

意味的類似度に基づいたソースコード とドキュメントの紐付け

オープンソースソフトウェア工学研究室
学籍番号：60230113
氏名：松尾 和輝

ドキュメント（ユーザマニュアル）

ソフトウェアに関する情報をユーザに提供する文書

- － 開発者がソフトウェアの機能説明，使用法などを記述

Latent Dirichlet allocation (LDA)

LDA is implemented as an Estimator that supports both `EMLDA0optimizer` and `OnlineLDA0optimizer`, and generates a `LDAModel` as the base model. Expert users may cast a `LDAModel` generated by `EMLDA0optimizer` to a `DistributedLDAModel` if needed.

Examples

Python [Scala](#) [Java](#) [R](#)

Refer to the [Python API docs](#) for more details.

```
from pyspark.ml.clustering import LDA

# Loads data.
dataset = spark.read.format("libsvm").load("data/mllib/sample_lda_libsvm_data.txt")

# Trains a LDA model.
lda = LDA(k=10, maxIter=10)
model = lda.fit(dataset)
```

ドキュメントの例¹

ソフトウェアの更新に伴いドキュメントも更新される

1. <https://spark.apache.org/docs/latest/ml-clustering.html>

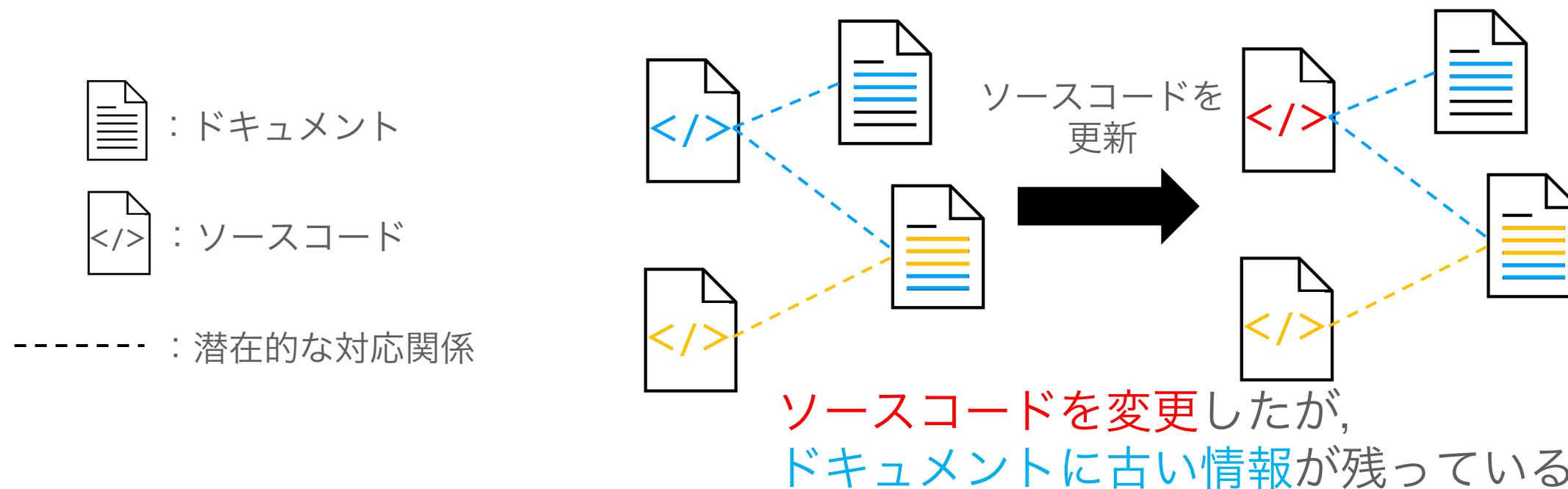
コードとドキュメントの不整合

ドキュメントの更新が遅れるor忘れられることがある

- コードとドキュメントが明示的に対応付けられておらず、開発者が更新すべきドキュメントを見逃してしまうため

問題

- ドキュメント参照者に間違った情報を与えてしまう^[1]
- ソフトウェアの品質低下^[2]



[1] Mira Kajko-Mattsson. 2005. A survey of documentation practice within corrective maintenance. Empirical Software Engineering 10, 1 (2005), 31–55.

