MULTIMEDIA COMMUNICATIONS TECHNICAL COMMITTEE
http://www.comsoc.org/~mmc

MMTC Communications - Frontiers

Vol. 17, No. 3, May 2022

CONTENTS

SPECIAL ISSUE ON Multimedia Data Mining Approaches and Applications

Guest Editor: Guangchi Liu
Stratifyd, Inc., U.S.A
luke.liu@stratifyd.com

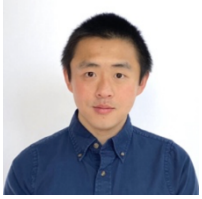
This special issue of Frontiers focuses on multimedia data mining approaches and applications. The research topics of the papers in this special issue include: 1) a comprehensive practice and analysis of FlyingSquid, a popular labeling framework for weakly supervision learning. 2) a comprehensive analysis of recent approaches on multi-aspect sentiment analysis, 3) a practice of how to conduct natural language model compression for efficient nature language processing (NLP) model inference, and 4) a overview of code2vec model in educational data mining.

The first paper studies the performance of FlyingSquid, a extensively used labeling framework aims at providing fast and accurate machine labeled training samples instead of human labelers. As a powerful labeling framework, FlayingSquid is currently extensively used in industry as an alternative to human labeling pipelines such as mechanical turkey. In this paper, the authors detail the performance of FLyingSquid upon various public dataset. In addition, the limitations and reasons behind that are also discussed.

The second paper implemented a weekly-supervised multiaspect sentiment analysis framework based upon target-opinion word pair extraction, target-aspect matching, and a variational sentiment classification model following the **VWS** methodology. The authors also proposed a modified version with shared representation layers and joint training, which achieved similar performance in terms of accuracy but with significantly faster time complexity. Based upon the results, a comprehensive analysis and future improvement discussion are also presented

The third paper introduce joint training and compressing of BERT model, an extensively used language model in modern natural language processing applications. While pretrained language model such as BERT brings significant model performance increase in various natural language analysis tasks, the inference speed makes it hardly be used in industrial applications. Hence, model compression plays as an important role in bringing BERT model into real-world applications. In this paper, a practice of how to compress BERT model with knowledge distilling techniques is introduced.

The fourth paper introduces an overview of the usage of code2vec model for computer science educational tasks. The authors discussed three recent publications about: 1) how the embeddings generated from the code2vec model be used for clustering student misconceptions; 2) how code2vec could use semi-supervised learning to use limited labels in a dataset for student bug detection; and 3) how code2vec can help the deep knowledge tracing model learn the structural information from student programming code for performance predictions. These papers show that in the programming education domain, the usage of student code in a model can help the performance of difficult tasks. This could also generate to other domains, that domain-specific information could be leveraged for other educational datasets as well.



Guangchi Liu is currently the Senior Director of Research and Development, Stratifyd, Inc., Charlotte, NC, USA. He received his Ph.D. in Computer Science from Montana State University, USA. His research interests include Social Computing and Security, Machine Learning, Natural Language Processing, Trust Modeling, etc. As an academia researcher, he has published 9 journal papers, 9 conference papers and 1 book chapter. As an industrial research lead, he has received 1 U.S. patent and several multimedia data analysis systems for commercial use.

Towards Fast Labeling for Supervised Learning: A Practice of FlyingSquid

Jiayu Wu
Amazon, Inc.
jiayuwu@ucla.edu

1. Introduction

FlyingSquid[1] is a novel weak supervision framework that makes exact inference from noisy, incomplete, multi-source 'weak labels' by modeling the ground truth label as latent variable in a probabilistic graphical model (PGM)[4]. The open-sourced python implementation are available on github. This report discusses how we may benefit from the FlyingSquid framework, in terms of potential use cases where the human labeling resources are at shortage and limitations based on empirical observations as well as theoretical analysis. Moreover, the current open-sourced implementation (as of April 2020) could use modifications to be more applicable to large-scale inference on complex model structures.

FlyingSquid is a **ground truth label (Y)** generation model accounting for the noise and abstention in the **observed label sources (Lambda)** by exploiting the dependency structure among variables. Here **Y** and **Lambda** are both variables that can be either univariate or multivariate, while Y is a latent variable and Lambda observed. We denote each dimension of them using the lower case **y**'s and **lambda**'s that are **vertices** in PGM, and undirected **edges** indicates correlations between vertices. FlyingSquid sought for exact inference with a Binary Ising Model on the graph with the latent variable Y and the transformed observed variable Lambda, without "training phase", e.g., gradient descent or MCMC sampling in conventional ML model[3].

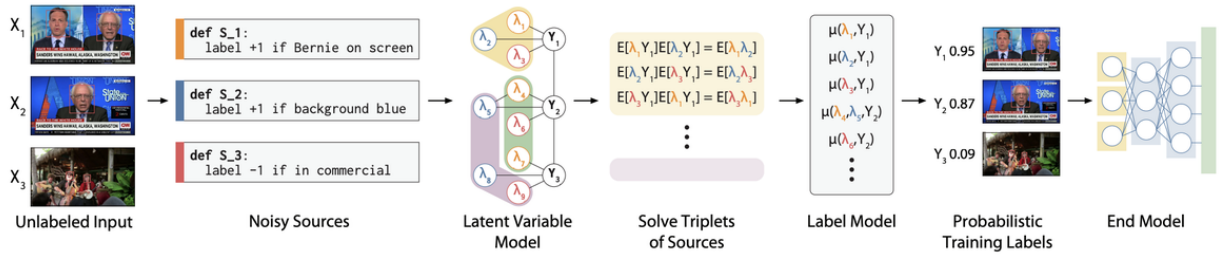


Figure 1. The FLYINGSQUID pipeline. Users provide weak supervision sources, which generate noisy labels for a set of unlabeled data. FLYINGSQUID uses a latent variable model and constructs triplets of sources to turn model parameter estimation into a set of minimal subproblems with closed-form solutions. The label model then generates probabilistic training labels to train a downstream end model.

Figure 1. FLYINGSQUID pipeline, which is quoted from [1]

The figure above is similar to its predecessor Snorkel[2], while it has the key advantages that:

1. **It is fast and theoretically bounded due to the use of exact inference instead of empirical approximation.**
2. **It is able to handle temporally or spatially correlated training samples with higher efficiency.**

The first advantage makes it promising in practice to improve existing rule-based labeling methods, including taxonomy, lexicon sentiments which are extensively used in industry, with little computation overhead, as FlyingSquid can be hundreds of times faster than the previous works. The original paper highlighted the potential of the second feature in sequence labeling (video), while competing methods have to work sequentially and disregard correlation between labeling functions. In our use case, it is not only applicable on **sentence/audio labeling**, but also has the potential to address **multi-label** problems with more complex dependency structures. In general, for learning with finite discrete variables, FlyingSquid provides a fresh idea to further simplify the question by converting to correlated binary variables, and apply triplet method for PGM inference.

2. Implementation

In this section, we introduce the practice of FlyingSquid and observations from the practices, including limitations, performances and insights in various downstream use cases.

2.1 General Limitations:

A major limitation of the current implementation is that it uses intensive space, and hence becomes slow or crashes due to memory errors.

First of all, the **number of vertices** in the graph (k), including latent variable y and observed variable lambda, should be less than 20. Since the joint distribution stores a probability for each possible combination of all variables, the space complexity scales exponentially as $O(2^k)$

Moreover, the **number of edges** should also be limited. The size of the maximal cliques increase along with the connectivity of the graph, which makes the linear transformation from estimated moments to mean parameters tremendously slower (see Algorithm 2 in the paper). On the other hand, the junction tree algorithm as part of the program fails (Value Error: no sepset found between these two edges) when the graph is not connected. It happens when we model multiple label variable Y and not all of them are connected. Therefore, in cases we have independent labels Y, it is better to fit a model for each label separately.

In the inference stage, we use an array with size $n \times (2^v)$ where n is the number of datapoints and v the number of labels, which scales exponentially with the number of labels. As a result, the inference on multiple labels can be slow if not crashed. Therefore, we rewrite an alternative version of predict_proba_marginalized that does batch inference without duplicate computations.

2.2 General Performance

One Label - Multiple Labeling Functions

We start with the case where we have a univariate label to predict. The following is the result on synthetic data. We synthesize the binary ground truth label with balanced class distribution, the labeling functions are sampled with accuracies varying between 50% to 95% and abstain rate between 47% to 90%, and the false prediction and abstaintion distribute evenly on positive and negative samples.

As shown in Table 1, we report training and inference time on 100000 datapoints, and accuracy on a development set with 500 data points, compared with the accuracy of MajotityVote and AnyMatch. AnyMatch predict positive if any labeling function vote positive, which is the labeling strategy currently used by taxonomy model, an extensively used rule-based labeling strategy in industry.

#data = 100000 independent LFs	**#LF = 5**	**#LF = 10**	**#LF = 15**	**#LF = 20**
FlyingSquid Accuracy	0.69	0.73	0.87	0.85
MajorityVote Accuracy	0.67	0.72	0.76	0.77
AnyMatch Accuracy	0.57	0.51	0.49	0.48
training time (s)	0.013	0.028	0.051	0.06
inference time (s)	5.692	11.539	18.176	24.209

Table 1

We observe that FlyingSquid **outperforms baselines consistently**, and in general more labeling functions improves the results. We also compare the time budget with different data size and correlated labeling functions (correlation assigned randomly), as shown in Table 2.

