Train One Get One Free: Partially Supervised Neural Network for Bug Report Duplicate Detection and Clustering

Lahari Poddar[1,2] **Leonardo Neves**[2] **William Brendel**[2]
Luis Marujo[2] **Sergey Tulyakov**[2] **Pradeep Karuturi**[2]

[1]School of Computing, National University of Singapore
lahari@comp.nus.edu.sg
[2]Snap Research
{lneves, willb, luis.marujo, stulyakov, pradeep.karuturi}@snap.com

Abstract

Tracking user reported bugs requires considerable engineering effort in going through many repetitive reports and assigning them to the correct teams. This paper proposes a neural architecture that can jointly (1) detect if two bug reports are duplicates, and (2) aggregate them into latent topics. Leveraging the assumption that learning the topic of a bug is a sub-task for detecting duplicates, we design a loss function that can jointly perform both tasks but needs supervision for only duplicate classification, achieving topic clustering in an unsupervised fashion. We use a two-step attention module that uses self-attention for topic clustering and conditional attention for duplicate detection. We study the characteristics of two types of real world datasets that have been marked for duplicate bugs by engineers and by non-technical annotators. The results demonstrate that our model not only can outperform state-of-the-art methods for duplicate classification on both cases, but can also learn meaningful latent clusters without additional supervision.

1 Introduction

User feedback is a key part of the development and improvement of software products. Each piece of feedback needs to be manually reviewed and assigned to the correct engineer responsible for maintaining the feature mentioned in the report. On online platforms with millions of users, different users tend to report the same issue, yielding a large number of duplicate reports. Sorting through these massive volumes of bug reports incur a significant amount of engineering time and cost. Additionally, these services are constantly releasing new product features. Therefore, we cannot rely on static annotated data for product feature classification because they rapidly become outdated. This motivates us to develop a framework that can automatically identify duplicate bug reports and cluster them without requiring additional labels.

Previous research in the software engineering domain has addressed duplicates detection and product feature identification as two separate problems framed as independent fully-supervised classification tasks (Nguyen et al., 2012; Budhiraja et al., 2018b; Jonsson et al., 2016; Mani et al., 2018). However, we observe that users generally report issues when accessing certain features of a product, e.g., '*app crashed when opening camera*', '*chat won't load*' and so on (here, `camera` and `chat` are the product features, respectively). Hence, two reports should *at least* discuss the same feature to be considered as duplicates. Therefore, we hypothesize that determining the feature discussed in a report is a *sub-task* of detecting whether or not a report is a duplicate to another one.

Inspired by the effectiveness of Siamese architectures for modeling pairs of texts (Tan et al., 2015; Bowman et al., 2015), we use a shared Recurrent Neural Network (RNN) to encode the two reports. We note that the latent vectors, learned by an RNN for a sequence of words, encode a multitude of semantic information; all of which may not be necessary to understand the topic of the report. We decompose the latent semantic vectors in order to distill only the topical information in a few designated dimensions. This allows us to perform the sub-task of feature-based clustering using only a subset of dimensions, and use the complete vector for the duplicate classification task. We propose a partially supervised learning framework that uses the label for duplicity through a similarity loss on the designated topic dimensions of the latent representation to learn topic clusters.

We use a two-step attention module, since the same words are often not crucial for both tasks. We first learn a self-attention for topic similarity

modeling and learn a conditional attention using a memory vector for duplicate classification.

To summarize, we present a systematic study of a classic problem in the software industry. The work has three major contributions.

- We propose a neural model for multi-task learning that requires supervision for only one of the tasks.
- The model uses semantic space decomposition and a hierarchical-conditional attention module to perform the tasks of duplicate detection and topic based clustering.
- We present the challenges we faced during our experience obtaining labels from non-technical annotators, and conduct extensive experiments on both engineer labeled and non-technical labeled datasets.

2 Related Work

The software engineering community has conducted some research work for detecting duplicate bugs. In Minh (2014), a combination of n-gram features and cluster shrinkage algorithm is used for duplicate classification, whereas, in Sun et al. (2011) the BM-25 based scoring has been used akin to information retrieval engines. A combination of tf-idf and topics learned by LDA have also been used (Nguyen et al., 2012; Budhiraja et al., 2018b). Recently, word embeddings have been used to compute similarity of two reports(Yang et al., 2016; Budhiraja et al., 2018a). However, in these approaches, the sequence information of a natural language sentence is not captured.