[2] David Lorge Parnas. 1994. Software aging. In Proceedings of International Conference on Software Engineering. 279–287.

先行研究[?]

目的 | ソースコードとドキュメントの明示的な紐付け

- 空間的観点 | コードエレメントを用いた単純マッチング
- 時間的観点 | 更新タイミングの時間的近接性
 - それぞれ単体, および組み合わせた手法で評価

コードエレメント

- 変数名, 関数名などのソースコードに関係する単語

Environment Variables	
Environment Variable	Meaning
SPARK_DAEMON_MEMORY	Memory to allocate to the history server
SPARK_DAEMON_JAVA_OPTS	JVM options for the history server (default: -Xmx1g)
SPARK_DAEMON_CLASSPATH	Classpath for the history server (default: .)
SPARK_PUBLIC_DNS	The public address for the history server, replace the internal address of the server, replace the internal address of the server, replace the internal address of the server
SPARK_HISTORY_OPTS	spark.history.* configuration options for the history server

ドキュメント²

正規表現により抽出

コードエレメント

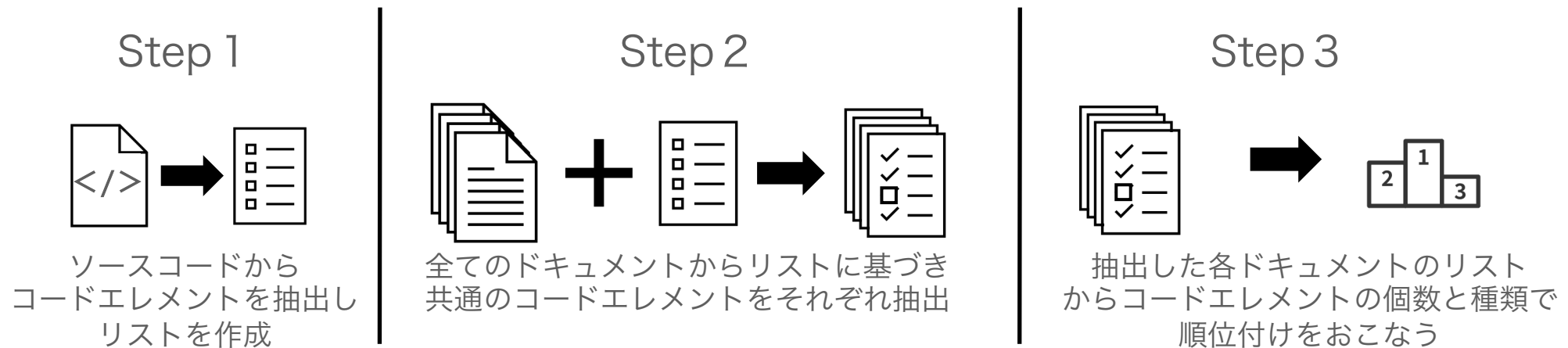
SPARK_DAEMON_MEMORY
SPARK_DAEMON_JAVA_OPTS
SPARK_DAEMON_CLASSPATH
SPARK_PUBLIC_DNS
SPARK_HISTORY_OPTS
⋮

