What Do Developers Ask About ML Libraries? A Large-scale Study Using Stack Overflow

***Abstract*—Modern software systems are increasingly including machine learning (ML) as an integral component. However, we do not yet understand the difficulties faced by software developers when learning about ML libraries and using them within their systems. To that end, this work reports on a detailed (manual) examination of 3,280 highly-rated Q&A posts related to ten ML libraries, namely *Tensorflow*, *Keras*, *scikit-learn*, *Weka*, *Caffe*, *Theano*, MLib, *Torch*, *Mahout*, and *H2O*, on *Stack Overflow*, a popular online technical Q&A forum. We classify these questions into seven typical stages of an ML pipeline to understand the correlation between the library and the stage. We also perform inter- and intra-library analyses to understand broad trends. Our findings reveal the urgent need for software engineering (SE) research in this area. Both static and dynamic analyses are mostly absent and badly needed to help developers find errors earlier. While there has been some early research on debugging, much more work is needed. API misuses are prevalent and API design improvements are sorely needed. Enabling reuse of trained models across libraries needs attention. Last and somewhat surprisingly, a tug of war between providing higher levels of abstractions and the need to understand the behavior of the trained model is prevalent. These findings suggest new paths for SE researchers to help improve the engineering of software that includes ML components.**

***Index Terms*—Machine learning, Q&A forums, API misuses**

I. Introduction

Machine learning (ML) is gradually becoming an essential computational tool in a software developer's toolbox for solving problems that defy traditional algorithmic approach. Software developers are fulfilling this need by development and refinement of a number of new ML libraries [18]. Recently it has also been suggested that ML can introduce unique software development problems [29], [28], [8]. However, we do not yet know about the problems that users of ML libraries face and those that they choose to ask about publicly.

Prior work has shown that studying question and answer (Q&A) forums such as *Stack Overflow* can give significant insights into software developer's concerns about a technology [32], [33], [35], [17], [19], [26], [5], [34], [25], [37], [27], [30], [20], [4], [16], but has not focused on ML libraries. More details of related work are discussed in Section V.

This work presents a study of the problems faced by developers while using popular ML libraries. Our study also leverages the posts on *Stack Overflow*. Since 2015, there has been growing interest and significant increase in ML related questions and distinct users making *Stack Overflow* a representative source of dataset for our study. We selected 10 ML libraries to study, identified by a survey [18] and confirmed by counting the number of posts on *Stack Overflow* related to those libraries. These libraries are *Caffe* [15], *H2O* [10], *Keras* [12], *Mahout* [23], *MLlib* [22], *scikit-learn* [24], *Tensorflow* [1], *Theano* [6], *Torch* [13], and *Weka* [14].

Caffe [15] is a deep learning library for Python and C++. ***H2O*** [10] is a deep learning library for Java, R, Python or Scala and its key feature is to provide a workflow-like system for building ML models. ***Keras*** [12] is a deep learning library for Python whose key feature is to provide higher-level abstractions to make creating neural networks easier. *Keras* also uses *Tensorflow* or *Theano* as the backend. ***Mahout*** [23] is aimed at providing scalable ML facilities for Hadoop clusters. ***MLlib*** [22] is aimed at providing scalable ML facilities for Spark clusters. ***scikit-learn*** [24] is a Python library that uses *Tensorflow* or *Theano* as the backend. This library provides a rich set of abstract APIs [9] to hide complexity of ML from the user in an effort to make ML features widely accessible.

Tensorflow [1] provides facilities to represent a ML model as data flow graphs. ***Theano*** [6] and ***Torch*** [13] are aimed at scaling ML algorithms using GPU computing. A unique aspect of *Theano* is that it provides some self-verification and unit testing to diagnose some runtime errors. ***Weka*** [14] is a ML library for Java. It provides API support for data preparation, classification, regression, clustering and association rules mining tasks and a GUI for making models easier.

All in all, this set is both representative and provides variety. We selected a total of 3,280 highly-rated *Stack Overflow* posts for this study. A team of three Ph.D. students, with experience in coursework on AI and ML, and using ML libraries, independently read and labeled each of the posts producing 9,840 labels that were then compared for consistency producing 177 conflicting labels on 177 different posts. All of these conflicts were resolved using mediated, face-to-face conflict resolution meetings between all three participants. The results of our study on this data, described in Section III and Section IV, suggest several directions for software engineering research.

Debugging and program analyses support is badly needed to detect errors earlier. Interestingly, API misuses are prevalent suggesting the need for API design improvements. Reuse of trained models across libraries is sought after. Lastly, it turns out, abstractions provided by some ML libraries can, in fact, make understanding a ML model's behavior difficult.

The contributions of this work include: (1) a labeled and verified, dataset of ML library-related Q&A on *Stack Overflow*, (2) a classification scheme for ML-related Q&A, (3) an intra-library analysis to identify strengths and weaknesses of ML libraries, and (4) an inter-library analysis to identify relative strengths and weaknesses.

TABLE I: Numbers of posts having different score (S) about ML libraries. The bold column represents selected posts.

Library	S ≥ 0	S ≥ 1	S ≥ 2	S ≥ 3	S ≥ 4	S ≥ 5
Caffe	2,339	1,320	620	318	192	**135**
H2O [10]	771	452	167	73	34	**20**
Keras [12]	5,708	3,323	1,751	953	568	**385**
Mahout [23]	1,186	610	293	160	103	**64**
MLlib [22]	1,688	929	498	272	173	**123**
scikit-learn [24]	9,246	5,302	2,898	1,759	1,188	**849**
Tensorflow [1]	21,115	10,109	4,962	2,769	1,827	**1,318**
Theano [6]	2,332	1,341	711	421	265	**198**
Torch [13]	1,226	640	312	161	91	**61**
Weka [14]	2,512	1,216	568	293	181	**127**
Total	48,123	25,242	12,780	7,179	4,622	**3,280**

II. Methodology

Our study uses Q&A posts on *Stack Overflow*, a popular platform used by developers. Our first step was to find the total number of questions asked about all 15 ML libraries highlighted by a recent survey [18]. Out of these, we selected 10 ML libraries for the study as shown in Table I. We excluded the other five libraries because the numbers of questions about them were too few (less than 20).

On *Stack Overflow*, each question is rated by the community. The score of a question is computed as $S = |N_U| - |N_D|$ where $|N_U|$ is the number of upvotes and $|N_D|$ is the number of downvotes. The higher score is an indicator of the higher quality of the question, which has been used in prior works [21]. Table I shows the entire distribution of the questions for each library based on the score S.

We selected questions with the score of 5 or higher (bold column in Table I) to focus on high-quality questions while keeping the workload of manually labeling each question manageable.

Next, we manually classified each *Stack Overflow* question into categories to study them further. We first discuss the classification of categories and then our labeling process.

A. Classification of Questions

We classify the questions in *Stack Overflow* into several categories. First, we classify the questions into two top-level categories based on whether the question is related to ML or not. Questions related to installation problems, dependency, platform incompatibility, Non-ML APIs, overriding the built-in functionality, adding custom functionality fall into Non-ML category as shown in Fig. 2. We classify the ML-related questions into six categories based on the stages of a typical ML pipeline [36], also reproduced in Fig. 1. Among those seven stages, data collection is out of the scope of this study because ML libraries do not provide this functionality which leaves us with six categories. The full classification is shown in Figure 2. Next, we discuss the categories in more detail.

1) Data Preparation: This top-level category includes questions concerning about converting the raw data into the input data format needed by the ML library.

Data adaption. Questions under this category are about reading raw data into the suitable data format required by the library. Data reader provided by the library usually provides this functionality. Questions about converting data, encoding, etc., also fall under this subcategory.

