LASCAD: LANGUAGE-AGNOSTIC SOFTWARE CATEGORIZATION AND SIMILAR APPLICATION DETECTION

DOAA ALTARAWY, HOSSAMELDIN SHAHIN, AYAT MOHAMMED, NA MENG

Arinjoy Basak, CS 6704 presentation, March 20th 2019

AN OUTLINE OF THIS PRESENTATION

- Problem Statement
- Related Work
- Approach
- Evaluation
- Conclusion
- Discussion

PROBLEM STATEMENT

INTRODUCTION AND MOTIVATION

INTRODUCTION: PROBLEM STATEMENT AND MOTIVATION

- Effective categorization and detection of similar software has become important
  - Cross platform software migration
  - Reimplementing software with a different programming language
  - Drawbacks in current step: very few manually labeled projects, impossible to do so by hand
- Solution: **L**anguage-**A**gnostic **S**oftware **C**ategorization and similar **A**pplication **D**etection (**LASCAD**)
  - Primarily draws upon information retrieval literature, with cross language support

INTRODUCTION: GOALS

Aim

- Transform Source Code Engines like GitHub
  - Categorize regardless of documentation
- Boosts research in automatic program repair, security vulnerability detection
  - Comparing similar software

Facilities to explore applications/projects directly based on their codebase information

RELATED WORK

SOFTWARE DETECTION, CATEGORIZATION, CODE SEARCH, LDA TUNING

PRIOR WORK

- Similar software detection
  - Google play “Similar” feature – manual labeling, not perfect at all
  - CodeWeb (Michail and Notkin, 1999) - only names, no implementation details
  - SSI (Bajracharya et al., 2010), CLAN (McMillan et al., 2012a) – API usage; similar invocation of library APIs; CLANDroid (Linares-Vasques et al., 2016) locates similar applications in Android
  - RepoPal (Zhang et al., 2017) – similar Github repos based on readme; but very, very restrictive
  - Similar Tech (Chen et al. 2016) - finding analogical application across languages
  - LASCAD – similar software, different languages, any repo, only source code

PRIOR WORK

- Automatic Software Categorization
  - JDK API invocation to train ML model (McMillan et al., 2011)
  - (State of the Art) MUDABlue (Kawaguchi et al, 2006), LACT (Tian et al, 2009) – topic modeling; textual relevance of terms in source code
    - Rationale: identifiers in code and words in comments used meaningfully indicate similar program semantics
    - Both produce uncontrollable number of categories, not a desired number of classes
  - LASCAD – novel in using LDA and hierarchical clustering, not requiring parameter tuning (discussion!), bounded number of classes (discussion!) more disciplined (later)
  - Note: Neither was directly available for comparison during evaluation!

PRIOR WORK

- Code Search Tools
  - SourceGraph
  - Google Code Search,
  - Sourcerer
    - But nobody retrieves similar applications
- LDA parameter tuning (see theory later)
  - Earliest work by Blei et al. (Blei et al., 2003)
  - Parameter tuning is challenging – even in source code analysis (Binkley et al., 2014)

BACKGROUND: LDA (SHORT INTRODUCTION)

- Generative statistical modeling – samples are generated from underlying distributions which are defined by parameters.
- Widely used, has a lot of advantages
- Idea: A collection of documents has a collection of topics (sometimes more than one, Blei and Lafferty, 2009), and words drawn from these topics. The list of topics are universally chosen for the collection of topics.
- Input: A collection of documents
- Output: Document-topic and Topic-word matrix.

BACKGROUND: LDA (SHORT INTRODUCTION)

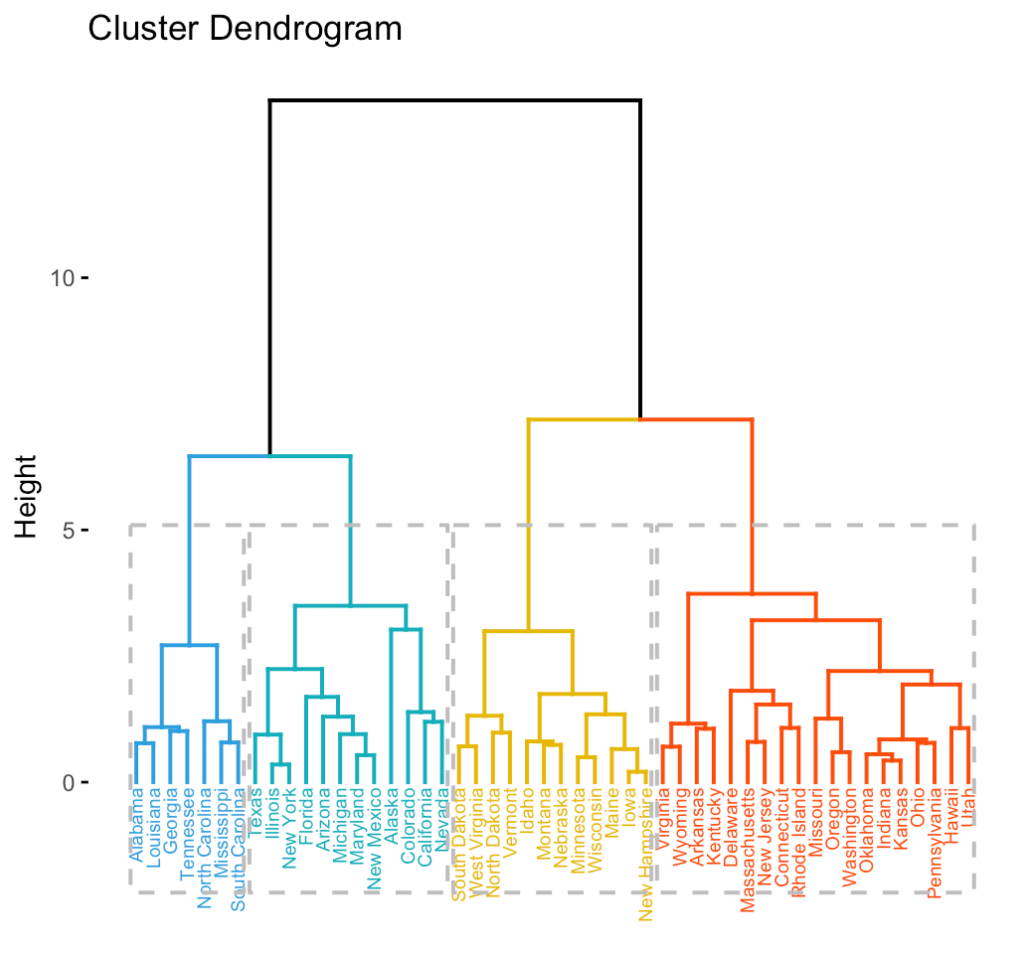
- Example of a generative model for tweets and mentioning users, used from Gong et al., 2015
- As in real life: topics are chosen in background distribution;
- A user creates document by choosing a topic, and selecting words from that topic; and then chooses users to mention based on topic

