

ライブチャットコミュニティからの 問題と解決策ペアの自動抽出

ISPY: Automatic Issue-Solution Pair Extraction from Community Live Chats

Lin Shi^{*§1}, Ziyu Jiang^{*§1}, Ye Yang[¶], Xiao Chen^{*§}, Yumin Zhang^{*§}, Fangwen Mu^{*§},
Hanzhi Jiang^{*§}, Qing Wang^{*†‡§2},

^{*}Laboratory for Internet Software Technologies, [†]State Key Laboratory of Computer Sciences,

[‡]Science&Technology on Integrated Information System Laboratory,

Institute of Software Chinese Academy of Sciences, Beijing, China;

[§]University of Chinese Academy of Sciences, Beijing, China;

[¶]School of Systems and Enterprises, Stevens Institute of Technology, Hoboken, NJ, USA;

Email: {shilin, ziyu2019, chenxiao2021, yumin2020, fangwen2020, hanzhi2021, wq}@iscas.ac.cn, yyang4@stevens.edu

問題解決ツールとしてのチャット

OSSコミュニティで利用されるチャットは、

- ・ソフトウェアのインストールやセットアップ
- ・ビルドやコンパイル
- ・バグ

などで発生する，様々な問題を解決するためのツールとして使用される^[7, 8]

先行研究では^[9, 10]，3つのOSSプロジェクトの約9千件のチャットメッセージを分析

- ・問題の報告：24%
- ・解決策の提案：51%

⇒ チャット履歴には，問題と解決策のペアが豊富に含まれている

[7] O. Ehsan, S. Hassan, M. E. Mezouar, and Y. Zou, “An Empirical Study of Developer Discussions in the Gitter Platform,” ACM Trans. Softw. Eng. Methodol., vol. 30, no. 1, pp. 8:1–8:39, 2021.

[8] L. Shi, X. Chen, Y. Yang, H. Jiang, Z. Jiang, N. Niu, and Q. Wang, “A First Look at Developers’ Live Chat on Gitter,” in ESEC/FSE ’21, Virtual Event. ACM, 2021.

[9] R. Alkadhi, T. Lata, E. Guzman, and B. Bruegge, “Rationale in Development Chat Messages: An Exploratory Study,” in MSR 2017. IEEE Computer Society, 2017, pp. 436–446.

[10] R. Alkadhi, M. Nonnenmacher, E. Guzman, and B. Bruegge, “How do Developers Discuss Rationale?” in 25th International Conference on Software Analysis, Evolution and Reengineering, SANER 2018. IEEE Computer Society, 2018, pp. 357–369.

障壁

チャット履歴から問題と解決策を抽出する際、以下のような障壁

- 複数の対話が同時並行で発生
→ 異なる対話の分離
- 崩れた表現や重複などのノイズ
→ 前処理 + 特徴表現学習（深層学習）
- 膨大な量の履歴
→ 自動化

目的とアプローチ

目的

OSSコミュニティのチャットから問題と解決策のペアを自動抽出する深層学習モデルを構築

アプローチ

1. 既存研究を使ってチャット履歴を個別の対話に分離
2. 局所的なコンテキストを考慮した特徴表現を学習し、対話の中から問題と解決策のペアを抽出

チャットから問題と解決策を抽出した一例

対話

- 一連の発話のかたまり
- モデルへの入力単位

 Tom Dec 21 2016 20:17
Hi. I am having a little bit of trouble setting up all of the dependencies for spark. With this import line: `import org.deeplearning4j.spark.impl.paramavg.ParameterAveragingTrainingMaster;`

 Jack Dec 21 2016 20:17
@Tom Make sure all your dl4j versions are the same first of all
You saw our examples, right?

 Mike Dec 21 2016 20:18
Our latest examples available here: <https://github.com/deeplearning4j/dl4j-examples>.
Please check. Those has the pom.xmls you need.

 Tom Dec 21 2016 20:18
Give me five mins. I will look more into that.

 Jack Dec 21 2016 20:18
Usually it's just dl4j-spark

 Tom Dec 21 2016 20:18
Which version?

 Jack Dec 21 2016 20:19
with the right scala version
dl4j-spark_2.10 or dl4j-spark_2.11

 Tom Dec 21 2016 20:25
Finally figured it out. Thank you so much. @Jack @Tom



Extracted Issue-solution Pair

Issue:

I am having a little bit of trouble setting up all of the dependencies for spark. With this import line: `import org.deeplearning4j.spark.impl.paramavg.ParameterAveragingTrainingMaster.`

Solution:

Make sure all your dl4j versions are the same first of all. Our latest examples available here: <https://github.com/deeplearning4j/dl4j-examples>. Those has the pom.xmls you need. Usually it's just dl4j-spark, with the right scala version. dl4j-spark_2.10 or dl4j-spark_2.11.

