Multimedia Tools and Applications
https://doi.org/10.1007/s11042-018-7020-3

Improving the system log analysis with language model and semi-supervised classifier

Guofu Li[1,2] · Pengjia Zhu[3] · Ning Cao[4] · Mei Wu[5] · Zhiyi Chen[1] · Guangsheng Cao[6] · Hongjun Li[6] · Chenjing Gong[6]

Received: 12 July 2018 / Revised: 22 November 2018 / Accepted: 3 December 2018 /
Published online: 23 March 2019

Abstract
Mining the vast amount of server-side logging data is an essential step to boost the business intelligence, as well as to facilitate the system maintenance for multimedia or IoT oriented services. Considering the vast volume of the data repository, designers of these logging-data analysis systems need to carefully balance the speed of the processing and the accuracy of the message classification. Conventional keyword-based log data monitoring and classification is sufficiently fast, but does not scale well in complex systems, especially when the target system is contributed by a large group of developers, each may differ in the way to encode the logging messages, and often carrying misleading labels. Conversely, many of the sophisticated approaches may suffer from their considerable time consumption, such that delayed processing jobs may begin to accumulate, and can hardly support the timely decision requirements. Meanwhile, we also suggest that the design of a large scale online log analysis should follow a principle that requires the least prior knowledge, in which unsupervised or semi-supervised solution is preferred. In this paper, we propose a two-stage machine learning based method, in which the system logs are regarded as the output of a *quasi-natural language*, pre-filtered by a perplexity score threshold, and then undergo a fine-grained classification procedure. Empirical studies on our web-services show that our method has obvious advantage in terms of processing speed and classification accuracy.

Keywords Log analysis · Language model · Machine learning

1 Introduction

An increasing portion of computation is now moving from the content consumption side to the server side due to the recent growth of the online services in various types. This trend is shared by areas including media steaming services and IoT services [11], strengthening the synergy between the user experience of interacting with the front-end interface

✉ Ning Cao
ning.cao2008@hotmail.com

Extended author information available on the last page of the article.

and its support from the back-end data service [2]. Large scale online system outputs tons of log data every day, containing different types of the sensitive and valuable information that requires the attention for both the business intelligence [20] and system maintenance, including optimization [1], diagnosis [26], security [5, 18] and privacy issues [24, 25], etc. In most scenarios, analyzing the log data by human labour is never an option, either due to its vast volume, or its real-time response that is needed by the business service, and thus requires a well-designed, automated process to monitor and classify this large volume.

Currently, researches of data analysis for the server-side multimedia systems mainly focus on how best to respond the user requests, such as searching queries or personalized content recommendation [8, 13], which are usually under the multi-modal context [6, 22]. Whereas, handling the log data produced by the server program itself is often thought to be simple enough for a few scripts task, and thus is often neglected. But the complication of the logging data in effect poses serious challenges for building desired intelligent programs, even by virtue of the state-of-the-art machine learning technologies.

A simple and intuitive way to handle these system logging automatically is by setting a list of keywords, based on which either online monitoring or off-line analysis can be easily implemented. However, the logging messages that set by the system developers are often similar to a type of natural language used in a special-domain, whose semantics often hard to be recovered by a simple set of word-based filter rules. When the word-choices made for different types of the system-exceptions often have large overlapping, full of the names of the sub-modules or functions, making it even harder to separate by keywords. Though many of the built-in logging modules provide the system developers with extra functionalities to include meta-information (e.g., a set of tags, warning levels, or the logging output time), a considerable portion of the coding practitioners do not have full consensus of the exact semantic of these labels, and often follow their own habit, and can even break the self-consistency in different part of the program. In practice, we observe a large amount of entries with label error which are actually normal, when actual error messages may be labeled as normal.

Another key challenge that requires careful examination by the designer of the the log analysis tools is that, the logging output volume commonly grows with system size and complexity. For a large commercial system, it is not uncommon to see several hundreds Tera-bytes of its historical logging data waiting to be processed, and Giga-bytes of new logging data are being produced every day. To ensure that the newly generated logging data do not stack and accumulate on the disk, the analysis algorithm has to be at least as fast as the generation speed, with comparatively limited computation power. This restriction blocks the log analyzer's developer from the choice of many time-consuming machine-learning algorithms, when still facing a complexity of the blurry semantics within the vast amount of data.

In addition to the previous two challenges, we emphasize that the develop team of the system logging analyzer is usually different from the one who build the target system.[1] Consequently, it is difficult, or sometimes impossible, for the algorithm designer of the logging analyzer to have full-knowledge of the conditions and implications of the logging messages. Besides, when a team is dedicated to the development of the logging analyzer, its goal is often to make the algorithm as generically applicable as possible. Therefore, we propose a "least prior knowledge" designing principal of such tools, which aims to

[1] We use the term "target system" to refer to the system which produce the logging data to be analyzed.

maximize the system flexibility by leveraging the large volume of the unlabeled logging data, with the minimum supervision.[2]

Few researches have been done on the quasi-language content produced on the back-end of the server side. One particular interesting side of the research is that methods originated from NLP or CV are in effect applicable to a variety of application scenarios. Current researches of the machine learning community mostly focuses on the NLP and computer vision, which are considered to be closely tied to the human-like intelligence. But this doesn't prevent other areas take advantages of the recent advancements of machine learning techniques, such as biometrics, [9, 10, 14], social network analysis [27].