[?]
2. <https://spark.apache.org/docs/3.5.1/monitoring.html>

先行研究 | アプローチ

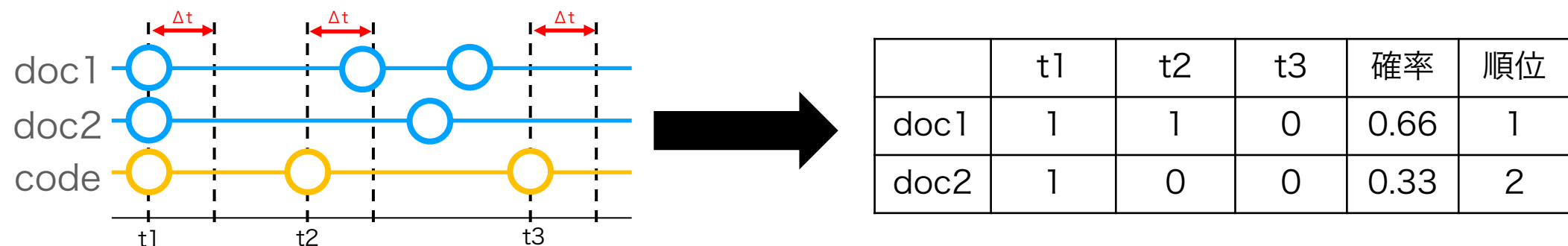
- 空間的観点

- ソースコードとドキュメントで共通するコード要素の個数 or 種類に基づき順位づけをおこなう



- 時間的観点

- ソースコードの更新から一定期間内にドキュメントが更新される確率に基づき順位付けをおこなう



先行研究 | 課題

- 共通するコードエレメントが無いと紐付けできない
 - 先行研究のデータセットの37%は共通するコードエレメントを持たない
 - このうち24%は時間的観点でも紐付けできない
- 原因
 - 同一ではないが類似するコードエレメントで記述されているものが多い

具体例

```
case object ClearCacheCommand extends Le  
  
override def run(sparkSession: SparkSe  
    sparkSession.catalog.clearCache()  
    Seq.empty[Row]  
}  
}
```

ソースコード³

コードエレメント

sparkSession

clearCache()

ClearCacheCommand

対応関係の
あるペア



CLEAR CACHE

Description

CLEAR CACHE removes the entries and as
and views.

ドキュメント⁴

コードエレメント

CACHE

CLEAR

コードエレメント
抽出

- 共通するコードエレメントが存在しないという判定

意味的類似度に基づいた紐付けが可能なのは？

→ Sentence-BERT^[4]

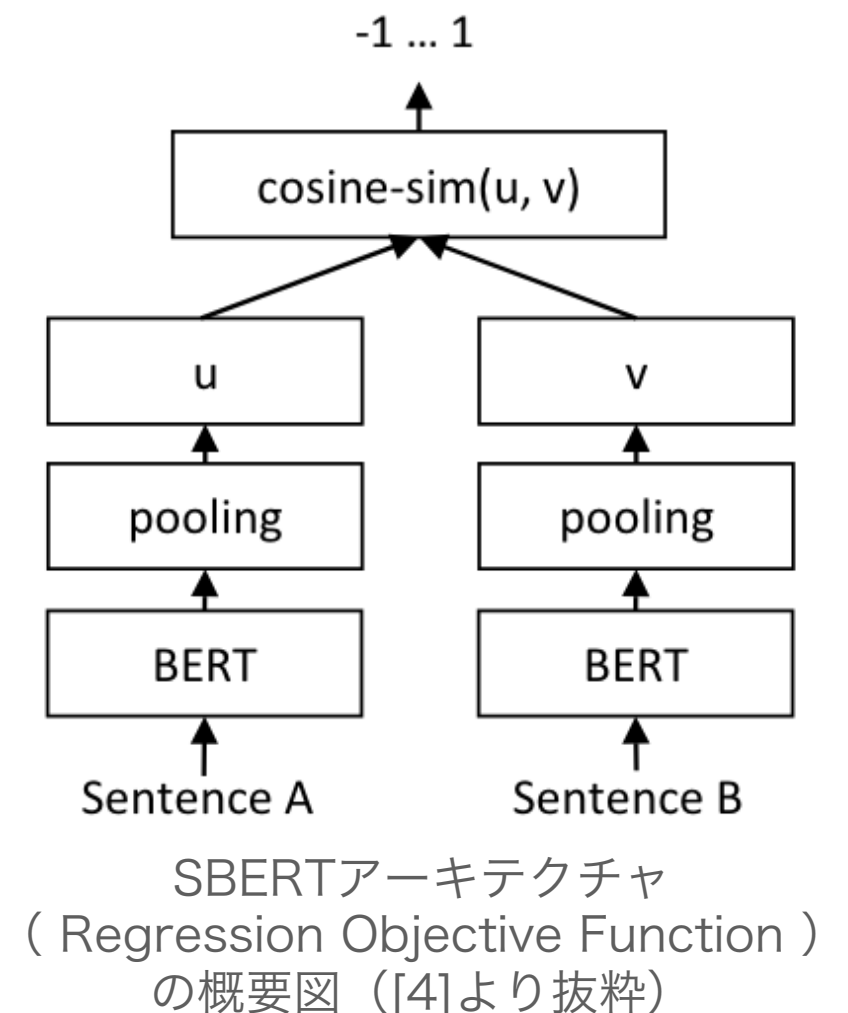
³ <https://github.com/apache/spark/blob/fd86f85e181fc2dc0f50a096855acf83a6cc5d9c/sql/core/src/main/scala/org/apache/spark/sql/execution/command/cache.scala>