Algorithm 1 The generation process of A-TTM model

for each topic $z \in T$ **do**
  Draw $\psi^z \sim Dir(\beta)$
  for each word $w \in W$ **do**
    Draw $\phi^{z,w} \sim Dir(\gamma)$
  end for
end for
for each user $u \in U$: **do**
  for each microblog $d \in D_u$: **do**
    Draw $\theta_d \sim Dir(.|\alpha)$
    for each word in microblog d,$w_m \in w_d$: **do**
      Draw a topic $z_m \sim Mult(.|\theta_d)$
      Draw a word from topic-word distribution $w_m \sim Mult(.|\psi^z)$
    end for
    for each user mentioned in microblog d, $a_n \in a_d$: **do**
      Draw a topic $z_n \sim Mult(.|\theta_d)$
      Draw a user $a_n \sim p(.|\mathbf{z}, w_d, \phi^{z,w})$
    end for
  end for
end for

BACKGROUND: HIERARCHICAL CLUSTERING

- Given N objects, group them based on similarity : top-down or bottom-up
- N clusters for N objects (in agglomerative) – find all pairs of distances
- Each round, combine two closest clusters based on linkage criteria to form single cluster
- Repeat until desired, choose cutoff for distance and number of clusters.

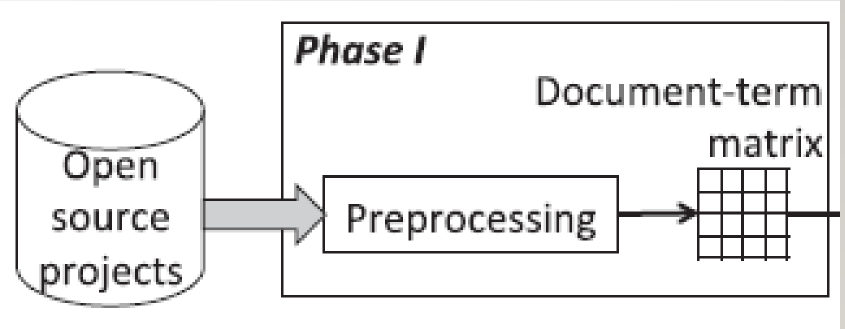


APPROACH

GENERATIVE MODELS, LDA, HIERARCHICAL CLUSTERING, PROCESS FLOW

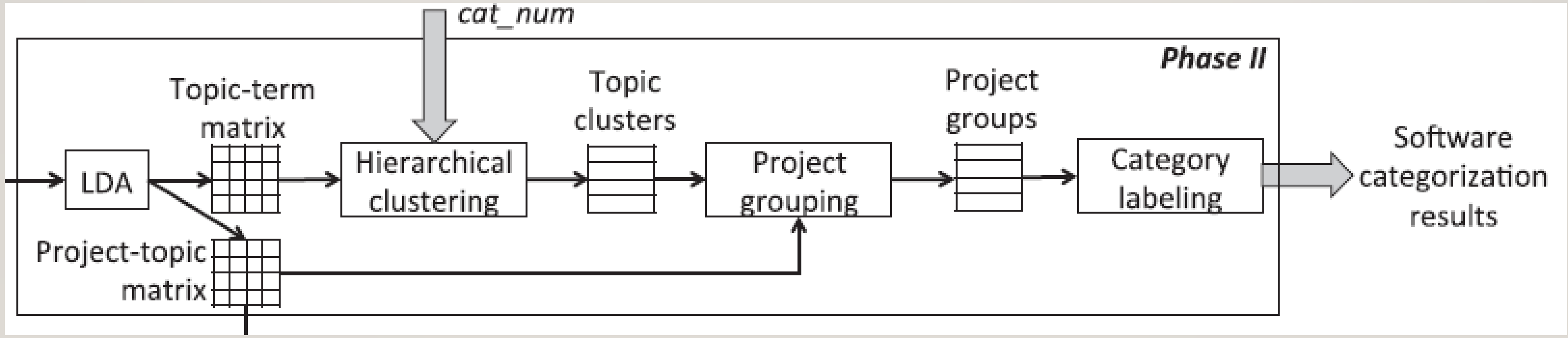
PHASE 1: SOURCE CODE PREPROCESSING

- Creating the document-term matrix based on frequency (number of appearances in a document)
- Step 1: Find words and identifiers
- Step 2: Refining based on language features – remove stop words, split identifiers: method_name or methodName -> (method, name)
- Step 3: Remove words below 0.2 and above 0.8 document frequency $df = \frac{k}{m}$