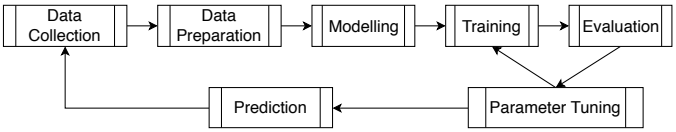


Fig. 1: Stages in a typical ML pipeline [36]

Featuring. Questions under this category are about feature extraction and selection. *Feature extraction* is a process to reduce dimensionality of the data where existing features are transformed into a lower dimensional space. *Feature selection* is another strategy of dimensionality reduction where informative features that have impact on the model are selected.

Type mismatch. Type mismatch happens when the type of data provided by the user doesn't match the type required by the ML API. For example, if an API needs `floating point` data as input but the client provides a `String` then the ML API will throw an exception due to type incompatibility.

Shape mismatch. Shape mismatch occurs when the dimension of the tensor or matrix provided by a layer doesn't match the dimension needed by the next layer. These kinds of errors are very common in deep learning libraries.

Data cleaning. Data cleaning phase, sometimes also called data wrangling, includes removal of null values, handling missing values, encoding data, etc. Without proper data cleaning the training may throw exceptions, and accuracy may be suboptimal. Data cleaning often perplexes developers because it has a direct impact on the performance of the trained model.

2) Modelling: The subcategories of this category include:

Model selection. This subcategory includes questions related to the choice of the best model and choice of the API version (e.g. whether to chose SVM or decision tree).

Model creation. This subcategory includes questions related to creating the model using the APIs, given a problem scenario how the APIs can be pipelined to create the model.

Model conversion. This subcategory includes questions related to conversion of a model trained using one library and then using the trained model for prediction in an environment using another library. For example, a model trained in *Torch* can be used for further training or prediction using *Theano*.

Model load/store. This subcategory contains questions about storing models to disk and loading them to use later.

3) Training: The subcategories of this category include:

Error/Exception. Questions about errors faced by users in the training phase fall into this subcategory. The errors may appear due to various reasons. If the errors are due to shape mismatch or type mismatch we put them into data preparation category. Otherwise, all errors are placed into this subcategory.

Parameter selection. Some frameworks have optional parameters, and developers have to choose appropriate values for these parameters and also pass relevant values to the compulsory parameters. Questions related to these problems fall into this subcategory.

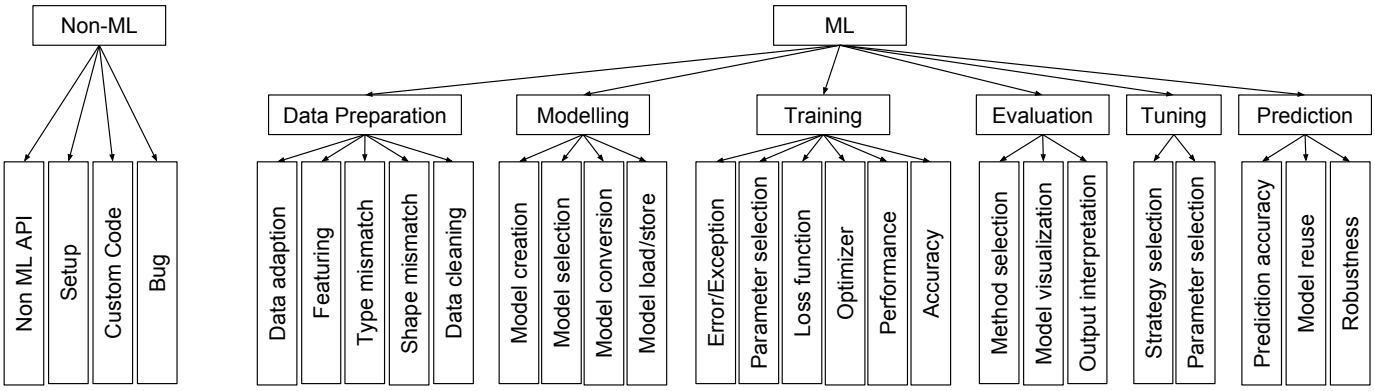


Fig. 2: Classification used for categorizing ML library-related Stack Overflow questions for further analysis

Loss function. Questions related to choosing and creating loss functions fall into this category, e.g., whether to use cosine distance, mean square error, etc. for an algorithm.

Optimizer. Questions related to the choice of optimizer are placed into this subcategory, e.g., which optimizer to use among Adam, AdaGrad, AdaDelta, etc.

Performance. In this subcategory, questions related to long training time and/or high memory consumptions are placed.

Accuracy. Questions related to training accuracy and/or convergence are placed into this subcategory.

4) Evaluation: The subcategories of this category include:

Evaluation method selection. Question related to the problems in the usage of APIs for doing validation fall into this sub category. For example, a question asked "*which of the eight APIs for eight different types of validations in scikit-learn, namely KFold, LeaveOneOut, StratifiedKFold, RepeatedStratifiedKFold, RepeatedKFold, LeaveOneGroupOut, GroupKFold and ShuffleSplit, should be used?*"

Visualizing model learning. The developers sometime need to visualize the behavior of the model to get better understanding of the training process and also to know the effects of evaluation on the change of loss function and accuracy. Those questions are placed in this subcategory.

5) Hyper-parameter Tuning: Hyperparameter tuning is used to improve the model's performance. The values of hyperparameters affect model accuracy. For example, a bad learning rate may cause a model to learn poorly and give low accuracy. The subcategories of this category include:

Tuning strategy selection. Questions about choosing among APIs for different tuning methodologies are placed into this subcategory. For example, one question asked about three APIs of *scikit-learn* for doing parameter tuning using grid search or randomized search or parameter sampling.

Tuning parameter selection. This subcategory covers discussions related to the selection of parameters for tuning. Some parameters may not have an effect on the model accuracy other than increasing the training time while some might have a significant effect on the accuracy, e.g. the code below specifically selects the parameters to tune at line 4.

```
1 from sklearn import svm, datasets
2 from sklearn.model_selection import GridSearchCV
3 iris = datasets.load_iris()
4 parameters = {'kernel':('linear','rbf'), 'C':[1,10]}
5 svc = svm.SVC()
6 clf = GridSearchCV(svc, parameters)
```

The user is trying to tune the kernel and C parameter of the support vector machine algorithm. So, in this case the hyper parameter tuning stage will find the best combination of parameters from values given at line 4.

6) Prediction: After the model is trained and evaluated, the model is used to predict new input data. Questions in this category are about problems faced by the developers during prediction and include the following subcategories.

Prediction accuracy. The discussions related to prediction accuracy, e.g. due to overfitting, are placed into this category.

Model reuse. Developers might have difficulty in reusing existing models with their own datasets for prediction to make use of the state of the art models from well-known providers.

Robustness. Questions in this subcategory concern with the stability of the models with slight changes, which could be noise, in the datasets.

B. Manual Labeling

Manual labeling of the Q&A dataset was the most important (and time-consuming) step before our analysis. To make the manual labeling bias-free we recruited three participants. Each participant had coursework in both AI and ML and had experience using ML libraries to solve problems. Each participant labeled all the questions producing 9,840 labels.

Participant Training. Before the labeling, the participants were provided with the classification shown in Fig. 2. Then, a training session was conducted where each (sub)category was discussed and demonstrated using examples.

Labelling Efforts. First, each participant gave each question one of the labels from top-level categories namely Non-ML, Data Preparation, Modelling, Training, Evaluation, Tuning, Prediction. Then, (s)he assigned a subcategory.