発話

• 発言の単位

(Teamsならメッセージを入力してEnterを押すまで)

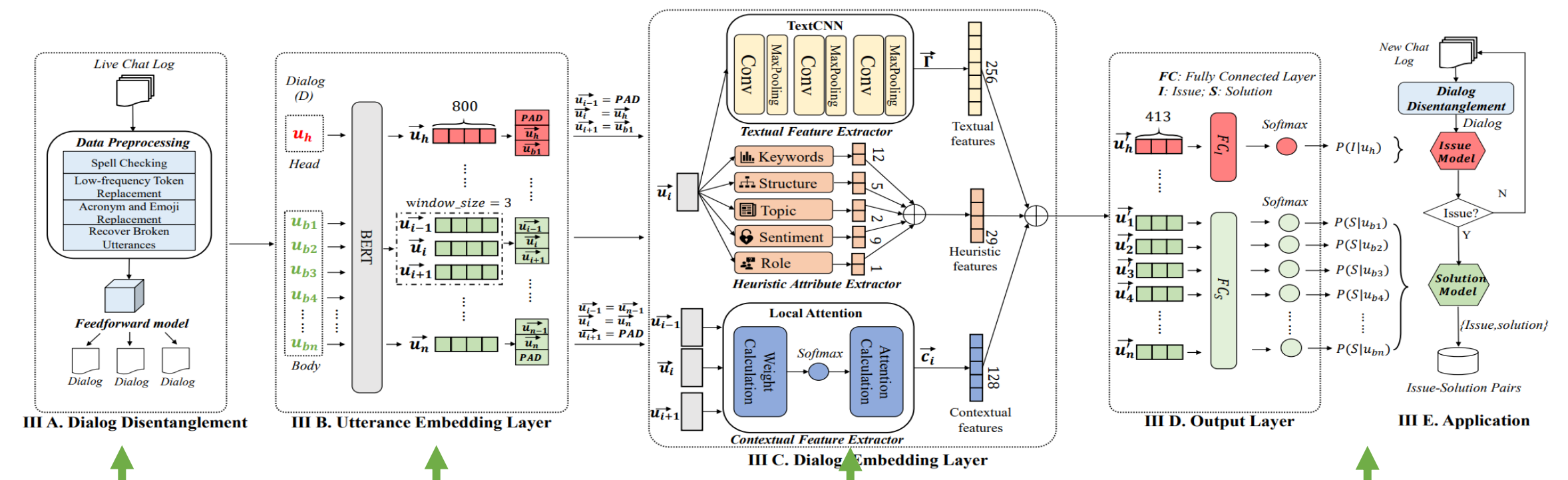
• 1~数文程度

• 特徴表現は 発話単位で学習

発話単位で
問題と解決策を抽出
することが最終目標

概要

大きく分けると4ステップ



Step1. 複数の対話を
個別の対話に分離

Step2. 発話を文章
ベクトルに埋め込む

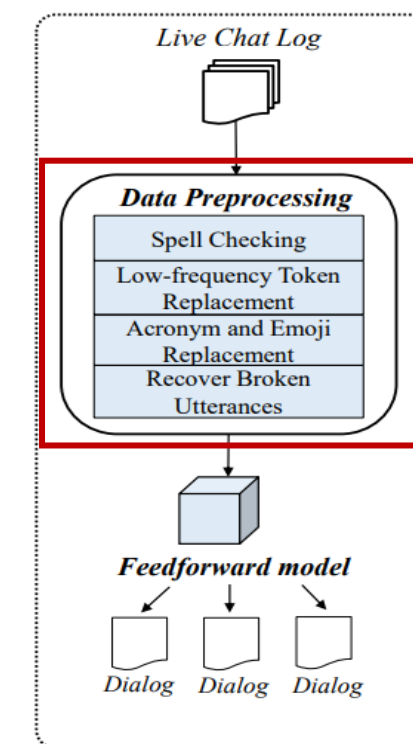
Step3. 発話をコンテキストを
考慮したベクトルに埋め込む

Step4. 発話が問題や
解決策かを判定

方法 | Step1. 複数の対話を個別の対話に分離

前処理

- 一般的な自然言語の前処理
ストップワード除去，正規化など
- チャット特有の前処理
略語を元に戻す，絵文字の置換
- OSSコミュニティ特有の前処理
URL，コード片，バージョン番号など，
低頻度トークンを特定のトークンに置換