Among the NLP community, the closest line of applicable work are the generic approaches of textual similarity (Cer et al., 2017; Wang et al., 2017; Neculoiu et al., 2016; Yin et al., 2016). They predominantly employ Siamese architecture (Mueller and Thyagarajan, 2016; Severyn and Moschitti, 2015) and more recently, attention mechanism (Wang et al., 2017; Shen et al., 2018; Wang et al., 2018; Tran and Niedereée, 2018). Inspired by these approaches, we propose a solution for duplicate detection while utilizing the partial supervision to achieve a sub-task of clustering *for free*.

Solving owner attribution of a report has been approached using feature-based methods (Jonsson et al., 2016; Xuan et al., 2012) and very recently using deep learning solutions (Mani et al., 2018). However, the connection between these two problems have not been explored and they require supervision for product feature identification.

3 Annotation Challenges

We consider a dataset consisting of user reported bugs collected from the Snapchat app. The bugs have been submitted by beta-testers using a bug-tracking functionality named Shake2Report(S2R) within the app. In S2R a user can submit a small textual description of the bug and attach a screenshot of the app while experiencing the issue. In this work, we only consider the textual descriptions of the reports.

Following previous work of duplicate bug tracking (Lazar et al., 2014), we first studied 500 pairs of reports marked as duplicates by engineers. However, these pose several challenges. Firstly, we realized that engineers often needed additional meta-data such as stack-traces and timestamps to accurately determine whether or not a pair of reports referred to the same problem. On the other hand, such additional meta-data cannot be made available to external crowd-sourced annotators due to legal and user privacy restrictions. Finally, the total amount of reports was far beyond what an engineering team could tackle, hence the need for our classification system, which helps scaling the bug report load while maintaining the annotation quality.

Our task is to automatically classify report pairs, and to add engineers to the loop only when pairs are textually ambiguous. We asked non-expert annotators to label the pairs only according to their semantic similarity (no metadata). The annotators had a high agreement among themselves with a Krippendorff's alpha (Krippendorff, 1970) score of 0.78. After adding engineers to the annotation pool, the score dropped to 0.58. To understand this disparity, consider the pairs:

- *"Crashed when posting story from memories"* and *"Crashed while exporting from memories"*
- *"Stuck on typing notification"* and *"Did not get notification for this group chat"*

Although not marked as duplicates by non-experts, they referred to the same underlying problem from the engineering side. While out-of-scope for this paper, we plan to conduct a dedicated study to explore these differences in future work.

As mentioned in the introduction, bug▸team assignments become rapidly outdated as the app

product features (and therefore the teams working on them) are constantly evolving. Also, non-expert annotators cannot be assumed to have any knowledge of these assignments, so we treat bug ▸ team assignments as an unsupervised task.

4 Proposed Method

We now describe the proposed learning framework as shown in Figure 1. Given a pair of reports P and Q, the system uses a single binary label $r(P, Q)$ (1 if the reports are duplicate, 0 otherwise) to supervise both tasks: duplicate classification and topic similarity modeling.

4.1 Text Encoder

This component takes a bug report represented as a sequence of words $\{w_1, w_2, \cdots, w_n\}$ as input and encodes it to latent vectors $\{\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_n\}$.

We first use word embeddings to transform all words in a text into finite d-dimensional vectors $\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n\}$. The vectors are then fed to a Gated Recurrent Unit (GRU) layer (Chung et al., 2014). We use bi-directional GRU units to encode the context information around a word. For a word w_i, outputs from the forward and backward GRUs are latent vectors of dimension g and denoted as $\mathbf{h}_i^f$ and $\mathbf{h}_i^b$, respectively. We concatenate these two vectors to form a single latent representation of the word as, $\mathbf{h}_i = \mathbf{h}_i^f \oplus \mathbf{h}_i^b$.

