SPEAR : Semi-supervised Data Programming in Python

Guttu Sai Abhishek[1*], **Harshad Ingole**[1*], **Parth Laturia**[1*], **Vineeth Dorna**[1*],
Ayush Maheshwari[1*], **Rishabh Iyer**[2], **Ganesh Ramakrishnan**[1]
{gsaiabhishek5, harshad.ingole99, parthlaturia, vineethdorna}@gmail.com,
{ayusham, ganesh}@cse.iitb.ac.in,
rishabh.iyer@utdallas.edu
[1]Indian Institute of Technology Bombay
[2]The University of Texas at Dallas

Abstract

We present SPEAR, an open-source python library for data programming with semi supervision. The package implements several recent data programming approaches including facility to programmatically label and build training data. SPEAR facilitates *weak supervision* in the form of heuristics (or rules) and association of *noisy* labels to the training dataset. These *noisy* labels are aggregated to assign labels to the unlabeled data for downstream tasks. We have implemented several label aggregation approaches that aggregate the *noisy* labels and then train using the *noisily* labeled set in a cascaded manner. Our implementation also includes other approaches that *jointly* aggregate and train the model for text classification tasks. Thus, in our python package, we integrate several cascade and joint data-programming approaches while also providing the facility of data programming by letting the user define labeling functions or rules. The code and tutorial notebooks are available at https://github.com/decile-team/spear. Further, extensive documentation can be found at https://spear-decile.readthedocs.io/. Video tutorials demonstrating the usage of our package are available here. We also present some real-world use cases of SPEAR.

1 Introduction

Supervised machine learning approaches require large amounts of labeled data to train robust machine learning models. For classification tasks such as spam detection, (movie) genre categorization, sequence labelling, and so on, modern machine learning systems rely heavily on human-annotated *gold* labels. Creating labeled data can be a time-consuming and expensive procedure that necessitates a significant amount of human effort. To reduce dependence on human-annotated labels, various techniques such as semi-supervision, distant supervision, and crowdsourcing have been proposed. In order to help reduce the subjectivity and drudgery in the labeling process, several recent data programming approaches (Bach et al., 2019; Chatterjee et al., 2020; Awasthi et al., 2020; Maheshwari et al., 2021) have proposed the use of *human-crafted* labelling functions or automatic LFs (Maheshwari et al., 2022a) to *weakly* associate labels with the training data. Users encode supervision in the form of labelling functions (LFs), which assign noisy labels to unlabeled data, reducing dependence on human labeled data. LFs can defined as first-order logic rules as a composition of semantic role attributes (Sen et al., 2020) or syntactic grammar rules (Sahay et al., 2021).

While most data-programming approaches cited above provide their source code in the public domain, a unified package providing access to all data programming approaches is however missing. In this work, we describe SPEAR, a python package that implements several existing data programming approaches while also providing a platform for integrating and benchmarking newer ones. Inspired by frameworks such as Snorkel (Lison et al., 2021; Ratner et al., 2017; Zhang et al., 2021) and algorithm based labeling in Matlab[1], we provide a facility for users to define LFs. Further, we develop and integrate several recent data programming models that uses these LFs. We provide many easy-to-use jupyter notebooks and video tutorials for helping new users get quickly started. Though we provide implementation on 5 text datasets, our package can be easily integrated with vision and speech datasets as well. The users can get started by installing the package using the below command.

```
pip install decile-spear
```

[*] Authors contributed equally

[1] https://www.mathworks.com/help/vision/ug/create-automation-algorithm-for-labeling.html

Proceedings of the The 2022 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 121 - 127