	#data = 100000	**#data = 500000**	**#data = 1000000**	
FlyingSquid Accuracy	0.87	0.84	0.85	
MajorityVote Accuracy	0.76	0.74	0.78	
AnyMatch Accuracy	0.49	0.5	0.51	
training time (s)	0.051	0.216	0.442	
inference time (s)	18.176	99.746	184.448	
#LF = 15 #data = 100000	**#edge = 0**	**#edge = 5**	**#edge = 10**	**#edge = 15**
training time (s)	0.051	53.154	126.788	210.691
inference time (s)	18.176	16.13	13.135	14.189

Table 2

Class Balance

In the previous toy example, we assume the ideal symmetric distribution of positives and negatives. In this section two alternatives are compared, and the prior is estimated by majority vote. As shown in Table 3, it indicates the importance of a correctly specified prior on Y, and Flyingsquid shows more advantages with unbalanced class distribution.

#LF = 5 #data = 100000	Balanced	90% postive			10% positive		
		assume balance prior	set prior (0.1, 0.9)	estimate prior	assume balance prior	set prior (0.9, 0.1)	estimate prior
FlyingSquid Accuracy	0.69	0.63	0.9	0.92	0.65	0.9	0.89
MajorityVote Accuracy	0.67			0.54			0.71
AnyMatch Accuracy	0.57			0.74			0.43

Table 3

Multiple (Correlated) Labels - One Labeling Function Each

We also test the model's potential on exploring Label Dependency by constructing the model with multiple labels (multivariate Y) and each with one labeling function. The real dataset used has 15 labels and the label dependency are estimated by co-occurrence. During training, we get maximal cliques consists of lambda-y pair or cliques with all labels. As we discussed in the Limitation section, the inference on the whole dataset with more than 70000 observations is infeasible, and it takes 1485 seconds to finish even with the re-implemented batch-inference. Moreover, only one labeling function for each label results in the predicted hard labels exactly the same as the input labeling function results, and the marginalized probabilities are always .5 for abstain and fixed probability otherwise on each label.

2.3 Usage Limitations (Taxonomy Label)

For Taxonomy Labels, we formulate the model such that the parent nodes are the labels (y) to be inferred, and corresponding child labels generated by the taxonomy rules are viewed as labeling functions (lambda). A separate label model is fit for each parent node y as discussed before.

The key characteristic of the taxonomy labeling is that it gives **no negative** prediction, but a **small proportion of precise positive labels** and abstains. It follows:

1. **no disputes** between labeling functions as there is no negative prediction whatsoever;
2. **high abstain** rate (>~97%), many datapoints are labeled by none of the labeling functions;
3. severely **imbalanced class** distribution as the labels are specific and only a small proportion are matched;
4. **higher precision** close to 1 (few false positive) and **relative large amount of false abstain** the rules fail to cover (low 'recall' if we see abstain as 'negative').

Because of those four features, there are three reasons that the capability of FlyingSquid is limited for Taxonomy Labels.

First of all, FlyingSquid is more effective in handling dispute between different labeling function than uncover new patterns. Intuitively, the Label Model learns from the differences between labeling functions to correct each of them and combine into a better guess. However, it takes nothing but the labeling function results as input, hence unable to generalize beyond the predefined labeling patterns. Consider an example where the ground truth is positive yet none of the labeling functions is able to predict, the model then only sees abstains and has nothing to infer with. Therefore, the first two features of taxonomy labels means the label model itself is not able to uncover more labels, and an End Model (ex. semi-auto) is required for better completion rate.

Secondly, the third feature suggests a prior need to be estimated and specified (see the toy example in the previous section for class balance). In the experiments, if we assume the default prior (P(positive)=0.5), the final positive prediction will be the same as AnyMatch. It is plausible since 0.5 is far greater than both the actual P(positive) and the proportion of datapoints with any positive label, and the model has to go aggressive to meet the assumption. Note that even in this case data points where all labeling function abstain are not predicted positive, which is another evidence to the no generalization argument.

The last question left, is then whether FlyingSquid improves the existing labeling functions, given high precision and high false abstention. It is not a trivial question to answer given no ground truth are available. To start with an extreme scenario, we assume all the labeling function are confident about their positive prediction (precision=1), then it is impossible for FlyingSquid to outperform AnyMatch because the former makes more

conservative prediction while AnyMatch don't care about false abstention. To prove that take one datapoint with five labeling functions as an example: if the votes are [0, 0, 0, 0, 0], AnyMatch and Flyingsquid both predict abstain; if the votes are [+1, 0, 0, 0, 0], AnyMatch makes it correct if any labeling function predicts positive, while FlyingSquid has lower confidence for positive due to the false abstention. The predicted probability should still be greater than 0.5 and positive if we assume balanced class, but assuming a prior P(positive)<<0.5 the model prones to predict negative in lower confidence cases. Note that even if the prior is set correctly, the predicted proportion could differ from the prior because the space is discrete and finite, i.e., fixed probability for each combination of the system {lambda_1, lambda_2, lambda_3, lambda_4, lambda_5, y}.

In short, an aggressive labeling method like AnyMatch is good enough when precision is close to perfect, so it is crucial to know FlyingSquid is meaningful on what precision level. We experiment on 100,000 synthetic data with five labeling function, and a rather high proportion of negative samples (small *P(positive)*). For fair comparison, we sample the labeling functions such that all of them has the same error distribution specified by True Positive(tp), False Positive(fp), False Abstain(fa). Note that 1) precision=1 if and only if fp=0; 2) a genuine recall is not defined for taxonomy labels (no false negative), whereas we are still concerned about false abstain.

P(positive)=0.05, (tp, fa)=(0.04, 0.01)								
precision	1	0.96	0.95	0.9	0.8	0.74	0.51	0.3
fp	0	0.0016	0.0024	0.005	0.0095	0.015	0.04	0.095
FlyingSquid Accuracy	0.99	0.99	0.99	0.99	0.99	0.99	0.98	0.92
AnyMatch Accuracy	0.99	0.99	0.98	0.97	0.95	0.92	0.82	0.61
P(positive)=0.05, precision=0.8								
fp	0.01125		0.0095		0.0075		0.00625	
(tp, fa)	(0.045, 0.005)		(0.04, 0.01)		(0.03, 0.02)		(0.025, 0.025)	
FlyingSquid Accuracy	0.99		0.99		0.99		0.98	
AnyMatch Accuracy	0.94		0.95		0.96		0.96	
P(positive)=0.1, fp=0.009								
precision	0.9		0.85		0.68		0.52	
(tp, fa)	(0.09, 0.01)		(0.05, 0.05)		(0.02, 0.08)		(0.01, 0.09)	
FlyingSquid Accuracy	0.95		0.95		0.92		0.9	
AnyMatch Accuracy	0.95		0.95		0.92		0.89	
P(positive)=0.1, fp=0.09								
precision	0.5		0.36		0.18		0.1	
(tp, fa)	(0.09, 0.01)		(0.05, 0.05)		(0.02, 0.08)		(0.01, 0.09)	
FlyingSquid Accuracy	0.92		0.9		0.85		0.83	
AnyMatch Accuracy	0.63		0.62		0.59		0.57	

Table 4

As shown in Table 4, these examples demonstrate the obvious effect of the proportion of false positives on the relative performance. Flyingsquid outperforms AnyMatch baseline only when there are a few false positives in the taxonomy label, and the more false positives the more relative improvements Flyingsquid makes, no matter we fix the precision or fix the false abstain.

By fixing the precision, we also show that FlysingSquid outperforms AnyMatch even when the majority (90%) of actual positive labels are not matched. With a fixed small number of false positives, Flyingsquid performs better with higher false abstain, presumably because the specified class prior encourages it to be more confident about labeling functions meanwhile AnyMatch is highly sensitive to false positive. It is also notable that, the above comparison are based on relative performance, in terms of absolute accuracy we still preferred label sources with higher precision, less false positive and lower false abstain. However, when the sources are contaminated Flyingsquid shows great potential over the baselines.

Conclusion

In conclusion, we introduced the FlyingSquid framework, implementation and use cases, and discussed the following limitations:

1. The implementation of PGM use intensive space, hence not suitable for high-dimensional label;
2. FlySquid does not generalize beyond the available labeling sources, therefore it is useful for handling disputes between different sources rather than enlarge the labeled set;
3. FlySquid has the key advantage that it is less sensitive to the contaminated label sources than naive baselines, and it is only meaningful when the labeling source are contaminated (in contrast to high precision yet with high abstain).

Although it is not suitable for current taxonomy rules featuring only positive prediction, high precision and high abstention, Flysquid still has great potential on sentiment polarity labeling that is more evenly distributed, sequential labeling and online labeling.

