Alpine Meadow : A System for Interactive AutoML

Zeyuan Shang, Emanuel Zgraggen, Tim Kraska
Massachusetts Institute of Technology
{zeyuans, emzg, kraska}@mit.edu

Abstract

AutoML has been widely used by domain experts without machine learning knowledge to extract actionable insights out of data. However, previous studies only emphasize the high accuracy of the final answer, which can take several hours, if not a couple of days to complete. In this paper we present *Alpine Meadow*, a first *Interactive Automated Machine Learning* tool. What makes our system unique is not only the focus on interactivity, but also the combined systemic and algorithmic design approach. We design novel algorithms for AutoML search and co-design the execution runtime to efficiently execute the ML workloads.

We evaluate our system on 300 datasets and compare against other AutoML tools, including the current NIPS winner, as well as expert solutions. Not only is *Alpine Meadow* able to significantly outperform the other AutoML systems while — in contrast to the other systems — providing interactive latencies, but also outperforms in 80% of the cases expert solutions over data sets we have never seen before.

1 Introduction

Truly democratizing Data Science requires a fundamental shift in the tools we use to analyze data and build models [16]. On one hand it requires to move away from Python-like scripting languages, SQL and batch processing to visual and interactive environments [9, 23, 14, 28, 18, 36]. On the other hand, it requires to significantly reduce the required expertise to build a machine learning pipeline. Ideally, a user should be able to specify a high-level task (e.g., predict label X based on my data), and the system automatically composes a machine learning pipeline to achieve that task, including all necessary data cleaning, feature engineering, and hyper-parameter tuning steps.

The latter challenge is largely referred to as *AutoML* or *Learning to Learn* and comes in various flavors. For example, there already exists a huge amount of work on a subset of the problem: automatic hyper-parameter tuning and model family selection. Most noticeable, TuPAQ [31, 29], Hyperband [19] and the various Bayesian Optimization approaches [33, 15, 10] all have the goal to automatically determine the best model family (e.g., SVM vs Linear regression) or parameters for a given algorithm (e.g., step-size, kernel, etc.). However, hyper-parameter and model selection is only one aspect of automatically finding the best ML pipeline for a given task. Rather an end-to-end solution also has to consider data cleaning operation, feature engineering, and potentially even data augmentation and transfer learning.

The closest existing solutions, which allow such end-to-end training are probably the recent Learning to Learn approaches to find neural net (NN) architectures [38, 4]. The view of some "purist" is that the input of a NN should be the raw data and that the model – if correctly tuned, for example, by an automatic NN architecture search – should do all the rest. However, deep learning based approaches only work with huge amount of training data and output a black box solution (i.e., a neural net), which is extremely hard to interpret. While this approach might be amenable for some scenarios, many real-world problems are rather small in terms of data size. For example, in the current DARPA D3M AutoML competition, only 5% out of the 300 datasets are actually larger than 10MB. We made similar observations when working with our partners in industry and hospitals.

Preprint. Under review.

More importantly though, we are not aware of a single AutoML solution, which can provide interactive response times to enable users to steer the computation and contribute to the optimization with their domain knowledge and quickly correct mistakes. For example, a doctor might decide to remove questionable features from the training set after seeing that the model starts to rely too much on it. Furthermore, as shown in interactive data exploration [37], interactive response times can improve the rate at which insights are uncovered: a team might try to build a model quickly during a meeting rather than having a week-long back and fourth between meetings, coding and running experiments.

In this paper, we present *Alpine Meadow*, a first interactive AutoML tool, which is intended to be integrated into a visual environment similar to Tableau or Vizdom [9]. Our end-to-end interactive and automated machine learning system makes the following contributions: (1) We present a novel architecture of an AutoML system with interactive responses; (2) We show rule-based optimization, can be combined with multi-armed bandits, Bayesian optimization and meta-learning to find more efficiently the best ML pipeline for a given problem. We devise an adaptive pipeline selection algorithm to prune unpromising pipelines early. (3) We co-design the runtime with the decision process and decouple these two components to achieve better scalability, and devise sampling, caching and scheduling strategies to further promote interactivity. (4) We show in our evaluation that *Alpine Meadow* significantly outperforms other AutoML systems while — in contrast to the other systems — provides interactive latencies on over 300 real world datasets. Furthermore, *Alpine Meadow* outperforms expert solutions in 80% of the cases for datasets we have never seen before.