4.2 Semantic Space Decomposition

A word's encoded representation $\mathbf{h}_i$ has various information intertwined in an abstract way within the vector dimensions. It is difficult to interpret individual dimensions and meaningfully use a subset of dimensions for a different sub-task.

We aim to distill the coarser topical information into designated dimensions within the vector, storing other finer pieces of information in remaining dimensions. Such disentanglement allows us to do the topic similarity modeling using only the designated dimensions and ignore the rest.

To this end, we force the network to learn topical information about each word in the first k dimensions of its latent vector space. For a bi-directional GRU, we have two encoded representations for the i^{th} word from the forward and backward passes i.e., $\mathbf{h}_i^f$, and $\mathbf{h}_i^b$. We define the topic vector of a word i as,

$$\boldsymbol{\theta}_i = \mathbf{h}_i^f[1:k] \oplus \mathbf{h}_i^b[1:k] \tag{1}$$

where $\mathbf{h}_i^f[1:k]$ and $\mathbf{h}_i^b[1:k]$ denote the first k dimensions in the GRU encoded latent vectors from the forward and backward passes respectively.

4.3 Topic Similarity Modeling

We aggregate the topic vectors of the constituent words to represent the topic of a report, and use a self-attention layer to learn the weights of the words important in determining the topic:

$$\mathbf{z}_i = \tanh(\mathbf{W} \cdot \mathbf{h}_i + \mathbf{b}_i) \tag{2}$$

$$\boldsymbol{\alpha}_i = \frac{\exp(\mathbf{z}_i)}{\sum_i \exp(\mathbf{z}_i)}, \tag{3}$$

$$\boldsymbol{\theta} = \sum_{i=1}^{n} \boldsymbol{\alpha}_i \cdot \boldsymbol{\theta}_i. \tag{4}$$

where $\boldsymbol{\alpha}_i$ is the weight for word w_i and $\boldsymbol{\theta}$ is the topic vector of the report. Note that only the designated topical dimensions of the words contribute to representing the topic vector of the report.

In order to learn the semantic space of topic vectors, we need to constrain them in such a way that if two reports (P and Q) have similar topics (e.g., both are talking about *camera*), their $\boldsymbol{\theta}^P$ and $\boldsymbol{\theta}^Q$ values should be closer and vice-versa. Since only the ground-truth label for duplicity is available, we consider this signal as *partially* observed for the topic modeling task. Following our hypothesis that determining the topic of a report is a sub-task for duplicate detection, there can be three possible cases for a pair of reports:

Case 1: Duplicates from the same topic;
Case 2: Non-duplicates from different topics;
Case 3: Non-duplicates from the same topic.

Since for duplicate tickets (P, Q) their topics are bound to be same, we reduce the distance between $\boldsymbol{\theta}^P$, and $\boldsymbol{\theta}^Q$ in our loss. However, non-duplicate pairs may or may not be from the same topic (Case 2, and 3).

We do not include Case 3 in the loss, as inferring it is difficult without explicit labels. We assume that if a pair of non-duplicate reports have no word overlap (apart from stopwords), then only they belong to different topics i.e., Case 2. For such pairs we wish to increase the distances between their topic vectors. We imbue this principle in the network through the following loss function:

$$\begin{aligned}\mathcal{L}_{\text{sim}} &= r(P,Q) \cdot S_{\text{C}}(\boldsymbol{\theta}^P, \boldsymbol{\theta}^Q) \\ &\quad - (1 - r(P,Q)) \cdot S_{\text{C}}(\boldsymbol{\theta}^P, \boldsymbol{\theta}^Q)\end{aligned} \tag{5}$$