⁴ <https://spark.apache.org/docs/3.5.1/sql-ref-syntax-aux-cache-clear-cache.html>

[4] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pp. 3982–3992. Association for Computational Linguistics, November 2019.

Sentence-BERT (SBERT) [4]

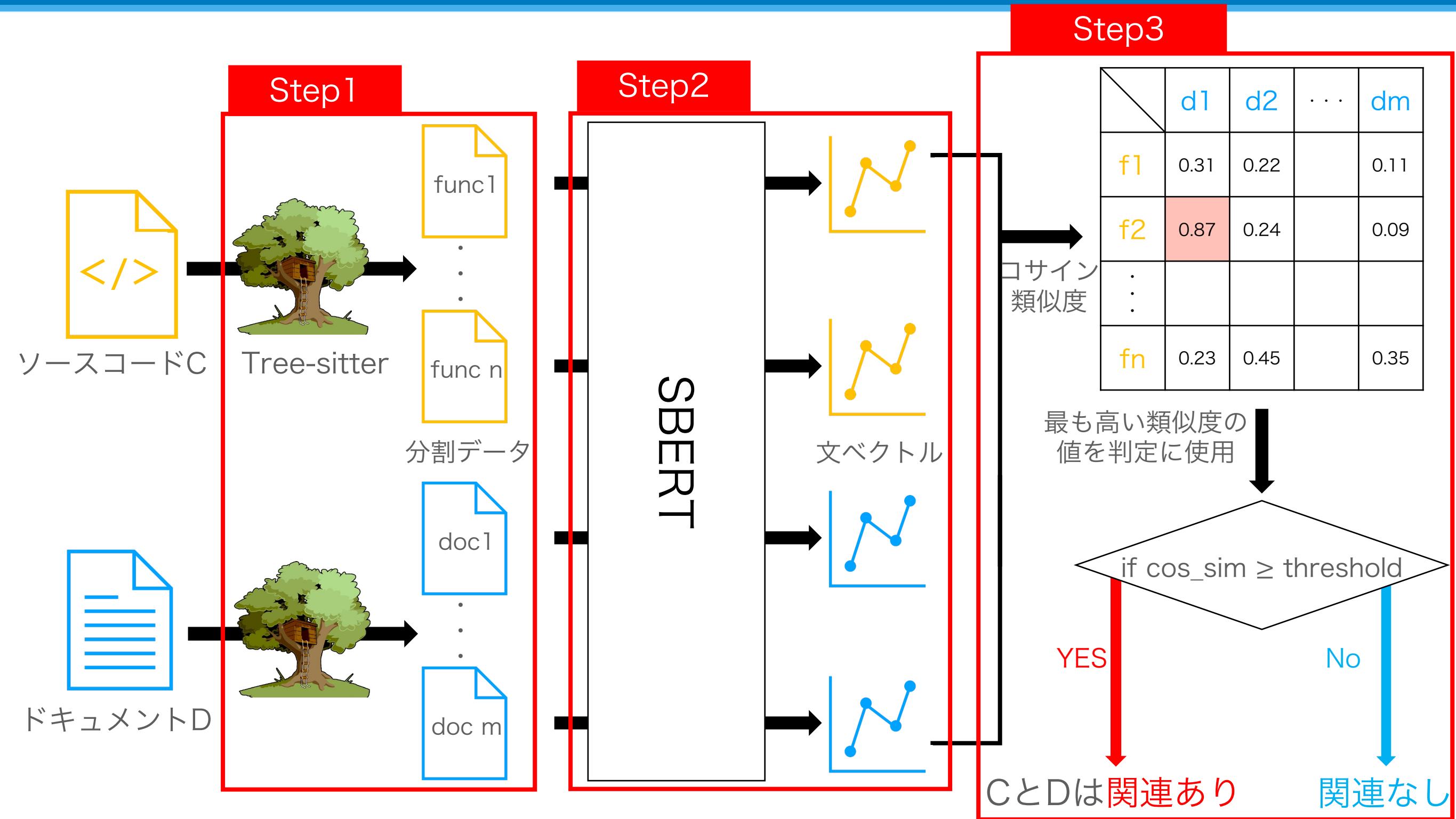
- Siamese networkを用いてファインチューニングさせることで**文章のベクトル化**に特化させたモデル
 - BERTモデルの出力にpooling層を追加
- [4]では、以下3種類の目的関数を紹介
 - Classification Objective Function
 - Regression Objective Function
 - Triplet Objective Function
- 文章同士の類似性/非類似性を学習
- BERTよりも**効率的に精度の高い**類似度計算が可能^[4]
 - BERTでは65時間かかる最も類似するペアの探索を約5秒でおこなうことが可能
 - BERTから出力される文ベクトルは質が低い



