

A time-based approach to automatic bug report assignment



R Shokripour,



J Anvik,



ZM Kasirun,



S Zamani

紹介者：柏 祐太郎

1. 概要

Bugtriageを支援する手法の多くは、tf-idfを利用

それぞれの文書の特徴付ける単語かどうかを判断する指標
⇒特定の文書にだけ現れる単語と、ありふれた単語に差

$$\text{tf-idf} = \text{tf} * \text{idf}$$

tf : Terms Frequency

それぞれの単語の文書内での出現頻度
⇒たくさん出現する単語は重要

$$\frac{\text{文書A内の単語aの出現回数}}{\text{文書A内の全単語数}}$$

idf : Inverse Document Frequency

それぞれの単語の出現する文書数の対数
⇒多くの文書で出現する単語は重要でない

$$\log \left(\frac{\text{文書数}}{\text{単語aが出現する文書数}} \right) + 1$$

2-2. とある日のブログのtf-idf

今日は肉と寿司とピザとラーメンを食べた後に肉を食べた(文書A)

◆TF (文書A, 肉) = 肉の出現数 / 単語数 = $2/5 = 0.4$

2-2. とある日のブログのtf-idf

今日は**肉**と**寿司**と**ピザ**と**ラーメン**を食べた後に**肉**を食べた(文書A)

他の日のブログが以下の場合,

- 昨日も**肉**, 今日も**肉**, 明日も**肉**, ずっと**肉**を食べたい
 - 今日は**寿司**と**天ぷら**と**蕎麦**と**ラーメン**しか食えなかった
 - **スイーツ**, **寿司**, **スイーツ**, **寿司**, なら無限に食える
- ◆IDF (肉) = $\log(\text{全文書数} / \text{肉が出現する文書数}) + 1 = \log(4/2) + 1 = 1.3$

2-2. とある日のブログのtf-idf

今日は肉と寿司とピザとラーメンを食べた後に肉を食べた

◆TF (文書A, 肉) = 肉の出現数 / 単語数 = $2/5 = 0.4$

◆IDF(肉) = $\log(\text{全文書数} / \text{肉が出現する文書数}) + 1 = \log(4/2) = 1.3$

◆tf-idf(文書A, 肉) = tf (文書A, 肉) * idf (肉) = $0.4 * 1.3 = 0.52$

とある日のブログのtf-idf

= {肉, 寿司, ピザ, ラーメン} = {**0.52**, 0.26, 0.32, 0.26}

2-3. Bugtriageに応用すると

$$\text{Expertise}_{\text{Dev}}(d, B) = \sum_{i=1}^{\# \text{CommonTerms}} \text{tf-idf}(t_i, d)$$

開発者d 新規不具合B 新規不具合Bに記載されている単語

$$\begin{aligned} \text{tf-idf}(t, d) &= \text{tf}(t, d) \times \text{idf}(t) \\ &= \text{TermFreq}(t, d) \times \log \left(\frac{\# \text{ProjDev}}{\# \text{DevTermFreq}(t)} \right) \end{aligned}$$

全開発者数

単語tが出現する
不具合の担当経験
がある開発者数

開発者dが過去に、
単語tが出現する不具合を何回修正したか

単語tは一般的な単語か

2-3. Bugtriageに応用すると

$$\text{Expertise}_{\text{Dev}}(d, B) = \sum_{i=1}^{\# \text{CommonTerms}} \text{tf-idf}(t_i, d)$$

開発者d 新規不具合B 新規不具合Bに記載されている単語

$$\begin{aligned} \text{tf-idf}(t, d) &= \text{tf}(t, d) \times \text{idf}(t) \\ &= \text{TermFreq}(t, d) \times \log \left(\frac{\# \text{ProjDev}}{\# \text{DevTermFreq}(t)} \right) \end{aligned}$$