In this paper, we based our experiments on the log data derived from two web-service modules of a large-scale online service, one for mobile apps and the other for customer service terminals. We choose a language model to first pre-filter a large amount of trivial log messages for acceleration, then a topic model based on the LDA (Latent Dirichlet Allocation) is applied before the filtered data are feed to the classification.

2 Backgrounds

Many of the conventional log analysis tools are built from a simple keyword list. For one reason, the system usually has a fixed template to output its runtime exception. And for another, each of the system developer usually has a style of output a certain type of error or message with a relatively fixed pattern. Therefore, for a small scale system, keywords do reflect the natural of the system logging, and implementing such an analysis tool via keywords has a good balance between the cost and effectiveness. However, as the system grows larger and the service it provides grows complex, this approach gradually becomes infeasible in practice.

2.1 Related works

Researches on analyzing system logs always evolve along with the advancements of the target systems themselves, including their underlying technology, volume capacity, etc. For instance, Silverstein et al. studies the task of query log analysis at web scale [19]. They analysis of an AltaVista Search Engine query log to study the interaction of terms within queries, aiming to find correlations between the searching items. Splunk[3] is one of the most widely used tool for log analysis. It provides several out-of-box services that are key to the system administrators with several built-in machine learning algorithms, and has been integrated inside the Google's cloud platform. However, Splunk only provides a command line wrapper of a few general-purpose machine learning algorithms that have not been tailored for log analysis tasks yet.

The runtime monitoring system proposed by Xu et al. [26], is another representative example to apply machine learning techniques on log data mining, which involves four submodules of *log parsing*, *feature creation*, *anomaly detection* and *visualization*. The SherLog tool proposed by Yuan et al. [29] shares a similar design principle of our method. It

[2]It may be confusing to use the term "unlabeled", as the log messages commonly carry labels when the are firstly generated, which may highly obscure and unrelated to their actual meaning.

[3]https://www.splunk.com/en_us/homepage.html

analyzes source code by leveraging information provided by run-time logs, and predicates the events that would happen in the production run without re-executing the program or prior-knowledge on the log's semantics. The *Beehive* project by Yen et al. [28] targets at the network security issue, and aims to extract useful knowledge by mining the dirty log data. Their experiments are conducted on the network packages and regard the meta-information of different layers of the network protocols as the main features for classification. Kobayashi et al. [12] suggest that processing the log data can be regarded as a special type of natural language processing problem, such that many of the NLP methods can be used for analyzing the data and extracting a repository of templates.

We share a similar intuition that regards log messages as quasi-natural language, and makes use of the language model [4] to perform pre-filtering. Language model has been applied to various types of natural language related tasks. For instance, Ponte et al. use a language model to improve the information retrieval system [15]. Similar to our idea, Salvetti et al. [17] applies the language model to another quasi natural language scenario, web URLs, for segmentation and analysis in a fast weblog classification task.

Du et al. are among the first who apply deep learning methods on log data analysis, and propose a system called *DeepLog* [7], which also regards the log data as a sequence of natural language output, and relies on a LSTM-based network architecture to find new patterns in the logs. Their LSTM-based approach is also motivated by the language model, similar to a part of our work, as the optimization objective is also to predicte the next token based on the previously seen context. However, their work focuses on a different set of tasks, which are anomaly detection and root cause analysis.

2.2 Task description

The logging messages written by an online service system is a rich repository of the event history of the system, as well as an indicator of the current status of service. They provide means more than the debug information for the system develop team, but also the source of intelligence for the system maintenance and audit. Therefore, various tasks can be defined as the goal of the log analyzer. In this paper, we assume the goal to be a multi-way classification, which is a common setting. More specifically, we consider three classes of the messages:

1. *information message*,
2. *operation error message*, and
3. *system error message*.

Note that the actual class label that each message should be assigned to is often different from what it received from the target system, though the labeling schema may seem similar to each other. For example, a "`Connection Timeout Exception`" may be set as a *system error* by the target system, but is actually harmless to the entire work flow of the system. It may be just a bad coding practice in the target system's code for handling the exception.

It is common for an online service system to run in 24×7 schedule, while the logging keeps constantly growing. In our specific scenario, the logging data that output from only two of the sub-modules that we experiment with are about 5GB and 90GB each day, when the volume of the logging data from all the sub-modules grows up-to 3TB each day. Thus, it imposes a type of constraint that the average processing speed for the analysis tool must be faster than the average speed of the growth of the log data. Therefore, our approach splits the whole process into two consecutive stages:

1. first, we use a language model to quickly filter out a large portion of the low-value messages;
2. then, we use a three-way classifier based on LDA topic modeling and feedforward neural networks.