PHASE II: SOFTWARE CATEGORIZATION

- Define parameter t_num (number of topics in LDA, fixed), cat_num (number of categories, fixed)
- Steps: perform LDA on document-term matrix to get topic-term and project-topic matrix, create hierarchical clusters at cat_num, group the projects to get project-cluster assignment, get category label by examining



PHASE II: SOFTWARE CATEGORIZATION
STEP 1: LDA

- LDA (**Latent Dirichlet Allocation)** is performed on the document-term matrix to get the parameters for the underlying distribution, estimate the following:
  - Parameter: number of topics t_num (made transparent)
  - Probability of a topic having the certain words/terms in corpus (TT)
    - Consider a vector $L = \{l_1,l_2,\ldots l_m\}$, with m extracted terms, and $\sum l_i=1$
  - Probability of a project having certain topics (PT)
    - Consider a vector $S = \{s_1,s_2,\ldots s_m\}$, with m extracted terms, and $\sum s_i=1$
  - Excellent reference: the paper by Blei et al., 2003

PHASE II: SOFTWARE CATEGORIZATION STEP 1: LDA (APPROXIMATE DETAILS)

for each topic $z \in T$: do
    Draw $\varphi^z \sim Dir(\beta)$
end for
for each project $d \in D$: do
    Draw $\Theta \sim Dir(\cdot|\alpha)$
    for each word in d, $w_m \in w_d$: do
        Draw a topic $z_m \sim Mult(\cdot \mid \Theta_d)$
        Draw a word from topic-word distribution
        $w_m \sim Mult(\cdot \mid \varphi^z)$

PHASE II: SOFTWARE CATEGORIZATION STEP 2: CLUSTERING

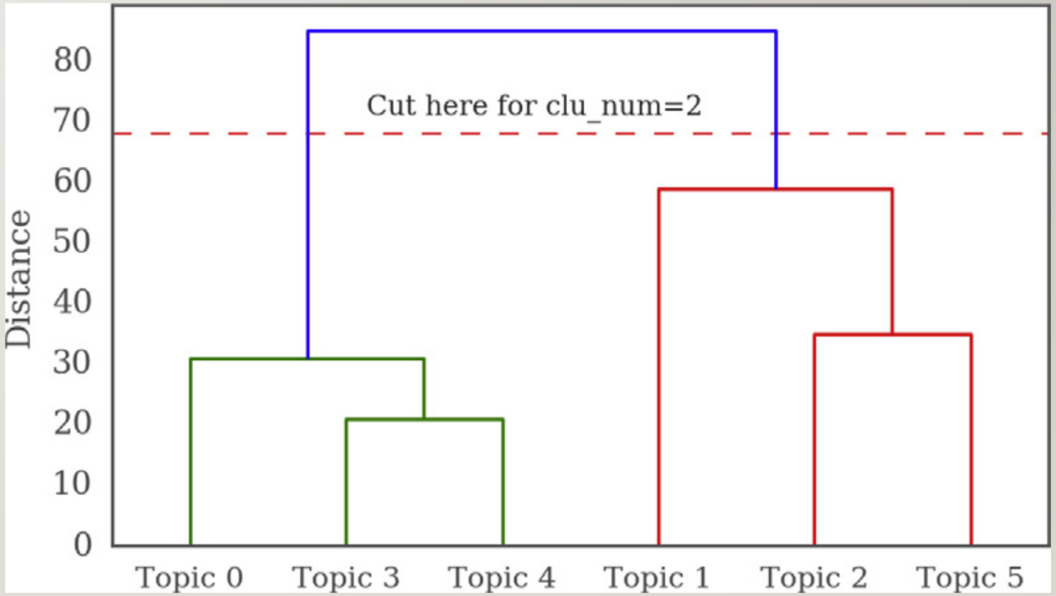
- Hierarchical clustering performed to get upto cat_num clusters
- Metric: Cosine similarity

$$Cos\_Sim_{ij} = \frac{L_i \cdot Lj}{||L_i||\ ||L_j||} = \frac{\sum_{k=1}^{m} l_{ik} l_{jk}}{\sqrt{\sum_{k=1}^{m} l_{ik}^2}\sqrt{\sum_{k=1}^{m} l_{jk}^2}}$$

- Linkage: centroid based:

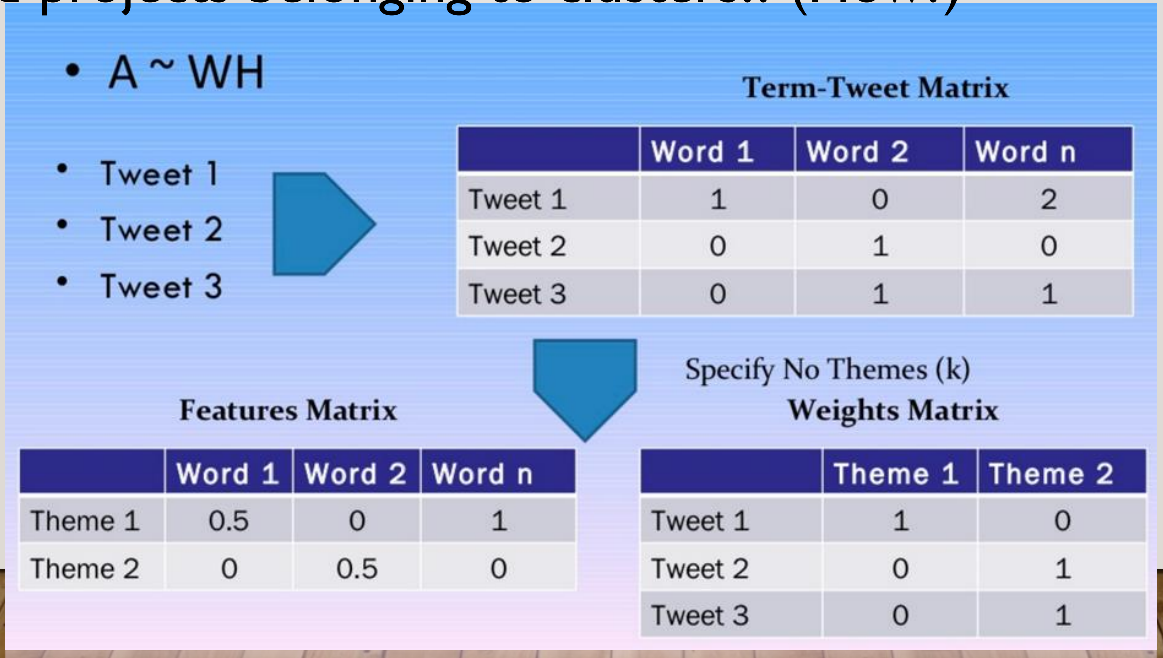
$$L_{cen} = [\frac{l_{i1} + l_{j1}}{2}, \frac{l_{i2} + l_{j2}}{2}, \ldots, \frac{l_{im} + l_{jm}}{2}]$$

- Merging clusters bottom, connecting two closest clusters at each step.



PHASE II: SOFTWARE CATEGORIZATION
STEP 3: PROJECT-TOPIC MAPPING

- Mapping of terms in topics (latent), clusters on these topics => terms in clusters
- Probability of belonging to the topics
- Next step: find projects belonging to clusters!! (How?)



	Word 1	Word 2	Word n
Tweet 1	1	0	2
Tweet 2	0	1	0
Tweet 3	0	1	1

	Word 1	Word 2	Word n
Theme 1	0.5	0	1
Theme 2	0	0.5	0

	Theme 1	Theme 2
Tweet 1	1	0
Tweet 2	0	1
Tweet 3	0	1

PHASE II: SOFTWARE CATEGORIZATION
STEP 3: PROJECT-TOPIC MAPPING

- Given: Clusters Cls = $\{cls_1, cls_2, \ldots cls_{cat\_num}\}$, Each project is S = $\{s1, s2, \ldots sm\}$
- Compute project cluster relevance matrix M_{ij}

$$M_{ij} = \sum_{k=1}^{t\_num} s_{ik} b_{kj}, \text{ where}$$

$$b_{kj} = \begin{cases} 0, & \text{if } k^{th} \text{ topic does not belong to } cls_j, \text{ or} \\ 1, & \text{if } k^{th} \text{ topic belongs to } cls_j \end{cases}$$

- The values are normalized per document
  - We get the probability of a document to belong to a cluster!
- Explanation:

$$M_{ij} = \sum_{k=1}^{t\_num} s_{ik} b_{kj}, \text{ where}$$

$$b_{kj} = \begin{cases} 0, & \text{if } k^{th} \text{ topic does not belong to } cls_j, \text{ or} \\ 1, & \text{if } k^{th} \text{ topic belongs to } cls_j \end{cases}$$

$$\forall i, \quad S_i = [s_{i1}, s_{i2}, \ldots s_{im}] \longrightarrow \sum_{k=1}^{m} s_{ik} = 1$$

$$\Rightarrow \sum_{k=1}^{m} P(k|i) = 1$$

$$M_{ij} = \sum_{k=1}^{t\_num} P(k|i)\, I(k \in cls_j) \propto P(j,i)$$

(together)

sort of expected no. of topics in doc i and cluster j together.

<u>Normalising</u>: $\tilde{M}_{ij} = P(j|i) = \dfrac{P(i,j)}{P(i)}$

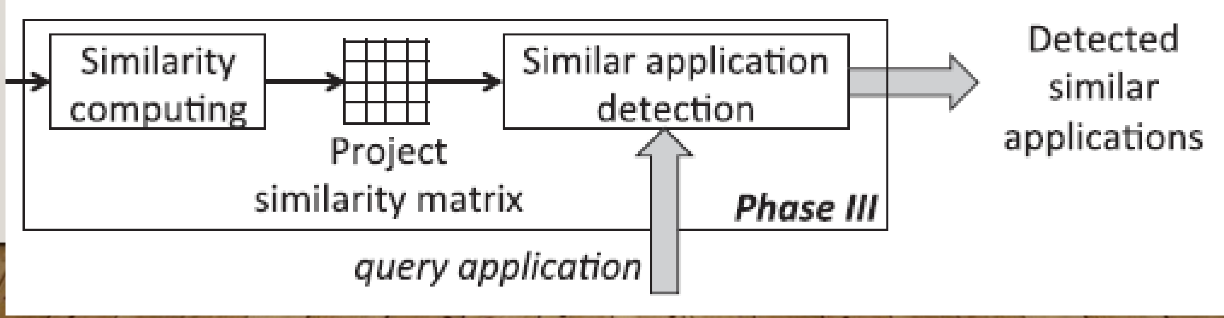
$$P(i) \longrightarrow \propto \sum_{\forall j} M_{ij}$$

PHASE II: SOFTWARE CATEGORIZATION STEP 4: ASSIGN MANUAL CATEGORIES

- One of the most time consuming steps
  - For labeled set of software, use application labels directly
  - Otherwise, read the projects in a group and assign a label
- Alternatives suggested (automation):
  - Label based on most relevant to topic clusters, and pick terms from these to label group
  - Use most frequent terms of each topic of cluster cls_j to name software group