We found that, at the steady state, a participant could label around 50-60 questions per hour. For labeling the whole dataset consisting of 3,280 questions, each participant took around 1 week time. In total, 168 person-hours were spent on labeling this dataset.

Reconciling Results. After collecting labels separately from each participant, a moderator then compared them. If there is

	R1	R2	R3
R1	1.00	0.94	0.92
R2	0.94	1.00	0.91
R3	0.92	0.91	1.00

(a) Kappa coefficients (κ).

0.00–0.20	slight agreement
0.21–0.40	fair agreement
0.41–0.60	moderate agreement
0.61–0.80	substantial agreement
0.81–1.00	perfect agreement

(b) Interpretation of κ value.

Fig. 3: Cohen's kappa coefficients for labeling process.

an inconsistency between participants for a question, the moderator created an issue in a repository for resolution. Among all 3,280 questions, 177 (5%) needed further discussion.

Then, the three participants had two in-person meetings to discuss those 177 questions. The participants read the questions carefully again and voted individually. If the votes matched we accepted those as resolved, otherwise participants discussed the reasons behind choosing a label and tried to achieve consensus. In most cases, the opinions differed due to the ambiguous nature of the questions. For example, for a question asking about suboptimal accuracy, it was difficult to say from the question without deeper observation whether it is talking about accuracy in the prediction stage or accuracy in the training or evaluation stage. We resolved these type of questions by carefully analyzing the text and manually inferring based on the description.

We measured the inter-participant agreements using Cohen's kappa coefficient (κ) as shown in Fig. 3a. It measures the observed level of agreement between raters of a particular set of nominal values and corrects for agreements that would appear by chance. The interpretation of κ's values is shown in Fig. 3b. From Fig. 3a, we see that the kappa coefficient between all the raters involved in the labeling process is more than 0.9 indicating perfect agreements.

III. Intra-Library Analysis

In this section, we discuss the potential strengths and weaknesses of each library w.r.t. our classification in Fig. 2. The statistics are shown in Table II and Table III.

1) Caffe: As shown in Table II, most of the questions about *Caffe* involve **modeling (32%)** and **training (24%)**. The third largest category is data preparation (14% of the total questions). The largest subcategory of questions for *Caffe* is **model creation** with **26.52%** questions. Even though prediction stage has only 6% of the questions at the top-level category, one of its subcategory, prediction accuracy, is in the top-5 most concerned problems.

2) H2O: H2O is the library that has the least amount of data from *Stack Overflow* among the 10 chosen libraries for analysis with only 20 questions. 41% of them are about data preparation that suggests that the design of data preparation stage can be improved for this library or tools could be made to ease data preparation. In the subcategories, we see data adaptation related question appear 35.29% of the time and the next major subcategory is model creation with 17.64%. From the outlier analysis, *H2O* has much more questions in the output interpretation and tuning strategy selection than other libraries. That would mean that its library users could benefit from **improved APIs for hyper-tuning strategy and displaying well-formatted results**.

3) Keras: Keras is a popular deep learning library which uses *Tensorflow* or *Theano* as backend. *Keras* provides a collection of abstract APIs that hide the details of *Tensorflow* or *Theano* computation. This library has most of the questions on modeling (28%), followed by training (25%) and data preparation (16%). In the subcategories, the majority of posts are on model creation (25.88%), and several others with similar contributions: data adaptation (7.90%), prediction accuracy (6.81%), performance (6.27%), shape mismatch (5.50%), error/exception (5.50%) and parameter selection (5.50%).

The result implies that the abstraction in *Keras* does not help reduce the difficulty with modeling in *Keras*. More research is needed on the **abstraction strategy for ML libraries**.

4) Mahout: Apache *Mahout* is a specialized library for performing ML on clusters. This library has the most questions in modeling stage with 44%, most of which are in model creation, followed by data preparation with 17%. This indicates that creation of model on clusters is comparatively more difficult. Other than model creation the prominent subcategories for this library are data adaptation (14.58%), prediction accuracy (4.2%) and error/exception(4.2%). The result suggests that **much work is still needed in distributed ML**.

5) MLlib: MLlib is another prominent library for distributed ML. This library has the highest percentage of questions on data preparation (33%). A reason could be that *MLlib* uses a special data structure called resilient distributed datasets (RDD). Most of the time developers need to be comfortable with this format of data while using this library. **Using a new data format may also create new technical difficulty in data preparation stage**. The other prominent category is modeling (29%). Among the subcategories, the prominent subcategories are data adaptation (25.20%), model creation (23.52%) and error/exception (5.88%).

6) scikit-learn: scikit-learn is a popular ML library in Python. Though it is not used for deep learning, its use for regression, supervised and unsupervised learning, and recommendation related tasks are well known. This library provides abstract APIs that hides the details of ML. In our study, the majority of questions about *scikit-learn* were about data preparation (26%), modeling (25%), and training (18%). Here again, we could see that abstractions does not resolve the problems on modeling, data preparation and training for this library either. It would suggest that **hiding all the details of ML tasks using abstract APIs might create difficulties in tracing and debugging** when the models are not learning properly or over-fitting.

We also notice from outlier analysis that *scikit-learn* has more questions in model creation and tuning parameter selection than others. *scikit-learn* provides a lot of optional parameters to be selected in their APIs, whose values are hard to select yet affect accuracy. That could be the reason why its users have more difficulties in selecting parameters. This calls for **research on designing APIs for parametric ML**.

TABLE II: Percentage of questions in each top-level category across libraries (in %).

	Caffe	*H2O*	*Keras*	*Mahout*	*MLlib*	*scikit-learn*	*Tensorflow*	*Theano*	*Torch*	*Weka*	Q1	Q3	IQR	Median	SD
Data preparation	14.0	41.0	16.0	17.0	33.0	26.0	16.0	17.0	23.0	30.0	16.5	29.0	12.5	20.0	8.7
Modelling	32.0	24.0	28.0	*44.0	29.0	25.0	27.0	27.0	33.0	20.0	26.5	31.2	4.7	27.0	5.5
Training	24.0	18.0	25.0	8.0	15.0	18.0	21.0	16.0	20.0	12.0	15.0	20.7	5.7	18.0	4.7
Evaluation	1.0	6.0	8.0	4.0	7.0	9.0	9.0	3.0	3.0	10.0	3.5	8.3	4.8	6.0	2.9
Tuning	1.0	*6.0	0.0	0.0	2.0	*4.0	1.0	1.0	0.0	0.0	0.0	1.5	1.5	1.0	1.9
Prediction	6.0	0.0	10.0	4.0	6.0	7.0	4.0	2.0	2.0	11.0	2.6	6.6	4.0	5.0	3.2
Non-ML	22.0	6.0	13.0	23.0	6.0	11.0	20.0	35.0	20.0	10.0	10.3	21.4	11.1	16.0	8.6

$*$ indicates the library in the column is an outlier for the category in the corresponding row. $IQR = Q3 - Q1$: inter-quartile range. SD: standard deviation.

TABLE III: Percentage of questions in each subcategory across libraries (in %).

