sim_L(M \rightarrow E_i) + Sim_L(E_i \rightarrow M)}{2}$$

Entity Similarity

$$Sim_J(M, E_i) = \frac{\sum_{w \in (M \cap E_i)} IDF_{entity}(w)}{\sum_{w \in E_i} IDF_{entity}(w)}$$

最終的に2つの類似度を足し合わせたスコアでランキング

$$Sim(m, e_i) = Sim_L(m, E_i) + Sim_J(m, E_i).$$

データセット

Stack Overflowから5つのPythonライブラリタグが付けられた質問, 回答, コメントを収集

訓練データセットと自動ラベリングの結果

Library	#Sentence	#Autolabel	#Augmentation
PyTorch	150,000	11,057	27,077
Tensorflow	150,000	8,925	21,899
Pandas	150,000	10,537	25,450
Numpy	150,000	11,501	27,941
Matplotlib	150,000	9,769	23,564
Sum	750,000	51,789	125,931

テストデータセット

Library	Version	#API	#Mention	#Sentence
PyTorch [12]	1.8.0	2,472	133	562
Tensorflow [14]	2.4.1	10,361	87	532
Pandas [13]	1.2.4	2,174	117	573
Numpy [11]	1.20	1,913	116	580
Matplotlib [10]	3.4.1	6,937	105	583
Sum	-	23,857	558	2,830

- テストデータセットはライブラリごとに600文収集したのち, 手動で検査し確実にAPIのメンションものをのみ使用

性能評価

	Approach	Precision Recall F1		
		Precision	Recall	F1
従来手法	RuleBase-Pure	1.00	0.070	0.131
	RuleBase-Knowl	1.00	0.314	0.478
	APIReal	0.787	0.604	0.683
	- w/o rules	0.823	0.477	0.599
	ARCLIN Ensemble	0.784	0.742	0.762
	- PyTorch	0.861	0.801	0.830
	- Tensorflow	0.576	0.840	0.683
	- Pandas	0.717	0.778	0.746
	- Numpy	0.865	0.741	0.798
	- Matplotlib	0.825	0.762	0.792

F1値で評価すると、従来手法より性能が向上

- State-of the-artであるAPIRealと比較して、F1値が8%向上

効いている要素は何か？

	Precision	Recall	F1
ARCLIN (ensemble)	0.784	0.742	0.762
- w/o recognizer	0.112	0.783	0.195
- w/o lib_pred	0.649	0.715	0.680
- w/o lexical_sim	0.814	0.439	0.570
- w/o entity_sim	0.645	0.719	0.680

API recognizerを利用しないと劇的にPrecisionが低下する

- API linkerではAPIに関係ないトークンのフィルターができない

まとめ・所感

目的

- 従来手法の課題に対応したAPIの認識と特定をおこなう手法（ARCLIN）の提案

結果

- 従来手法より良い性能を発揮した
 - State-of-the-artである手法に比べて、F1値が8%向上

所感

- Library Predictorの存在意義がわからない
 - 関連するライブラリのラベルが投稿についているのでは？
 - ラベルがつけられないことが多いとかがあるのか？