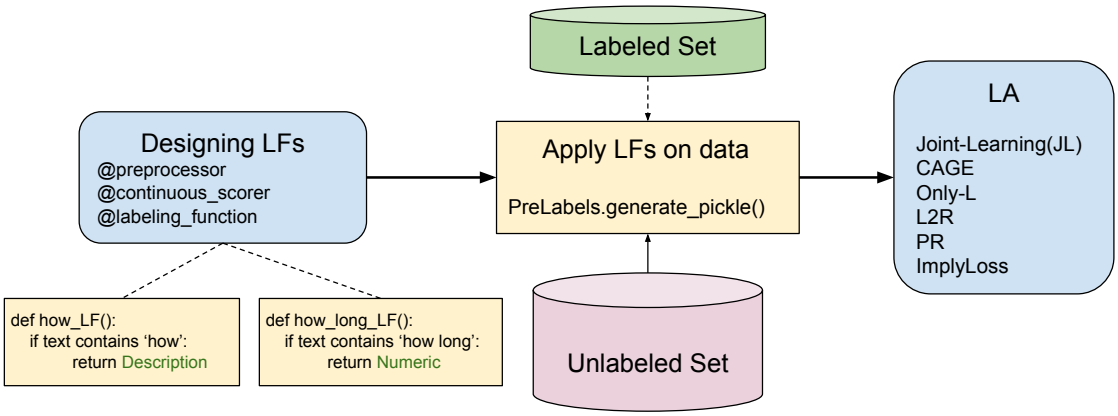

Figure 1: Flow of the SPEAR library.

In Table 1, we compare our library with other existing packages such as Wrench (Zhang et al., 2021), SkWeak(Lison et al., 2021), Imply Loss (Awasthi et al., 2020), Snorkel (Bach et al., 2019) and Matlab. Wrench (Zhang et al., 2021) provides facility for semi-supervised and unsupervised label aggregation approaches, however, it does not provide mechanism to find useful subset of unlabeled data and defining continuous LFs. SkWeak (Lison et al., 2021) does not integrate semi-supervised LA approaches in the package. SPEAR addresses the shortcomings of existing packages by providing features such as designing of discrete and continuous LFs, integrating unsupervised and semi-supervised aggregation approaches and facility to choose labeled set using subset selection approaches.

2 Package Flow

The SPEAR package consists of three components (and they are applied in the same order): (i) Designing LFs, (ii) applying LFs, and (iii) applying a label aggregator (LA).

Initially, the user is expected to declare an *enum* class listing all the class labels. The *enum* class associates the numeric class label with the readable class name. As part of (i), SPEAR provides the facility for manually creating LFs. LFs can be in the form of regex rules as well. Additionally, we also provide the facility of declaring a *@preprocessor* decorator to use an external library such as *spacy*[2], nltk, *etc.* which can be optionally invoked by the LFs. Thereafter, as part of (ii), the LFs can be applied on the unlabeled (and labeled) set using an *apply* function that returns a matrix of dimension #LFs $\times$ #instances. The matrix is then provided as input to the selected label aggregator (LA) in (iii), as shown in Figure1. We integrate several LA options into SPEAR. Each LA aggregates multiple noisy labels (obtained from the LFs) to associate a single class label with an instance. Additionally, we have also implemented in SPEAR, several joint learning approaches that employ semi-supervision and feature information. The high-level flow of the SPEAR library is presented in Figure 1.

3 Designing and Applying LFs

User interacts with the library by designing labeling functions. Similar to Ratner et al. (2017), labeling functions are python functions which take a candidate as an input and either associates class label or abstains. However, continuous LFs returns a continuous score in addition to the class label. These continuous LFs are more natural to program and lead to improved recall (Chatterjee et al., 2020).

3.1 Designing LFs

SPEAR uses a `@labeling_function()` decorator to define a labeling function. Each LF, when applied on an instance, can either return a class label or not return anything, *i.e.* abstain. The LF decorator has an additional argument that accepts a list of preprocessors. Each preprocessor can be either declared as a pre-defined function or can employ external libraries. The pre-processor transforms the data point before applying the labeling function.

```
@labeling_function(cont_scorer, resources,
    preprocessors, label)
def CLF1(x,**kwargs):
```