Reference

[1] Daniel Y. Fu, Mayee F. Chen, Frederic Sala, Sarah M. Hooper, Kayvon Fatahalian, Christopher Re. Fast and Three-rious: Speeding Up Weak Supervision with Triplet Methods. Proceedings of the 37th International Conference on Machine Learning (ICML 2020)
[2] Ratner, A., Bach, S. H., Ehrenberg, H., Fries, J., Wu, S., & Ré, C. (2017, November). Snorkel: Rapid training data creation with weak supervision. In Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases (Vol. 11, No. 3, p. 269.
[3] Joglekar, M., Garcia-Molina, H., and Parameswaran, A. Evaluating the crowd with confidence. In Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 686–694, 2013.
[4] Chaganty, A. T. and Liang, P. Estimating latent-variable graphical models using moments and likelihoods. In International Conference on Machine Learning, pp. 1872– 1880, 2014.

Mutli-Aspect Sentiment Analysis Framework via Weakly Supervised Classification and Heuristic Target-opinion Mining

JIayu Wu
Amazon, Inc.
jiayuwu@ucla.edu

1. Introduction

We implemented a weekly-supervised Multi-Aspect Sentiment Analysis (MSA) framework by target-opinion word pair extraction, target-aspect matching, and a variational sentiment classification model following the **VWS** methodology proposed by (Zeng et al., 2019). We also proposed a modified version with shared representation layers and joint training, which achieved similar performance with significantly faster computation time. Comprehensive analysis and future improvement directions are also presented.

Multi-Aspect Sentiment Analysis (MSA) is motivated by the interests in text sentimental classification on a more fine-grained level than overall polarity/rating. Specifically, we are interested in predicting the sentimental orientation for different aspects based on the document. Take Figure 1 as an example, given a hotel review the task is to extract the sentiment towards value, rooms, location and service respectively besides the overall sentiment.

The main challenge for MSA is the lack of direct supervision, because manual labeling is not only expensive but also hardly generalizable to other aspects and other corpus. Therefore, we seek for unsupervised learning or weakly supervised learning with indirect supervision. Whereas, in this report we assume a predefined set of aspects for each corpus, which in practice could be derived through domain-specific manual labeling, topic modeling, etc.

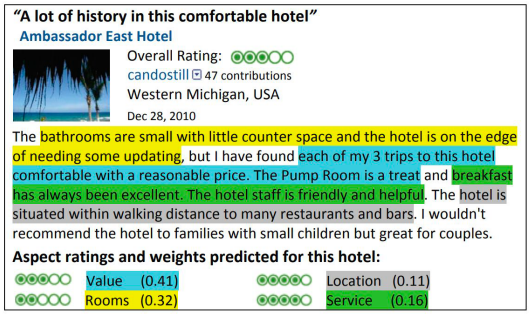


Figure 1

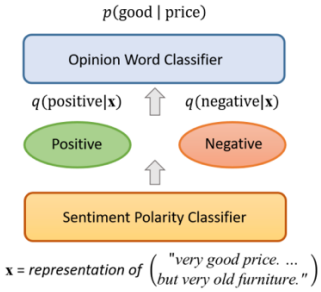


Figure 2

2. Related Work

Weakly-supervised learning is a common method when it is hard to obtain sufficient direct supervision. Weak supervision often takes form of indirect supervision from an alternative source that is related to the intended quantity. Given the relation, an auxiliary task can be defined with supervised objective, and the intended quantity can be modeled and estimated as a latent variable. For MSA task, the intended quantity is the sentiment polarity/rating of each aspect given the document. We review two STOA methods for MSA with different weak supervision and auxiliary tasks:

VWS (Zeng et al., 2019) proposed to use target-opinion word pairs as weak supervision. Those word pairs are extracted based on dependency parser, and then the target words are matched to each aspect by word similarity. They define the auxiliary task as predicting the opinion words given the aspect sentiment polarity (Figure 2). For example, if the sentiment towards the aspect 'value' is positive, the probability of seeing an opinion word 'good' for target word 'price' should be high. A complete data likelihood is defined on P(opinion word | target word), and an ELBO objective is derived for variational EM optimization. In this report we implement this method with some modifications.

LARA (Wang et al., 2010) proposed to make better use of the overall sentiment rating, and learn opinion-specific word distributions in a way analogous to topic modeling. They assume overall rating is the product of the two latent variables – aspect opinion and aspect importance of a document. In order to factorize the overall rating in a reasonable way, aspect segmentation of the document based on frequency pattern is performed to start from good aspect sentiment prediction, and various regularization techniques are adopted.

3. Methodology

This section describes in detail how is the methodology for MSA, given a pre-defined set of aspects. Each aspect is given as a set of seed words in our experiments. The MSA framework consists of two stages: 1) extract target-opinion word pairs given dependency parser and match to aspects; 2) multi-aspect sentiment orientation classification.

3.1 Target-Opinion Word Pair Extraction and Matching

Target-opinion word pairs are relevant to aspect sentiment in that, the opinion word indicates the sentimental orientation, and the targets word aligns the expressed sentiment with a specific aspect. To obtain target-opinion word pairs aligned with pre-defined aspects, we first extract the word pairs base on dependency parser, and then match them to aspects based on similarity between target word and aspect word.

3.1.1 Extraction Algorithm

After tagging the part of speech for a document with a pretrained dependency parser, we could use syntactic rules to extract target-option word pairs. Four rules are defined as follows:

Condition	**Example**
amod (adjectival modifier)	**spacious* room
nsubj (nominal subject) && ADJ among children of the head word	the *room is spacious
dobj (direct object) && head word in ("like", "dislike", "love", "hate")	I like this *room
xcomp (open clausal complement) && head word in ("seem", "look", "feel", "smell", "taste")	it feels *spacious

Table 1

The conjunct word of each target/opinion word are also included, and negation are marked. Note that the VWS paper also use a fifth rule that is domain-specific, we discard it to avoid manual labeling.

Since a major source of noise is amod with neutral sentiment, ex. 'desk-front', 'day-first', 'room-next', etc., we eliminate pairs where opinion words are neutral, based on open-source lexicon sentiment dictionary. This step excludes nearly one half of the extracted pairs.

3.1.2 Match Algorithm

We match target-option pairs to predefined aspects by embedding similarity between target word and aspect seed words. For each pair, we calculate the cosine similarity between the target word and all seed words based on pretrained w2v embedding (Mikolov et al., 2013). Each pair is assigned to the aspect where one of its seed words has the highest similarity to the target word.

3.2 Multi-Aspect Sentiment Classification

3.2.1 Baseline: Lexicon Opinion Scoring

Lexicon Opinion Scoring is a weak baseline for sentiment analysis. It is a ruled-based decision that does not require any supervision but only a word sentiment dictionary. If an opinion word from the extracted pair is a positive lexicon according to the sentiment dictionary it votes for positive, and vice versa. Then the polarity of an aspect is determined by majority voting by all opinion words. When there is a tie, Lexicon-O uses the overall polarity as the prediction and Lexicon-R randomly assigns a polarity.

3.2.2 Variational Weakly Supervised Model (VWS)

We train a variational document-level multi-aspect sentiment classification model following (Zeng et al., 2019). For each aspect, a sentiment polarity classifier and an opinion word classifier are trained. The sentiment polarity classifier takes as input a document representation x and output the sentiment polarity R_a for the aspect (Eq 1), the opinion word classifier takes as input the predicted sentiment polarity r_a and a target word w_t, and output the probability a distribution over all opinion words w_o (Eq 2).

$$q(R_a = r_a|\mathbf{x}) = \frac{\exp\left(\mathbf{w}_{r_a}^T\mathbf{x}\right)}{\sum_{r'_a}\exp\left(\mathbf{w}_{r'_a}^T\mathbf{x}\right)}, \quad (1)$$

$$p(w_o|r_a, w_t) = \frac{\exp\left(\varphi(w_o, w_t, r_a)\right)}{\sum_{w'_o}\exp\left(\varphi(w'_o, w_t, r_a)\right)}, \quad (2)$$

Define the complete data log likelihood over opinion word given target word, an ELBO can be derive as:

$$\mathrm{L} = \log p(w_o|w_t) \geq \sum_{r_a} q(r_a|x) \left[log \frac{p(w_o, r_a|w_t)}{q(r_a|x)} \right] = \mathrm{E}_{q(r_a|x)}[\log p(w_o|\, r_a, w_t)\, p(r_a)] + \mathrm{H}(q(R_a\,|\mathrm{x})).$$

Further assuming that the sentiment polarity R_a follows uniform distribution, i.e., $p(r_a)$ is a constant. The training objective is thus $\mathrm{E}_{q(r_a|x)}[\log p(w_o|\, r_a, w_t)] + \mathrm{H}(q(R_a\,|\mathrm{x}))$, which can be computed with the observed data to update the two aforementioned classifiers via SGD. Due to the large corpus size for opinion word, we could apply negative sampling technique in training. The document representation is obtained from a hierarchical iterative attention model (Yin et al., 2017), trained together with the two classifiers.
To ensure good sentiment classification, the network was initialized through supervised overall-rating classification, and then a separate model is trained for each aspect. In the section below, we replicate the VWS method and also show that training multiple aspects with shared representation layer could achieve a similar prediction accuracy with significant reduction in run time.

4. Implementation

We implement the MSA framework with word pair extractions and sentiment classification, as illustrated in Fig-3.