To see how Alpine Meadow can be used in an interactive visual environment we refer the interested reader to `http://www.einblick.ai/assets/northstar_demo.mp4`.

2 Overview

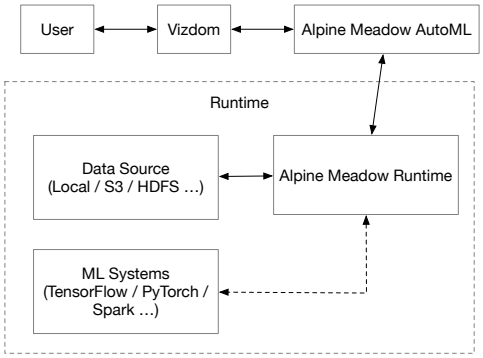


Figure 1: Overview of Alpine Meadow

Alpine Meadow is part of *Northstar*[16], a system for Data Science platform where domain experts interact with data through an interactive visual environment called *Vizdom*[9]. In this environment, a prediction problem can be specified through drag and drop gestures and can be as simple as binary classification (i.e. spam detection) or as complex as graph community detection. Based on such a problem specification, *Alpine Meadow* will automatically begin to search and progressively return machine learning pipelines to the end-user.

Alpine Meadow essentially consists of two parts: the AutoML component which makes decisions on which pipelines to evaluate, and the runtime component is responsible for evaluating pipelines (i.e., training and testing). By decoupling these two components, we can easily scale up *Alpine Meadow* by adding more runtime workers. The runtime is able to read data from different file systems (including local, S3 and HDFS) in various formats (e.g., CSV, Parquet, image). We are working on integrating with other ML systems (e.g., TensorFlow, PyTorch and Spark) such that we are able to push computations down to these systems.

3 Alpine Meadow AutoML

The general optimization loop of *Alpine Meadow* is shown in Figure 2. *Alpine Meadow* first creates a search space and employ a two-step selection algorithm to find promising pipelines, then we evaluate and prune these candidate pipelines, and use the evaluation results as the feedbacks to update the search space model. In the following, we outline the different steps in more detail.

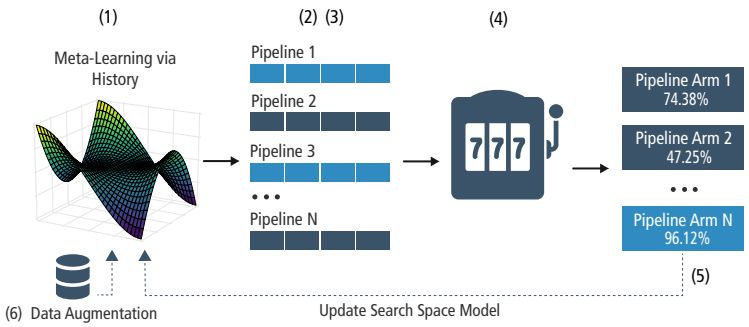


Figure 2: Optimization loop: (1) search space model, (2) logical-plan selection, (3) physical-plan selection, (4) pipelines evaluation and pruning, (5) search space model update, (6) data augmentation

3.1 Rule-based Search Space

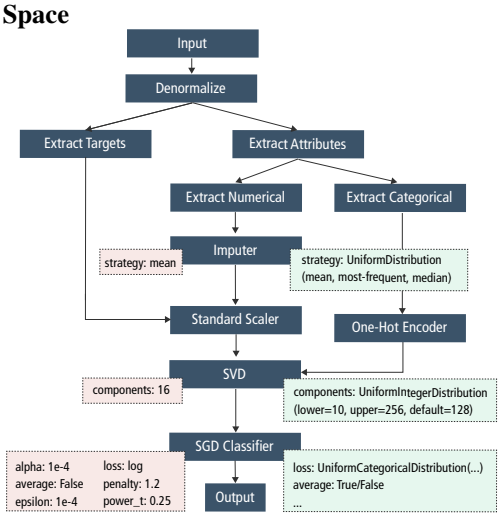


Figure 3: An example pipeline. The boxes in red show fixed hyper-parameters and they compose a *physical pipeline* plan with this DAG. While the boxes in green give distribution of hyper-parameters and they compose a *logical pipeline* with this DAG.