III A. Dialog Disentanglement

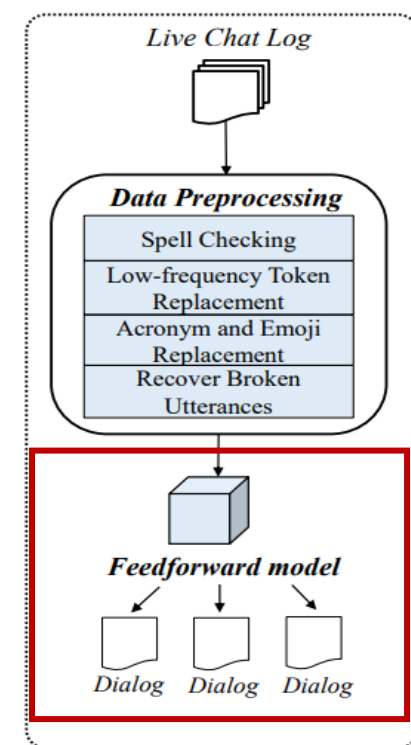
方法 | Step1. 複数の対話を個別の対話に分離

チャット履歴から複数の対話を個別の対話に分離

既存手法^[24]の学習済みモデルを使って2つの発話同士を紐づけ

- 入力：2つの発話
一致する単語数や、発話の中に互いのメンションが含まれるかなど、77次元の特徴量
- 処理：2層512次元のFeedforwardNN
- 出力：2つの発話が繋がっているかどうか

→ 紐づけられた一連の発話を、個別の対話とする



III A. Dialog Disentanglement

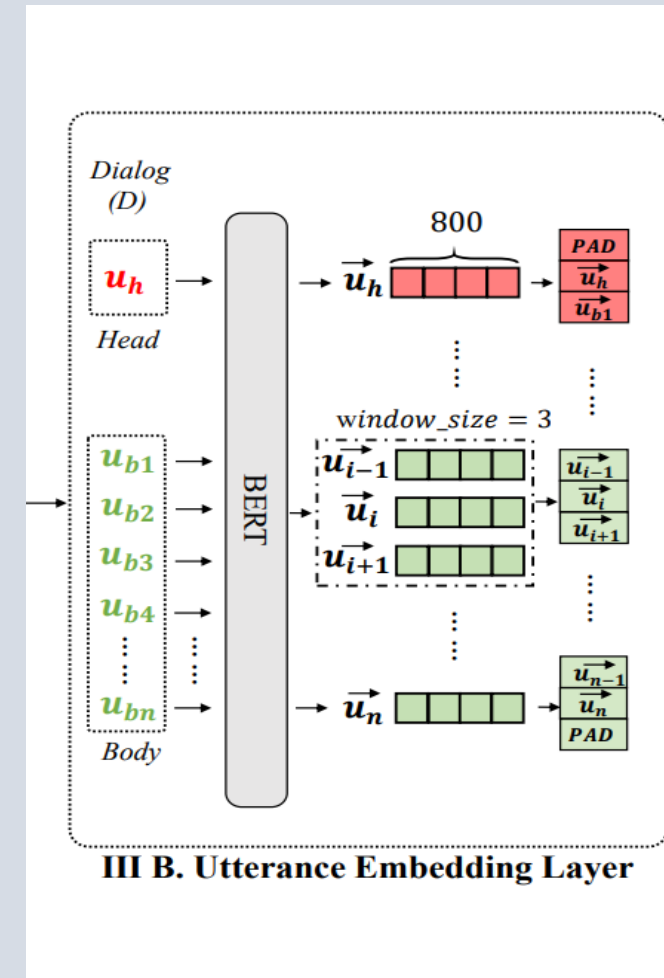
[24] J. K. Kummerfeld, S. R. Gouravajhala, J. Peper, V. Athreya, C. Gunasekara, J. Ganhotra, S. S. Patel, L. Polymenakos, and W. S. Lasecki, "A Large-Scale Corpus for Conversation Disentanglement," in Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers).

方法 | Step2. 発話を文章ベクトルに埋め込む

BERT

事前学習済みのBERTモデル^[25]を使って、
各発話を800次元の埋め込みベクトルに変換

- 入力：発話
- 出力：発話の埋め込みベクトル



[25] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," CoRR, vol. abs/1810.04805, 2018.