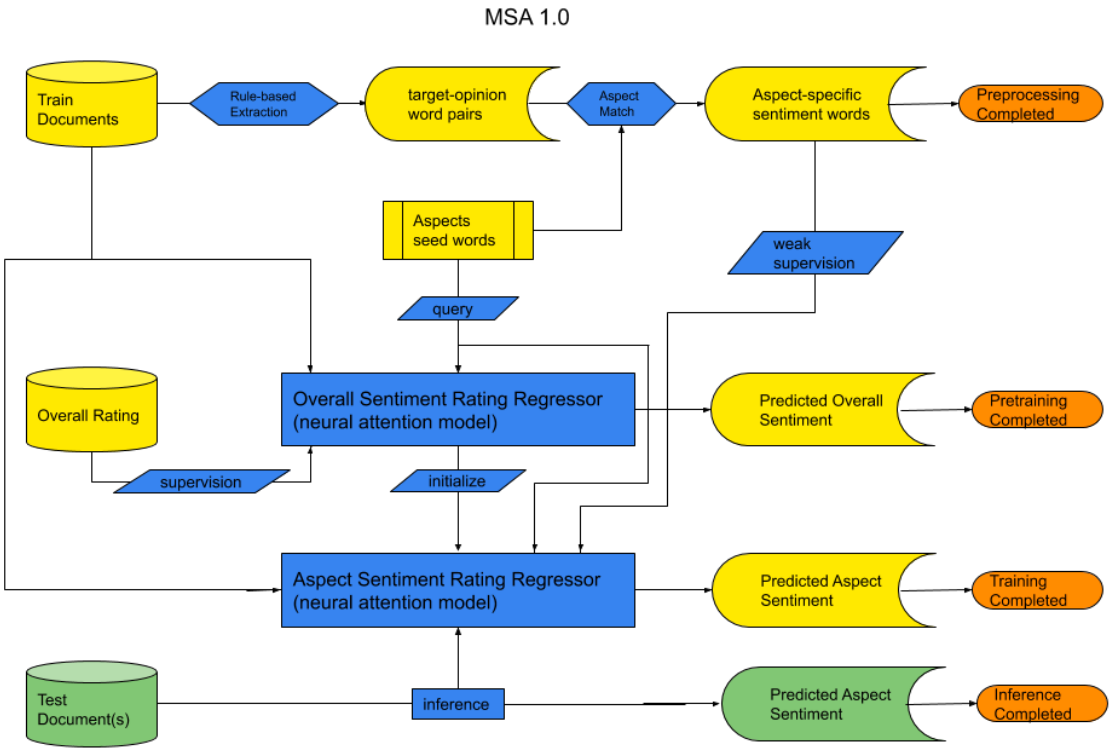


Figure 3

For target-opinion word pairs extraction and matching, we implement the heuristic algorithm in section 3.1 in Python, and the pretrained dependency parser for part of speech tagging is spaCy en_core_web_sm. We also implemented the Lexicon Opinion Scoring algorithm to examine the target-opinion extraction quality as well as baseline the sentiment classification task. For opinion lexicon dictionary, we combine two popular ones used by Hu and Liu (2004) and Wilson et al. (2005). With the two different ways to break tie in lexicon voting, we have two variants as **Lexicon-O** using the overall polarity as the prediction and **Lexicon-R** randomly assigning a polarity.

For the VWS model, we used the open source tensorflow code to replicate the original model, and compare to that reported in the paper, termed as **VWS-replicate** and **VWS-report** respectively. We also implemented a modified version for significantly improved computation speed with marginal accuracy decrease, termed as **VWS-opt**.

The major difference between our VWS-opt and the original VWS-replicate is that: we construct the training model with fixed representation layer plus variable output layer, such that an arbitrary number of aspects can be learned and inferred simultaneously. It is more efficient that than training separate models for each aspect. We also use the same learnable scaling factor for the sentiment word weights across all aspects for joint training, which ensures the loss components for different aspects are balanced. Other trivial modification for training

efficiency includes packing bi-lstm inputs to variable lengths to reduce redundant computation, and use the same embedding weights for aspect-query and document-key instead of learning a separate set of word embedding. The latter is not only more efficient, but also enables expansion to a new aspect without redo the pretraining for query embedding.

For fair comparison to the original paper, we adopted the same training setting to use Adadelta optimizer with initial learning rate=1.0 and weight decay=1e-5 on mini-batches of size 8. A validation set is used to select the best model along the training, and the learning rate is decayed by a factor of 0.95 when the validation accuracy stops increasing.

5. Experiment Results

We experiment on two public datasets: TripAdvisor and Beer with 7 and 4 predefined aspects respectively.

4.1 Target-Opinion Extraction and Matching

We compare the extracted word pairs from our heuristic extraction algorithm to the resulted word pairs disclosed by the original paper in Table 2:

Method	**Num. Pairs**	**Avg Pairs per document**	**Num. Pairs**							**Run Time**	
			Aspect 1	**Aspect 2**	**Aspect 3**	**Aspect 4**	**Aspect 5**	**Aspect 6**	**Aspect 7**	**Extract**	**Match**
TripAdvisor: 28543 documents (train: 22665, val: 2939, test: 2939)											
VWS Results	281955	9.88	21889	110619	38137	16093	5818	79114	10285	-	-
Our Heuristic	282264	9.89	27543	72080	42633	8630	34113	85334	11931	376 s	1.6 s
Beer: 27583 documents (train: 22067, val: 2758, test: 2758)											
VWS Results	290975	10.55	53470	76961	59520	101024	-	-	-	-	-
Our Heuristic	230328	8.35	38855	78143	51708	61622	-	-	-	206 s	1.2 s

Table 2

It appears that we extract a few less word pairs, which is expected as we use fewer rules (four rather than five).

By closer examination there are indeed some noise in our results, the common sources of noise are:

1. amod with neutral sentiment, ex. ‘desk-front’, ‘day-first’, ‘room-next’;
2. dependency parsing inaccuracy especially when the sentences are incomplete or grammatically wrong;
3. failure to understand idiom, ex. ‘away from' indicates negation, 'in general’ identified as amod.

We also compare the Lexicon Scoring result based on our extracted word pairs to that based on VWS word pairs and report in Table 3. It can be observed that for Beer dataset our extracted word pairs give comparable performance to the original. For the TripAdvisor dataset, we further break down the comparison on each aspect in Table 4. The aspect ‘clean' mainly accounts for the inferior results.

Accuracy	**Lexicon-O**		**Lexicon-R**		**Lexicon-O**		**Lexicon-R**	
	dev	**test**	**dev**	**test**	**dev**	**test**	**dev**	**test**
TripAdvisor					**Beer**			

Our Heuristic	0.6667	0.6691	0.6425	0.6444	0.6649	**0.6623**	0.6381	0.6345
VWS-replicate	0.7166	0.7134	0.6753	0.6536	0.6564	0.66	0.6136	0.6195
VWS-report	0.7153	**0.7153**	0.5914	0.5973	0.651	0.651	0.5895	0.5881

Table 3

O-Assign		**value**	**room**	**location**	**clean**	**staff**	**service**	**business**
Dev Acc.	VWS-replicate	0.7191	0.6977	0.7746	0.7692	0.627	0.7203	0.6718
	Our Heuristic	0.667	0.7011	0.7518	0.6399	0.6061	0.7232	0.5627
Test Acc.	VWS-replicate	0.7263	0.7117	0.7824	0.8079	0.7037	0.7199	0.5205
	Our Heuristic	0.6872	0.6954	0.7241	0.7056	0.6112	0.7183	0.5403

Table 4

4.2 VWS Benchmark

We replicate the original VWS multi-aspect sentiment classification model with their open source code, and compare the performance between the reported result in the paper, the replicated results with our extracted word pairs and the replicated results with the original word pairs. Table 5 showed that our replication achieved a similar performance to the original paper, except for one TripAdvisor aspect 'location'.

Dataset	**Aspects**	**VWS-report**		**VWS-replicate our extracted word pairs**		**VWS-replicate VWS original word pairs**	
		dev	test	dev	test	dev	test
TripAdvisor	Avg.	0.7577	**0.7561**	0.7212	0.7145	0.7644	0.7554
	value	-	-	0.8434	0.85	0.8509	**0.8529**
	room	-	-	0.7507	0.74	0.7639	**0.7405**
	location	-	-	0.5896	0.5772	0.8502	**0.8527**
	clean	-	-	0.79	**0.7616**	0.7839	0.7616
	staff	-	-	0.6999	**0.7209**	0.654	0.6448
	service	-	-	0.7554	0.7555	0.7541	**0.7657**
	business	-	-	0.5817	**0.5614**	0.5921	0.5514
Beer	Avg.	0.7502	**0.7538**	0.7121	0.7105	0.7199	0.7155
	feel	-	-	0.6818	**0.6884**	0.6893	0.68576
	look	-	-	0.6586	0.6394	0.6617	**0.6453**

	smell	-	-	0.6856	0.6862	0.7086	**0.7152**
	taste	-	-	0.8198	**0.8282**	0.8171	0.8156

Table 5

4.4 Modified VWS

The original VWS implementation train a separate model for each aspect, which is very time consuming. Therefore, we implement a modified version VWS-opt in pytorch, where the representation layer for document encoding is shared for overall sentiment pretrain and sentiment classification for each aspect.