[2]https://spacy.io

Package	Designing & applying LFs	Continuous LFs	Unsup LA	Semi-sup LA	Labeled-data subset selection
Snorkel(Ratner et al., 2017)	✓	✗	✗	✓	✗
Imply Loss (Awasthi et al., 2020)	✗	✗	✗	✓	✗
Matlab	✓	✗	✗	✗	✗
SkWeak (Lison et al., 2021)	✓	✓	✓	✗	✗
Wrench (Zhang et al., 2021)	✓	✗	✓	✓	✗
SPEAR	✓	✓	✓	✓	✓

Table 1: Comparison of SPEAR against available packages.

```
    return label if kwargs["continuous_score"] >=
        threshold else ABSTAIN
```

The LF can express pattern matching rules in the form of heuristics, distant supervision by using external knowledge bases and other data resources to label datapoints. LFs on SMS dataset can be seen in the example notebook here.

Continuous LFs: In the discrete LFs, users construct heuristic patterns based on dictionary lookups or thresholded distance for the classification tasks. However, the keywords in hand-crafted dictionaries might be incomplete. Chatterjee et al. (2020) proposed a comprehensive alternative that design continuous valued LFs that return scores derived from soft match between words in the sentence and the dictionary.

SPEAR provides the facility to declare continuous LFs, each of which returns the associated label along with a confidence score using the `@continuous_scorer` decorator. The continuous score can be accessed in the LF definition through the keyword argument `continuous_score`. As evident from Table 1, no other existing package provisions for both semi-supervised aggregation and subset selection modules.

```
@continuous_scorer()
def similarity(sentence,**kwargs):
  word_vecs = featurizer(sentence)
  keyword_vecs = featurizer(kwargs["keywords"])
  return similarity(word_vecs,keyword_vecs)
```

3.2 Applying LFs

Once LFs are defined, users can analyse labeling functions by calculating coverage, overlap, conflicts, empirical accuracy for each LF which helps to re-iterate on the process by refining new LFs. The metrics can be visualised within the SPEAR tool, either in the form of a table or graphs as shown in Figure 2.

PreLabels is the master class which encapsulates a set of LFs, the dataset to label and enum of class labels. PreLabels facilitates the process of applying the LFs on the dataset, and of analysing and refining the LF set. We provide functions to store labels assigned by LFs and associated metadata such as mapping of class name to numeric class labels on the disk in the form json file(s). The pre-labeling performed using the LFs can be consolidated into labeling performed using several consensus models described in Section 4.

```
sms_pre_labels = PreLabels(name="sms",
    data=X_V, gold_labels=Y_V,
data_feats=X_feats_V, rules=rules,
    labels_enum=ClassLabels, num_classes=2)
```

4 Models

We implement several data-programming approaches in this demonstration that includes simple baselines such as fully-supervised, semi-supervised and unsupervised approaches.

4.1 Joint Learning (Maheshwari et al., 2021)

The joint learning (JL) module implements a semi-supervised data programming paradigm that learns a joint model over LFs and features. JL has two key components, *viz.*, feature model (fm) and graphical model (gm) and their sum is used as a training objective. During training, the JL requires labeled ($\mathcal{L}$), validation ($\mathcal{V}$), test ($\mathcal{T}$) sets consisting of true labels and an unlabeled ($\mathcal{U}$) set whose true labels are to be inferred. The model API closely