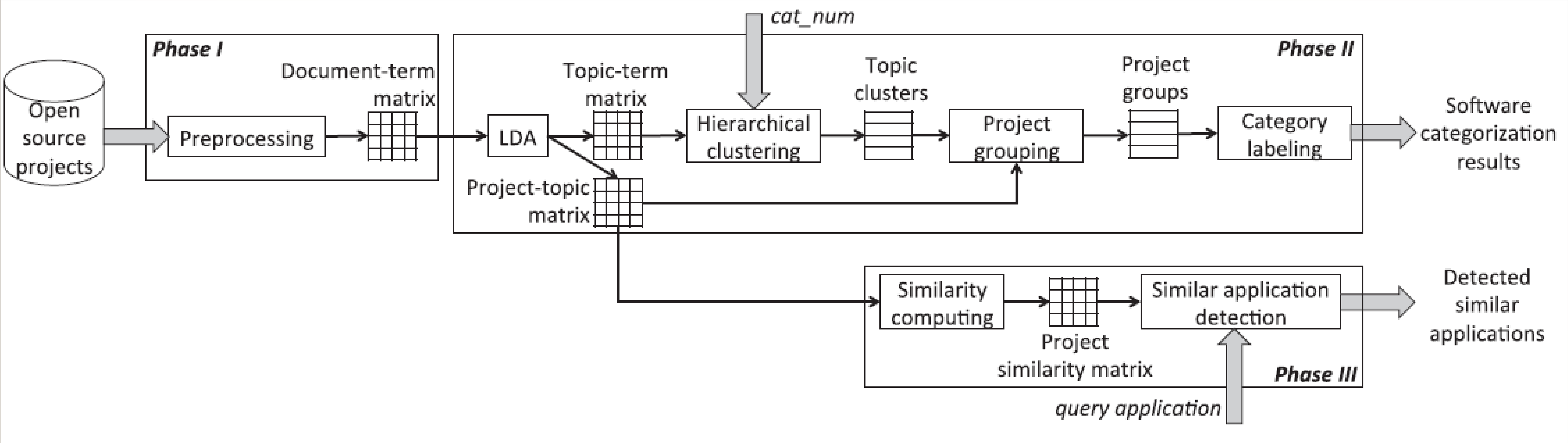
PHASE III: DETECTING SIMILAR APPLICATIONS

- Users select application from pool
- Existing: vectors of project-cluster similarity, a probability distribution
- Calculate Jensen-Shannon Divergence – similarity of distributions
  - Based on very popular KL Divergence
  - This one is symmetric, so better suited
- Select projects with highest scores



TOTAL SYSTEM

- Implementation in Python using NLTK, Scikit-Learn for LDA and hierarchical clustering, Pandas and Scipy for data preprocessing



EVALUATION

CRITERION, MEASURES, EXPERIMENTS, CASE STUDIES

DATASETS

Name	Reference	Size	Language	Labeled	Multi-category
MUDABlue	Kawaguchi et al., 2006; SourceForge	41	C	Yes	Yes (13 categories)
LACT	Tian et al., 2009	43	6 languages	Yes	No (probably) (6 categories)
New Labeled	This paper	103	19 languages	Yes	No
New Unlabeled	This paper	5220	17 languages	Yes	Unknown

CRITERIA OF MEASUREMENT OF SUCCESS

$$precision = \frac{\sum_{s \in S} precision_{soft}(s)}{|S|}$$

$$precision_{soft}(s) = \frac{|C_A(s) \cap C_{Ideal}(s)|}{|C_A(s)|}$$

$$recall = \frac{\sum_{s \in S} recall_{soft}(s)}{|S|}$$

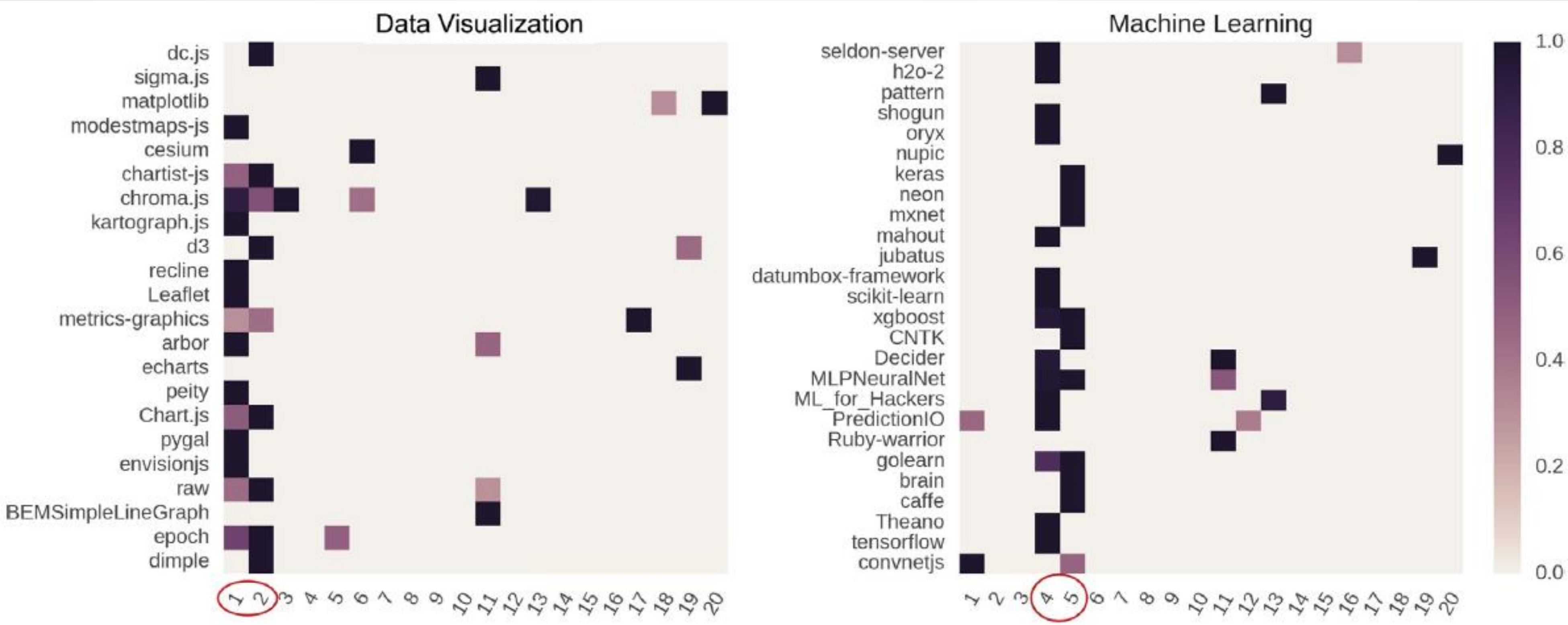
$$recall_{soft}(s) = \frac{|C_A(s) \cap C_{Ideal}(s)|}{|C_{Ideal}(s)|}$$