Since our implementation is capable of joint training of arbitrary number of aspects, we compare in Table 6 the classification accuracy for joint training and separate training, and in Table 7 the total training time for 10 epochs (the convergence is normally achieved within 6 epochs).

We observe that sharing representation layer cut the multi-aspect training time by 50% while the classification accuracy is only slightly inferior. Either training scheme achieved a better accuracy than weak baseline of Lexicon-O and the original open-source implementation.

Method	**TripAdvisor**				**Beer**			
	Aspect Accuracy		**Overall Accuracy**		**Aspect Accuracy**		**Overall Accuracy**	
	dev	**test**	**dev**	**test**	**dev**	**test**	**dev**	**test**
Lexicon-O	0.6667	0.6691	0.6565	0.6745	0.6649	0.6623	0.6188	0.6001
VWS-reproduce	0.7212	0.7145	0.9149	0.9032	0.7121	0.7105	0.8391	0.8435
VWS-opt (joint)	0.7133	0.7231	0.9179	0.9023	0.7106	0.7055	0.8385	0.8452
VWS-opt (sep)	0.7246	**0.7377**			0.7175	**0.7297**		

Table 6

10 Epoch Training Time	**Pre-train**	**Train**	**Pre-train**	**Train**
VWS-opt (joint)	1221s	1334s	1029s	739s
VWS-opt (sep)		2894s		1824s
VWS-reproduce	1140s	4512s	782s	1920s

Table 7

Note that for this experiment, we keep the hyperparameters the same as the original paper for fair comparison, except that we increase the decay factor to 0.99 for joint training because in this case the same batch sees less examples over each aspect. Whereas, further improvement could be achieved with further modification to the optimization scheme. For example, we observe faster convergence (in less than 3 epochs) with adam optimizer and learning rate at 1e-3.

7. Conclusion and Future Works

We implemented a Multi-Aspect Sentiment Analysis (MSA) framework in two primary stages: 1) a heuristic algorithm for target-option word pairs extraction and matching; 2) a variational multi-aspect sentiment classification model with target-option word pairs as weak supervision. We adopt the methodology propose by (Zeng et al., 2019), and also made modification fot shared representation layers and joint training for reduced computation time with similar classification accuracy. Our implemented framework achieved better performance than naïve lexicon scoring baseline and than the original open source implementation.

For future improvements, we could consider alternative model structure under the same framework. Although representation sharing is enabled by our implementation, the current prediction layers is a single softmax layer that predicts each aspect independently. We could consider alternative structures for joint prediction to capture correlation between different aspects. Current framework uses a hierarchical attention LSTM for document

representation side, while we could consider alternative document encoder like transformer for further performance boost.

Besides VWS framework, there are also other promising weakly supervised multi-aspect classification methods we could explore:

1. VWS defines the objective as maximizing the likelihood of opinion word given target word, and derives a variational EM by factorizing the distribution of aspect polarity. Whereas, another weakly-supervised objective is also possible, that instead of defining an opinion word classifier p(opinion word | aspect rating) and factorizing the aspect rating, we may also define the posterior as p(aspect rating | opinion word) and minimize the divergence between q(aspect rating | document) and the posterior directly.
2. Inspired by LARA, another weakly supervised MSA method we review in section 2, we may also assume another latent variable as relative aspect importance to the overall sentiment. LARA uses simple language model (BOW) because it relies on an analytical solution to the latent variable estimation. We could substitute this objective with a variational method as well (ex. sample aspect importance variable from an inference network following VAE). Moreover, it is also possible to optimize jointly both VWS objective and LARA objective. They can share the same model structure for document representation and for aspect sentiment classifier, and updated by alternating SGD between two objectives with different auxiliary prediction model.

References

[1] Zeng, Ziqian and Zhou, Wenxuan and Liu, Xin and Song, Yangqiu. 2019. A Variational Approach to Weakly Supervised Document-Level Multi-Aspect Sentiment Classification VWS, arXiv preprint arXiv:1904.05055 (2019).

[2] Hongning Wang, Yue Lu, and Chengxiang Zhai. 2011. Latent aspect rating analysis without aspect keyword supervision. In Proceedings of KDD, pages 618– 626

[3] Yichun Yin, Yangqiu Song, and Ming Zhang. 2017. Document-level multi-aspect sentiment classification as machine comprehension. In Proceedings of EMNLP, pages 2034–2044.

[4] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In Proceedings of NIPS, pages 3111–3119.

[5] Jonathon Read and John Carroll. 2009. Weakly supervised techniques for domain-independent sentiment classification. In Proceedings of CIKM workshop on Topic-sentiment Analysis for Mass Opinion, pages 45–52. ACM.

[6] Aitor Garc´ıa Pablos, Montse Cuadros, and German Rigau. 2015. V3: Unsupervised aspect based sentiment analysis for semeval2015 task 12. In Proceedings of SemEval, pages 714–718.

Layer-wise Knowledge Distillation for BERT

Yang Cai
Steven.yangcai@gmail.com

1. Introduction

Language models like BERT [1] have been proved to be consistently better than the other language models proposed at the time and perform well in many downstream tasks. There's a trend towards bigger and bigger models. However, due to the large number of parameters, models like BERT are very slow for inference comparing to traditional deep learning models and not practical for business usage.

To improve the speed of inference while keeping the performance of BERT, we implemented a novel Transformer distillation method for knowledge distillation of the BERT from the paper TinyBERT [2]: Distilling BERT for Natural Language Understanding.

In this work, we reduced the size of the embedding layer and hidden layers of the original BERT model and build the new model TinyBERT [2], which is 8x times smaller and 10x times faster for inference than the base model of BERT. With knowledge distillation, the plenty of knowledge encoded in original model, e.g., BERT, can be effectively transferred to a smaller "student" model, e.g., TinyBERT [2].

2. Knowledge Distillation

There has been works show that it is possible to reach similar performances on many downstream tasks using much smaller language models pre-trained with knowledge distillation, resulting in models that are lighter and faster at inference time, while also requiring a smaller computational training budget.

Knowledge distillation is a model compression technique in which a compressed model (the student) is trained to learn the behavior of the original model (the teacher). Some prior works focus on building task-specific distillation [5] setups. For example, building an LSTM model to learn the behavior of BERT for text classification tasks.

Previous work like DistilBERT [3] only trained the student model to replicate the teacher based on its output distribution which in comparison greatly reduces the knowledge transfer. DistilBERT [3] also used the teacher model for initializing the weight of the student, which required both to have the same internal dimensions while allowing a different number of layers.

Instead of task specific distillation setup, in this work, a compressed BERT model is used as the student model and the original BERT base model as the teacher model. We encourage the student model to learn the behavior of each layer and the linguistic knowledge of the teacher model. This allows the internal representations of student and teacher to be compared elementwise despite their difference in dimensions.

The linguistic knowledge of BERT includes syntax and coreference information, which is essential for natural language understanding. The layer-wise distillation is to encourage that the linguistic knowledge can be transferred from teacher (BERT) to student (TinyBERT [2]).

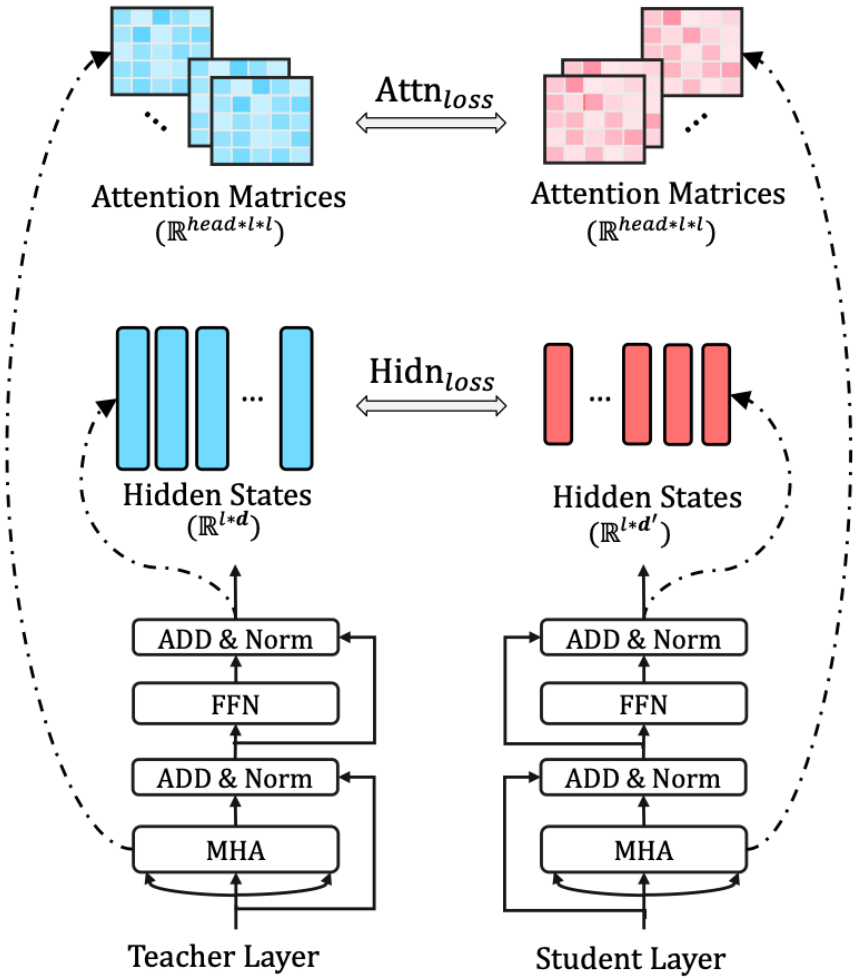
2.1 Model Setup

The distillation process includes two transformer-based models with different sizes: the BERT base model and TinyBERT [2].