	Caffe	*H2O*	*Keras*	*Mahout*	*MLlib*	*scikit-learn*	*Tensorflow*	*Theano*	*Torch*	*Weka*	Q1	Q3	IQR	Median	SD
Data adaptation	9.84	35.29	7.90	14.58	25.2	8.64	9.22	10.41	22.95	20.51	9.37	22.3	12.93	12.5	8.70
Featuring	0	5.88	1.09	0	4.20	9.34	0.74	0.52	0	4.27	0.13	4.3	4.17	0.92	3.03
Type mismatch	1.52	0	1.09	0	2.52	2.92	2.02	2.08	0	1.71	0.27	2.07	1.80	1.61	1.02
Shape mismatch	1.52	0	*5.50	0	0	1.86	2.62	2.08	0	0	0	2.03	2.03	0.75	1.70
Data Cleaning	1.52	0	0.55	2.10	2.52	3.62	2.09	1.60	0	3.41	0.79	2.40	1.61	1.82	1.22
Model creation	26.52	17.64	25.88	*43.75	23.52	21.37	23.01	23.43	22.95	21.36	21.77	25.30	3.53	23.22	6.70
Model selection	0	0	0.55	0	0.84	*2.10	0.60	0	0	0	0	0.58	0.58	0	0.64
Model conversion	3.79	0	0.27	0	0.84	0.33	2.25	2.60	4.91	3.41	0.24	3.21	2.97	1.54	1.71
Model load/store	1.50	5.88	1.63	0	5.04	1.75	1.94	1.01	4.91	1.71	1.55	4.20	2.65	1.73	1.88
Error/Exception	0.76	5.88	5.50	4.20	5.88	4.78	5.32	5.72	1.64	2.56	2.96	5.67	2.71	5.10	1.80
Parameter selction	9.10	5.88	5.50	0	2.52	3.97	3.74	2.60	8.20	5.13	2.89	5.78	2.89	4.50	2.60
Loss function	6.10	0	4.09	0	0.84	1.40	3.74	2.60	3.30	1.71	0.98	3.60	2.62	2.16	1.86
Optimizer	2.30	0	1.09	2.10	0	0.70	2.77	1.04	3.30	0.85	0.74	2.20	1.46	1.07	1.07
Performance	2.30	5.88	6.27	2.10	5.04	3.27	4.87	3.12	1.64	0.85	2.13	5.00	2.87	3.20	1.80
Accuracy	3.78	0	2.45	0	0.84	3.62	0.90	1.04	1.64	0.85	0.84	2.30	1.46	0.97	1.30
Eval. strategy selection	0.75	0	2.18	2.08	5.04	3.85	5.24	0	1.64	8.54	0.84	4.7	3.86	1.86	2.64
Visualization	0	0	1.63	0	0	2.68	1.65	0	0	2.68	0	1.23	1.23	0	0.95
Output interpretation	0	*5.88	*3.82	2.1	1.68	2.21	2.17	2.60	1.64	1.71	1.69	2.50	0.81	2.13	1.47
Tuning strategy selection	0.75	*5.88	0.27	0	1.68	3.50	0.45	1.04	0	0	0.07	1.52	1.45	0.60	1.82
Tuning param. selection	0	0	0	0	0	*0.81	0.08	0	0	0	0	0	0	0	0.24
Prediction accuracy	6.10	0	6.81	4.20	5.04	5.25	3.97	2.08	1.64	8.54	2.55	5.85	3.30	4.60	2.44
Model reuse	0	0	*1.37	0	0	0.23	0.22	0	0	*1.71	0	0.23	0.23	0	0.60
Robustness	0	0	1.65	0	0.84	1.28	0.30	0	0	0.85	0	0.85	0.85	0.15	0.59
Non-ML API	2.27	0	2.72	4.20	2.52	2.80	4.40	2.08	3.27	1.71	2.13	3.15	1.02	2.63	1.19
Setup	16.67	5.88	9.53	18.75	2.52	5.95	14.50	30.72	16.39	7.69	6.39	16.60	10.21	12.05	7.90
Custom code	1.52	0	0.81	0	0.84	1.51	1.05	1.56	0	0.85	0.21	1.40	1.19	0.84	0.60
Bug	*1.52	0	0	0	0	*0.23	0.07	0	0	0	0	0.06	0.06	0	0.45

$*$ indicates the library is an outlier for the subcategory in the corresponding row. $IQR = Q3 - Q1$: inter-quartile range. SD: standard deviation.

7) Tensorflow: This library, which is actively maintained by Google, covers the highest percentage of questions in *Stack Overflow*. This library comparatively faces less difficulty in data preparation (16%) but the issues in modeling and training take the higher share (27% and 21%, respectively). If we go deeper into subcategories, we see that model creation (23%), data adaptation (9.22%), error/exception (5.32%), evaluation strategy (5.24%), performance (4.87%) and prediction accuracy (3.97%) are the dominant subcategories. Our conjecture is that **automatic static analysis and verification would be helpful in designing tensor networks**.

8) Theano: Theano is a deep learning library that shows major difficulties in modeling stage (27%). Data preparation (17%) and training (16%) follow next. Among the subcategories, model creation (23.43%), data adaptation (10.41%) and error/exception (5.72%) are the most problematic subcategories. This suggests that, besides tool support for data processing and data adaptation, **runtime checkers/verifiers for *Theano* models** are also needed.

9) Torch: Torch is also a specialized deep learning library. For this library, modeling (33%), data preparation (23%) and training (20%) show the highest levels of difficulties. Among the subcategories, data adaptation (22.95%) and model creation (22.95%) are the most problematic subcategories.

10) Weka: Weka is a popular data mining and ML library for Java. This library has the highest volume of posts about the data preparation (30%), most of which are about data adaptation. This is much higher than the median for data preparation of all the libraries (20%). *Weka* uses a custom data format called ARFF. This would suggest that using a custom data format introduces additional technical difficulty in the data preparation stage like in *MLlib*. So **when a custom data format is used automatic conversion with correctness guarantee should also be provided** to the developers. This can be a direction of research for both software and data engineering researchers. The next dominant stage is modeling (20%). Training stage for *Weka* looks comparatively better: 12% compared to the median among all libraries of 18%.

IV. Inter-Library analysis

In this section, we present our analysis and observations across ML libraries. We study the strengths and weaknesses of different ML libraries in terms of each of the top-level

categories defined in Fig. 2. The distribution of questions over stages of all libraries are shown in Table II and Table III.

A. Data Preparation

1) Data adaptation: As shown in Table III, the percentages of questions about data adaptation across libraries have SD of 8.70% and IQR of 12.93% which indicates a considerable variation across the libraries. We also see that the libraries like *H2O*, *Torch* and *Weka* have respectively 35.29%, 22.95% and 20.51% of questions related to using library APIs to adapt data to pipelines. That shows significant difficulties and would call for research effort to improve this stage. We also see questions about *Weka*'s custom data format to which developers are not familiar. This leads us to the first finding and implication.

Finding 1: *H2O*, *Torch* and *Weka* have 35.29%, 22.95% and 20.51% of posts, respectively, about data adaptation.
Implication: The tradeoff in the design of data preparation APIs, e.g. use of custom formats, needs more study.

2) Featuring: From Table III, percentages of questions related to feature extraction/selection has SD of 3.03%, median of 0.92% and IQR of 4.17%. Among the libraries, users of *scikit-learn* have the most questions of 9.34% even though this library has APIs for auto-selecting features. *H2O*, *Weka* and *MLlib* follow with 5.88%, 4.27% and 4.20%, respectively. This suggests that existing feature extraction/selection support might not yet be sufficient to ease the ML practice. Surprisingly, *Tensorflow* users have comparatively fewer difficulties in data preparation, especially in feature engineering. The reason could be attributed to multiple adapters *Tensorflow* provides to read and store dataset. The dataset is stored as elements with the same size and each element can contain one or more Tensor objects. Each Tensor object also has type and shape information added to it. Using Tensor objects and providing functionality to read raw data and forming dataset using Tensor objects reduces adaptation and data cleaning related errors.

