Iterative Stratified Testing and Measurement for Automated Model Updates

Elizabeth Dekeyser[*], **Nicholas Comment**, **Shermin Pei**, **Rajat Kumar**[†],
Shruti Rai, **Fengtao Wu**, **Lisa Haverty**, **Kanna Shimizu**

Amazon Alexa

Abstract

Automating updates to machine learning systems is an important but understudied challenge in AutoML. The high model variance of many cutting-edge deep learning architectures means that retraining a model provides no guarantee of accurate inference on all sample types. To address this concern, we present Automated Data-Shape Stratified Model Updates (ADSMU), a novel framework that relies on iterative model building coupled with data-shape stratified model testing and improvement. Using ADSMU, we observed a 26% (relative) improvement in accuracy for new model use cases on a large-scale NLU system, compared to a naive (manually) retrained baseline and current cutting-edge methods.

1 Introduction

Research in automated machine learning, including Auto ML and Auto AI, has primarily focused on automating the steps between data preparation and creating a purpose-relevant machine learning model, which can include choosing model architecture, tuning hyperparameters, and data preprocessing (Truong et al., 2019; Feurer et al., 2020; He et al., 2021).

However, in production machine learning systems, there is an equally important task of automating the maintenance and updates of existing models currently in use. This task embodies two essential requirements: (i) sufficiently covering new use cases; and (ii) not regressing already working use cases.

In academic literature, there exist a range of methods to meet these requirements, including Bayesian approaches (Kirk, 2017; Theodoridis, 2015), mixture of experts models (Yuksel et al., 2012; Shazeer et al., 2017), finetuning (Käding et al., 2016; Finn et al., 2017; Yu et al., 2020), and training data selection (Moore and Lewis, 2010; Axelrod et al., 2011; Iyer and Bilmes, 2013). However, production systems present unique challenges that are not addressed by these methods.

First, in deep learning models, high model variance can result in changing interpretations across model builds (Gal and Ghahramani, 2016; Belkin et al., 2019; Pham et al., 2020). Because of this, simply retraining a model with new training data can result in uneven coverage across a new use case, i.e. failing performance requirements for a subset and, thus not providing sufficient coverage for the new use case. Furthermore, retraining the model for the new use case can shift formerly correct interpretations, i.e. failing performance requirements for previously existing use cases. A standard solution to this challenge in the industry is repeated manual retraining and failure analysis, requiring weeks of modeler work for a simple model update.

Second, experimentation in production systems is further hampered by the costs of changing model architectures. Production systems can be in use by hundreds of millions of customers and are supported by complex engineering systems. Changes to these systems require extensive testing of both inference and engineering performance. Because of this, introducing architectural solutions to challenges of model variance or data diversity can be prohibitively costly.

To address these challenges, we present **A**utomated **D**ata-Shape **S**tratified **M**odel **U**pdates (ADSMU). ADSMU builds and iteratively improves models to optimize toward data-shape stratified metrics. It can be applied on top of pre-existing model architectures in any production system, making it compatible with both legacy and cutting-edge model architectures and methods for model updating.

Through stratified optimization, ***ADSMU*** ensures that models cover a diverse range of the data space without requiring manual intervention. Auto-

[*] edekeyse@amazon.com
[†] kmardpl@amazon.com

Proceedings of EMNLP 2022 Industry Track, pages 208–215
December 9–11, 2020. ©2022 Association for Computational Linguistics

mated model updates using this framework resulted in a 26% (relative) decrease in new user experiences error rate compared to alternate model update approaches (refer to Table 1). In addition, the adoption of this AutoML framework corresponded to a 94% timeline reduction in its application to selected model retraining for supporting new NLU use cases, compared to manual retraining.

In the following sections, we discuss related work, examine the details of ADSMU and how it can be applied to automated model updates, and discuss the limitations of ADSMU.

2 Related Work

AI automation can encompass all the steps of building an ML model, including problem definition, data collection, data cleaning, data coding, metric selection, algorithm selection, parameter optimization, post-processing, deployment, online evaluation, and debugging (Feurer et al., 2020; He et al., 2021). Nevertheless, these steps tend to focus on the importance of building a first, robust model, with the assumption that updating and retraining can be done using an identical pipeline and framework (Truong et al., 2019).