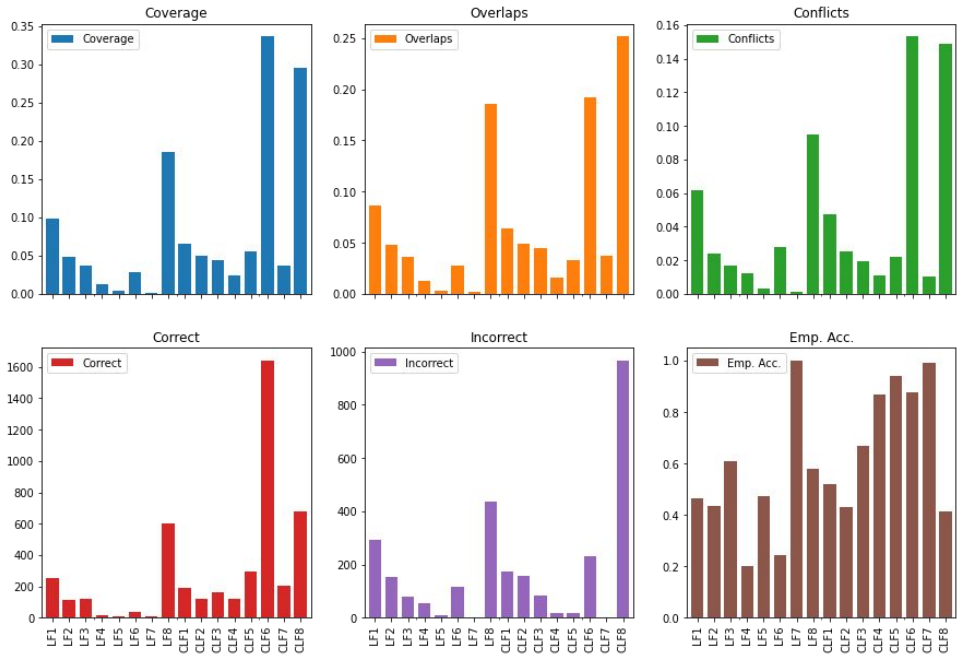

Figure 2: LF analysis on the SMS dataset presented in the form of graph visualization within the SPEAR tool. The statistics include precision, coverage, conflict and empirical accuracy for each LF.

follows that of *scikit-learn* (Pedregosa et al., 2011) to make the package easily accessible to the machine learning audience. The primary functions are: (1) `fit_and_predict_proba`, which trains using the prelabels assigned by LFs and true labels of $\mathcal{L}$ data and predicts the probabilities of labels for each instance of $\mathcal{U}$ data (2) `fit_and_predict`, similar to the previous one but which predicts labels of $\mathcal{U}$ using maximum posterior probabilities (3) `predict_(fm/gm)_proba`, predicts the probabilities, using feature model(fm)/graphical model(gm) (4) `predict_(fm/gm)`, predicts labels using fm/gm based on learned parameters. We also provide functions `save` or `load_params` to save or load the trained parameters.

As another unique feature (*c.f.* Table 1), our library supports a *subset-selection framework* that makes the best use of human-annotation efforts. The $\mathcal{L}$ set can be chosen using submodular functions such as facility location, max cover, *etc.* We utilise the submodlib[3] library for the subset selection algorithms. Some of the function alternatives for subset selection are `rand_subset`, `unsup_subset`, `sup_subset_indices` and `sup_subset_save_files`.

[3] https://github.com/decile-team/submodlib

4.2 Only-$\mathcal{L}$

In this, the classifier $P(y|\mathbf{x})$ is trained only on the labeled data. Following Maheshwari et al. (2021), we provide facility to use either Logistic Regression or a 2-layered neural network. Our package is flexible to allow other architectures to be plugged-in as well.

4.3 CAGE (Chatterjee et al., 2020)

This accepts both continuous and discrete LFs. Further, each LF has an associated quality guide component, that refers to the fraction of times the LF predicts the correct label; this stabilises training in absence of $\mathcal{V}$ set. In our package, CAGE accepts $\mathcal{U}$ and $\mathcal{T}$ sets during training. CAGE has member functions similar to (except there are no fm or gm variants to `predict_proba`, `predict` functions in Cage) JL module, with different arguments, serving the same purpose. It should be noted that this model doesn't need labeled($\mathcal{L}$) or validation($\mathcal{V}$) data.