$$\text{F-score} = \frac{2 * precision * recall}{precision + recall}$$

$$\text{relDiff} = \frac{|\#\text{of identified categories} - \#\text{of ideal categories}|}{\#\text{of ideal categories}}$$

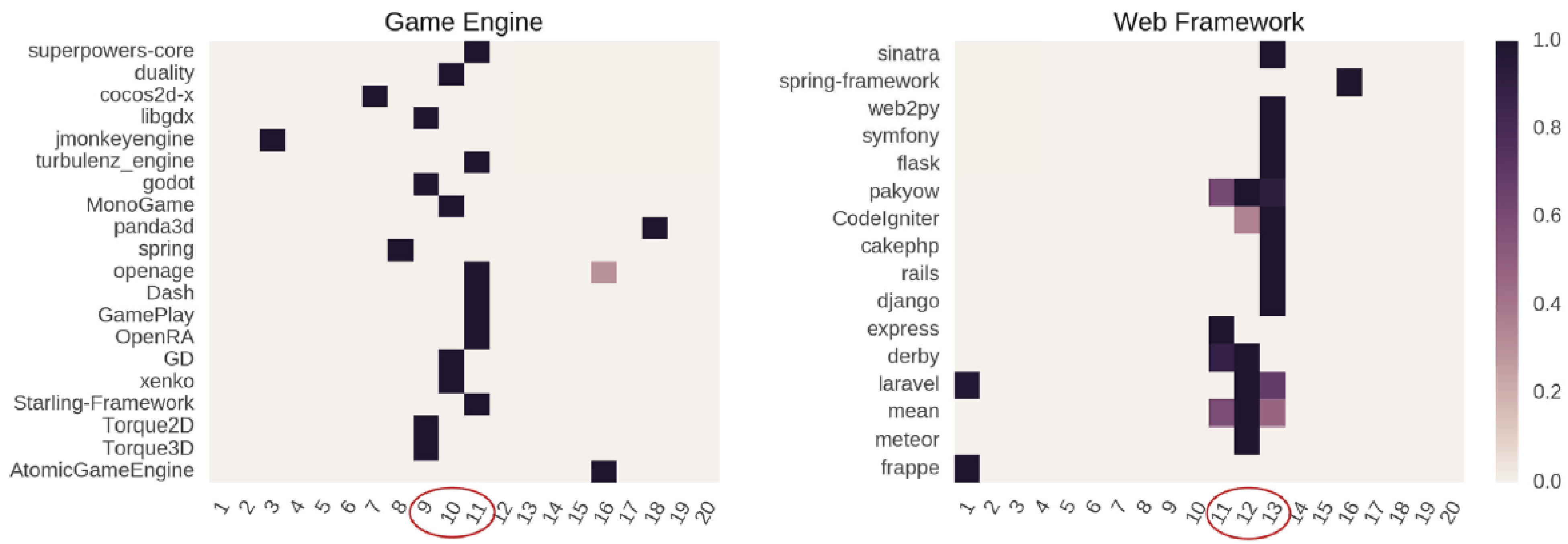
Source: Kawaguchi et al., 2006; drawn mainly from Information retrieval domain