Research on optimized model updates, separate from AutoML, has focused on two primary fields: First, model architectures, such as finetuning (Käding et al., 2016; Finn et al., 2017; Yu et al., 2020) or Bayesian approaches (Kirk, 2017; Theodoridis, 2015), and second, training data selection, which encompasses work on submodular optimization (Moore and Lewis, 2010; Axelrod et al., 2011; Iyer and Bilmes, 2013), cross-entropy comparison (Moore and Lewis, 2010), selection by proxy (Coleman et al., 2019), and active learning (Cohn et al., 1996; Settles, 2009; Liu et al., 2021).

Many of these methods are premised on developing rich training data that performs well across feature spaces. However, despite this focus on data diversity, performance is tested, measured, and improved based on average performance across development test sets. This inherently biases models to perform well on the majority data type, allowing more challenging or rarer test cases (corresponding to newly launched user experiences) to fall through the cracks.

3 Automated Data-Shape Stratified Model Updating

ADSMU provides a framework for (1) automating data stratification, (2) using this stratification for model measurement, and (3) boosting performance in failing stratification groups using model iteration. We discuss an application of this method using an iterative approach to model retraining, which is compatible with any model architecture.

3.1 Data-shape Based Stratification

3.1.1 Stratification Methods

ADSMU requires the implementation of a stratification "rule" that is used throughout the model update process. This rule takes advantage of the labelled nature of training data and uses those labels to split data up meaningfully. This can be done using various stratification methods, including exact match, fuzzy match, or model-based.

Exact match stratification relies on holding one part of the data constant and stratifying based on other division areas. In the case of natural language understanding, this could rely on utterance label shapes. For instance, the utterances "{please: Other} {play: Other} {moana: VideoName}" and "{please: Other} {play: Other} {frozen: VideoName}" would have identical utterance shapes and therefore, in a simplistic exact-match setting, would be part of the same stratification group.

Fuzzy match stratification relies on similar manually-input heuristics but allows for more variation. In the case of natural language understanding, this could be through matching slot trails rather than labels, in which case "{play: Other} {moana: VideoName}" and "{please: Other} {play: Other} {frozen: VideoName}" could be part of the same utterance shape.[1]

The final method of data stratification is model-based. This uses unsupervised strategies to group together training data in a meaningful way and ensures data is optimized within those groups. For instance, a clustering method could be used on top of sentence embeddings that would take into account both content and syntax, placing "please play Moana" and "might you play Frozen" in Cluster

[1] Both of the above methods require modeler insight to determine principled stratification heuristics, but this should depend on the use case. In one use case, stratifying based on entity content could be more valuable (with "Moana" and "Frozen" being the meaningful inputs), while in others, the semantic shape might matter more.

A and “it would be fantastic if you played Moana” and “I would really appreciate if you played Frozen” in Cluster B due to different carrier phrase structures i.e. the model learns to differentiate between two possible classes of carrier phrases for slotting in a named entity recognition task. While this method has the benefit of relying less on labels and manual input, it has the potential to introduce more noise into the modeling process.

3.1.2 Measuring Stratified Performance

Once established, these stratification groups become the foundation for future modeling and optimization.

During model training, model performance is not judged by a single accuracy metric but rather as an average of the performance of each of the stratification groups. This ensures each stratification group is equally weighted and under-performance in a less common test set is not overlooked.

In other words, traditional approaches measure performance as $\bar{x} = \frac{1}{n}(\sum_{i=1}^{n} x_i)$ where n is the number of test cases and x_i is a binary or continuous success metric for observation i.

However, we propose a measurement method that equally weights each stratification group, regardless of the number of observations within it:

$$\bar{x} = \frac{1}{L}(\sum_{h=1}^{L} \bar{x}_h) \tag{1}$$

where L is the number of stratum and $\bar{x}_h$ is the average of the binary or continuous success metric for stratification group h.[2]

3.1.3 Resolving Stratification Group Failures

The final area in which stratification is applied is failure resolution. When a model shows failures in a stratification group, it adds synthetically-generated training data to improve performance in that stratification group.