3 Language model based pre-filter

The goal of the first stage of our method is to perform a fast pre-filtering on the large volume of logging data, similar to the intuition behind the "roll-out policy" in for reinforcement learning.

3.1 Fitting the language model

The term "language model" in the context of common NLP (Natural Language Processing) refers to a way to find the joint probability of a word (token) sequence $P(w_{1:T})$, by breaking the join probability to the product of a chain of conditional probabilities:

$$P(w_{1:T}) = \prod_{t=1}^{T} P(w_t|w_{1:t-1}), \tag{1}$$

in which $w_{1:T}$ denotes the word-sequence from position 1 to T. In many of the circumstances, we refer to the sequence behind the condition bar as the *context* that the probability is conditioned on. With an n-order Markovian assumption, we may have

$$P(w_t|w_{1:t-1}) \simeq P(w_t|w_{t-n:t-1}), \tag{2}$$

such that

$$P(w_{1:T}) \simeq \prod_{t=1}^{T} P(w_t|w_{t-n:t-1}) \tag{3}$$

Recent researches in the NLP field have witnessed the shift of the language model from the *statistic* language model to the *neural* language model, in which distributed representation is used to represent the semantic of each word or token. Comparatively, traditional statistic language model is a non-parametric model that trivially estimates the probabilities based on the word counts in the training corpus, while the neural language model creates an embedding layer on which semantic of the context is modeled. Although empirical studies shows that the neural language model has advantages in smoothing and generalization, calculating the conditional probability by looking up a count-table in a controlled language scope is much faster than running a forward pass in a deep neural network (usually some variations of the recurrent neural network). Therefore, in our experiment, we choose to follow a traditional 3-gram language model to perform a fast pre-filtering. Comparing to the keyword-based filter, language model has the capability to take the word-order into consideration, making it ideal to handle the log messages whose meaning is more obvious by recognizing its template rather than its wording.

Though language model itself can be trained to make text classification directly, our approach however considers the perplexity derived from it to be an indicator of how common a certain type that the message in question is seen in the training corpus. It can be counter-intuitive that the most notable log (error) message that the system administrator should pay attention to, is the message whose type is rarely seen previously. However, considering a system that has been running stably for a sufficient long time, we should have

the confidence that it behaves as expected in most cases. Therefore, regardless of whether the message was labeled as "error" by the target system, we care more about the "unusual" events, while the "commonly-seen" messages are trivial in natural.

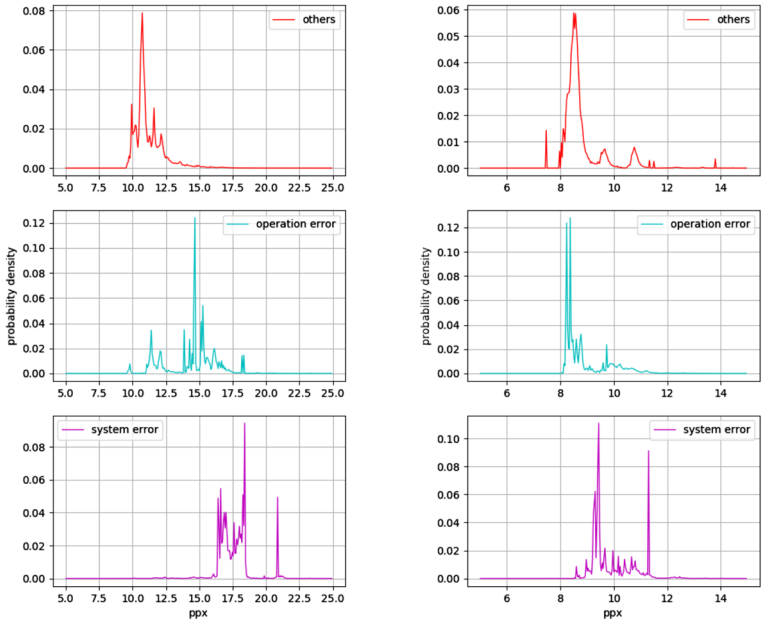
3.2 Fast filtering by perplexity

Perplexity (or ppx) is a score implied by the language model, meaning the averaged joint probability of the token sequence when trained on corpus. Formally speaking, we may calculate the ppx score by

$$ppx = P(w_{1:T})^{\frac{1}{T}}. \tag{4}$$

One way to interpret the ppx as the "average branching factor" of a long sequence given by a language model. Thus, a less ambiguous sequence prefix always implies a more steep distribution of the next token. Since the impact of the length of the sequence is removed from the ppx score, with a fixed model and training corpus, the relative ppx difference is manly determined by how familiar certain sequence of tokens are observed in the training data, while the absolute values of the ppx is less interesting to our task.