方法 | Step3. 発話をコンテキストを考慮したベクトルに埋め込む

概要

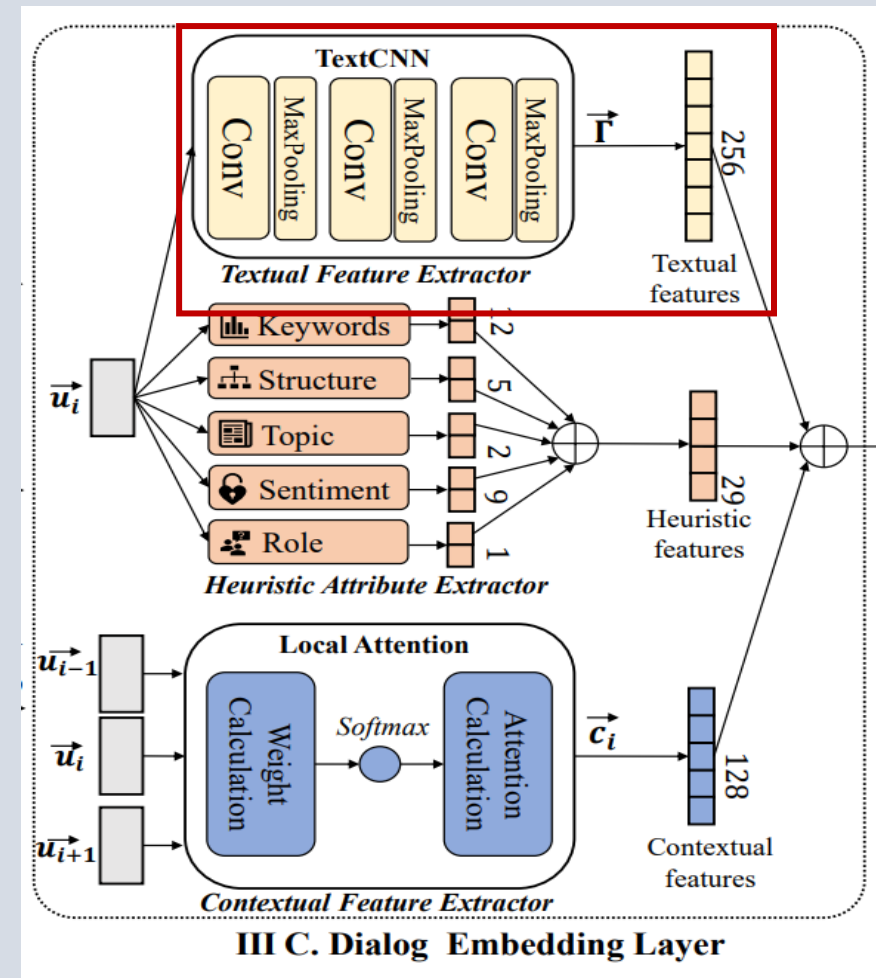
3つの方法で、コンテキストを考慮した発話の特徴表現を学習し、結合

- TextCNN
- ヒューリスティックな特徴抽出
- Local Attention

TextCNN[29]

データ数が少ないので、パラメータ数が少ないモデルを使う必要

- 入力：発話の埋め込みベクトル
- 出力：256次元のコンテキストベクトル



[29] Y. Kim, "Convolutional Neural Networks for Sentence Classification," arXiv preprint arXiv:1408.5882, 2014.