This data is generated based on the user-defined heuristic for stratification. For instance, in an exact match example where the semantic shape is “play|Other TOKEN|VideoName” the system can generate fill-in-the-blank data for VideoName, either resampling from pre-existing training data (e.g. finding all VideoName instances and using that as a catalog) or alternately using a user-provided catalog for data regeneration.[3]

3.2 Integration into Automated Model Updates

In this section, we discuss one method for integrating stratification groups into an end-to-end model update pipeline, following the steps indicated in Figure 1 and addressed in more detail below.

1. The user provides updated data. The user inputs data to be added and iterated on top of a pre-existing model and dataset.

2. The system builds a stratified synthetic data pool. The AI system ingests the user-provided data and splits it using a pre-defined stratification method (see Section 3.1.1).[4]

3. Training data addition and model buiding. An initial amount of training data is added.[5] Model is then trained agnostic to architectures, hyperparameters, and loss functions.

4. Results of model training evaluated. The model results are evaluated on each of the stratification groups. Note that while the system may have added varying amounts of training data from different stratification groups, every stratification group is tested, and test performance is weighted equally.

5. Data addition for supporting failing stratification groups. If a stratification group is underperforming, more data is added to support this failing shape using methods discussed in Section 3.1.3.

Steps 3-5 are then repeated until net performance, measured by the methods discussed in Section 3.1.2, reaches an acceptable level or, alternately, a preset maximum number of iterations is reached.

3.3 Experimental Setup

We examine the impact of the ADSMU framework applied to a machine learning task that mimics a production-scale system and compare its results with other standard model update approaches.

[2] There is a range of potential extensions to these measurement metrics. For instance, semantic groups could be weighted based on modeler-defined heuristics, such as predicted popularity. Alternately, other metrics, such as harmonic mean, could be used to calculate semantic group performance.

[3] This data could also be created using cutting-edge, model-based synthetic generation methods (Wan et al., 2017; Zhang et al., 2019). As with model-generated semantic groups, the additional value of these methods must be balanced against the noise and lack of precision introduced by model-generated data.

[4] This step can also include data generation or supplementation to increase the data size if necessary.

[5] This can be naive (equal amounts across stratification groups) or more principled (using model-based training data selection methods, e.g. Moore and Lewis (2010); Axelrod et al. (2011); Iyer and Bilmes (2013)).

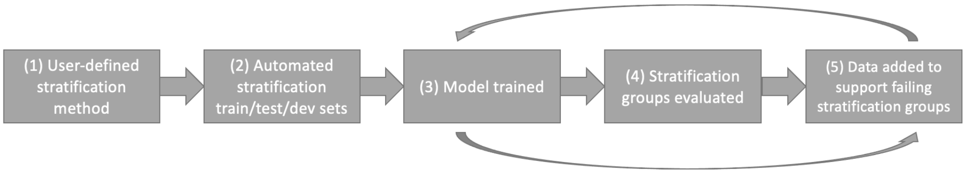
(1) User-defined stratification method
(2) Automated stratification train/test/dev sets
(3) Model trained
(4) Stratification groups evaluated
(5) Data added to support failing stratification groups

Figure 1: Exemplar integration of ADSMU in a model retraining pipeline. Data shape stratification is used for model performance measurement and improvement. Steps 3-5 are performed in iteration to boost model performance on weak stratification groups.

Experiment	**Data Selection**	**Model Training**	**Overall Accuracy Change**	**New Use Case Accuracy Change**
(1, Baseline)	Naive	Naive	0%	0%
(2)	Naive	Finetuning	1%	-2%
(3)	Submodular	Naive	0.6%	0.1%
(4)	Submodular	Finetuning	1.5%	-0.5%
(5)	ADSMU	ADSMU	1%	26%
(6)	ADSMU + Submodular	ADSMU	1.5%	28%

Table 1: Results from experimentation on experimental domain, intent, and named entity recognition tasks. Results reported are overall accuracy across classification tasks based on percent improvement from experiment (1, Baseline). ADSMU results used three model iterations.