Perplexity is a single digested scalar value that tells the uncertainty of a sequence. Our empirical study suggests that this score alone is a strong indicator of the natural of the log entry, and presents valuable evidence to differentiate messages of different types. Figure 1a and b illustrate the distributions of the perplexity that drawn from the system log carrying different labels.



(a) The *scrmapp* submodule. (b) The *scrmweb* submodule.

Fig. 1 The distribution of the perplexity of log entries drawn from three different log types. The *information* type is written as *other* here

Both figures show that there is exist a peak in the distribution in the perplexity range from 8 to 9, regardless of the types of the message type. Moreover, log messages with the label *system error* tend to have another large spike in the high-perplexity range. This spike is more obvious in Fig. 1a, which is much higher than what it has around 8 to 9. We assume that a special and rarely-seen type of system error tend to have a very long trace of exceptions into the calling stack, causing both its lexicon and word sequence more obscure for the language model. The threshold to cut the high-frequency message types is a hyper-parameters, which balance the trade-off between the recall and speed acceleration. In practice, we choose a threshold of 13.5 for the *scrmapp* submodule, and 9.5 for the *scrmweb* submodule, to filter out all valueless logging messages for further analysis.

4 Semi-supervised classification

The second stage of our method consists of two steps. We first use a classic topic modeling technique to leverage the unlabeled data to construct a semantic space, then apply a multilayer perceptron neural network to perform a supervised learning for the three-way classification task.

4.1 LDA topic modeling

Latent Dirichlet allocation (LDA) [3] is a directed graphic model based on a Bag-of-Word representation of texts. It makes the assumption that each of the word occurrence is a sample drawn from a multi-nominal distribution of the vocabulary, conditioned on the topic the text wants to express, and uses the Dirichlet as its conjugate prior for smoothing. Previous studies show that in non-conventional semantic space construction scenarios, LDA can better capture the semantics that critical to the task than other dimension reduction methods, such as LSA (Latent Semantic Analysis) or word2vec [21], probably due to the smoothing effect imposed by the Dirichlet prior. In practice, we often use an extended version of LDA which has an extra Dirichlet-distributed topic-word distribution, as shown in Fig. 2.

The three values of shown in Fig. 2 are K the number of topics, M the number of documents, and N the number of words in each document. Besides, the interpretation of several key parameters are as follows:

- $\boldsymbol{\alpha}$ is the Dirichlet prior on the per-document topic distributions,
- $\boldsymbol{\beta}$ is the Dirichlet prior on the per-topic word distribution,
- θ_m is the topic distribution for document m,
- ϕ_k is the word distribution for topic k,
- $z_{m,n}$ is the topic for the n-th word in document m,
- and $w_{m,n}$ is the specific word.

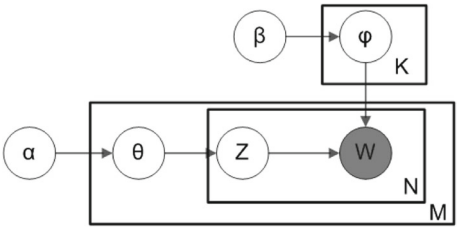


Fig. 2 A plate notation for LDA with Dirichlet-distributed topic-word distributions

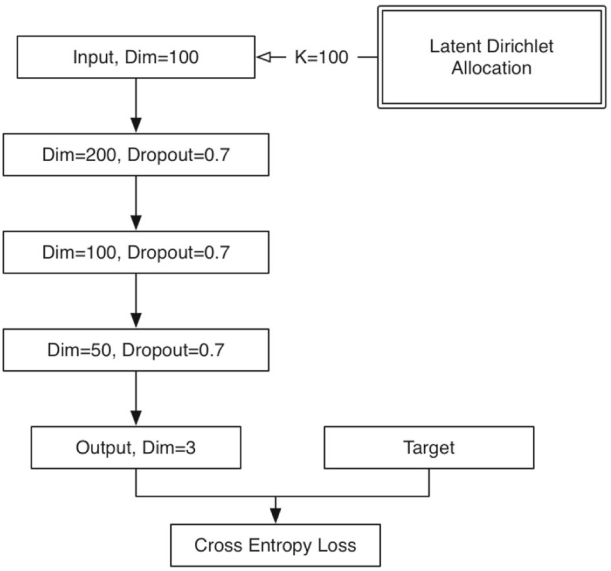


Fig. 3 The architecture of the 5-layer feed-forward neural network as the final step for classification task

The only observable variables (i.e., the node with gray background) is the occurrence of n in document m, denoted as $w_{m,n}$. We use the gensim[4] implementation of the LDA in our experiment [23], and choose hyper-parameter K (i.e., the number of topics) to be 100, $\alpha_{k=1...K} = \frac{1.0}{K}$, and $\beta_{w=1...V} = \frac{1.0}{K}$.