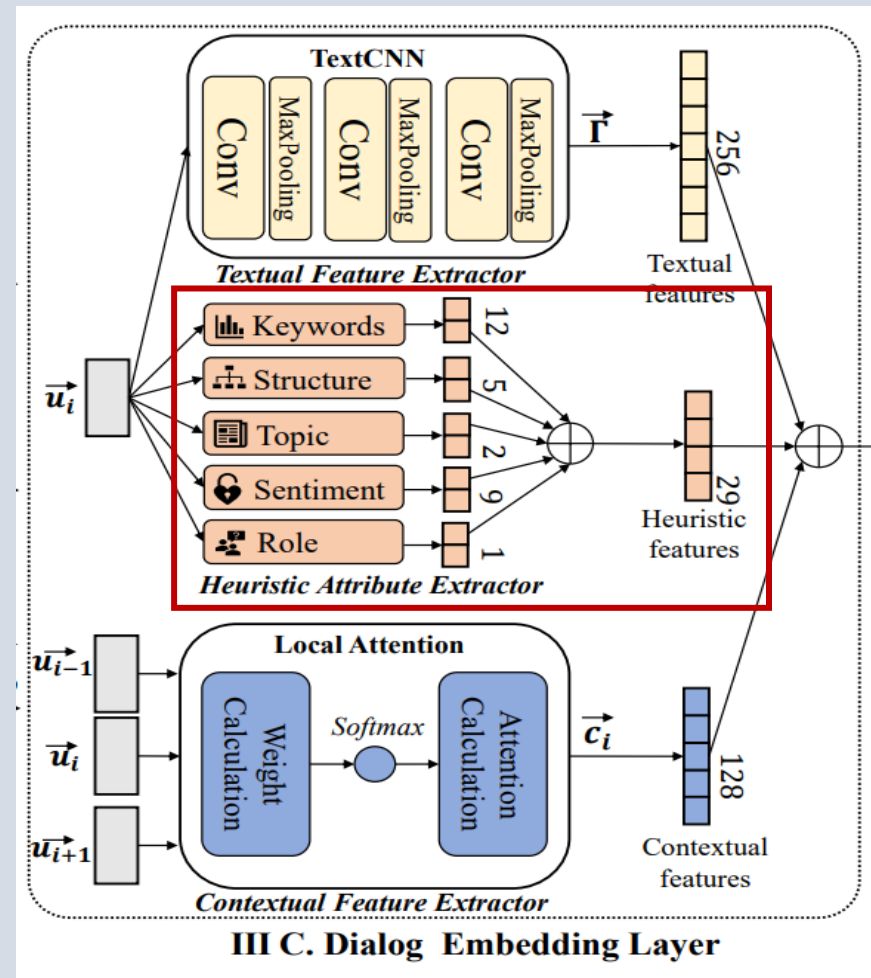
方法 | Step3. 発話をコンテキストを考慮したベクトルに埋め込む

ヒューリスティックな特徴量抽出

いくつかの観点で、発話から特徴量を抽出

- Keyword : 5W1Hや「？」などを含んでいるか
- Structure : 単語数, ユニーク単語数など
- Topic : 対話全体のトピックと発話のトピックの差
対話全体のTF-IDFベクトルと発話のTF-IDFベクトルのユークリッド距離
- Sentiment : 発話の感情極性
- Role : 発話者のGitter上での役職

出力 : 29次元のベクトル



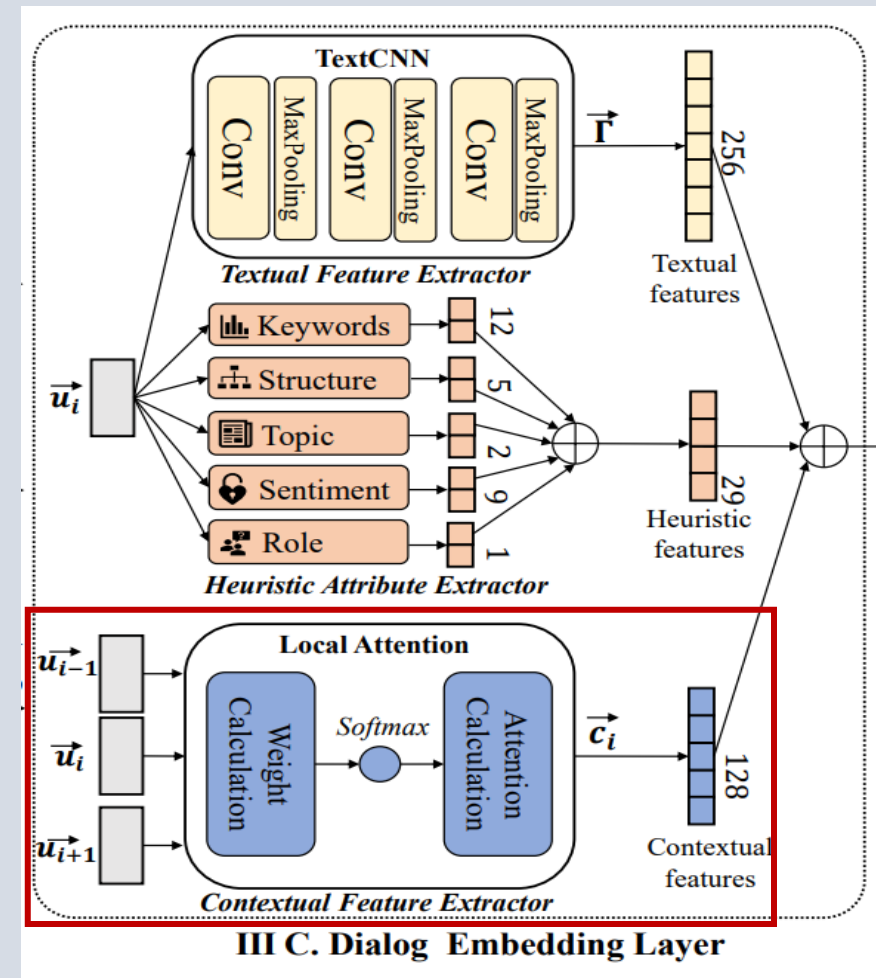
方法 | Step3. 発話をコンテキストを考慮したベクトルに埋め込む

Local Attention^[33]

- 入力：発話とその前後の発話の埋め込みベクトル
- 出力：前後の発話によって重みづけされた128次元のベクトル

Local Attentionの狙い

- 問題検出
問題の説明と，それに対する最初の返答の関係を捉えられる
- 解決策検出
解決策が複数の発話にまたがる場合でも，その関係を捉えられる



[33] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," CoRR, vol. abs/1409.0473, 2015.