The BERT base model consists of 12 layers with hidden size 768 and 12 attention heads and contains 110M parameters. By reducing the number of layers from 12 to 4 and reducing the size of hidden layers from 768 to 312, we build TinyBERT [2]. TinyBERT [2] contains only 14.5M parameters.

Sharing similar structures between the teacher model and the student model has its advantage comparing using completely different models, e.g. using LSTM and BERT for task-specific distillation [5]. Unlike LSTM, since TinyBERT [2] and BERT share similar structures, we are able to learn the layer-wise behavior of BERT with TinyBERT [2].

Details of layer-wise distillation from the original paper of TinyBERT [2]:



2.2 Loss Function

A set of loss functions are designed to help the student learn fine-grained layer-wise behavior from its teacher. The purpose of designing these loss functions is to allow the student model to observe how the teacher's embedding layer, attention matrices, hidden representations, and prediction layer behave given input data. The loss function consists of the following four parts:

$$\mathcal{L}_{embd} = \texttt{MSE}(\boldsymbol{E}^S \boldsymbol{W}_e, \boldsymbol{E}^T)$$

$$\mathcal{L}_{attn} = \frac{1}{h}\sum_{i=1}^{h} \texttt{MSE}(\boldsymbol{A}_i^S, \boldsymbol{A}_i^T)$$

$$\mathcal{L}_{hidn} = \texttt{MSE}(\boldsymbol{H}^S \boldsymbol{W}_h, \boldsymbol{H}^T)$$

$$\mathcal{L}_{pred} = -\texttt{softmax}(z^T) \cdot \texttt{log\_softmax}(z^S)$$

- MSE loss between the embeddings of student and teacher networks. A linear transformation is added to student's embedding layer due to the different size of embeddings of student and teacher.
- Average MSE loss of all heads between attention matrices of student and teacher networks.
- MSE loss of the hidden layers of student and teacher networks. Like embedding loss function, a linear transformation is added to student hidden layers.
- The soft cross-entropy loss between the student network's logits against the teacher's logits.

3. Experiment

We apply knowledge distillation to TinyBERT [2] and BERT-base-uncased (English) models with a combination of the English Wikipedia dataset and Amazon review dataset (about 100M customer reviews provided). Furthermore, we also pretrained TinyBERT without the teacher model and compared the performance.

Before training, there are usually two strategies for the initialization of the student model: using randomized parameters and using layers extracted from the teacher model. In previous work, i.e., DistilBERT [2], the student model is initialized with layers extracted from the teacher model. Since TinyBERT [2] has different sizes of hidden layers compared to BERT-base-uncased, extracting layers from the teacher model is no longer practical. Hence, we use randomized parameters as initialization of the student model in the experiment.

3.1 Performance

TinyBERT [2] is significantly smaller and faster than DistilBERT [3] and BERT-base models.

Model	# Parameters (millions)	Infer Time (seconds)
TinyBERT	14.5	120
DistilBERT	66	410
BERT-base-uncased	145	895

TinyBERT [2] recovers 98% of performance on downstream classification task:

Model	Accuracy (%)
TinyBERT (distillation)	91
TinyBERT (pretraining only)	89
DistilBERT	93
BERT-base-uncased	93

3.2 Distributed Training

To reduce the total training time by splitting the training across multiple GPUs, a ring topology can be utilized. This enables the GPUs to share memories evenly with their neighboring nodes in the ring and avoid over memory overflow of one single GPU. The Horovod [4] framework was used to perform the synchronized update during the training. We ran the distillation process on eight GPUs and took 300 hours to converge.

4. Conclusion

In this work, we reimplemented the layer-wise knowledge distillation from BERT into compressed TinyBERT model. Using only one-tenth of the parameters of BERT base model, TinyBERT retains 98% performance of the original BERT. The comparison between the distillation and pretraining also suggests that distillation not only makes training process converge faster but also boosts the performance for downstream tasks.

5. Future Improvements

In the knowledge distillation process, the loss is calculated only between the student model and the teacher model. In order to retain the linguistic information of the teacher model and keep the distillation process robust, we use a combination of the same data set with which the teacher model is pretrained, i.e., wiki data set and the domain specific data set, i.e. Amazon reviews. In the future we will split the distillation process into two steps:

a. Distillation on the original data set on which the teacher model is pretrained.
b. Pretraining the student model on domain specific data set without distillation.

References

1. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. Attention Is All You Need.
2. Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang and Qun Liu. TinyBERT: Distilling BERT for Natural Language Understanding.
3. Victor SANH, Lysandre DEBUT, Julien CHAUMOND, Thomas WOLF. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.
4. LF AI & Data. Horovod. https://github.com/horovod/horovod
5. Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. Distilling task-specific knowledge from bert into simple neural networks.

An Overview of Code2vec in Student Modeling for Programming Education

Yang Shi, Thomas Price
Department of Computer Science
North Carolina State University, USA
{yshi26, twprice}@ncsu.edu

1. Introduction

Student modeling [1] has been a key task of building Intelligent Tutoring Systems [2]. The modeling process includes aspects such as student performance prediction, feedback generation, etc. For example, we can automatically generate formative feedback [3] to students according to their assignment submissions, giving students chances to fix issues before submitting them. If we have the information about student previous submissions, predictions on student future performance on a similar assignment can also help instructors determine if they need help on some specific concepts before going further. Using student models can help the learning process, while automating the process can give students and instructors more timely benefits with less human efforts, assisting *in* the learning process instead of *after* the learning process.

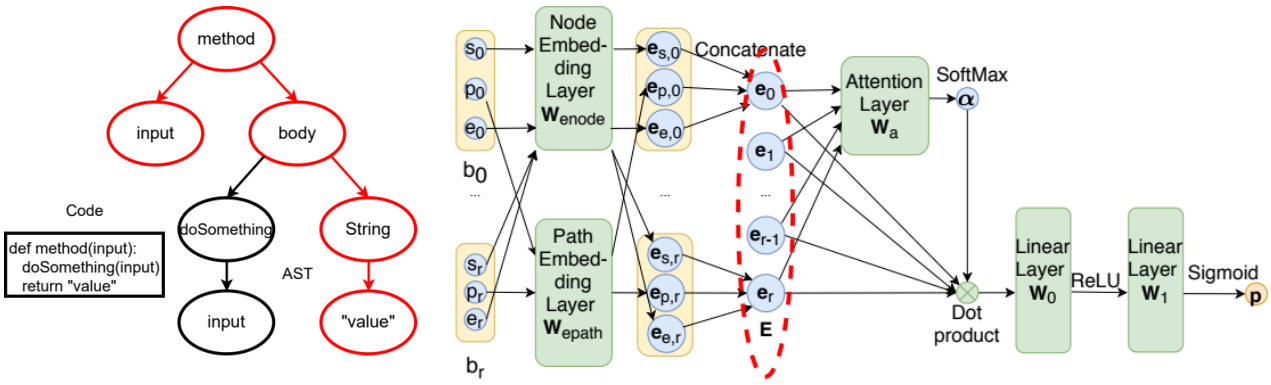
However, producing accurate modeling results and generating explanations without lots of human specifications for automated methods are challenging. There are mainly two types of automatic student modeling systems: rule-based or data-driven systems. In rule-based systems, modeling students takes expert efforts to understand the learning process of students with their expertise, and translate their understanding to rules for the automatic modeling. While these systems are often accurate and have clear explanations, this process usually still takes a lot of human effort. Recent study using data-driven methods for student modeling are getting more attention with the development of machine learning, however, should be re-designed to fit in the educational datasets. Specifically, in computer science education domain, students are often asked to submit programs to achieve requirements for some specifications. Using data-driven methods to process student programming code data is even more challenging, since programming data is highly structured. Programming code can be represented using graphs or trees [4], however, it cannot be directly used as the input of machine learning algorithms. Using proper methods to represent student code for data-driven student modeling tasks will help the modeling performance. One example of models that use such representation for data-driven student modeling in CS education is code2vec.

This communication paper presents our three work leveraging machine learning methods such as code2vec to solve student modeling tasks in the computer science education domain. In the first paper [5], we introduce an application of the code2vec model [6] for student code auto-grading tasks, generating distributed representations for student misconception cluster embedding. These generated clusters can be further used for feedback propagation, helping instructors give formative feedback to students by batch. In our second paper [7], we further used semi-supervised learning strategies for data-driven student bug detection. We specifically focused on the model performance when expert labels of student bugs are limited. Our results show that even without many labeled data points we can still use the unlabeled data for student bug detection tasks. Our third paper [8] used code2vec model to enhance the ability of deep knowledge tracing [9] by introducing code structural information to the model, helping the accurate prediction of student performance in future problems. These three papers show that code2vec model can help data-driven student models learn structural information from their code submissions, and improve the modeling quality.