4.2 Classification by MLP

Multilayer Perceptron (MLP) is one of the basic type of feed-forward neural network architecture with densely connected layers one after another, whose weights are tuned by a supervised learning technique called back-propagation [16]. The input to the MLP's input units are the K dimensional semantic vectors derived from the transformation of the Bag-of-Word representation of the original text by the previously learnt LDA topic model. In our experiment, we create a 5 layers of the densely connected feedforward network, consisting of an input layer, three hidden layers with ReLU activation, each with a drop-out probability of 0.3 to keep from overfitting, and a $softmax$ layer to produce the final output distribution. The architecture of this MLP is illustrated in (Fig. 3).

5 Empirical evaluation

We select two consecutive days' log data from two representative system modules, namely *scrmweb* and *scrmapp* from a national-wide telecom company. The volume sizes of these

[4] https://radimrehurek.com/gensim/index.html

Table 1 The log processing time consumption of the *ppx*-based pre-filtering

Sub-module	Volume	#Core Alloc	Avg CPU	Total time	Avg Speed
scrmapp	∼200GB	20	1403%	∼23:59	∼165KB/s per-core
scrmweb	∼10GB	30	1923%	∼1:09	∼124KB/s per-core

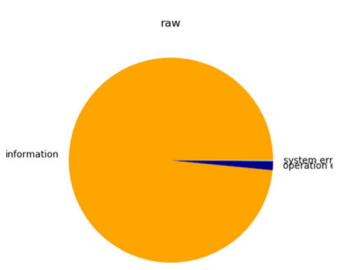
log data files are about 10GB and 200GB, containing 8M+ and 90M+ message instances respectively. To ensure that our analyzer is able to process the day-round logging output sufficiently fast, the average processing time of one day's log data must be shorter than 24 h.

5.1 Language model filtering

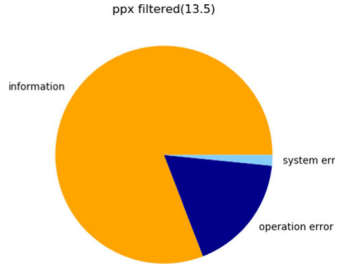
In our method, the major time consumption lies in the first stage, which uses the language model to pre-filter out most of the logging data. In the experiment, we allocate 20/30 cores for language modeling, multiple cores of E5 2630v4 and 1 Nvidia Titan XP for classification. Table 1 shows the log processing time consumption of the two modules.

Our experiment shows that even by a simple 3-gram language model, the pre-filtering stage still costs the most of the time consumption. For a two days' data, the pre-filtering time consumption is almost an entire day. The average processing speed, depending on the different module it works on, varies from 124KB/s per-core to 165KB/s per-core, but is still several orders of magnitude faster than the inference stage of the LDA + MLP classifier.

The speed boosting effect is only the positive side of the perplexity filtering stage. To further study how well the *ppx* score can behave as a filter, we construct two datasets by selecting a subset of the log data from both submodules randomly, each consisting of 100,000 entries. We communicate with the module developers, whose expertise knowledge enables us to build an oracle rule-based classifier that generates the ground truth class labels (namely, the *information*, *system error*, and) for each dataset. The we see how the distribution of the three class labels among the log entries are influenced by the filter. The result is shown in Figs. 4 and 5.

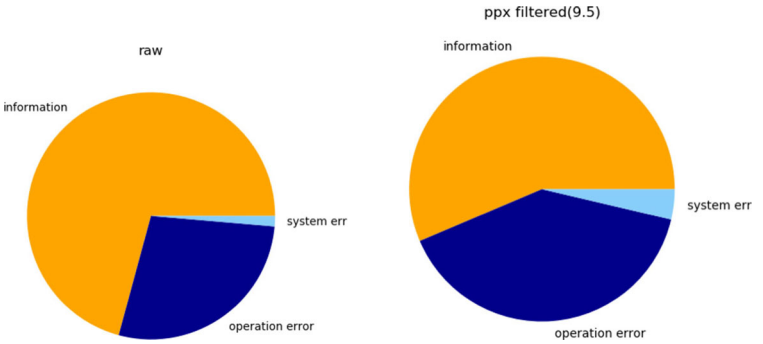


(a) The distribution of the three types of the log entries (by number of counts) before filtering.



(b) The distribution of the three types of log entries (still by number of counts) after filtering.