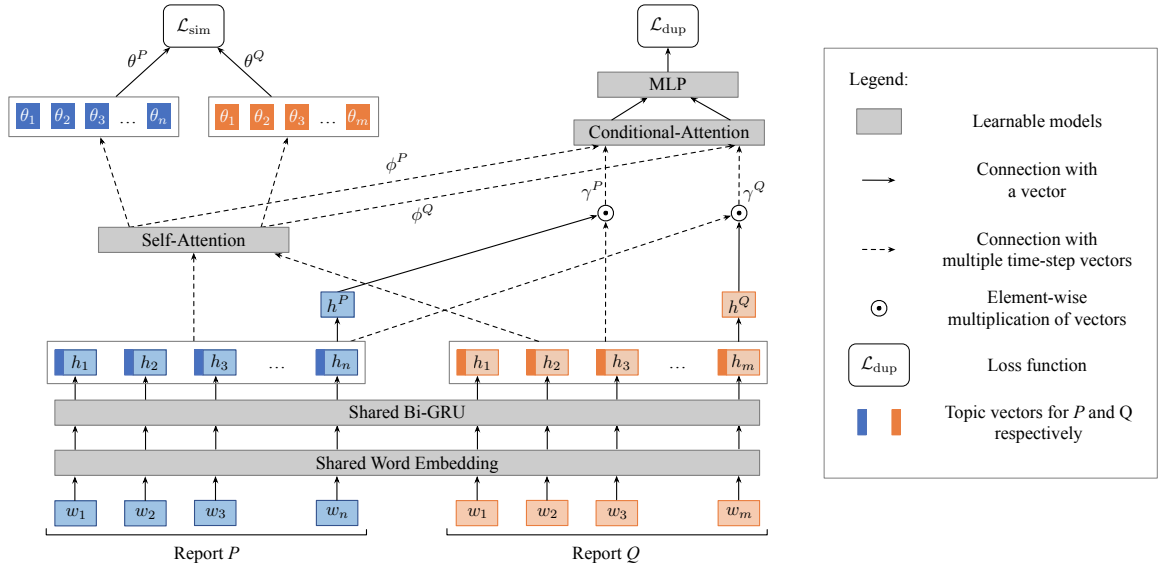


Figure 1: **The proposed partially supervised learning framework.** The framework takes two user reports P and Q as inputs. The left branch performs topic clustering with self-attention mechanism using topic dimensions. The right branch performs duplicates detection with conditional attention taking a pair of encoded reports as inputs.

where $r(P,Q)$ is the ground-truth duplicity label (0 or 1), S_{C} is cosine similarity. A careful reader might observe that the function minimizes distance of duplicates pairs ($r(P,Q) = 1$), while exhibiting the exact opposite behaviour for non-duplicates ($r(P,Q) = 0$). To account for the skewed distribution of duplicate vs. non-duplicates, we normalize the loss using proportional class weights.

4.4 Duplicate Classification

The final component of the network considers the complete latent vectors of the words for the task of duplicate classification. We use another attention layer for this component to allow the network to focus on important words. However, it is not necessary that the same words will be important for both the tasks. For example, consider the following pair of tickets: *"Can't import these pics from camera roll to memories"* and *"No pics in memory"*. In order to determine the product feature, the self attention module needs to focus on the words *memories* and *memory* in the reports, respectively. However, to decide whether they are duplicates or not, the second attention module needs to focus on the specific error, i.e. *Can't import pics* and *No pics* in the two reports respectively. For report P, we create a memory vector $\boldsymbol{\phi}^P$ to guide this attention layer using the hidden representations of the words as well the previously learned self-attention weights for topic modeling.

$$\boldsymbol{\phi}^P = \sum_i \boldsymbol{\alpha}_i^P \cdot \mathbf{h}_i^P + \sum_i \mathbf{h}_i^P \tag{6}$$

We also note that unlike the self-attention used for topic similarity modeling, for duplicate classification we need to use a form of conditional attention. For determining whether a report (P) is duplicate to another one (Q), the words that are important in P are dependent on the words in Q. Therefore, for each word position i in P we compute its relevance to the report Q as,

$$\boldsymbol{\gamma}_i^P = \mathbf{h}_i^P \odot \mathbf{h}^Q \tag{7}$$

where $\mathbf{h}_i^P$ is the hidden state of the i^{th} word in report P, $\odot$ denotes element-wise multiplication, and $\mathbf{h}^Q$ is the representation of report Q obtained using averaging the hidden states of all words in Q i.e. $\mathbf{h}^Q = \text{Avg}(\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_m)$, m is the number of words in Q.