目的・アプローチ

- 目的
 - 意味的類似度に基づいたソースコードとドキュメントの紐付け手法の構築
- アプローチ
 - Step1 :
 - ソースコードをコードブロックに分割
 - ドキュメントを見出しごとに分割
 - Step2 :
 - 分割したソースコードとドキュメントのベクトル化
 - Step3 :
 - すべてのソースコードとドキュメント間のコサイン類似度を計測
 - コサイン類似度が閾値以上のペアがあれば関連があると判定

イメージ図



正解集合

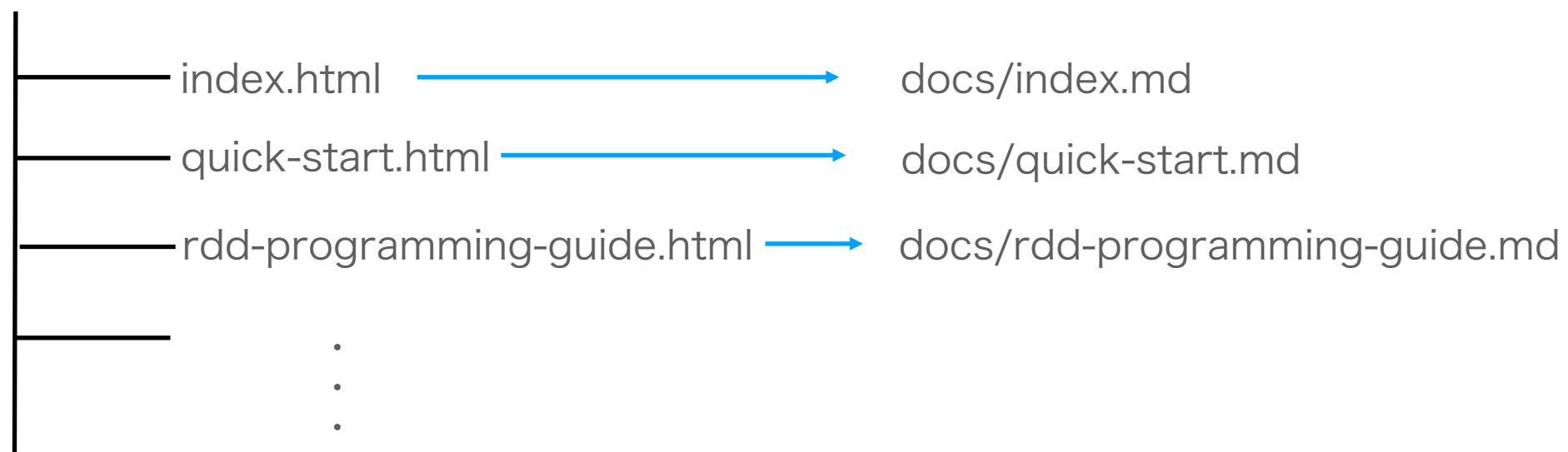
- 本研究には、明示的なソースコードとドキュメントのペアが必要となる
 - そのようなデータセットは存在しないため、自作必要
- GitHubにホスティングされているapache/spark⁵というプロジェクトを対象
 - apache/sparkのバージョン：3.5.1 (latest)
 - hash : fd86f85e181fc2dc0f50a096855acf83a6cc5d9c