Fig. 4 The influence of the *ppx* filter on the distribution of the three types of log entries of the *scrmapp* sub-module. The threshold of *ppx* is set to 13.5

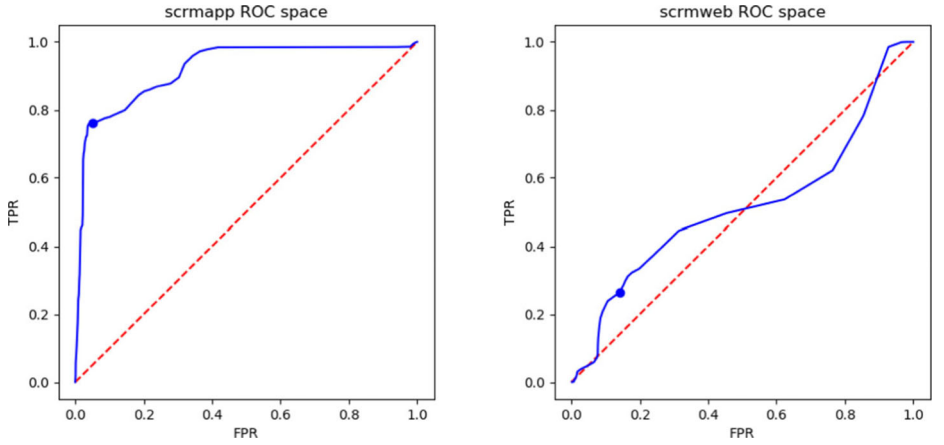


(a) The distribution of the three types of the log entries (in terms of entry counts) before filtering.

(b) The distribution of the three types of log entries (in terms of entry counts) after filtering.

Fig. 5 The influence of the *ppx* filter on the distribution of the three types of log entries of the *scrmweb* sub-module. The threshold of *ppx* is set to 9.5

The changes in the distributions complies with the initial intuition about the perplexity score, and convince us that the *ppx* score is an effective criterion to filter out unwanted data. In addition, we found that by choosing the threshold to be 13.5 and 9.5 for the two sub-modules respectively, only 6.1% and 17.7% of the data (in term of entry counts) are left to enter the next stage of process. But the above results alone do not provide more quantitive measure on the side-effects when the filtering process is working, making us to turn to the ROC curve. Different threshold of the *ppx* score, as an important hyper-parameter here, would lead to different ratio of true-positive and false-positive. The ROC curve tells us the overall behavior of a filtering system when different values of the threshold is chosen. In



(a) The ROC curve of the *ppx* filtering for the *scrmapp* submodule.

(b) The ROC curve of the *ppx* filtering for the *scrmweb* submodule.

Fig. 6 The ROC curve of the perplexity based filter. We treat the log entries of type *system error* and *operation error* as valuable entries (thus the *true* targets), and the log entries that pass the filter as the *positive* signals. The blue dots on both curves denote the picked *ppx* thresholds (13.5 and 9.5) for the two submodules

Table 2 The distribution of the pre-set labels of the data in sub-module *scrmweb*

Log type	#Instance	Percentage
Normal	5.6M+	70.8%
Operation	2.2M+	27.8%
System	0.1M+	1.4%

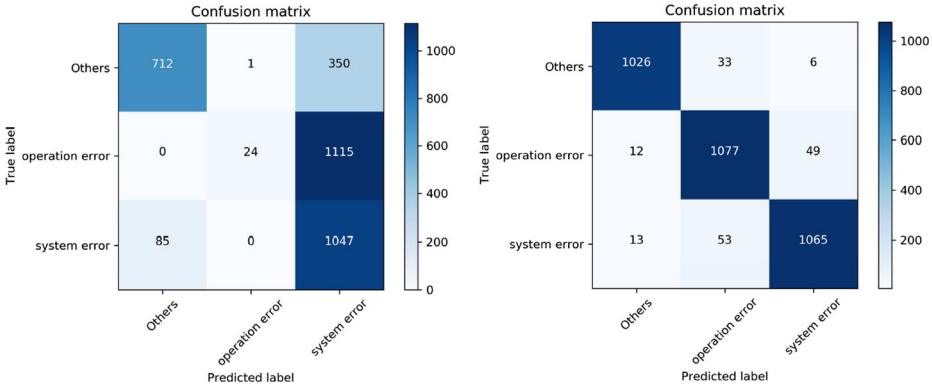
this scenario, we regard the log entries of type *system error* and *operation error* as valuable entries (thus the *true* targets) that we would like to keep, and the log entries that pass the filter as the *positive* signals. We continue to use the same datasets, and the experiment results are shown in Fig. 6.

It is interesting that the *ppx* filter works astonishingly well on the *scrmapp* submodule, but is relatively weak on the *scrmweb* submodule. This is largely due to the nature of the complexity of the sentences in log data, and can be partially told from Fig. 1: Notice that there is an considerable amount of overlap between the *information* log entries and *operation error* log entries within *ppx* range from 8 to 9 in Fig. 1b. The blue dots on both curves denote the picked *ppx* thresholds (13.5 and 9.5) for the two submodules, which are located at the lower-left corner of the entire curve, indicating that we still bias towards the recall in practice. Even though, the *ppx* filter still allow us to discard a large proportion of the log entries.

5.2 Classification accuracy

Our next experiment evaluates the classification accuracy of our proposed method, and is tested on the *scrmweb* sub-module only. The ground truth labels in this experiment are again created by the oracle rule-based classifier as before. Table 2 shows the data distribution from two-day's logging outputs.