We now use this conditional representation $\boldsymbol{\gamma}_i^P$ and the memory vector $\boldsymbol{\phi}^P$ to learn the conditional attention weights ($\boldsymbol{\beta}$) and compute the weighted representation $\mathbf{c}^P$ as,

$$\mathbf{c}_i^P = \tanh(\mathbf{W}_1 \cdot \boldsymbol{\gamma}_i^P) \cdot \tanh(\mathbf{W}_2 \cdot \boldsymbol{\phi}^P) \tag{8}$$

$$\boldsymbol{\beta}_i^P = \frac{\exp(\mathbf{c}_i^P)}{\sum_i \exp(\mathbf{c}_i^P)} \tag{9}$$

$$\mathbf{c}^P = \sum_{i=1}^{n} \boldsymbol{\beta}_i^P \cdot \mathbf{c}_i^P \tag{10}$$

We concatenate the weighted representations of P and Q as $\mathbf{c}^{PQ} = \mathbf{c}^{P} \oplus \mathbf{c}^{Q}$. We pass this concatenated vector through a Multi-Layer Perceptron, the final layer of which outputs a prediction $\hat{r}(P, Q)$ whether they are duplicates or not. We use binary cross-entropy loss to train this objective.

$$\mathcal{L}_{\text{dup}} = H(r(PQ), \hat{r}(PQ)), \tag{11}$$

where H is binary cross-entropy. The overall loss for the whole network is a weighted combination of the two losses.

$$\mathcal{L} = \lambda \mathcal{L}_{\text{sim}} + (1 - \lambda)\mathcal{L}_{\text{dup}} \tag{12}$$

In our experiments, we use $\lambda = 0.5$ but it can be varied depending on the importance of the tasks. We optimize the network using Adam optimizer and train in an end-to-end fashion through back-propagation.

5 Evaluation

5.1 Dataset

We experiment on four real-world datasets.

(1) Snap S2R : Composed of bugs reported through an in-app tool. The data was labeled as duplicate or not by non-technical annotators (described in section 3).

(2-4). Open Source Projects [1] : Repository of bugs submitted for the open-source projects of **Eclipse Platform**, **Mozilla Firefox** and **Eclipse JDT**. Each report consists of a short title and a longer description which often describes the steps to reproduce the issue. We only consider the title of the report in our experiments. The reported bugs have been marked for duplicates by engineers while resolving them.

We augment the S2R data with randomly sampled negative pairs to resemble a positive to negative ratio that is estimated in production. For the Eclipse and Firefox datasets, we randomly sample negative pairs, keeping the positive to negative ratio close to what was previously mentioned (Lazar et al., 2014). Table 1 shows the statistics of the four datasets. These datasets were selected to capture different training sizes and variations of vocabulary used.

5.2 Experimental Settings

We initialize the words using pre-trained Glove embeddings (Pennington et al., 2014) of 300 dimensions but tune it during training to capture the intrinsic features of the specific task and dataset at hand. For the GRU layers we use 150 dimensions and the first 20 dimensions of them are used for topic representation. For the MLP in duplicate classification we use 2 layers of fully connected layers of 100 hidden neurons each, with $relu$ activation and 20% dropout rate. For fair comparison we use the same number of parameters in all baselines. The learning rate is set to 0.003 and a batch size of 128 samples is used while training all the models.

[1] https://github.com/logpai/bugrepo

Dataset	#pairs	#vocab	%dups	#reports	avg #words /report
Snap S2R	66,945	7763	9%	17,255	14.6
Eclipse Platform	170,312	29,702	12%	83,608	7.9
Eclipse JDT	90,592	17,228	14%	44,670	8.1
Firefox	231,628	34,590	26%	113,262	10.0

Table 1: Statistics of the datasets used

5.3 Evaluating Duplicate Classification

We first start evaluating the proposed approach for detecting duplicate bug reports. We compare with the following six baselines:

- **Logistic Regression** uses a bag of n-grams representation with n ranging from one to three words. Tf-idf scores are used for feature weighting.
- **Siamese CNN** (Severyn and Moschitti, 2015) encodes the texts with a shared convolution network followed by an MLP for classification.
- **Siamese Bi-GRU** uses a shared bidirectional GRU for encoding text.
- **Siamese Bi-GRU with Attention** uses an attention layer on top of the GRU outputs to encode the texts
- **DWEN** (Budhiraja et al., 2018a) is the state-of-the-art deep learning approach for duplicate bug detection using word embeddings.
- **BiMPM** (Wang et al., 2017) is the state-of-the-art sentence similarity modeling that uses multi-perspective symmetric matching for a pair of texts.