4.4 Learning to Reweight (L2R) (Ren et al., 2018)

This method is an online meta-learning approach for reweighting training examples with a mix of $\mathcal{U}$ and $\mathcal{L}$. It leverages validation set to determine and adaptively assigns importance weights to examples based on the gradient direction. This does

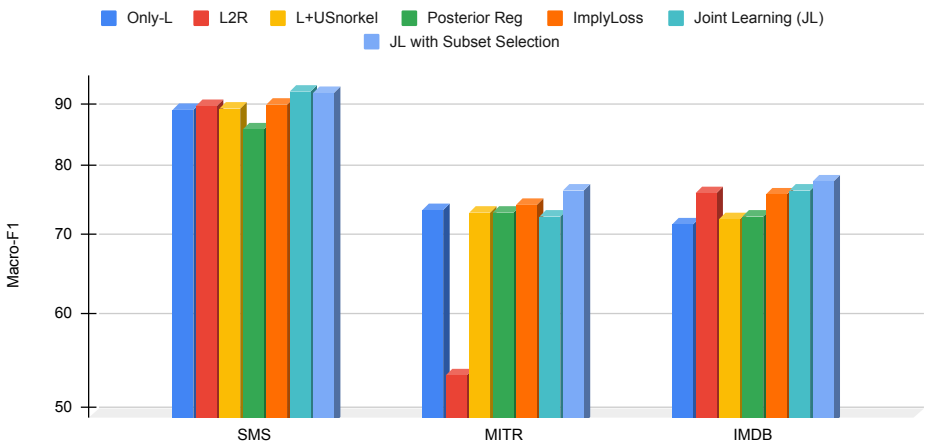

Figure 3: Experiments on SMS, IMDB and MIT-R dataset and comparison with various approaches. We use JL combined with supervised subset selection for obtaining numbers.

not employ additional parameters to weigh or denoise individual rules.

4.5 $\mathcal{L} + \mathcal{U}_{\text{Snorkel}}$ (Ratner et al., 2017)

This method trains a supervised classifier on $\mathcal{L}$ set and Snorkel's generative model on $\mathcal{U}$ set. Snorkel is a generative model that models class probabilities based on discrete LFs for consensus on the noisy and conflicting labels. It assigns a linear weight to each rule based on an agreement objective and label examples in $\mathcal{U}$.

4.6 Posterior Regularization (PR) (Hu et al., 2016)

This is a method that enables to simultaneously learn from $\mathcal{L}$ and logic rules by jointly learning a rule and feature network in a teacher-student setup. The student network learns parameter θ using the $\mathcal{L}$ set and teacher networks attempts to imitates the student network in a joint learning manner. The teacher network encodes logic rules as a regularization term in the overall loss objective.

4.7 Imply Loss (Awasthi et al., 2020)

This approach uses additional information in the form of labeled rule exemplars and trains with a denoised rule-label loss. They leverage both rules and labeled data by mapping each rule with exemplars of correct firings (i.e., instantiations) of that rule. Their joint training algorithms denoise overgeneralized rules and train a classification model. It has two main components:

1. Rule Network: It learns to predict whether a given rule has overgeneralized on a given sample using latent coverage variables.

2. Classification Network: It is trained on $\mathcal{L}$ and $\mathcal{U}$ to predict the output label and maximize the accuracy on unseen test instances using a soft implication loss.

This module contains the following primary classes:

1. DataFeeder - It will essentially take all the parameters as input and create a data feeder class with all these parameters as its attributes.

2. HighLevelSupervisionNetwork (HLS) - It will take the 2 networks, the mode or the approach that needs to be used to train the model, the required parameters, the directory storing model checkpoints at different instances and the instances and labels from the labeled dataset ($\mathcal{L}$) and create an object named "hls".

HLS object will have many member functions of which the 2 significant are:
(a) **hls.train**: This function, when called with the required mode, will train the 2 network attributes of the object.
(b) **hls.test**: It supports 3 types of testing:
(i) test_w: this will test the rule network and the related model of the object.
(ii) test_f: this will test the classification network and the related model of the object.
(iii) test_all: this will test both the networks and models of the class.