5. <https://github.com/apache/spark>

正解集合 | 対象ドキュメント

- <https://spark.apache.org/docs/3.5.1/>以下の階層にあるドキュメントほぼ全てが対象
 - Javadoc, scaladoc等のソースコードコメントから生成されたドキュメントは除外
- 合計311件
 - ※ソースコードと関係のないドキュメントも含むのもっと減る
 - web urlとGitHub内のmdファイル, rstファイルとを紐づける

<https://spark.apache.org/docs/3.5.1/>



正解集合 | 対象ソースコード

- apache/sparkの主要なプログラミング言語
 - Scala (拡張子: scala) → 2,787件
 - Python (py) → 441件
 - Java (java) → 829件
 - R (R) → 66件
- テストに関連するものは除外
 - ドキュメントと関連がないと思われるため
 - “test”という名称のフォルダ内にあるファイル全て除外
 - core/src/test/java/test/org/apache/spark/JavaAPISuite.java

正解集合 | 作成方法

1. ドキュメントのコミット履歴から，同時に変更されているソースコードのうち，関連がありそうなものを目視（時間的観点をカバー）
2. 単純マッチングにより関連がありそうなものを目視（空間的観点をカバー）
3. ドキュメントと関係のありそうなテストコードを探し，テストコードと同時に変更されているコードを目視（1, 2以外をカバー）

CLEAR CACHE

Description

CLEAR CACHE removes the entries and as
and views.

ドキュメント⁴

```
+  
+ test("Clear all cache") {  
+   sql("SELECT key FROM testData LIMIT  
10").registerTempTable("t1")  
+   sql("SELECT key FROM testData LIMIT  
5").registerTempTable("t2")  
+   cacheTable("t1")  
+   cacheTable("t2")  
+   clearCache()  
+   assert(cacheManager.isEmpty)
```

テストコード⁶

6. <https://github.com/apache/spark/pull/4694/files>

データセット

- 明確に関連のあるソースコードが存在すると判断したドキュメントをデータセットとする
 - ドキュメント数：147件

SBERTの実装

- sentence-transformers⁷ライブラリを使用

- 使用モデル

- BERT (bert-base-uncased)
- CodeBERT (microsoft/codebert-base)

```
from sentence_transformers import SentenceTransformer, models

word_embedding_model = models.Transformer("bert-base-uncased")
pooling_model = models.Pooling(
    word_embedding_model.get_word_embedding_dimension(),
    pooling_mode_mean_tokens=True,
)
model = SentenceTransformer(modules=[word_embedding_model, pooling_model])
```

- pooling層：Mean-pooling

- 各単語ベクトルの平均を文章ベクトルとする方法

7. <https://sbert.net/>

設定

- データセットの分割
 - 147件のドキュメントを
訓練：検証：テスト = 8：1：1
に分割（117：15：15）
 - 関連するソースコードとドキュメントペアの数
訓練：検証：テスト = 389：57：109
 - ブロックレベルのペア数（ファインチューニングに使用）
訓練：検証 = 82,704：5,262