We report results after averaging five independent trials using 80% data for training, 10% for

Method	Snap S2R			Eclipse Platform			FireFox			Eclipse JDT		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Logistic Regression	0.67	0.63	0.65	0.71	0.88	0.78	0.92	0.95	0.94	0.76	0.88	0.81
Siamese CNN (2015)	0.67	0.63	0.65	0.81	0.81	0.81	0.93	0.94	0.93	0.79	0.79	0.79
Siamese Bi-GRU	**0.76**	0.61	0.68	**0.86**	0.84	0.85	**0.95**	0.93	0.94	0.85	0.84	0.84
Siamese Bi-GRU w Att	**0.76**	0.62	0.68	**0.86**	0.86	0.86	**0.95**	0.94	0.94	0.84	0.84	0.84
DWEN (2018a)	0.69	0.55	0.62	0.83	0.74	0.78	0.93	0.92	0.93	0.81	0.71	0.76
BiMPM (2017)	0.73	0.65	0.69	**0.86**	0.82	0.84	0.94	0.92	0.93	0.85	0.79	0.82
Our Method	0.73	**0.67***	**0.70**	0.84	**0.91***	**0.87***	0.94	**0.96***	**0.95***	**0.86**	**0.90***	**0.88***

Table 2: Comparison of different methods for duplicate classification task on multiple datasets. * denotes statistical significance with the runner up for $p\text{-}value < 0.01$

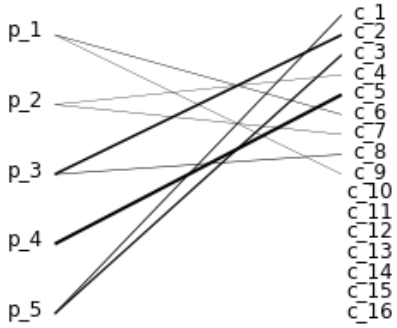
validation, and 10% for testing. Due to the imbalanced class distribution, we use precision, recall and F1-score of the positive class as evaluation metrics.

From the results in Table 2 we see that our proposed method largely outperforms all other models in terms of recall and in most cases on F1 score as well. Using a sequence encoder like GRU boosts the performance significantly compared to using n-grams as in Logistic Regression or CNN. We note that the overall scores on Eclipse and Firefox datasets are much higher than the scores on Snap S2R data. One of the reasons could be the difference in data size where the Snap S2R dataset is much smaller in size compared to others. Noisy labels from non-technical annotators could also be a potential reason. Manually examining the posts we also note that the bug reports submitted for the open projects are often by engineers, who use concrete technical terms to describe the problem. In contrast, for the Snap S2R dataset, the reports are submitted by end-users using free text with non-technical terms that make it harder for a machine learning model to disambiguate.

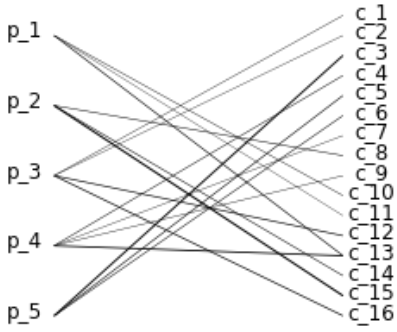
5.4 Evaluating Clusters

A major advantage of our model is its ability to perform clustering of the bug reports to aid in ownership attribution. In this set of experiments, we show that our model is able to learn a semantic space so that nearby reports in the space indeed belong to the same product feature.

We use the topic vector (θ) of a report learned by our model and run an off-the-shelf clustering algorithm K-means[2]. Ideally, there should be a one-to-one mapping between the obtained clusters and the actual project features. For comparison, we use one of the most popular unsupervised topic model, LDA (Blei et al., 2003) to learn latent topics from the bug reports. For implementation of LDA we use the Gensim library[3]. We perform standard pre-processing steps, and remove stopwords before training the LDA model. We evaluate the learned latent clusters by comparing them to the ground truth project features, as have been marked by users while submitting reports for S2R dataset. For fair comparison, we use the same (20) number of clusters for both methods, which is close to the actual number of product features.