方法 | Step4. 発話が問題や解決策かを判定

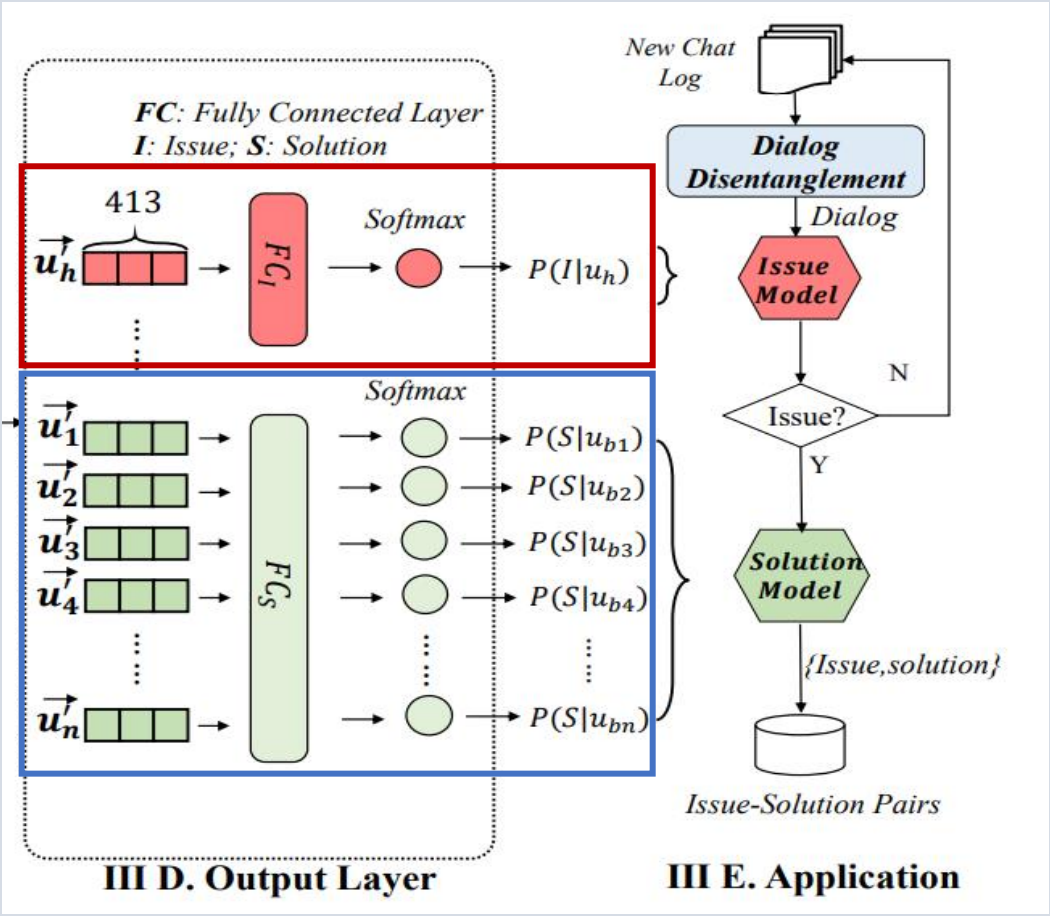
対話を「最初の発話」と「最初以降の発話」に分離

最初の発話

発話が問題かどうかを全結合層で2値分類

最初以降の発話

各発話が解決策かどうかを全結合層で2値分類



提案モデルを評価するため，以下のリサーチクエスチョンに取り組む

RQ1：発話が問題かどうかを検出する性能はどの程度か？

RQ2：発話が問題の解決策かどうかを検出する性能はどの程度か？

RQ3：個々のコンポーネントは検出性能にどの程度貢献しているか？

データセット

異なる8つのドメインから，Gitter（チャットサービス）の参加者上位1位のOSSプロジェクトを選択し，Gitterのチャット履歴を収集

アノテーション

- ・各プロジェクトから100件の対話をランダムサンプリング
- ・対話中の各発話を4人の博士課程の学生がアノテーション
 - ・最初の発話が問題かどうか
 - ・最初以降の各発話が問題に対する解決策かどうか

訓練方法

評価対象以外のプロジェクトのデータでモデルを訓練（クロスプロジェクト）

RQ1：発話が問題かどうかを検出する性能はどの程度か？

比較対象のベースラインモデル

- 典型的な文章分類モデル
ナイーブベイズ（NB） ， ランダムフォレスト（RF） ， 勾配ブースティング決定木（GBDT）
- 既存研究のモデル
LSTMベース（Casper[42]） ， CNNベース（CNC_PD[44]） ， ルールベース（DECA_PD[46]）

