Comparison of Baselines - Language Model for Change Message Classification

Vivek
Computer Science, North Carolina State University
Raleigh, NC
vvivek@ncsu.edu

1 BASELINES

The first classification system of software changes was put forward by Swanson in 1976 [6]. Swanson defined that a change can be classified in one of the following 3 categories:

- Corrective: Changes that are fix for prior failures.
- Adaptive: Changes in environment.
- Perfective: Changes to improve efficiency.

Based on the above classification we discuss 5 models proposed in literature. We argue for one of them to be contestant for baseline for our project.

1.1 Fixed Dictionary

Mockus et al. [5] studied and created classification based on word frequency analysis and keyword clustering. Mockus et al. derives inspiration from Swanson's classifications [6]. They have created a generic keyword based dictionary that corresponds to Swanson's classes [6]. It has been envisioned that there are 3 primary types of maintenance: fault fixes for keywords such as *fix, problem, incorrect, correct*, new code development for keywords *add, new, modify, update* and code improvement for keywords *cleanup, unneeded, remove, rework*. These keywords help is clustering of similar words based on some heuristics, followed by a simple classification algorithm. The keywords used in our implementation of the baseline is shown in Table-1. A developer survey has been used as validation method. Kappa coefficient is 0.5 indication moderate agreement. A log linear model was employed ti further this validation that indicated strong agreement between automatic classification and developer's classification. A claim is made that this method can be used for different projects as only basic information from version control system is used. This is a very simple algorithm and very comprehensible. It can be easily reproduced. So, it is a good candidate as baseline for our project.

1.2 Generalized Evolving Dictionary

Mauczka et al. [4] proposed a variation of keyword based approach to automated classification change messages. This work has been based on Mockus et al. [5] natural language "modification request" words. Weights for certain keywords were also defined. Since, the weight was subject to researchers tuning of their algorithms, it may have suffered a bias. A first set of 3 keywords is for each category is selected based on prior work [2] [5]. This keywords dictionary is extended by new words basis a frequency analysis algorithm until 80% of the project is classified. This dictionary was localized to a project starting from an initial globally accepted seed. A modified version of the original dictionary is shown in Table-2. However, they go on to do validation over cross-projects to establish a general keyword dictionary. Mauczka et al. [4] made an extension of Swanson's original classification by adding a *Blacklist* category. This dictionary can be seen as an extension of exemplary work of Mockus et al. [5]. Moreover, it has been validated across different projects. Hence, this model can also be a viable candidate for baseline.

1.3 Latent Dirichlet Allocation with Manual Annotation of Topics

Hindle et al. [3] discusses Latent Dirichlet Allocation (LDA). The proposal says discovering change message classification topics over a window of time. LDA is a mixture model based method used to analyze unstructured data. Hindle et al. used certain parameters to conclude topic meaning. Top 10 words were seen among all 20 topics. Then they were manually assigned a topic name. If some topic did not give proper meaning it was discarded. The main objective is to create a trend in topics for a software project. For this purpose, similarity between two topics needs to be computed. A simple algorithm of 8 matches among top 10 words implied similar topic. There are many "magic" parameters used in this study. Nevertheless, as the generation of topic from commit message is an automated process. The final classification of a topic is based on human intervention. Hence, it is kind of semi-supervised classification based on LDA with software domain knowledge injection. Such a model is viable candidate for our baseline.

1.4 Latent Dirichlet Allocation with Automated Classification

Inspired by Hindle et al. [3] and success of LDA in topic modeling and feature transformation, we also look at simple Random Forest classifier with features as transformed by LDA. This model can be established as classical LDA and classical classifier to be a baseline for using LDA as a language model. Hence, it is also a candidate for baseline for our project.

1.5 Semi-supervised Latent Dirichlet Allocation

Ying et al. [1] proposes an automated discovery of change message features using LDA. The algorithm includes a semi-supervised LDA method for change message classification. It is called semi-supervised as topic generation and data transformation through LDA is largely depended on *signifier documents*. A set of documents is built as signifier documents corresponding to Mauczka et al. [4] extended dictionary that is validated across project. The exact dictionary used is shown in Table-2. In a way, Ying et al. method is

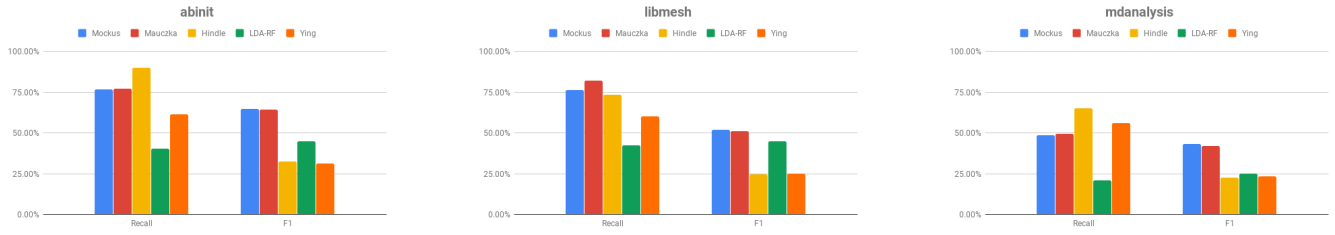

Figure 1: Evaluation of the baseline models on different data-sets

initialized using Mockus et al. [5] "keywords" and is based on Swanson's original change message classification [6]. Key difference of this algorithm is two folds:

- Injection of software engineering domain knowledge to generate topics and words distribution.
- Classification algorithm is similarity between topic distribution of signifier documents and target documents.