(a) Our Method



(b) LDA

Figure 2: Mapping between the learned latent clusters and top-5 product features for Snap S2R dataset.

[2] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[3] https://radimrehurek.com/gensim/models/ldamodel.html

Figure 2 shows the mapping between learned latent clusters and top-5 ground truth product features for the proposed model and LDA. To reduce the effect of noisy labels, we consider a cluster-product feature mapping only if more than 10 reports about a product feature are assigned to a cluster. The width of the connecting edge is proportional to the number of times a cluster-product feature association is observed.

Firstly, we observe that LDA requires more clusters to represent the same number of features. This demonstrates that LDA over-clusters, failing to identify reports that talk about the same product feature and assigns them to different clusters. Secondly, some clusters in LDA (e.g., c_13) are connected to multiple product features denoting impurity of the cluster. Finally, for most features, their mappings to a cluster are much stronger in our model as demonstrated by the width of the edges in the graph. This shows that with partial supervision from duplicity labels, we are able to learn clusters that better correspond to ground truth product features.

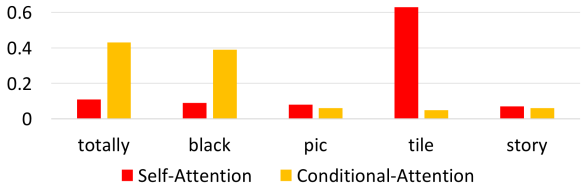
Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
video	load	lens	app	specs
audio	long	lenses	map	spec
videos	story	carousel	viewing	importing
playback	time	preview	crashed	v2
sync	taking	applied	navigating	unpaired

Table 3: Top Words according to their tf-idf scores from few clusters learned by our model.

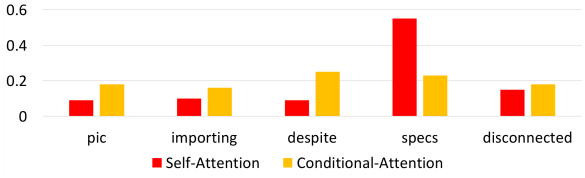
Table 3 shows top words according to their tf-idf scores appearing in bug reports assigned to a cluster. The learned clusters are coherent and represent specific product features. For example, *Cluster 1* talks about issues regarding `video` or `audio`, while *Cluster 2* is mostly about issues related to loading of a `story` (a feature in the app). In *Cluster 3* reports related to various `lenses` and `filters` are placed, while *Cluster 4* seems to be about `maps` and `navigation` features. It is also interesting to note that generic issues like *'crashed'* appear in some of the clusters. Although one could identify most of the bugs as crashes, this indicates that users have a specific vocabulary when referring to a particular product feature.

5.5 Analyzing Attention Weights

Finally, we present case studies of the attention weights learned by our proposed model. As we have two-steps of attention layers optimized by the two objectives, they can learn to focus on words that are important for the specific tasks.



(a) bug report: *totally black pic for **tile** story*



(b) bug report: *pic importing despite **specs** disconnected*

Figure 3: Visualization of the attention weights learned by the two attention modules on few sample bug reports

Figure 3 shows the attention weights for two sample bug reports from Snap S2R dataset. We observe that the two attention modules tend to focus on different parts of the text. The self-attention gives more weight to the words that determine the overall topic of the report (*tile* in Figure 3a, *specs* in Figure 3b). In contrast, the conditional-attention focuses on the set of words describing the specific issue to aid in detecting duplicates.

6 Conclusion

In this paper we have studied bug-tracking, which is a widespread problem in the software industry. We develop a neural architecture that can learn to classify duplicates and cluster them into meaningful latent topics without additional supervision. The architecture decomposes the latent semantic space of a word to only distill the topical information into a few designated dimensions, and uses a two-step attention module to focus on different textual parts for the two tasks. We share the challenges of annotating a user reported bug dataset with non-technical annotators, as opposed to using annotations from engineers. This is a direction we plan to further explore in future work. Experimental results on different types of datasets indicate that the proposed approach is promising compared to existing techniques for both tasks. Most importantly, our model's construction is generic and presents new possibilities in various domains for modeling sub-tasks for free, with partial supervision from another task.