Finding 2: *scikit-learn* has the highest number of posts on feature selection/extraction subcategory (9.34%).
Implication: Blackbox abstraction for feature selection/extraction APIs may not be suitable for ML libraries.

3) Type mismatch: Type mismatch questions have median of 1.61%, SD of 1.02%, and IQR of 1.80%. The smaller IQR indicates that type mismatch appears in most of the libraries. *scikit-learn*, *MLlib*, *Theano* and *Tensorflow* have higher difficulties in type-related problems with 2.92%, 2.52%, 2.08% and 2.02%, respectively. *MLlib* uses a custom data format called RDD that seems to make type-related problems more common to this library. There are also questions about failures due to type mismatch in *scikit-learn*, *Tensorflow* and *Theano* as their APIs have type requirements that are not currently checked. A static analysis tool might be able to prevent the majority of these problems.

TensorFlow ValueError: Cannot feed value of shape (64, 64, 3) for Tensor u'Placeholder:0', which has shape '(?, 64, 64, 3)'

Fig. 4: Question 40430186: An example showing dimension or shape mismatch problem in training in ML.

Finding 3: Type mismatches appear in most ML libraries.
Implication: Type checkers are desirable for ML libraries.

4) Shape mismatch: Shape mismatch related questions have median of 0.75%, SD of 1.70%, and IQR of 2.03%. This problem appears in all deep learning library in which *Keras* is an outlier with 5.50%. In these libraries, shapes of neurons at adjacent layers must be compatible otherwise the library will throw exceptions during training or fail during prediction. Such a problem is shown in Fig. 4.

Abstract APIs that hide the details of inner-working of the deep networks can further complicate matters. To illustrate consider the following *Keras* code.

```
1 def CreateModel(shape):
2   if not shape:
3     raise ValueError('Invalid shape')
4   logging.info('Creating model')
5   model = Sequential()
6   model.add(LSTM(4, input_shape=(31, 3)))
7   model.add(Dense(1))
8   model.compile(loss='mean_squared_error', optimizer='adam')
9   return model
```

The error is at line 6 where an invalid value of $(31, 3)$ is passed to $input\_shape$. The accepted answer suggests that $input\_shape$ should be $(32, 1)$ instead. The user could not verify statically whether the built network is matching in shape or there is any unconnected or extra port while building the model. If we had the tools that could tell the developer that using dimension (32,1) can cause 2 out of 3 ports of the next layer to be unconnected then it would be much easier for the developer to find these errors by themselves. This kind of errors could be detected by program analysis and by providing feedback to the users. In fact, many discussions in these high-scored posts call for richer analysis features.

Finding 4: Shape mismatch problems appear in higher percentage in deep learning libraries. *Keras* is an outlier in this sub category with 5.5% of posts.
Implication: Tool support for verifying shape and dimension compatibility is needed for deep learning libraries. Dependency of data on model architecture needs to be verified and dynamic modification of the network as per data shape may be needed.

5) Data Cleaning: As shown in Table III, data cleaning related questions across the libraries have median of 1.82%, SD of 1.22% and IQR of 1.61%. Most of the libraries have questions about data cleaning stage except for *H2O* and *Torch*. In fact, this is obviously needed step in any data science pipelines. Libraries *scikit-learn*, *Weka* and *MLlib* have the most questions with 3.62%, 3.42% and 2.52%, respectively. The abstract APIs in these libraries sometimes make cleaning fail. For example, the `nan` values in the `dataframe` needs

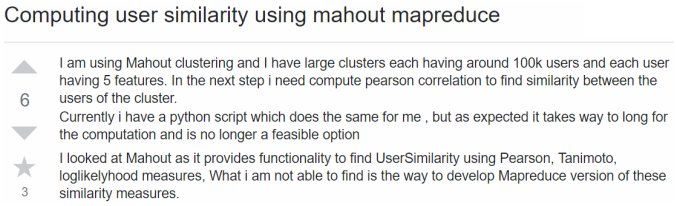


Fig. 5: Question 12319454: An example question on model creation for doing machine learning in cluster using *Mahout*.

to be converted first into numpy `nan` type before they can be cleaned using APIs provided by *scikit-learn*. Furthermore, these failures do not clearly indicate the root cause making diagnostics difficult.

Finding 5: Most libraries have problems in data cleaning.
Implication: Data cleaning needs verification tools to ensure that all required steps in the process are performed.

B. Modeling

This category has the most questions in our study with median of 27%. We now present observations about its subcategories.

1) Model creation: As shown in Table III, questions about model creation has median of 23.22%, SD of 6.70% and IQR of 3.53% which indicates that all libraries have major problems in this step. *Mahout* is an outlier with almost half number of questions about this. Since *Mahout* is used for ML on clusters, this could be attributed to the difficulty in creating models on clusters or doing distributed ML.

Fig. 5 shows an example question asking about creating a MapReduce version of a model in *Mahout*.

Deep learning libraries like *Caffe*, *Keras*, *Theano* and *Tensorflow* also have higher percentages of questions about model creation with 26.52%, 25.88%, 23.43% and 23.01%, respectively. This shows that model creation for deep neural networks is difficult as well.

Finding 6: Model creation is the most challenging (yet critical) in ML pipeline, especially for libraries supporting distributed ML on clusters like *Mahout* and *MLlib*.
Implication: This calls for tool support in creating models, especially in distributed machine learning.

When we study the questions about *Caffe* we see that *Caffe* users have problems in model creation due to the dependency of the model on multiple files. To create a model successfully, one needs to make a schema file in protobuf format, create a solver file and write code in C++ or Python to build the model [2]. Having several components complicates matters. In our study, 36 out of 135 questions about *Caffe* are about model creation problems.

Finding 7: Multilingual form complicates model creation.
Implication: Monolingual model form need to be studied.

2) Model selection: This subcategory has fewer questions than others with median of 0%. *scikit-learn* is an outlier with 2.10%. While *scikit-learn* APIs abstracts the details of underlying ML algorithms, it does not provide enough guidelines on which model would be the best for given data.

Finding 8: *scikit-learn* has abnormally high percentage of questions about model creation compared to other libraries due to its lack of guidelines for model selection.
Implication: Recommendation support would be helpful in selecting models.

3) Model conversion: Converting models trained using one library and using it with another library is a common problem. Some libraries provide support to convert model for use with other libraries but such support is not always adequate. The developers often have questions about reusing trained models in other platforms. Percentages of question in this subcategory have median of 1.54%, SD of 1.71% and IQR of 2.97%. *Torch*, *Caffe*, *Weka* lead this subcategory with 4.91%, 3.79% and 3.41%, respectively. This suggests that interoperability of models across libraries is needed.

4) Model Load/Store: Loading/Storing model has mean of 2.54, SD of 1.87 and IQR of 2.62 across the libraries. *H2O* and *MLlib* leads this category with 5.88% and 5.02% respectively.

C. Training

We now present observations about the training category that has on average 18% questions across libraries.

1) Error/Exception: Error/Exception subcategory has the median of 5.10%, SD of 1.80% and IQR of 2.71%. All the libraries have issues on runtime error/exception. Surprisingly, though model creation seems problematic in *Caffe*, runtime failure is very low in *Caffe* with 0.76%. *MLlib*, *H2O*, *Keras*, *Tensorflow* and *scikit-learn* have higher percentage of runtime errors with 5.88% and 5.88%, 5.50%, 5.32% and 4.78%, respectively. This suggests that debugging and monitoring facilities for ML needs much improvement to help developers resolve error/exception independently.

Finding 9: Questions on exceptions/errors are prevalent.
Implication: Deep learning and distributed ML libraries show more error at training time. This indicates static and dynamic analysis tools are needed for such libraries.