We create a subset of the logging consisting of 20,000 instances (about 25M in size), in which the three categories are sampled evenly, and then allocate 5/6 of the entries for training, 1/6 for testing. A baseline system that simply uses a trivial list of keywords as



(a) The confusion matrix of the baseline system on the test-set.

(b) The confusion matrix of our proposed method on the test-set.

Fig. 7 The confusion matrix of the three-way classification results on the test dataset

Table 3 The Precision/Recall/F-score of the baseline system and our proposed method

System	Precision	Recall	F-Score
Baseline	0.549	0.954	0.697
Proposed method	**0.973**	**0.976**	**0.975**

The scores of best performance are highlighted with bold font

the criteria for the three way classification is constructed,[5] which gives the performance as shown in Fig. 7a. Figure 7b shows the confusion matrix of our proposed method this test-set. Rows in the matrices are the numbers of the instances for each true labeled class, columns in the matrices are the numbers of the instances for each predicated class of our method. Our empirical experiment shows that the average Precision/Recall/F-score reach 0.973, 0.976 and 0.975 respectively. Table 3 gives a comparison of the Precision/Recall/F-score between our proposed method and the baseline implementation.

Although the baseline system has a similar performance at the Recall (due to the general setting of the keyword set), it has far worse performance on the Precision side, and thus has a much lower F-score. Notice that most of the log data are pre-filtered, only a small portion of the entire dataset is left for the LDA topic modeling and MLP classifier. The processing speed in this stage is $\sim$1.9KB/s per-core, which is approximately 100 times slower than the language model based filtering stage, making pre-filtering stage an essential part of the entire process.

6 Conclusions

Designing a log data analyzer for a large scale online system faces several key challenges, including the vague semantics of the logging messages, the need of a sufficiently fast processing speed, and above all, the lack of prior knowledge of how the developers of the target system choose the message templates and vocabularies. Based on these considerations, we propose a "least prior knowledge" principle for designing such a system, and present an implementation based on several classic natural language processing techniques. We use a fast n-gram language model to pre-filter the trivial types of messages, which considers the complexity of the grammar and vocabulary. Then we use Latent Dirichlet allocation (LDA) to construct a semantic space from a large repository of unlabeled bag-of-word representation of the messages. Finally, a standard feed-forward multilayer-perceptron is used to perform the three-way classification, with the minimum amount of human supervision. Out empirical results provide convincing evidences that the proposed method achieves nice performance in terms of speed and accuracy.

Acknowledgments This research is supported by Shanghai University Youth Teacher Training Funding Scheme (ZZslg16054), National Social Science Foundation (16BXW031), Grant of Shandong Province Vocational Education Educational Reform Research Project "Study on Vocational Colleges" Professional Building Service Regional Upgrade Industries" (2017209).

[5]In the baseline system, we use the keyword-set {"Exception"} to capture the *system error* log entries, and the keyword-set {"Error", "Failure"} to capture the *operation error* log entries.