SOFTWARE CATEGORIZATION EFFECTIVENESS



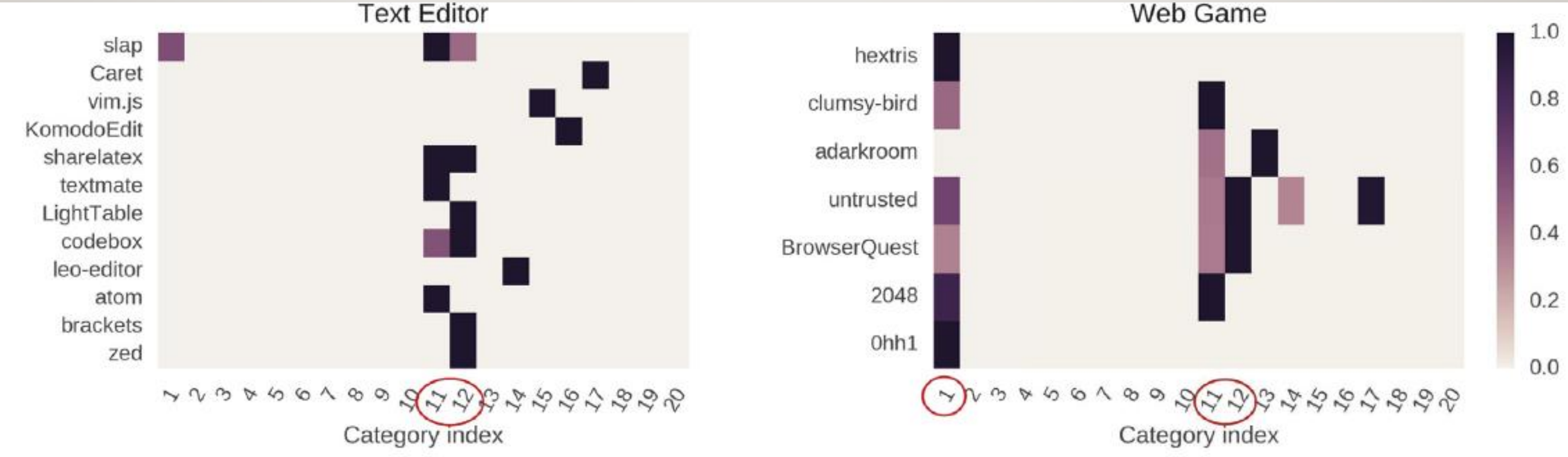
- Final values: 67% precision, 85% recall, 75% F-score, and 2.33 relDiff (except some cases)

SOFTWARE CATEGORIZATION EFFECTIVENESS



- Final values: 67% precision, 85% recall, 75% F-score, and 2.33 relDiff (except some cases)

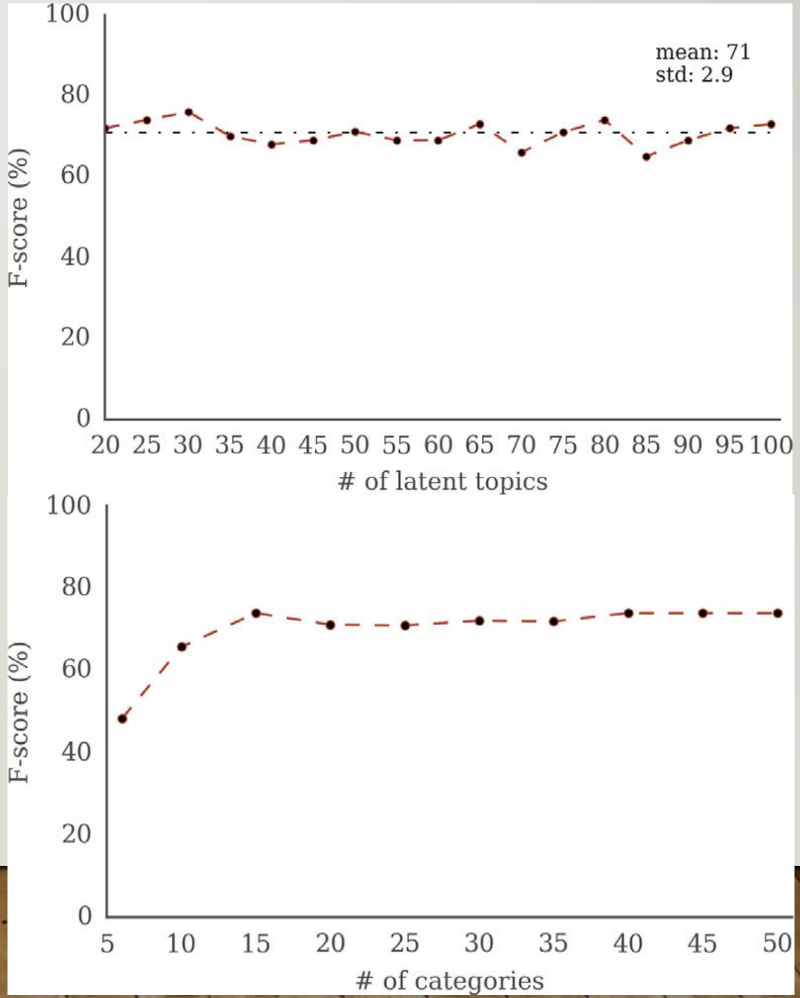
SOFTWARE CATEGORIZATION EFFECTIVENESS



- Final values: 67% precision, 85% recall, 75% F-score, and 2.33 relDiff (except some cases)
- For each ideal category, finds 2-3 clustered categories

SENSITIVITY TO PARAMETER SETTINGS

- Parameters: t_num (deep) and cat_num (higher level)
- Shown that it is not sensitive to parameter choice, choose stable parameters
  - cat_num=20
  - t_num=50
- Opinion: not the correct way to do it; look at clustering distances



COMPARISON WITH PRIOR TOOLS

- MUDABlue and LACT were not available for direct comparison
  - MUDABlue could not be implemented; only results compared
  - LACT was implemented (but details of verification not specified)

Tool comparison based on MUDABlue's 41 C programs of 13 ideal categories.

Tool	# of categories	Precision	Recall	F-score	RelDiff
MudaBlue	40	-	-	72%	5.67
LACT	23	76%	65%	70%	2.83

Previous tool LACT's categorization results on the 103-application data set with *t*_ ... iabil- ity in t...

Comparing LASCAD and LACT's categorization results on the 103-application data set with approximately similar *cat_num*.

70	38	70%	40	76%	
80	47	71%	45	74%	
90	50	73%	50	76%	
100	Avg.	**68.67%**		**74.67%**	

Lascad ***categorized software stably better than prior approaches on different data sets. It allows users to flexibly control the number of generated categories, without producing over- whelming numbers of categories as previous tools do.***

Tool comparison based on our 103 applications of 6 ideal categories.