We have observed that the lack of rich and interactive debugging tools makes building ML models difficult. A number of problems faced by developers, e.g. when a model is throwing an exception at training time, a model is not converging or learning as the iteration of training goes on, a model is not predicting well, etc, can be minimized to a great extent with effective debugging tools. Fortunately, some recent work has started to address these issues [11], [31], but much more work is needed. Due to the lack of debugging tools to monitor pipelines causes of failure are hard to identify. More abstract deep learning libraries throw more runtime exception during training, e.g. see Figure 6.

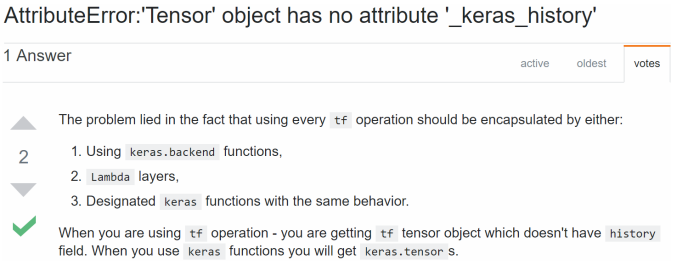


Fig. 6: Question 45030966: An example question about *Keras* showing abstraction in deep learning libraries could lead to misuse and error-prone development.

2) Parameter selection: Questions about parameter selection have median of 4.50%, SD of 2.60% and IQR of 2.89%. This shows a wide variation between libraries. *Caffe* and *Torch* have comparatively more problems with 9.10% and 8.20%, respectively. Libraries like *Keras*, *Weka*, *H2O*, *MLlib* shows larger percentage of questions on choice of parameters.

Finding 10: Parameter selection can be difficult.
Implication: Meta-heuristic strategies can be helpful.

3) Loss function: Choice of Loss function has the median of 2.16%, SD of 1.86% and IQR of 2.62%. This shows that the loss function choice varies across the libraries. All deep learning libraries like *Caffe*, *Keras*, *Tensorflow* and *Torch* have the highest percentages of 6.10%, 4.09%, 3.74% and 3.30%, respectively. This indicates necessity of further research on the usage of loss function in deep learning libraries.

Finding 11: Choice of loss function is more problematic in deep learning libraries.
Implication: This indicates the necessity to develop automatic suggestion algorithms and tools for selecting function for deep learning.

4) Optimizer: Optimizer subcategory has median of 1.07%, SD of 1.07% and IQR of 1.46% across libraries. *Torch*, *Tensorflow*, *Caffe* have respectively 3.30%, 2.77% and 2.30% of questions about optimizer which is higher compared to other libraries. This suggests that choice of optimizer is more relevant for the deep learning libraries.

5) Performance: This subcategory has median of 3.20%, SD of 1.80% and IQR of 2.87%. We see that the distributed ML and deep learning libraries like *Keras*, *H2O*, *MLlib* and *Tensorflow*, have higher percentage of questions related to performance having 6.27%, 5.88%, 5.04% and 4.87%, respectively. Performance related problems here appear due to runtime, memory usage and convergence.

6) Training accuracy: This subcategory has fewer questions than others with median of 0.97%. *Caffe* and *scikit-learn* have the highest share with 3.78% and 3.62%, respectively. *scikit-learn* is a library with highly abstract APIs and large number of optional parameters to be selected. Due to the abstraction, the impact of parameter values on training accuracy could be unclear to developers.

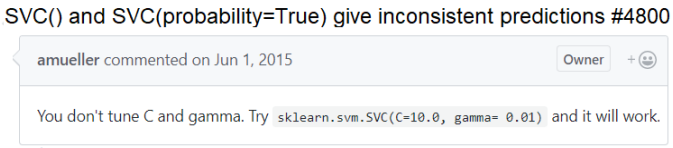


Fig. 7: *scikit-learn* issue #4800: An example of hyperparameter tuning problem. The user filed a GitHub issue that the library probably has a bug. However, a developer of the library responded that the problem was with hyperparameter tuning.

Finding 12: *scikit-learn* and *Caffe* have higher percentage of questions about training time accuracy and convergence (3.62% and 3.78% respectively).
Implication: This indicates the necessity to embed dynamic analysis of learning behavior along with abstract APIs and the development of these analysis tools.

D. Evaluation of models

This category has median of 6.0%, SD of 2.9% and IQR of 4.8%. The visualization and output interpretation subcategories do not have enough data to make meaningful interpretations.

1) Evaluation Strategy: Selection of evaluation strategy has median of 1.86%, SD of 2.64% and IQR of 3.86%. This indicates that this functionality varies more across different libraries. *Weka*, *Tensorflow* and *MLlib* have more questions on evaluation strategy using the APIs having 8.54%, 5.24% and 5.04%, respectively. We believe that this might be because these three libraries provide a rich set of features for this task.

E. Tuning

This category involves questions about selection of hyper parameters and tuning strategies.

1) Tuning strategy selection: This subcategory has median of 0.60%, SD of 1.82% and IQR of 1.45% suggesting a variation. *H2O* is an outline in this subcategory having 5.88% questions. We believe *H2O* APIs for tuning strategy selection are newer and need improvement.

2) Tuning paramater selection: This subcategory has median of 0, SD of 0.24, and IQR of 0 because not too many libraries have questions about it. *scikit-learn* and *Tensorflow* are the outliers in tuning parameter category. This was expected as *scikit-learn* has a lot of optional parameters to be selected and those parameters have effect on the convergence and accuracy.

As an example, consider the API below to create `AdaBoostClassifier` with 5 optional parameters initialized to some default values.

```
class sklearn.ensemble.AdaBoostClassifier(
base_estimator=None, n_estimators=50, learning_rate=1.0,
    algorithm='SAMME.R', random_state=None)
```

The `base_estimator` is set to `None` but the user may need to choose an estimator to get the best performance. Learning rate is by default set to 1.0. At this learning rate, it is highly likely that the model will not learn anything. So the user may often use these APIs incorrectly and wonder why ML model is not producing the useful result. Since these

are optional parameters, the user will not even get any error or warning. Finding good values for these parameters and tuning them to make the best model, avoiding over-fitting are frequent questions among developers using *scikit-learn*. There have been some GitHub issues filed to the repository of *scikit-learn* as bugs (See Fig. 7 for an example) but the underlying problem was that the developer was not able to trace why the model is not showing expected accuracy, and unable to tune hyperparameters. We have found 36 questions out of 849 in *scikit-learn* asking help about hyperparameter tuning.

Finding 13: *scikit-learn* has more difficulty in hyper parameter tuning compared to other libraries
Implication: Due to the presence of a lot of optional parameters in the APIs, the tuning of these parameters is more problematic than other libraries. This gives intuition of having tools to suggest parameters that may have effect on a particular model and particular problem.

F. Prediction

1) Prediction Accuracy: Questions about prediction accuracy across the libraries have median of 4.60%, SD of 2.44% and IQR of 3.30%. Libraries with abstract APIs like *Weka*, *Keras*, *Caffe*, *scikit-learn* and *MLlib* have the highest volume of questions with 8.54%, 6.81%, 6.10%, 5.25% and 5.04%, respectively. Though *Caffe* has fewer questions in model creation, it has higher difficulties in prediction stage.

2) Model reuse: This subcategory has median of 0%, SD of 0.60% and IQR of 0.23%. *Keras* and *Weka* are outliers with 1.37% and 1.71% of questions about model reuse.