References

David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. 2015. A large annotated corpus for learning natural language inference. *Conference on Empirical Methods in Natural Language Processing*.

Amar Budhiraja, Kartik Dutta, Raghu Reddy, and Manish Shrivastava. 2018a. Dwen: deep word embedding network for duplicate bug report detection in software repositories. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, pages 193–194. ACM.

Amar Budhiraja, Raghu Reddy, and Manish Shrivastava. 2018b. Lwe: Lda refined word embeddings for duplicate bug report detection. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, pages 165–166. ACM.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*.

Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS Workshop on Deep Learning and Representation Learningp, December*.

Leif Jonsson, Markus Borg, David Broman, Kristian Sandahl, Sigrid Eldh, and Per Runeson. 2016. Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. *Empirical Software Engineering*, 21(4):1533–1578.

Klaus Krippendorff. 1970. Estimating the reliability, systematic error and random error of interval data. *Educational and Psychological Measurement*, 30(1):61–70.

Alina Lazar, Sarah Ritchey, and Bonita Sharif. 2014. Generating duplicate bug datasets. In *Proceedings of the 11th working conference on mining software repositories*, pages 392–395. ACM.

Senthil Mani, Anush Sankaran, and Rahul Aralikatte. 2018. Deeptriage: Exploring the effectiveness of deep learning for bug triaging. *arXiv preprint arXiv:1801.01275*.

Phuc Nhan Minh. 2014. An approach to detecting duplicate bug reports using n-gram features and cluster chrinkage technique. *Int. J. Sci. Res. Publ.(IJSRP)*, 4(5):89–100.

Jonas Mueller and Aditya Thyagarajan. 2016. Siamese recurrent architectures for learning sentence similarity. In *AAAI*, volume 16, pages 2786–2792.

Paul Neculoiu, Maarten Versteegh, and Mihai Rotaru. 2016. Learning text similarity with siamese recurrent networks. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 148–157.

Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N Nguyen, David Lo, and Chengnian Sun. 2012. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 70–79. ACM.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Aliaksei Severyn and Alessandro Moschitti. 2015. Learning to rank short text pairs with convolutional deep neural networks. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pages 373–382. ACM.

Chenlin Shen, Changlong Sun, Jingjing Wang, Yangyang Kang, Shoushan Li, Xiaozhong Liu, Luo Si, Min Zhang, and Guodong Zhou. 2018. Sentiment classification towards question-answering with hierarchical matching network. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3654–3663.

Chengnian Sun, David Lo, Siau-Cheng Khoo, and Jing Jiang. 2011. Towards more accurate retrieval of duplicate bug reports. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 253–262. IEEE Computer Society.

Ming Tan, Cicero dos Santos, Bing Xiang, and Bowen Zhou. 2015. Lstm-based deep learning models for non-factoid answer selection. *arXiv preprint arXiv:1511.04108*.

Nam Khanh Tran and Claudia Niedereée. 2018. Multihop attention networks for question answer matching. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 325–334. ACM.

Lu Wang, Shoushan Li, Changlong Sun, Luo Si, Xiaozhong Liu, Min Zhang, and Guodong Zhou. 2018. One vs. many qa matching with both word-level and sentence-level attention network. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2540–2550.

Zhiguo Wang, Wael Hamza, and Radu Florian. 2017. Bilateral multi-perspective matching for natural language sentences. *IJCAI*.

Jifeng Xuan, He Jiang, Zhilei Ren, and Weiqin Zou. 2012. Developer prioritization in bug repositories. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 25–35. IEEE.

Xinli Yang, David Lo, Xin Xia, Lingfeng Bao, and Jianling Sun. 2016. Combining word embedding with information retrieval to recommend similar bug reports. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 127–137. IEEE.

Wenpeng Yin, Hinrich Schütze, Bing Xiang, and Bowen Zhou. 2016. Abcnn: Attention-based convolutional neural network for modeling sentence pairs. *Transactions of the Association of Computational Linguistics*, 4(1):259–272.