Our experiments focused on updating a two-stage model for domain classification (DC) and a joint intent classification (IC) and named entity recognition (NER) task. For conducting experiments, we chose a deep bidirectional language model architecture based on their established success on several NLP tasks (Peters et al., 2018). The pre-existing training data (user data + synthetic data; user data used was already preprocessed for de-identification) proxied that of a large and complex production system and represented utterances across multiple domains and intents. We compare ADSMU to other model update approaches (Table 1) using a single model update task, wherein the model update was performed in a single pre-existing domain and introduced one new intent and new slots. We used exact match stratification to create 142 stratification groups for the training data and measured performance using the method from Equation (1). We set a goal of 95% accuracy for stratification groups, meaning that more training data was added only when groups did not meet that bar. This accuracy threshold is an experimental variable and can be tuned based on the trade-off between performance requirements and training data volume because error falls off as a power of the training data volume (Sorscher et al., 2022). Stratification group failures were generated using the exact match heuristic resampling from a user-provided catalog. All model update approaches used comparable amounts of training data. For purposes of scale estimation, ADSMU started with training data measuring 0.004% of the pre-existing training data (in-domain and out-of-domain data), with each iteration (step 3-5 in Figure 2) adding more data for the failing stratification groups. The total utterances used during end-to-end ADSMU measured 0.008% of all the pre-existing training data.

We compared ADSMU to a naive retrained baseline and two common model refresh methods– submodular optimization and finetuning. The baseline model update was performed by an expert modeler by manual training data sampling to support the new intent in a pre-existing domain. To implement submodular optimization, we used methods presented in Schreiber et al. (2020); specifically, the implementation of the feature-based optimization method of Wei et al. (2014). This is based on a generalized feature-based function:

$$f(X) = \Sigma_{d=1}^{D} \phi(\Sigma_{i=1}^{N} X_{i,d}) \tag{2}$$

For the optimizer, we used an approximate lazy greedy algorithm (Schreiber et al., 2020). We added the same amount of training data for the naive baseline, ADSMU and submodular optimization. For comparison with finetuning, we froze the weights of the encoder layers and retrained the decoder to learn the new intent category. We retrained for 35% of the epochs of the baseline/ADSMU model.