References

1. Añorga J, Arrizabalaga S, Sedano B, Goya J, Alonso-Arce M, Mendizabal J (2018) Analysis of youtube's traffic adaptation to dynamic environments. Multimed Tools Appl 77(7):7977
2. Bhuiyan MZA, Wang G, Wu J, Cao J, Liu X, Wang T (2017) Dependable structural health monitoring using wireless sensor networks. IEEE Trans Depend Secure Comput 14(4):363
3. Blei DM, Ng AY, Jordan MI (2003) Latent dirichlet allocation. J Mach Learn Res 3:993
4. Charniak E (1996) Statistical language learning. MIT, Cambridge
5. Cheng R, Xu R, Tang X, Sheng VS, Cai C (2018) An abnormal network flow feature sequence prediction approach for ddos attacks detection in big data environment. Comput Mater Contin 55(1):95
6. Datta D, Singh SK, Chowdary CR (2017) Bridging the gap: effect of text query reformulation in multimodal retrieval. Multimed Tools Appl 76(21):22871
7. Du M, Li F, Zheng G, Srikumar V (2017) In: Proceedings of the 2017 ACM SIGSAC conference on computer and communications security. ACM, pp 1285–1298
8. Elayeb B, Romdhane WB, Saoud NBB (2018) Towards a new possibilistic query translation tool for cross-language information retrieval. Multimed Tools Appl 77(2):2423
9. He P, Deng Z, Wang H, Liu Z (2016) Model approach to grammatical evolution: theory and case study. Soft Comput 20(9):3537
10. He P, Deng Z, Gao C, Wang X, Li J (2017) Model approach to grammatical evolution: deep-structured analyzing of model and representation. Soft Comput 21(18):5413
11. Kaur J, Kaur K (2017) A fuzzy approach for an iot-based automated employee performance appraisal. Comput Mater Contin 53(1):23
12. Kobayashi S, Fukuda K, Esaki H (2014) In: Proceedings of the ninth international conference on future internet technologies. ACM, p 11
13. Liu Q, Guo Y, Wu J, Wang G (2017) Effective query grouping strategy in clouds. J Comput Sci Technol 32(6):1231
14. Liu Y, Ling J, Liu Z, Shen J, Gao C (2018) Finger vein secure biometric template generation based on deep learning. Soft Comput 22(7):2257
15. Ponte JM, Croft WB (1998) In: Proceedings of the 21st annual international ACM SIGIR conference on research and development in information retrieval. ACM, pp 275–281
16. Rumelhart DE, Hinton GE, Williams RJ (1985) Learning internal representations by error propagation. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science
17. Salvetti F, Nicolov N (2006) In: Proceedings of the human language technology conference of the NAACL, companion volume: short papers. Association for Computational Linguistics, pp 137–140
18. Shen J, Gui Z, Ji S, Shen J, Tan H, Tang Y (2018) Cloud-aided lightweight certificateless authentication protocol with anonymity for wireless body area networks. J Netw Comput Appl 106:117–123
19. Silverstein C, Marais H, Henzinger M, Moricz M (1999) In: ACm SIGIR forum, vol 33. ACM, pp 6–12
20. Sylaiou S, Mania K, Paliokas I, Pujol-Tost L, Killintzis V, Liarokapis F (2017) Exploring the educational impact of diverse technologies in online virtual museums. Int J Arts Technol 10(1):58
21. Veale T, Chen H, Li G (2017) I read the news today, oh boy, international conference on distributed, ambient, and pervasive interactions. In: International conference on distributed, ambient, and pervasive interactions. Springer, Cham, pp 696–709
22. Venkitasubramanian AN, Tuytelaars T, Moens MF (2017) Entity linking across vision and language. Multimed Tools Appl 76(21):22599
23. Řehůřek R, Sojka P (2010) In: Proceedings of the LREC 2010 workshop on new challenges for NLP frameworks. ELRA, Malta, pp 45–50
24. Xia Z, Xiong NN, Vasilakos AV, Sun X (2017) Epcbir: an efficient and privacy-preserving content-based image retrieval scheme in cloud computing. Inf Sci 387:195
25. Xia Z, Zhu Y, Sun X, Qin Z, Ren K (2018) Towards privacy-preserving content-based image retrieval in cloud computing. IEEE Trans Cloud Comput 6(1):276
26. Xu W, Huang L, Fox A, Patterson D, Jordan MI (2009) In: Proceedings of the ACM SIGOPS 22nd symposium on operating systems principles. ACM, pp 117–132
27. Yang W, Wang G, Bhuiyan MZA, Choo KKR (2017) Hypergraph partitioning for social networks based on information entropy modularity. J Netw Comput Appl 86:59
28. Yen TF, Oprea A, Onarlioglu K, Leetham T, Robertson W, Juels A, Kirda E (2013) Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks. In: Proceedings of the 29th annual computer security applications conference. ACM, pp 199–208
29. Yuan D, Mai H, Xiong W, Tan L, Zhou Y, Pasupathy S (2010) Sherlog: error diagnosis by connecting clues from run-time logs, architectural support for programming languages and operating systems. 38(1):143

Guofu Li is a Lecturer at College of Communication and Art Design in University of Shanghai for Science and Technology. He received his BSc in Software Engineering from the Fudan University in 2007, and his PhD in Computer Science and Informatics from the University College Dublin in 2014. He was a Post-doctoral Research Fellow at the CNGL research centre (ADAPT now) and Computer Science and Informatics of the University College Dublin from 2014 to 2015. His research interests include digital communication, machine learning and natural language processing.

Ning Cao is an academic leader of College of Information Engineering at Qingdao Binhai University. He is also a master supervisor at College of Information Technology of Hebei University of Economics and Business. He received his B.S. degree from Harbin Institute of Technology in Software Engineering in 2008 and his Ph.D degree from University College Dublin in Computer Science in 2015. His research interests include IoT, wireless sensor networks. Up to date, he has published over 60 publications.

Affiliations

Guofu Li[1,2] · Pengjia Zhu[3] · Ning Cao[4] · Mei Wu[5] · Zhiyi Chen[1] · Guangsheng Cao[6] · Hongjun Li[6] · Chenjing Gong[6]

Guofu Li
li.guofu.l@gmail.com

Pengjia Zhu
zhupengjia@gmail.com

Mei Wu
wumei0604@hotmail.com

Zhiyi Chen
iamedithchen@gmail.com

Guangsheng Cao
52605560@qq.com

Hongjun Li
911845094@qq.com

Chenjing Gong
821670587@qq.com

1 College of Communication and Art Design, University of Shanghai for Science and Technology, Shanghai, China

2 Computer Science and Informatics, University College Dublin, Dublin, Ireland

3 State Street Corporation, Boston, MA, USA

4 College of Information Engineering, Sanming University, Sanming, China

5 School of Computer Science and Engineering, Wuhan Institute of Technology, Wuhan, China

6 College of Information Engineering, Qingdao Binhai University, Qingdao, China