2. Code2vec Model

The code2vec model was first introduced by Elon et al. [6], introducing how to build a model leveraging structural information to understand programming code. It was originally designed for classifications of function names when given the content of the code. Specifically, we can represent a code snippet as a abstract syntax tree (AST). For example, in Figure 1(a), a simple code snippet is represented as an AST with the root *method* and leaf nodes *inputs* and *"value"*. The model code2vec uses code paths to represent the elements of code, extracting leaf-to-leaf node paths to represent the structural information. One example of the code paths can be (*input*|*method*|*body*|*String*|*"value"*) in the red-colored paths in Figure 1(a), covering a wide range in the source code including the *method* definition, its input, and the returned *"value"* of the program. While it is possible to represent long term dependencies of nodes in a code snippet, some code paths can also represent shorter local areas, only representing a sequence of consecutive two statements.

The extracted paths are then used as the inputs of code2vec model. The model structure is shown in Figure 1(b). In Figure 1(b), every code path *b* are passed into the code2vec model. The paths are hashed into indices as strings, embedded through different embedding layers than the leaf nodes. The embedded nodes and paths are then concatenated into a vector, and stacked into a matrix representing the information from a code snippet. The matrix then uses the attention layer to learn the importance of every code path, using a weighted average on the embedding of every path to summarize the information from a code snippet into a single vector. The vector is a distributed and continuous representation of the code snippet, then pass through linear layers to make the final classifications, which in the case of automatic grading of student code, will be a probability of whether students succeed on all rubrics for a problem.



(a) (b)

Figure 1 (a): An example of AST representing a code snippet, shown in [5]. (b): The code2vec model used by [5] and [7].

3. Methods and Results

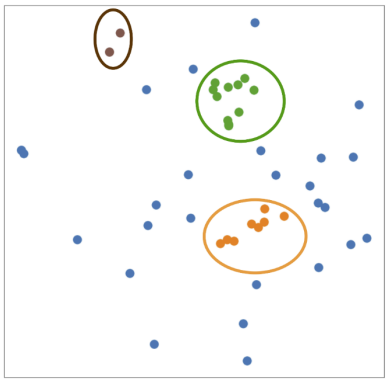
3.1 Code2vec for Misconception Clustering

Code2vec can be directly used for automatically grading student code. Given an input of student code submission, the code2vec model can turn the code path inputs into probabilistic outputs of the submission passing all rubrics. However, only knowing *whether* they are correct or not cannot directly help students without giving them explanations on *why* they are graded as fail, leaving students confused about the code submissions. Our paper [5] not only used code2vec for automatic grading, but also focused on how to interpret the embedding information inside the model, specified in the red circle of Figure 1(b), to propagate formative feedback to students, turning the confusion into learning.

We trained a code2vec model using a dataset collected in Srping 2016 through Fall 2017 from an introductory programming course in NC State University for non-majors. Specifically, we used students' submissions from one assignment requiring students to draw a spiral shape, which is particularly difficult for students with a low success rate. The language taught in the class was Snap [10], a block-based programming language designed to teach students introductory programming concepts. Our dataset collected 207 submissions from students, graded with six rubric items. If a submission passes all six items, we label it as a positive, otherwise the submission has at least one aspect failing the rubric. The details of the rubrics used can be found in our paper [5]. We trained our model using the training split of the dataset, and tuned the hyperparameters of the model with repeated sampling of the validation data from the training split. The testing performance is reported as the final results.

Our results show that the grading accuracy of code2vec model is better than the other baseline models. While the automatic clustering may still yet to be ready for classroom deployment due to a low F1 score, it still outperforms the other baseline models. We extended our analysis on the embeddings to cluster these submissions, based on the DBSCAN clustering algorithm [11]. After extracting the red-circled embedding information in Figure 1(b) from the code2vec model, we clustered the embedding vectors to see if submissions have similar misconceptions. The clustering results and some examples are shown in Figure 2(a) and (b). Three clusters are detected in the embeddings shown in Figure 2(a), and they are labeled as green, orange, and brown clusters respectively. In the green cluster, student submissions do not use *repeat* or *for* loops, and instead only used repeated code statements, showing a lack of understanding of loop from students. The orange cluster

shows students who don't use variables in their loop to control the length of strokes in their drawing process. The brown cluster is a small cluster with students misused custom input in the assignment, which is not required. Our clustering used deep learning embeddings, which are often referred to as "black-boxes" for the misconception discovery task. While propagating feedback to students, this process does not require extra expert labels on the misconceptions, although instructors still need to inspect samples of certain clusters for the actual types of misconceptions.



Green

```
forward(10);
turn(90);
forward(15);
turn(90);
forward(20);
turn(90);
…
```

Orange

```
repeat(10)
{
  x = x + 10;
  forward(x);
  turn(90);
  …
}
```

Brown

```
answer = input();
var Rotations = answer;
var Length = 10;
repeat(Rotations*4)
{
  Length = Length + 10;
  forward(Length);
  turn(90);
…
}
```

(a) **(b)**

Figure 2 (a): The generated code embedding visualizations in [5]. (b): Example pseudocode of the corresponding code clusters in (a), where the colored lines shows different types of misconceptions from [5].

3.2 Semi-Supervised Learning for Student Bug Detection

Although generating expert labels is costly, there are scenarios in the educational domain where a limited subset of fine-grained feedback labels such as program bugs are available. The small label size creates hardship for supervised machine learning methods, especially deep learning methods as they are often known as data hungry. Our paper [7] addresses this issue by introducing semi-supervised learning to the educational data mining domain by asking a question: how much labeled data we would need to create a model using semi-supervised learning comparable to a model trained with the fully labeled dataset. Besides using code2vec as a baseline model, we also used ASTNN [12] for the performance comparison. For the code2vec part, we still used the model structure introduced in Figure 1(b), while the ASTNN model structure can be found in the paper [7].

We manually labeled one assignment in the dataset collected by CodeWorkout [13] from an introductory Java course from Virginia Tech, assigning bug labels for every student submissions/compiles. We only focused on a limited combination of three kinds of bugs in our experiment, two logical and one syntactical, and compared the model performance in detecting different bugs. The results are shown in Figure 3. While solid lines show the semi-supervised F1 scores, compared with dashed lines (supervised method), even using about 30% of labeled data can train models to get similar performance with models trained with full training splits of the dataset (80% of the whole dataset).

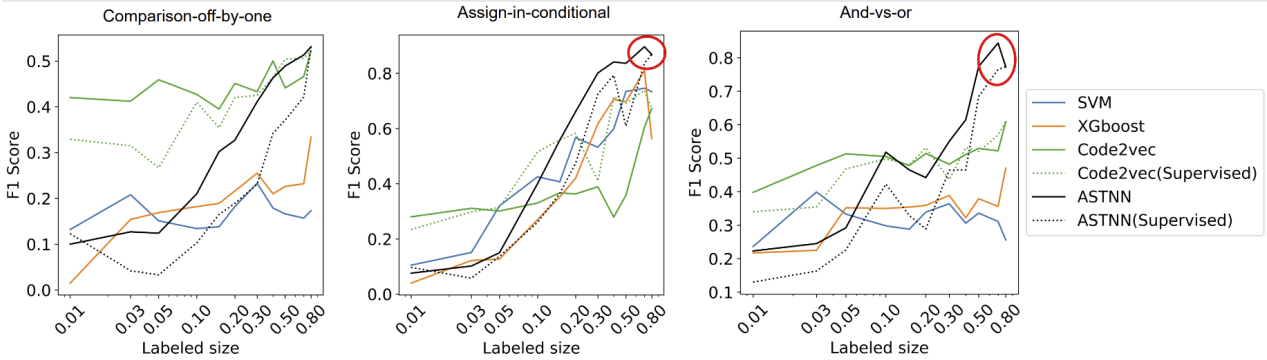


Figure 3: Comparisons of model performance in semi-supervised learning setting in [7].

3.3 Code-DKT

The model code2vec brings the structural information from student code to computation, allowing general student models to improve their performance in the CS educational domain. Our work [8] shows how code2vec can be integrated to the deep knowledge tracing (DKT) model [9] to improve the performance of knowledge tracing (KT) , and some case studies further show how code information can be leveraged to make the improvement. In KT tasks, we model students score history of attempts on a set of problems and predict their success rate on any problems in the set. After integrating code2vec extracted structural information from student code submissions, we improved the AUC score of DKT by 3.07% to 4.00%. Figures 4 and 5 show a comparison between the predicted probabilities of success for students over time. In both figures, the shade of color indicates the probability of students getting the corresponding problem correct, while the frames indicate the problems the students attempted as their next one. The numbers inside the frames are the ground truth, where 1 indicates a success. Gray frames are wrong predictions by the models, whereas black frames are successful predictions. The paper [8] used three case studies to show why code can help the model predictions in Code-DKT, marked in red in the graphs. Please refer to the original article for more details. The comparison of the two figures show that Code-DKT uses code information to make stronger predictions compared with the DKT predictions, which are usually close to the 0.5-thresholded probabilities. While code structural features helped DKT model, it could as well be able to help other student models for wider applications.