Tool	# of categories	Precision	Recall	F-score	RelDiff
LACT	38	57%	91%	70%	5.33
LASCAD	**20**	**67%**	**85%**	**75%**	**2.33**

SIMILAR APPLICATION DETECTION

- Metric relevance is defined.
- Used the unlabeled projects (5220), random 38 projects as queries
  - Top 1 relevance – 71%
  - Top 5 relevance – 64%
- Used labeled set (103) (completely identical for relevance)
  - Top 1 relevance – 70%
  - Top 5 relevance – 64%

$$r_i = \frac{\sum_{j=1}^{m} b_j}{m}, \text{ where } b_j = \begin{cases} 1, & \text{if } a_j \text{ is similar, or} \\ 0, & \text{if } a_j \text{ is not similar} \end{cases}$$

Correspondingly, the overall relevance for the n queries is

$$relevance = \frac{\sum_{i=1}^{n} r_i}{n}$$

Interesting:

- Random query search – 11%
- Title/Description search – 8.3%
- Readme File search – LDA and similarity (RepoPal) – 23% (Top 1) 19% (Top 5)

CONCLUSIONS

- Contributions
  - Usable, reliable, language agnostic software categorization and similar application detection
  - First to design based on LDA and clustering, removing parameter tuning
  - Direct control over number of desired categories
  - Case studies on failures
- Major Findings
  - 67% precision, 85% recall, 75% f-score, 2.33 relDiff, multiple categories for real-world categories
  - Not sensitive to t_num variations, only for cat_num <= 15
  - Categorized better than prior approaches, allows flexible control, no over-categorization

CONCLUSIONS

Case Study observations and results

- Difference from oracle
  - Incompleteness of labels
  - Incorrect labels
  - Red Herrings – latent features shared, but different functionalities
- Incorrect retrieval
  - Small codebase
  - Similar fn.alities, different implementatn.

- Threats to validity
  - Unlabeled dataset, no ground truth – open to human error – user study?
  - Small query size
  - LACT reimplementation
  - Parameter tuning removed – but…
  - Non-sensitive to cat_num, but…useful?
  - Underestimate performance due to multicategory membership – different metric?

DISCUSSION POINTS

- Poorly maintained projects may lack comments and have confusing identifiers
  - Topic free word alignment?
- LDA parameter tuning is avoided by hard-coding it – but it is not recommended?
  - Fixing? Different way to do this?
  - Also, evaluation method – look at cluster distance at cutoff, not just F score
- Only chooses from pool of existing projects to check similarity
  - New project arrival? (potentially, recalculate)
  - Large scale implementation efficiency (offline and online similarity scoring for rank)
  - Only uses code, fails on name – address this?

DISCUSSION POINTS

- Why even topic modeling and clustering, and not smaller number of topics overall?
  - Allowing overlaps probably – but alternatives?
  - LDA purely doesn't work as well
- Comments: Evaluation criteria, main theory well founded
  - Could use more details of formulation for LDA estimation.
  - Tool comparison could be better? (Ask)
- Future direction: directly look at unknown source code and find suggestions for porting/similar libraries/plugin for conversion of projects using templates – How?

REFERENCES

- Michail, A., Notkin, D., 1999. Assessing software libraries by browsing similar classes, functions and relationships. In: Proceedings of the 21st International Conference on Software Engineering. ACM, New York, NY, USA, pp. 463–472. doi: 10.1145/302405.302678.
- Bajracharya, S.K. , Ossher, J. , Lopes, C.V. , 2010. Leveraging usage similarity for effective retrieval of examples in code repositories. In: Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering.
- McMillan, C. , Grechanik, M. , Poshyvanyk, D. , 2012. Detecting similar software applications. In: Software Engineering (ICSE), 2012 34th International Conference on. IEEE, pp. 364–374.
- Linares-Vásquez, M. , Holtzhauer, A. , Poshyvanyk, D. , 2016. On automatically detecting similar Android apps. In: 2016 IEEE 24th International Conference on Program Comprehension (ICPC). IEEE, pp. 1–10.
- Zhang, Y., Lo, D., Kochhar, P.S., Xia, X., Li, Q., Sun, J., 2017. Detecting similar repositories on GitHub. In: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 13–23. doi: 10.1109/SANER.2017. 7884605.
- Chen, C., Xing, Z., 2016. SimilarTech: Automatically recommend analogical libraries across different programming languages. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. ACM, New York, NY, USA, pp. 834–839. doi: 10.1145/2970276.2970290.
- Kawaguchi, S. , Garg, P.K. , Matsushita, M. , Inoue, K. , 2006. MUDABlue: An automatic categorization system for open source repositories. J. Syst. Softw. 79 (7), 939–953.
- Tian, K. , Revelle, M. , Poshyvanyk, D. , 2009. Using Latent Dirichlet Allocation for automatic categorization of software. In: Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on. IEEE, pp. 163–166.
- Blei, D.M. , Ng, A.Y. , Jordan, M.I. , 2003. Latent dirichlet allocation. J. Mach. Learn. Res. 3, 993–1022 .
- Binkley, D. , Heinz, D. , Lawrie, D. , Overfelt, J. , 2014. Understanding lda in source code analysis. In: Proceedings of the 22Nd International Conference on Program Comprehension. ACM, pp. 26–36 .
- Blei, D. , Lafferty, J. , 2009. Text mining: Classification, clustering, and applications. chapter Topic Models, Chapman & Hall/CRC.
- Y. Gong, Q. Zhang, X. Sun, and X. Huang. Who will you@? In Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, pages 533–542. ACM, 2015.

533–542. ACM, 2015.