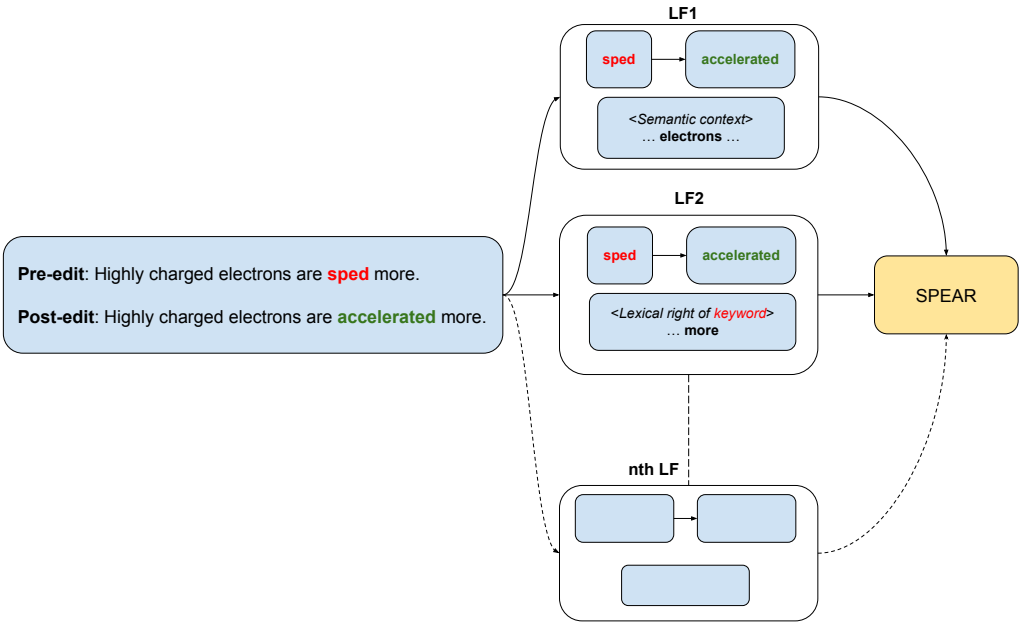

Figure 4: Mutiple LFs generated from post-editor edits based on semantic and lexical features while editing science (domain-specific) document in English.

5 Experiments

We prepared jupyter tutorial notebooks for two standard text classification datasets, namely SMS, YouTube and TREC. We took LFs on these datasets from Awasthi et al. (2020) and train using approaches implemented in this paper. Figure 3 shows performance of various approaches implemented using our package on additional two datasets, MIT-R and IMDB. We can integrate image classification tasks by defining appropriate feature extraction module and rules.

6 Use Cases

SPEAR is employed in project UDAAN[4] for reducing post editing efforts. UDAAN (Maheshwari et al., 2022b) is an end-to-end translation and post-editing eco-system for domain-aware, target vocabulary-constrained translation. Specifically, based on the post editor's patterns of changes to the target language document, candidate labeling functions are generated (based on a combination of heuristics and linguistic patterns) by the UDAAN workbench (*c.f.* Figure 4 for examples of LFs). SPEAR is then used to invoke these LFs on a combination of the edited (*i.e.*, labeled) data and the not yet edited (*i.e.*, unlabeled) data to present consolidated edits to the post-editor. This use case has been presented in the flow chart in Figure 4 – we present the appropriate incorporation of SPEAR into the post-editing environment of an ecosystem such as for translation (UDAAN) or even for Optical Character Recognition[5] or Automatic Speech Recognition (ASR).

As a part of the COVID-19 third wave preparedness, SPEAR was deployed for the Municipal Corporation of Greater Mumbai (MCGM)'s Health Ward[6] for predicting the COVID-19 status of patients, to help in preliminary diagnosis.

6.1 Demonstration Case

For the purpose of demonstration, apart from the use cases outlined above, we can choose a text classification dataset and form regex or continuous rules after observing a few data points. Once the LFs are developed, they can be deployed in conjunction with any of the semi- and un-supervised algorithms present in the package (*c.f.* Section 4) and to compare these algorithms against each other.