全開発者数
単語tが出現する
不具合の担当経験
がある開発者数

- 多くの研究は、時系列を考慮していない
 - 活動していない開発者データを入れるとゴミになる
 - より最近に活発な活動をしている開発者に推薦した方が良い

3. アプローチ

最近のデータに重付けするtime-based tf-idfを提案

$$\text{Expertise}_{\text{Dev}}(d, B) = \sum_{i=1}^{\text{\#CommonTerms}} \text{Time-tf-idf}(t_i, d)$$

$$\text{Time-tf-idf}(t, d, B) = \text{tf-idf}(t, d) \times \text{Recency of use}(t, d, B)$$

$$\text{Recency of use}(t, d, B) = \left(\frac{1}{\text{\#DevTermFreq}(t)} \right) + \left(\frac{1}{\sqrt{\text{BugDate}(B) - \text{DateDevTerm}(d, t)}} \right)$$

単語tがどれだけ一般的か

単語tが出現する不具合を
最後に修正してから何日か

4-1. Research Questions

1. Bugtriageの精度を改善できるか？

- Naive baysやSVM, tf-idfと比較
- 精度＝推薦する開発者が過去に, 新規不具合のコンポーネントを修正したことがあるかを指標

2. 直近のデータが少ない場合でも精度が出るか

- データセットにおける, それぞれ最古と最新の不具合100件をテストデータとして, 精度を比較

4-2. Bugtriageの精度を向上できるか

全プロジェクト・全試行において精度向上

Averages of top 1 to top 5 accuracies for five random test sets on Eclipse, NetBeans and ArgoUML projects.

Project	Test set	Approaches					
		SVM (%)	NB (%)	VSM (%)	SUM (%)	ABA-tf-idf (%)	ABA-Time-tf-idf (%)
Eclipse	1	1.6	24.6	22.2	19.8	21.6	32.4
	2	3.6	27	17.6	17.4	21.8	34.8
	3	4.6	27.4	23.4	17.8	20.4	30.8
	4	0.2	23	20.6	17.6	25.6	37.4
	5	1.8	26.2	26	14	18.6	31.6
NetBeans	1	3.6	21.6	34.2	24.6	43	44.8
	2	0	21	32.6	29	42.6	47.4
	3	0.8	17.6	39.8	29.8	44	48.4
	4	1.2	18.2	41	28	47.4	51.4
	5	4	14.8	39.6	26.4	42.2	45.2
ArgoUML	1	4.4	15.2	22.2	17.8	19.6	29.6
	2	4.2	14.6	17.6	22.2	23	34.6
	3	6.8	12.8	16.8	21.6	18	27.6
	4	7.8	15.2	18.6	14.2	15.4	25.2
	5	7.6	12.6	16.4	19	17.2	26.2

4-3. 直近のデータが少ない場合でも精度が出るか 11

直近のデータが少ないと，精度が低下する場合もある

		Avg. randoms	Early	Late
Eclipse	SUM (%)	6.2	9	5
	VSM (%)	9.2	7	7
	NB (%)	12.2	11	11
	SVM (%)	0.6	1	1
	ABA-tf-idf (%)	9	5	7
	ABA-Time-tf-idf (%)	14.4	10	16
NetBeans	SUM (%)	11.4	7	16
	VSM (%)	19	2	22
	NB (%)	7.6	6	12
	SVM (%)	0.8	1	1
	ABA-tf-idf (%)	32.2	9	36
	ABA-Time-tf-idf (%)	34.8	11	41
ArgoUML	SUM (%)	6	5	9
	VSM (%)	6.4	3	7
	NB (%)	5.2	8	4
	SVM (%)	2.6	1	2
	ABA-tf-idf (%)	4.4	4	7
	ABA-Time-tf-idf (%)	7.2	4	7

5. まとめ

- バグトリージ手法で一般的にtf-idfが使われる
- 多くの研究は、時系列を考慮していない
- 最近のデータに重付けするtime-based tf-idfを提案
 - tf-idfに比べて、推薦の精度が向上
 - 直近のデータが少ないと精度の低下する場合もある

6. 所感

- ☀ やろうとしていることは正しい（当たり前. . .）
- ☀ 多くの論文を引いていて参考になる
- ☁ アルゴリズムが複雑で読みにくい
- ☁ Recency of useなど、数式の意味が書かれていない（なぜ今回の結果になるか説明ができていない）
- ☁ 結局、頑張っている人に割当てるため、タスクが集中