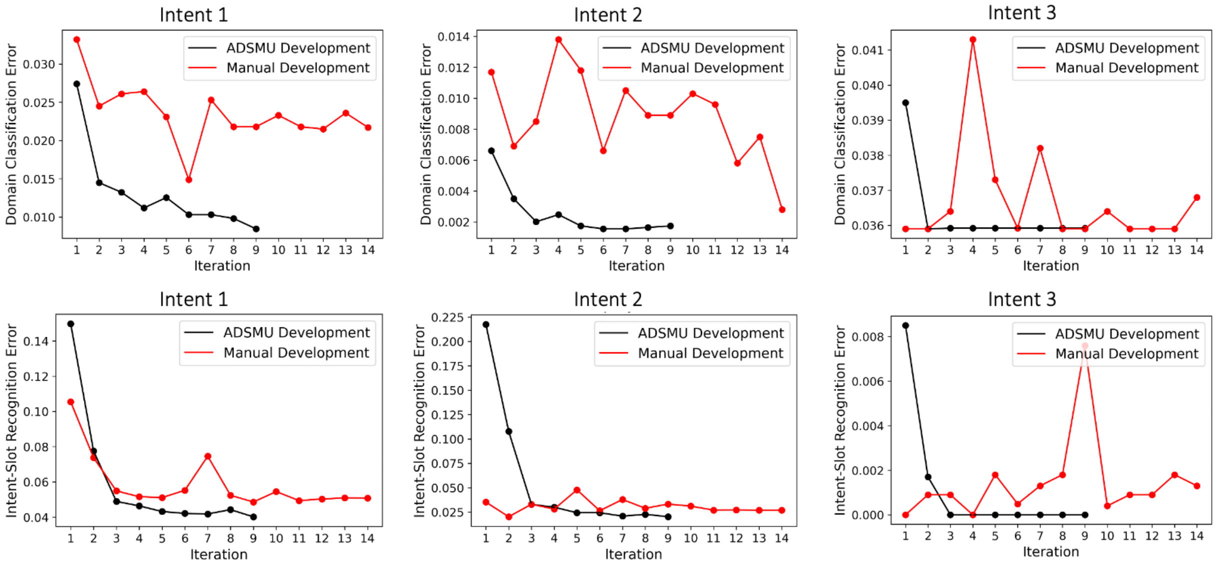

Figure 2: Comparison of manual model update vs. ADSMU. We obtained best model for the manual update after 14 iterations, and after 9 iterations for the ADSMU. Each manual iteration involved in-depth failure analysis and data addition for failure resolution by an expert modeler. Figure shows error rate (range: 0 to 1).

3.4 Results

For experiments in section 3.3, we report results based on percent improvement from current performance (baseline) based on overall accuracy across the three classification tasks (domain, intent, named entity recognition). While we only compared ADSMU to other approaches using a single model update, we repeated use of ADSMU for several other model building exercises that mimicked large-scale NLU production systems and obtained similar robust results. The results can be found in Table 1.

We find that finetuning performs slightly better overall than a naive retrained baseline but worse on new use cases. Submodular optimization combined with finetuning presents similar underperformance issues on new model use cases, though it slightly outperforms baseline when combined with a full retraining. ADSMU shows marked improvements in new use case accuracy compared with a naive baseline and slight improvements in overall accuracy. ADSMU combined with submodular optimization shows even more significant gains on overall and new use case accuracy, highlighting the potential to combine ADSMU with other data selection and model retraining approaches.

Additionally, we provide a specific exemplar case, wherein we compared iteration-level performance of model update performed by an expert modeler possessing domain knowledge against the ADSMU for a spoken language understanding use case (Sarikaya et al., 2016) (same model and task as in section 3.3). This model update was performed in a single pre-existing domain and across multiple pre-existing intents but with new slots (Figure 2). The stop-criteria (best model) for the manual development work and the ADSMU iterations required: a) no regression of the pre-existing use cases; and b) less than 5% error rate for domain classification and intent and slot recognition for the new use case. To support the new use case, the expert modeler added training data measuring 0.5% in total of the pre-existing in-domain training data, and the ADSMU added data measuring 0.83% for obtaining the best model. The ADSMU converged faster than the manual model update process and performed better for both the domain classification and joint intent and named entity recognition task. Overall, the best model for ADSMU performed 0.8% better than the model provided by the expert modeler. ADSMU showed a relatively 24% lower error rate than the modeler model for the new use case. The iteration-level error rates (range: 0-1) for the new use case (across various intents) are shown in Figure 2.

The key to contextualizing these results is an analysis on a second axis – modeler cost. Accuracy results are compared against a single naive model rebuild baseline. Yet in practice, modelers can invest time to improve model accuracy through multiple rebuilds, offline testing, and tweaking training data. Comparison of ADSMU to manual development represented a 94% timeline decrease in model updates from months to days.

4 Conclusions and Next Steps

ADSMU presents a framework for using stratified testing and measurement to iteratively improve models in the setting of automated model updates. This highlights the need that focuses on automating model creation while largely ignoring the challenges associated with automated model refresh processes (Truong et al., 2019; Feurer et al., 2020; He et al., 2021).

One key extension of this approach is applications that remove the need for model iteration and improve performance within stratified sub-groups in a single model build. For instance, optimizing on stratified loss functions might help ensure that models cover diverse use cases in a more consistent manner. More simplistically, a model could have training data added incrementally in between epochs based on weaker performing stratification groups.

5 Limitations

There are three key limitations to this approach. First, the reliance on user-defined stratification groups requires a certain degree of domain knowledge. Second, this framework is best suited to use with high-variance model architectures, since these are the most likely to show variable performance across data types and model builds. Finally, the reliance on iterations imposes a higher computational cost than a single model rebuild.

References

Amittai Axelrod, Xiaodong He, and Jianfeng Gao. 2011. Domain adaptation via pseudo in-domain data selection. In *Proceedings of the 2011 conference on empirical methods in natural language processing*, pages 355–362.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. 2019. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854.