Task	Methods	Angular			Appium			Docker			DL4j			Ethereum			Gitter			Nodejs			Typescript			Average		
		P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Issue	ISPY	76	77	76	75	68	71	84	74	79	77	68	72	82	73	77	80	69	74	79	70	74	86	78	82	80	72	76
	NB	36	40	38	41	30	35	47	36	41	70	56	62	08	25	13	22	42	29	30	50	37	15	40	22	34	40	36
	RF	56	25	34	69	30	42	75	23	35	84	44	58	100	17	29	50	25	33	33	13	18	23	30	26	61	26	36
	GBDT	27	75	40	40	70	51	50	79	61	73	44	55	21	76	33	19	67	29	30	88	44	18	90	30	35	65	46
	Casper	39	35	37	08	03	05	59	26	36	46	40	43	19	42	26	14	17	15	05	06	06	15	40	22	26	26	26
	CNC_PD	20	55	29	23	50	32	23	36	28	12	32	17	24	42	30	12	42	19	10	50	17	05	40	10	16	43	24
	DECA_PD	33	50	40	28	37	31	33	36	34	64	28	39	42	42	42	44	67	53	32	50	39	04	10	06	35	40	37

- 平均すると，Precision，Recall，F1すべての指標で提案モデルが優位
 - ベースラインでF1が最も高いモデルと比べても，Precision +45%，Recall +7%，F1 +30%
- 個々のプロジェクトにおいても，F1は提案手法が最も高い

RQ2：発話が問題の解決策かどうかを検出する性能はどの程度か？

比較対象のベースラインモデル
先ほどとほぼ同じ

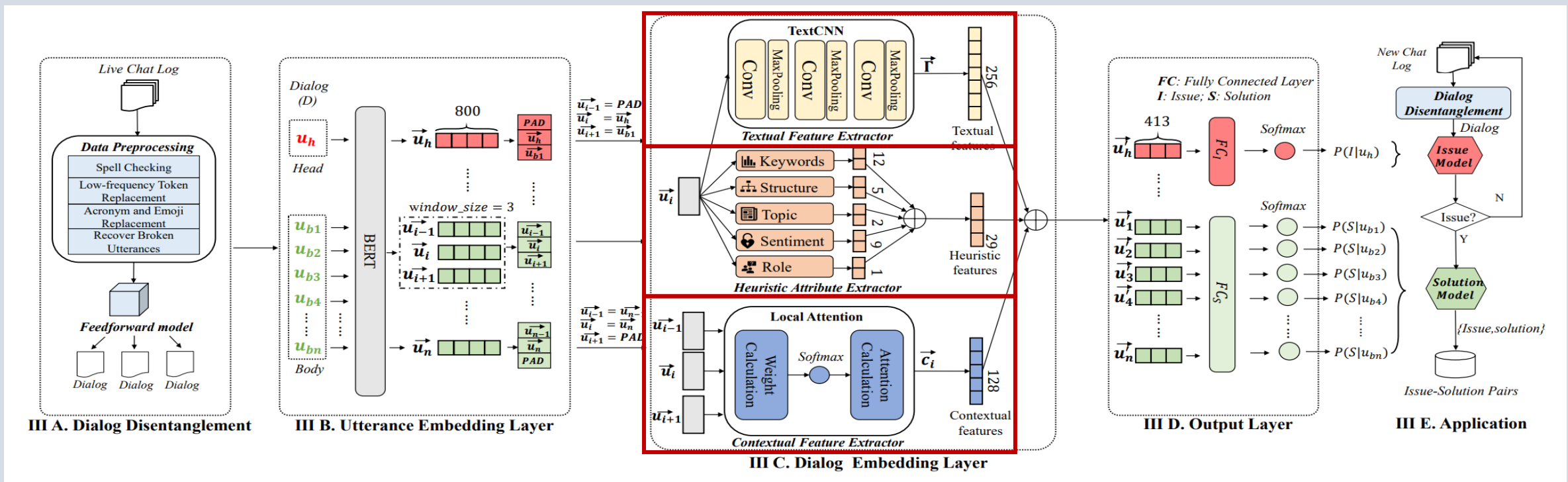
Task	Methods	Angular			Appium			Docker			DL4j			Ethereum			Gitter			Nodejs			Typescript			Average		
		P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Solution	ISPY	61	58	69	68	57	62	71	60	65	66	62	58	73	63	68	62	56	59	68	58	63	72	67	69	68	59	63
	NB	21	58	30	24	48	32	31	59	40	58	49	53	33	80	47	10	67	17	37	55	44	08	09	09	28	53	37
	RF	26	83	39	27	52	36	31	56	40	80	13	22	15	40	22	10	33	15	50	65	57	54	64	58	37	51	43
	GBDT	26	92	40	26	74	38	22	62	32	71	16	26	11	40	17	10	67	17	42	75	54	42	73	53	31	62	41
	UIT	29	17	21	21	13	17	30	12	17	24	07	12	37	16	22	27	11	17	19	15	18	31	18	23	27	14	18
	CNC_SP	18	17	17	26	28	27	21	31	25	46	20	28	30	14	19	46	54	50	41	28	33	38	25	30	33	27	30
	DECA_SP	00	00	00	00	00	00	20	09	12	25	02	24	00	00	00	00	00	00	57	20	30	33	09	14	17	05	08