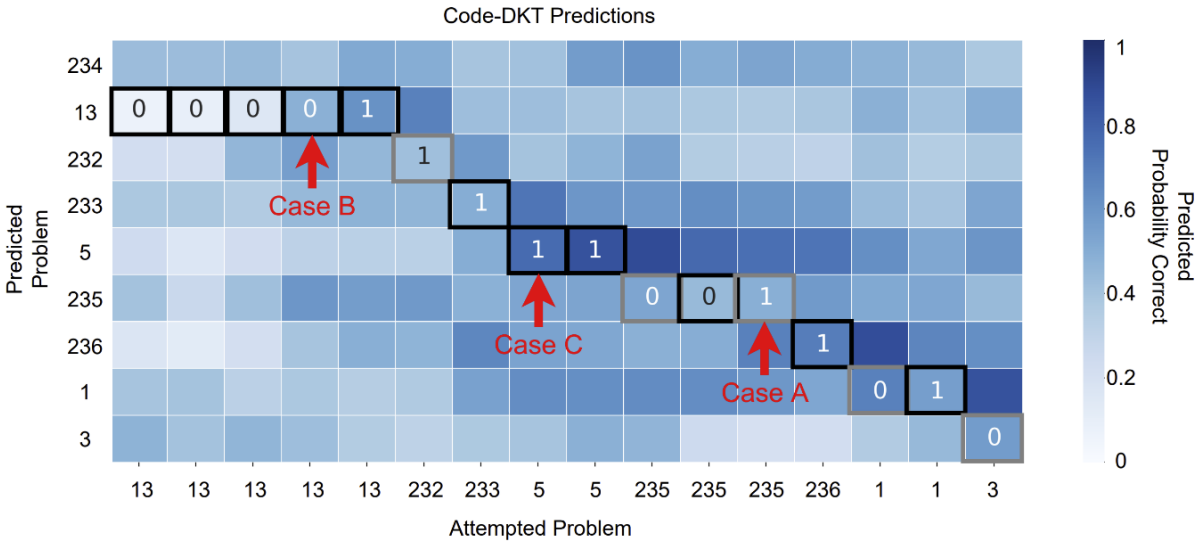


Figure 4: Predicted probability of students succeeding in their next submissions for problems by Code-DKT [8].

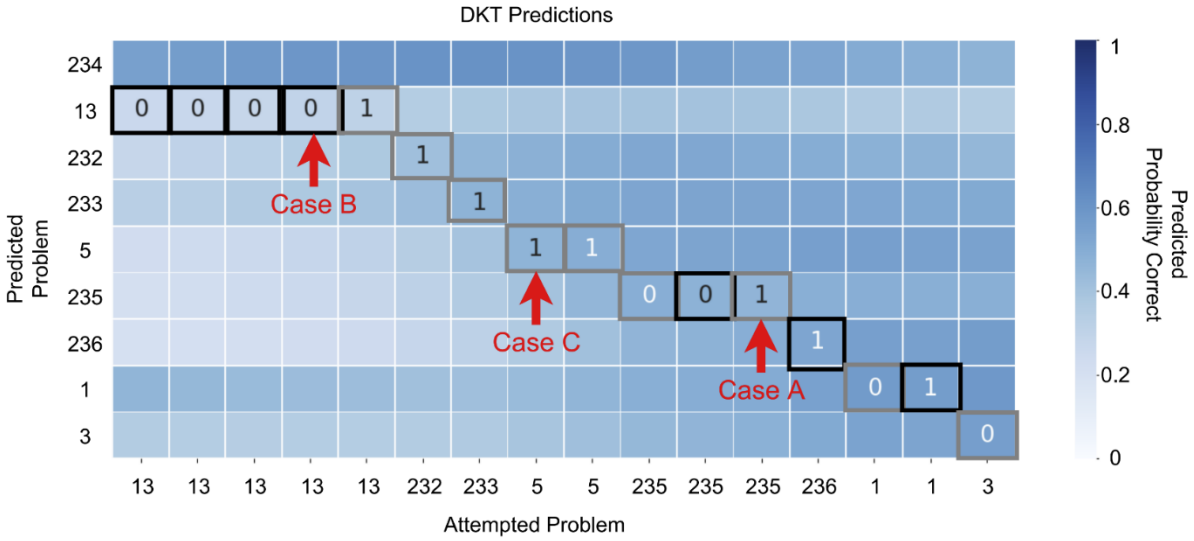


Figure 5: Predicted probability of students succeeding in their next submissions for problems by DKT [8].

4. Conclusions

In this communication paper, we introduced our three-work using code2vec to solve student modeling tasks in computer science educational domain. Our papers show that while domain-specific models such as code2vec can generate automated grade [5] or detect bugs [7] on student code, the model can also raise explanations when students submitted wrong code, propagating feedback to students by batch [5]. The model, along with other deep learning models such as ASTNN, is also proven to perform as good when using semi-supervised learning strategy when there is no enough label for the supervised learning tasks [7]. Furthermore, the code2vec model can also be integrated into other student models for better performance in tasks such as knowledge tracing [8]. Our work only provides some examples of leveraging domain-specific information for student modeling tasks in educational datasets. While there are also other technologies developing better performing models for general educational domain, computer science educational domain still needs special customization in the student models. It is not an intuitive task to process the information from student programming datasets, and thus the related tasks in CS education would have both challenges and opportunities to leverage structural information, for better modeling performances to provide more quality assistance, helping both students and instructors.

References

[1] VanLehn, Kurt. "Student modeling." Foundations of intelligent tutoring systems 55 (1988): 78.
[2] VanLehn, Kurt. "The behavior of tutoring systems." International journal of artificial intelligence in education 16.3 (2006): 227-265.
[3] Shute, Valerie J. "Focus on formative feedback." Review of educational research 78.1 (2008): 153-189.
[4] Alon, Uri, et al. "A general path-based representation for predicting program properties." ACM SIGPLAN Notices 53.4 (2018): 404-419.
[5] Shi, Yang, et al. "Toward semi-automatic misconception discovery using code embeddings." LAK21: 11th International Learning Analytics and Knowledge Conference. 2021.
[6] Alon, Uri, et al. "code2vec: Learning distributed representations of code." Proceedings of the ACM on Programming Languages 3.POPL (2019): 1-29.
[7] Shi, Yang, et al. "More with less: Exploring how to use deep learning effectively through semi-supervised learning for automatic bug detection in student code." In Proceedings of the 14th International Conference on Educational Data Mining (EDM) 2021. 2021.
[8] Shi, Yang, et al. "Code-DKT: A Code-based Knowledge Tracing Model for Programming Tasks." In Proceedings of the 15th International Conference on Educational Data Mining (EDM) 2022. 2022.
[9] Piech, Chris, et al. "Deep knowledge tracing." Advances in neural information processing systems 28 (2015).
[10] Harvey, Brian, et al. "Snap!(build your own blocks)." Proceeding of the 44th ACM technical symposium on Computer science education. 2013.
[11] Ester, Martin, et al. "A density-based algorithm for discovering clusters in large spatial databases with noise." KDD. Vol. 96. No. 34. 1996.
[12] Zhang, Jian, et al. "A novel neural source code representation based on abstract syntax tree." 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 2019.

[13] Edwards, Stephen H., and Krishnan Panamalai Murali. "CodeWorkout: short programming exercises with built-in data collection." Proceedings of the 2017 ACM conference on innovation and technology in computer science education. 2017

MMTC OFFICERS (Term 2020 — 2022)

CHAIR

Jun Wu
Fudan University
China

STEERING COMMITTEE CHAIR

Joel J. P. C. Rodrigues
Federal University of Piauí (UFPI)
Brazil

VICE CHAIRS

Shaoen Wu (North America)
Illinois State University
USA

Liang Zhou (Asia)
Nanjing University of Post and Telecommunications
China

Abderrahim Benslimane (Europe)
University of Avignon
France

Qing Yang (Letters & Member Communications)
University of North Texas
USA

SECRETARY

Han Hu
Beijing Institute of Technology
China

STANDARDS LIAISON

Weiyi Zhang
AT&T Research
USA

MMTC Communication-Frontier BOARD MEMBERS (Term 2016—2018)

Danda Rawat	Director	Howard University	USA
Sudip Misra	Co-Director	IIT Kharagpur	India
Guanyu Gao	Co-Director	Nanjing University of Science and Technology	China
Rui Wang	Co-Director	Tongji University	China
Lei Chen	Editor	Georgia Southern University	USA
Tasos Dagiuklas	Editor	London South Bank University	UK
ShuaiShuai Guo	Editor	King Abdullah University of Science and Technology	Saudi Arabia
Kejie Lu	Editor	University of Puerto Rico at Mayagüez	Puerto Rico
Nathalie Mitton	Editor	Inria Lille-Nord Europe	France
Zheng Chang	Editor	University of Jyväskylä	Finland
Dapeng Wu	Editor	Chongqing University of Posts & Telecommunications	China
Luca Foschini	Editor	University of Bologna	Italy
Mohamed Faten Zhani	Editor	University of Quebec	Canada
Armir Bujari	Editor	University of Padua	Italy
Kuan Zhang	Editor	University of Nebraska-Lincoln	USA
Bin Tan	Editor	Jinggangshan University	China