Validation is done through developer survey and accuracy is reported. Nearly, 85% of the messages were classified with nearly 70% accuracy. The algorithm is claimed to work on cross project analysis of software change message classification because of developer agreement. This algorithm is a viable candidate as our baseline. It's dependence on both software domain knowledge and automated change message classification with features specific to a project makes it a very strong candidate.

Table 1: Mockus et al. [5] dictionary

Category	Associated words
Corrective	fix, bug, problem, incorrect, correct, error, fixup, fail
Adaptive	new, change, patch, add, modify, update
Perfective	style, move, removal, cleanup, unneeded, rework

Table 2: Mauczka et al. [4] dictionary

Category	Associated words
Corrective	wrong, except, cause, null, warn, correct, valid, fail, bug, dump, opps, error, invalid, failure, incorrect, bug-fix, fix, problem, bad, miss
Adaptive	context, introduce, future, appropriate, information, add, compatibility, simple, structure, configuration, available, require, init, function, internal, security, current, config, provide, easier, switch, default, faster, text, release, install, user, feature, old, useful, method, change, trunk, protocol, patch, version, replace, necessary, new, create
Perfective	clean, include, whitespace, dead, consistent, prototype, static, style, remove, definition, header, documentation, variable, inefficient, useless, cleanup, move, unused, declaration
Blacklist	cvs2svn, cvs, svn

2 COMPARING BASELINES

For the purpose of establishing a baseline, a version of each of the above is implemented. We sample the data with replacement 20 times and report the results as shown in Figure-1.

The model for Mockus et al. [5] is implemented as a look-up dictionary as in Table-1. If any keyword is present first in the change message tokens, it is classified according to the category. Similar implementation is for Mauczka et al. [4]. The dictionary look-up is mentioned in Table-2. This is not the actual algorithm implemented in the paper. However, this is based on the final dictionary proposed for cross project change message analysis. The algorithm proposed by Ying et al. [1] is implemented exactly as mentioned. The signifier documents used are as in Table-2. Taking inspiration for using LDA as language model as in Hindle et al. [3]. A similar model is implemented. Instead of manual annotation of a topic classification, we generated only 2 topics and use *Corrective* keywords from Table-1 to annotate one of the topics as "corrective" topic. After a topic is annotated as buggy based on analysis of top 10 words with the dictionary, we classify the change message according to the probability of that topic. We also implement a model using LDA as language model and Random Forest as classifier on transformed features which are topic distributions. The difference between this LDA implementation and Ying et al. [1], is that Ying et al. model is semi-supervised and is classified basis similarity to topic distributions in signifier documents using cosine similarity.

We compare all the three projects based on the implementation of the baseline models as described above. Although, Mockus et al. [5] and Mauczka et al. [4] are not automated models but are based on dictionary lookup, they tend to perform better than the other models for the given dataset. One other thing to note is that all the models based on "corrective keywords" is better. That is to say, number precision is low for all these models as they have been specifically designed for bug-fix indicating words. If we say that we require an automated language model rather a dictionary based model, Hindle et al. [3] model seems to be the most acceptable baseline.

REFERENCES

[1] Ying Fu, Meng Yan, Xiaohong Zhang, Ling Xu, Dan Yang, and Jeffrey D. Kymer. 2015. Automated classification of software change messages by semi-supervised Latent Dirichlet Allocation. *Information and Software Technology* 57 (2015), 369–377.

[2] Ahmed E. Hassan. 2008. Automated Classification of Change Messages in Open Source Projects. In *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC '08)*. ACM, New York, NY, USA, 837–841. https://doi.org/10.1145/1363686.1363876

[3] A. Hindle, M. W. Godfrey, and R. C. Holt. 2009. In *What's hot and what's not: Windowed developer topic analysis*. IEEE, 339–348.

[4] Andreas Mauczka, Markus Huber, Christian Schanes, Wolfgang Schramm, Mario Bernhart, and Thomas Grechenig. 2012. Tracing Your Maintenance Work — a Cross-project Validation of an Automated Classification Dictionary for Commit Messages. In *Proceedings of the 15th International Conference on Fundamental Approaches to Software Engineering (FASE'12)*. Springer-Verlag, Berlin, Heidelberg, 301–315. https://doi.org/10.1007/978-3-642-28872-2_21

[5] Audris Mockus and Lawrence G. Votta. 2000. Identifying Reasons for Software Changes Using Historic Databases. In *Proceedings of the International Conference on Software Maintenance (ICSM'00) (ICSM '00)*. IEEE Computer Society, Washington, DC, USA, 120–. http://dl.acm.org/citation.cfm?id=850948.853410

[6] E. Burton Swanson. 1976. The Dimensions of Maintenance. In *Proceedings of the 2Nd International Conference on Software Engineering (ICSE '76)*. IEEE Computer Society Press, Los Alamitos, CA, USA, 492–497. http://dl.acm.org/citation.cfm?id=800253.807723