- 平均すると、Recall以外の指標は提案モデルが優位
 - Recallが最も高いモデルと比べても、Precision +37%，Recall -3%，F1 +22%
- 個々のプロジェクトにおいても、PrecisionとF1は提案手法が最も高い

RQ3：個々のコンポーネントは検出性能にどの程度貢献しているか？

アブレーションスタディの方法

Step3の発話のコンテキスト埋め込みで，それぞれの特徴表現の性能への影響を評価

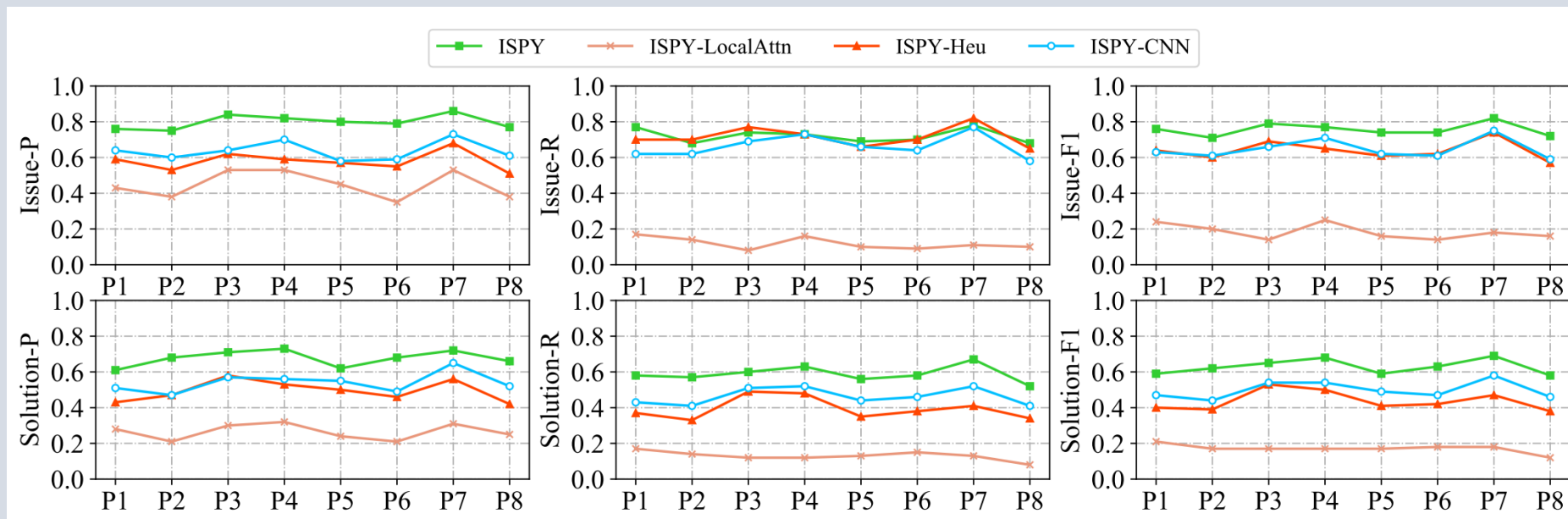


RQ3：個々のコンポーネントは検出性能にどの程度貢献しているか？

結果

問題検出

解決策検出



- **Local Attention**の特徴表現が特に重要 (ISPY-LocalAttn)
 - 平均すると，問題検出ではF1 -57%，解決策検出ではF1 -46%
- TextCNN，ヒューリスティック特徴表現でも，F1 10~20%影響

目的

OSSコミュニティのチャットから問題と解決策のペアを自動抽出する深層学習モデルを構築

アプローチ

1. 既存研究を使ってチャット履歴を個別の対話に分離
2. 局所的なコンテキストを考慮した特徴表現の学習

結果

- ・ベースラインと比較して、問題検出ではF1が30%，解決策検出では22%向上
- ・アブレーションスタディを行い，Local Attentionを使った特徴表現が特に重要

所感

いわゆるGlobal Attentionとの比較はなかったが，局所的な関係性を捉える方が重要な場合もある