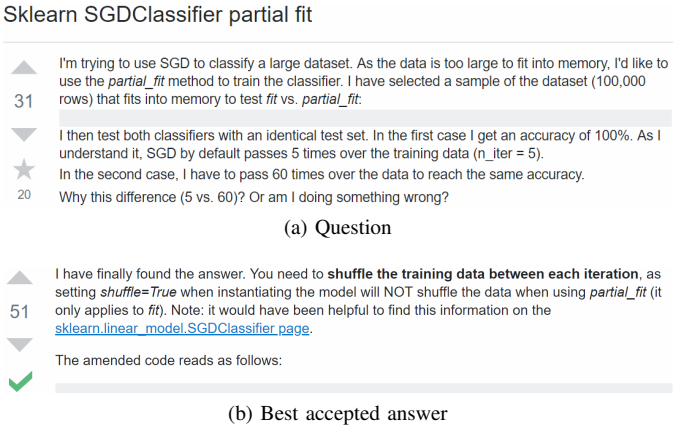
3) Robustness: This subcategory has median of 0.15%, SD of 0.59%, and IQR of 0.85%.

For both model reuse and robustness subcategories, developers are just becoming aware of the possibility and seeking clarifications about it.

G. API Misuses in All the Stages of ML Pipelines

The ML libraries have APIs that are very often misused. API misuse is seen across all the stages of ML pipeline. The general guideline of API misuse happens in the APIs at almost all stages of the ML pipeline.

For example, see Figure 8 where a user is asking that their training takes much time or longer number of iterations to get a certain training accuracy. When they use one API they are able to achieve the desired accuracy in 5 iterations where in the other API they need 60 iterations to reach the same accuracy. The second API works fine, without any error and eventually reaches the same accuracy. But still, the user is puzzled that almost 12 times higher number of iterations are required when using the second API. The answer in Figure 8b suggests that the second API needs the data to be shuffled properly before passing to the API in every iteration. Making that change solves the performance problem. This is an example of API misuse where the precondition of the second API is not satisfied which leads to a performance bottleneck. For another example, let's consider a problem related to the creation of a



(a) Question

(b) Best accepted answer

Fig. 8: Question 24617356: An example showing the API misuse problem in ML libraries. Code snippets are omitted.

NaiveBayes model. Only a part of the code snippet where API misuse occurred is shown below:

```
def convert_to_csr_matrix(vectors):
  logger.info("building the csr_sparse matrix representing tf−idf")
  row = [[i] * len(v) for i, v in enumerate(vectors)]
  row = list(chain(*row))
  column = [j for j, _ in chain(*vectors)]
  data = [d for _, d in chain(*vectors)]
  return csr_matrix((data, (row, column)))
```

The code failed to work successfully giving dimension mismatch error in some parts of the code. The solution to the problem is to properly use the API $csr\_matrix()$. This API needs to have a shape parameter defined explicitly and the correct way to use the API is to explicitly define the shape shown in the code below.

```
return csr_matrix((data, (row, column)), shape=(len(vectors), dimension))
```

We have observed another kind of API misuse due to API update by the library provider. To illustrate, consider the code below that worked well in Apache Spark *MLlib* version $<$ 2.0. For Apache Spark version $>=$ 2.0, this API doesn't work. This is one of the top voted questions on Apache Spark *MLlib* category.

```
1 from pyspark.mllib.clustering import KMeans
2 spark_df = sqlContext.createDataFrame(pandas_df)
3 rdd = spark_df.map(lambda data: Vectors.dense([float(c) for c in data]))
4 mdl = KMeans.train(rdd, 2, maxIterations=10, runs=30, initializationMode="random")
```

Since version 2.0 the code at Line 3 became invalid and the valid API call is the following

```
rdd = spark_df.rdd.map(lambda data: Vectors.dense([float(c) for c in data]))
```

We have found that similar version incompatibility problems are also prevalent in other ML libraries.

Besides, the API misuse scenarios discussed above, many other kinds of API misuse are common in ML libraries, and a more detailed analysis and categorization of errors is needed (much like MUBench [3]). Some common problems include failure to find important features, improperly preparing the dataset, performance, over-fitting problems, suboptimal prediction performance, etc. A detailed analysis of API misuse is beyond the scope of this work.

H. Grouping Libraries

We studied the libraries to see if there is some similarity of patterns among the libraries compared by the distribution of problems at different stages.

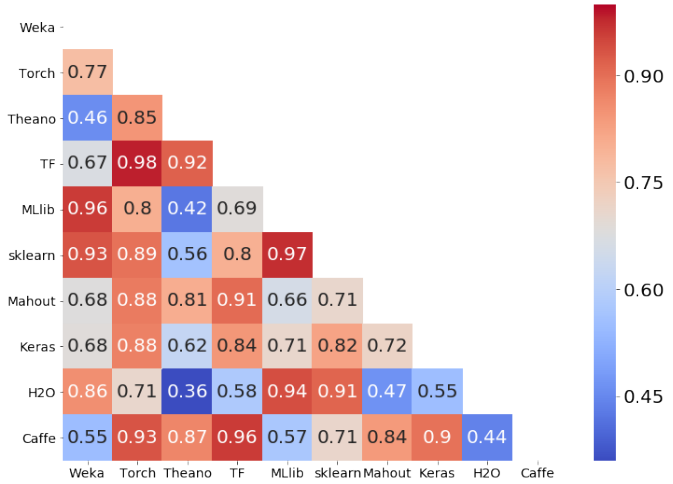


Fig. 9: Correlation between distributions of percentage of questions over stages of the libraries.

From the chart in Fig. 9, we group the libraries into the following categories:

Group 1. *Weka*, *H2O*, *scikit-learn*, and *MLlib* form a strongly correlated group with correlation coefficient greater than 0.84 between the pairs. Other than *H2O*, other libraries in this category are not used for deep learning. This suggests that the problems appearing in these libraries have some correlation and the difficulties of one library can be described by the difficulty of other libraries in the group.

Finding 14: *Weka*, *H2O*, *scikit-learn*, *MLlib* form a strong correlated group with correlation coefficient greater than 0.84 between the pairs.
Implication: Analysis framework for one library can be reused in other library in this group.

Group 2. *Torch*, *Keras*, *Theano*, and *Tensorflow* form another group with strong correlation of more than 0.86 between the pairs. These libraries are specialized for deep learning. This suggests that deep study of one library can also be useful in the analysis for other deep learning libraries. It means that the problems are cross related to the libraries for this category.

Finding 15: Deep learning libraries *Torch*, *Keras*, *Theano* and *Tensorflow* form another group with strong correlation of more than 0.86 between the pairs
Implication: Problems of deep learning libraries currently follow a common pattern which can be leveraged by developing analysis framework for one library and studying how much solution is transferable from one library to another library.

V. Related Work

Stack Overflow is the widely used platform to study the software engineering practice from the developer's perspective. Meldrum *et. al.* [21] studied 266 papers using *Stack Overflow* platforms to show the growing impact of *Stack Overflow* on software engineering research. Treude *et. al.* [32] did a manual labeling of 385 questions to manually classify 385 questions into 10 different categories (how-to, discrepancy, environment, error, decision help, conceptual, review, non-functional, novice, and noise) to identify the question types. This study is useful to learn the general categories of questions asked by developers. Kavaler *et. al.* [17] used *Stack Overflow* data to study the queries on APIs used by Android developers and showed the correlation between APIs used in producing Apps in the market and the questions on APIs asked by developers. Linares-Vásquez *et. al.* [19] studied the effect of the changes in Android API on the developer community. They used the discussions arising on *Stack Overflow* immediately after the API is changed and behavior of the API is modified to study the impact of the change among the developers. Barua *et. al.* [5] studied the *Stack Overflow* posts and used LDA topic modeling to extract topics to study the trend of different topics over time. Rebouças *et. al.* [25] studied the usage pattern of swift programming language among developers using *Stack Overflow* data. Schenk *et. al.* [27] studied the geographical distribution of usage and knowledge of different skills using *Stack Overflow* posts and users data. Stanley *et. al.* [30] proposed a technique based on the Bayesian probabilistic model to predict the tags of a *Stack Overflow* post. McDonnel *et. al.* [20] presented a study of API stability using *Stack Overflow* data and as a test case they used Android Ecosystem. Baltadzhieva *et. al.* [4] proposed a technique to predict the quality of a new *Stack Overflow* question. Joorabchi *et. al.* [16] studied the challenges faced by computer science learners in different topics and subjects using the *Stack Overflow* data. However, existing work has not studied the usage of ML libraries using *Stack Overflow*.