設定

- Loss : Triplet Loss

- anchorと類似する文章positive, 類似しない文章negativeの3つ組からなる

- 以下の損失関数を最小化するように学習

$$\max(\|s_a - s_p\| - \|s_a - s_n\| + \varepsilon, 0)$$

s_a : anchorの文章ベクトル
 s_p : positiveの文章ベクトル
 s_n : negativeの文章ベクトル
 ε : マージン

- 本研究ではanchorをコードブロック, positiveとnegativeをドキュメントブロックとする

- negativeサンプルの選択

- ① anchorと関連しないドキュメントブロックをランダムに選択
- ② anchorと関連しないドキュメントブロックのうち, 最も類似するものを選択

評価（思案中）

- テストデータ109件（正例） に対して関連しないペア109件（負例） を追加
 - 計218件のペアを用いて評価
- 評価指標
 - precision, recall, f1
- 関連するかを決定する閾値
 - 0.5から1.0まで0.01刻みで閾値を変更し, 評価
 - f1が最も高いときの閾値を採用

結果

		閾値	precision	recall	f1
BERT	FTなし	結果待ち			
	random				
	high-similar				
CodeBERT	FTなし				
	random				
	high-similar				

今後の予定

- 改善案考える
 - コードエレメントとの併用
 - 特徴量の追加？
- BERT使ってみる？
- 他プロジェクトではどういう結果になるのかの確認

補足資料

正規表現リスト

- キャメルケース, スネークケース, 関数名など

表 3.1: code element の抽出に用いる正規表現一覧 [4]

正規表現	抽出される code element の例
<code>(?<=(?<!'') [!_a-z~]+(?='))</code>	<code>'CREATE'</code>
<code>[A-Z] [a-zA-Z]+ ?<[A-Z] [a-zA-Z]*></code>	<code>PairMap<T></code>
<code>[a-zA-Z0-9\.] + ([a-zA-Z_ , \.]* [a-zA-Z_ , \.])</code>	<code>main()</code>
<code>[a-zA-Z]+ \. [A-Z]+</code>	<code>sub.PIPE</code>
<code>[@] [a-zA-Z]+</code>	<code>@Override</code>
<code>([a-zA-Z]{3,} \. [A-Za-z]+ _ [a-zA-Z_]+)</code>	<code>command.config_value</code>
<code>([A-Z]{2,})</code>	<code>AWS</code>
<code>([A-Z]+ _ [A-Z0-9_]+)</code>	<code>CLUSTER_CONF_DIR</code>
<code>([a-z]+ _ [a-z0-9_]+)</code>	<code>get_classpath</code>
<code>([A-Z]+ [a-z0-9]+ [A-Z] [a-z0-9]+ \w*)</code>	<code>VersionInfo</code>
<code>([a-z0-9]+ [A-Z]+ \w*)</code>	<code>configurationFile</code>
<code>(\w+ \([^\)]*\))</code>	<code>join(STORM_DIR, "lib")</code>
<code>([A-Z] [a-z]+ [A-Z] [a-zA-Z]+)</code>	<code>ConfigParser</code>
<code>([a-z]+ [A-Z] [a-zA-Z]+)</code>	<code>configurationFile</code>
<code>\{\{ [^\}]* \}\}</code>	クラスブロックとメソッドブロック
<code>\{\% [^\%]* \% \}</code>	<code>{% endautoescape %}</code>
<code>' [^']* '</code>	<code>'like'</code>
<code>-- [^_]* --</code>	<code>__main__</code>
<code>\\$ [A-Za-z_]+</code>	<code>\$STORM</code>

先行研究の課題メモ

- 先行研究の正解ペア166件を，各手法で紐付けできるか調べた結果
 - 空間的観点で紐付けできるとは
 - コードとドキュメントで共通するコードエレメントが1つ以上ある（出現回数の多いコードエレメントを除外した上で）
 - 時間的観点で紐付けできるとは
 - コードが更新されてから，1日以内にドキュメントが更新されたことがある

空間的観点で紐付け 時間的観点で紐付け	空間的観点で紐付け	
	できない	できる
できない	15	20
できる	47	84

- 両方の手法で紐付けできないものは15件（9%）
- 空間的観点で紐付けできないもののうち，時間的観点でも紐付けできないものの割合は（24%）