7 Conclusion and Future Work

SPEAR is a unified package for semi-supervised data programming that enables quick annotation of training data and facilitates training of machine learning models. It eases the use of developing

[4] https://www.udaanproject.org/

[5] https://www.cse.iitb.ac.in/~ocr/

[6] https://colab.research.google.com/drive/1tNUObqSDypUos7YNvnqvemALlkrrsB0z

LFs and label aggregation approaches. This allows for better reproducibility, benchmarking and easier ML development in low-resource settings such as in textual post-editing. Presently, we are integrating automatic LF induction approaches such as Snuba (Varma and Ré, 2018) that employ a small labeled set to induce LFs automatically. This will significantly increase the scope of labeling datasets, reducing the extent of human intervention in designing LFs. The package is written in Python3 and open-sourced with a MIT License[7], open for community contribution.

8 Acknowledgements

We thank anonymous reviewers for providing constructive feedback. Ayush Maheshwari is supported by a Fellowship from Ekal Foundation (www.ekal.org). Ganesh Ramakrishnan is grateful to IBM Research, India (specifically the IBM AI Horizon Networks - IIT Bombay initiative) as well as the IIT Bombay Institute Chair Professorship for their support and sponsorship.

References

Abhijeet Awasthi, Sabyasachi Ghosh, Rasna Goyal, and Sunita Sarawagi. 2020. Learning from rules generalizing labeled exemplars. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Stephen H Bach, Daniel Rodriguez, Yintao Liu, Chong Luo, Haidong Shao, Cassandra Xia, Souvik Sen, Alex Ratner, Braden Hancock, Houman Alborzi, et al. 2019. Snorkel drybell: A case study in deploying weak supervision at industrial scale. In *Proceedings of the 2019 International Conference on Management of Data*, pages 362–375.

Oishik Chatterjee, Ganesh Ramakrishnan, and Sunita Sarawagi. 2020. Robust data programming with precision-guided labeling functions. In *AAAI*.

Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. 2016. Harnessing deep neural networks with logic rules. *arXiv preprint arXiv:1603.06318*.

Pierre Lison, Jeremy Barnes, and Aliaksandr Hubin. 2021. skweak: Weak supervision made easy for NLP. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 337–346, Online. Association for Computational Linguistics.

[7] https://opensource.org/licenses/MIT

Ayush Maheshwari, Oishik Chatterjee, KrishnaTeja Killamsetty, Rishabh K. Iyer, and Ganesh Ramakrishnan. 2021. Data programming using semi-supervision and subset selection. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*.

Ayush Maheshwari, Krishnateja Killamsetty, Ganesh Ramakrishnan, Rishabh Iyer, Marina Danilevsky, and Lucian Popa. 2022a. Learning to robustly aggregate labeling functions for semi-supervised data programming. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1188–1202.

Ayush Maheshwari, Ajay Ravindran, Venkatapathy Subramanian, Akshay Jalan, and Ganesh Ramakrishnan. 2022b. Udaan – machine learning based post-editing tool for document translation.

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

Alexander J Ratner, Stephen H Bach, Henry R Ehrenberg, and Chris Ré. 2017. Snorkel: Fast training set generation for information extraction. In *Proceedings of the 2017 ACM international conference on management of data*, pages 1683–1686.

Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. 2018. Learning to reweight examples for robust deep learning. In *International Conference on Machine Learning*, pages 4334–4343.

Atul Sahay, Anshul Nasery, Ayush Maheshwari, Ganesh Ramakrishnan, and Rishabh Iyer. 2021. Rule augmented unsupervised constituency parsing. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4923–4932.

Prithviraj Sen, Marina Danilevsky, Yunyao Li, Siddhartha Brahma, Matthias Boehm, Laura Chiticariu, and Rajasekar Krishnamurthy. 2020. Learning explainable linguistic expressions with neural inductive logic programming for sentence classification. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4211–4221.

Paroma Varma and Christopher Ré. 2018. Snuba: automating weak supervision to label training data. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, volume 12, page 223. NIH Public Access.

Jieyu Zhang, Yue Yu, Yinghao Li, Yujing Wang, Yaming Yang, Mao Yang, and Alexander Ratner. 2021. Wrench: A comprehensive benchmark for weak supervision. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.