VI. Conclusion

This work is motivated by the need to improve the usage of ML libraries in practice. To understand the problems, we retrieved a significant dataset of Q&A from *Stack Overflow*, classified these questions into categories and subcategories and performed analysis within a library and across libraries. We came across some expected findings, e.g. static and dynamic analysis are needed debugging support is sorely missed, and API misuse is prevalent. Somewhat surprisingly, higher-level of abstractions, at least in their traditional form, seem to complicate matters for ML developers. Initial evidence suggests that visibility into the abstractions is often necessary, and thus opaque abstractions could be problematic, but further study is needed to understand the phenomenon, and to suggest the remedy. Last but not least, model reuse across libraries is needed. We call SE researchers to action to solve these problems as development and refinement of software with ML components is likely to be routine in the next decade.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] Alexandr Honchar. Using Caffe with your own dataset, 2017. https://medium.com/machine-learning-world/using-caffe-with-your-own-dataset-b0ade5d71233.

[3] Sven Amann, Sarah Nadi, Hoan A. Nguyen, Tien N. Nguyen, and Mira Mezini. Mubench: A benchmark for api-misuse detectors. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 464–467, New York, NY, USA, 2016. ACM.

[4] Antoaneta Baltadzhieva and Grzegorz Chrupała. Predicting the quality of questions on stackoverflow. In *Proceedings of the International Conference Recent Advances in Natural Language Processing*, pages 32–40, 2015.

[5] Anton Barua, Stephen W Thomas, and Ahmed E Hassan. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering*, 19(3):619–654, 2014.

[6] James Bergstra, Frédéric Bastien, Olivier Breuleux, Pascal Lamblin, Razvan Pascanu, Olivier Delalleau, Guillaume Desjardins, David Warde-Farley, Ian Goodfellow, Arnaud Bergeron, et al. Theano: Deep learning on gpus with python. In *NIPS 2011, BigLearning Workshop, Granada, Spain*, volume 3. Citeseer, 2011.

[7] Joshua Bloch. How to design a good api and why it matters. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 506–507. ACM, 2006.

[8] Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, and D Sculley. What's your ml test score? a rubric for ml production systems. In *NIPS Workshop on Reliable Machine Learning in the Wild*, 2016.

[9] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[10] Arno Candel, Viraj Parmar, Erin LeDell, and Anisha Arora. Deep learning with h2o. *H2O. ai Inc*, 2016.

[11] Aleksandar Chakarov, Aditya Nori, Sriram Rajamani, Shayak Sen, and Deepak Vijaykeerthy. Debugging machine learning tasks. *arXiv preprint arXiv:1603.07292*, 2016.

[12] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[13] Ronan Collobert, Samy Bengio, and Johnny Mariéthoz. Torch: a modular machine learning software library. Technical report, Idiap, 2002.

[14] Geoffrey Holmes, Andrew Donkin, and Ian H Witten. Weka: A machine learning workbench. In *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*, pages 357–361. IEEE, 1994.

[15] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[16] Arash Joorabchi, Michael English, and Abdulhussain E Mahdi. Text mining stackoverflow: An insight into challenges and subject-related difficulties faced by computer science learners. *Journal of Enterprise Information Management*, 29(2):255–275, 2016.

[17] David Kavaler, Daryl Posnett, Clint Gibler, Hao Chen, Premkumar Devanbu, and Vladimir Filkov. Using and asking: Apis used in the android market and asked about in stackoverflow. In *International Conference on Social Informatics*, pages 405–418. Springer, 2013.

[18] kdnuggets. Top 15 Frameworks for Machine Learning Experts, 2016. https://www.kdnuggets.com/2016/04/top-15-frameworks-machine-learning-experts.html.

[19] Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. How do api changes trigger stack overflow discussions? a study on the android sdk. In *proceedings of the 22nd International Conference on Program Comprehension*, pages 83–94. ACM, 2014.

[20] Tyler McDonnell, Baishakhi Ray, and Miryung Kim. An empirical study of api stability and adoption in the android ecosystem. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*, pages 70–79. IEEE, 2013.

[21] Sarah Meldrum, Sherlock A Licorish, and Bastin Tony Roy Savarimuthu. Crowdsourced knowledge on stack overflow: A systematic mapping study. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, pages 180–185. ACM, 2017.

[22] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016.

[23] Sean Owen and Sean Owen. *Mahout in action*. Manning Shelter Island, NY, 2012.

[24] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[25] Marcel Rebouças, Gustavo Pinto, Felipe Ebert, Weslley Torres, Alexander Serebrenik, and Fernando Castor. An empirical study on the usage of the swift programming language. In *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, volume 1, pages 634–638. IEEE, 2016.

[26] Tirath Prasad Sahu, Naresh Kumar Nagwani, and Shrish Verma. Selecting best answer: An empirical analysis on community question answering sites. *IEEE Access*, 4:4797–4808, 2016.

[27] Dennis Schenk and Mircea Lungu. Geo-locating the knowledge transfer in stackoverflow. In *Proceedings of the 2013 International Workshop on Social Software Engineering*, pages 21–24. ACM, 2013.

[28] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 2503–2511, Cambridge, MA, USA, 2015. MIT Press.

[29] D Sculley, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, and Michael Young. Machine learning: The high-interest credit card of technical debt, 2014.

[30] Clayton Stanley and Michael D Byrne. Predicting tags for stackoverflow posts. In *Proceedings of ICCM*, volume 2013, 2013.

[31] Tensorflow. Debugging TensorFlow Programs, 2016. https://www.tensorflow.org/programmers_guide/debugger.

[32] Christoph Treude, Ohad Barzilay, and Margaret-Anne Storey. How do programmers ask and answer questions on the web?: Nier track. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 804–807. IEEE, 2011.

[33] Shaowei Wang, David Lo, and Lingxiao Jiang. An empirical study on developer interactions in stackoverflow. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1019–1024. ACM, 2013.

[34] Wei Wang and Michael W Godfrey. Detecting api usage obstacles: A study of ios and android developer questions. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 61–64. IEEE Press, 2013.

[35] Jie Yang, Ke Tao, Alessandro Bozzon, and Geert-Jan Houben. Sparrows and owls: Characterisation of expert behaviour in stackoverflow. In *International Conference on User Modeling, Adaptation, and Personalization*, pages 266–277. Springer, 2014.

[36] Yufeng Guo. The 7 Steps of Machine Learning, 2017. https://towardsdatascience.com/the-7-steps-of-machine-learning-2877d7e5548e.

[37] Yang Zhang, Gang Yin, Tao Wang, Yue Yu, and Huaimin Wang. Evaluating bug severity using crowd-based knowledge: An exploratory study. In *Proceedings of the 7th Asia-Pacific Symposium on Internetware*, pages 70–73. ACM, 2015.