David A Cohn, Zoubin Ghahramani, and Michael I Jordan. 1996. Active learning with statistical models. *Journal of artificial intelligence research*, 4:129–145.

Cody Coleman, Christopher Yeh, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. 2019. Selection via proxy: Efficient data selection for deep learning. *arXiv preprint arXiv:1906.11829*.

Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. 2020. Auto-sklearn 2.0: Hands-free automl via meta-learning. *arXiv preprint arXiv:2007.04074*.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR.

Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR.

Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622.

Rishabh K Iyer and Jeff A Bilmes. 2013. Submodular optimization with submodular cover and submodular knapsack constraints. *Advances in neural information processing systems*, 26.

Christoph Käding, Erik Rodner, Alexander Freytag, and Joachim Denzler. 2016. Fine-tuning deep neural networks in continuous learning scenarios. In *Asian Conference on Computer Vision*, pages 588–605. Springer.

Matthew Kirk. 2017. *Thoughtful machine learning with Python: a test-driven approach*. " O'Reilly Media, Inc.".

Zhuoming Liu, Hao Ding, Huaping Zhong, Weijia Li, Jifeng Dai, and Conghui He. 2021. Influence selection for active learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9274–9283.

Robert C Moore and William Lewis. 2010. Intelligent selection of language model training data. In *Proceedings of the ACL 2010 conference short papers*, pages 220–224.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *CoRR*, abs/1802.05365.

Hung Viet Pham, Shangshu Qian, Jiannan Wang, Thibaud Lutellier, Jonathan Rosenthal, Lin Tan, Yaoliang Yu, and Nachiappan Nagappan. 2020. Problems and opportunities in training deep learning software systems: An analysis of variance. In *Proceedings of the 35th IEEE/ACM international conference on automated software engineering*, pages 771–783.

R. Sarikaya, P. A. Crook, A. Marin, M. Jeong, J.P. Robichaud, A. Celikyilmaz, Y.B. Kim, A. Rochette, O. Z. Khan, X. Liu, D. Boies, T. Anastasakos, Z. Feizollahi, N. Ramesh, H. Suzuki, R. Holenstein, E. Krawczyk, and V. Radostev. 2016. An overview of end-to-end language understanding and dialog management for personal digital assistants. pages 391–397.

Jacob M Schreiber, Jeffrey A Bilmes, and William Stafford Noble. 2020. apricot: Submodular selection for data summarization in python. *J. Mach. Learn. Res.*, 21:161–1.

Burr Settles. 2009. Active learning literature survey.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

Ben Sorscher, Robert Geirhos, Shashank Shekhar, Surya Ganguli, and Ari S. Morcos. 2022. Beyond neural scaling laws: beating power law scaling via data pruning. *arXiv preprint arXiv:2206.14486v3*.

Sergios Theodoridis. 2015. *Machine learning: a Bayesian and optimization perspective*. Academic press.

Anh Truong, Austin Walters, Jeremy Goodsitt, Keegan Hines, C Bayan Bruss, and Reza Farivar. 2019. Towards automated machine learning: Evaluation and comparison of automl approaches and tools. In *2019 IEEE 31st international conference on tools with artificial intelligence (ICTAI)*, pages 1471–1479. IEEE.

Zhiqiang Wan, Yazhou Zhang, and Haibo He. 2017. Variational autoencoder based synthetic data generation for imbalanced learning. In *2017 IEEE symposium series on computational intelligence (SSCI)*, pages 1–7. IEEE.

Kai Wei, Yuzong Liu, Katrin Kirchhoff, Chris Bartels, and Jeff Bilmes. 2014. Submodular subset selection for large-scale speech training data. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3311–3315. IEEE.

Yue Yu, Simiao Zuo, Haoming Jiang, Wendi Ren, Tuo Zhao, and Chao Zhang. 2020. Fine-tuning pre-trained language model with weak supervision: A contrastive-regularized self-training approach. *arXiv preprint arXiv:2010.07835*.

Seniha Esen Yuksel, Joseph N Wilson, and Paul D Gader. 2012. Twenty years of mixture of experts. *IEEE transactions on neural networks and learning systems*, 23(8):1177–1193.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.