Data scientists rely on their expertise and past experience to solve challenging problems. We imitated this process by adapting the idea of rule-based search space definition commonly used in database optimizers to our AutoML system. Rules in our system encapsulate best practices similar to those data scientists might use. We propose three kinds of rules: *primitive*, *parameter* and *enforcement* rules to generate the search space of *logical pipelines* (we will use pipeline and plan interchangeably). A **logical pipeline** is a Directed Acyclic Graph (DAG) of primitives, with their hyper-parameters' domain specification (not fixed), and by assigning a set of hyper-parameter values to a logical pipeline, we get a *physical pipeline*, which is an end-to-end solution for the given problem. Figure 3 gives an example of *logical pipeline* and *physical pipeline*.

Primitive Rules add new primitives to the search space dependent on the task (e.g., using different algorithms for classification, regression, recommendation, or graph-related problems) or the dataset schema (e.g., applying one-hot encoding for categorical features). Until now, we have integrated close to hundred primitive rules derived from winning Kaggle competitions, expert solutions to problems provided by DARPA, and interviews with data scientists. These rules, for example, include things like encoding categorical features, scaling numerical values, imputation of empty values, selection of features, choosing models for different problem types, extracting features from raw text and images, building the graphs for graph datasets etc.

Parameter Rules generate reasonable distributions for hyper-parameters of primitives. For example, a rule might be that the set of possible values for the kernel of a SVM are *linear*, *poly*, *sigmoid* or *rbf*, or that the value for the regularization factor λ should be sampled from a log uniform distribution.

Enforcement Rules check the feasibility of a *logical pipeline*. Not every generated *logical pipeline* is feasible. For example, most algorithms will fail if not all the categorical features are encoded into numerical values or raw data (e.g., text) are not featurized. *Alpine Meadow* uses enforcement rules to validate *logical pipeline* and aborts the generation of unfeasible ones.

When executing primitive rules, we introduce some randomness such that it is possible to build very complicated pipelines (e.g., run standard scaling on some subset of columns and min-max scaling on another set of columns, and then combine them and do PCA).

3.2 Pipeline Selection

Logical Pipeline Selection. To extract *logical pipelines* out of the search space, we formulate it as a multi-armed bandit problem and employ a combination of *Upper Confidence Bound* (UCB) [8] and *ϵ-greedy* [32] algorithm to achieve the tradeoff between exploitation and exploration.

To promote interactivity, we devise a cost-aware scoring model when computing the upper confidence bound as follows:

$$s_k = \mu_k + \frac{\theta}{c_k} \cdot \delta_k \tag{1}$$

where μ_k and δ_k are the mean and standard deviation of the rewards (i.e., quality of the *logical pipeline* plan, e.g., accuracy) and c_k is the cost, or execution time, for *logical pipeline* k based on the past history, and θ is a factor on how much risk we want to take to try a pipeline, which might have a high upside (i.e., variance).

This selection model requires the past history of *logical pipelines*, to "warm-start" the selection process, we adopt some meta-learning techniques [11] and integrate it into the above scoring model.

Physical Pipeline Selection. We employ Bayesian Optimization, specifically *Sequential Model-Based Optimization* (SMBO) [13], to build a model of f to keep track of which are the most promising regions in the search space. Based on SMBO we create k *physical pipelines* for each *logical pipeline* (we set k as 10 in our implementation), and pushes them into a shared queue that is consumed by the evaluation module as described next. Here, again we not only pick the most promising hyper-parameters, but also introduce some random candidates in its *physical pipeline* candidate-list to avoid to get stuck in local optimum point. Once those *physical pipelines* evaluation is complete, their scores are returned and we use them to update the scoring model for logical pipeline selection and the surrogate model of SMBO.

3.3 Evaluation and Pruning

To achieve incremental computation and return results early, as well as reduce the computational resources spent on bad pipelines we devised **Adaptive Pipeline Selection** (APS), a bandits-based pruning strategy able to detect bad performing pipelines, without using the whole training set.

APS gets as input a dataset $\mathcal{D}$, and it splits into a training dataset D_{train} and a validation dataset $D_{validation}$. As shown in Algorithm 1, it further splits the train-set into N smaller samples of the same size $\mathcal{D}^1_{train}, \ldots, \mathcal{D}^N_{train}$ (we set N as 10 in our implementation). At each sub-epoch i, APS trains the pipeline over the partial training sample $\mathcal{D}^1_{train} \cup \mathcal{D}^2_{train} \cup \ldots, \mathcal{D}^i_{train}$ (union of first i data splits), and evaluate the errors over the partial training sample err_{train} and the validation sample $err_{validation}$. We use err_{train} as the lower bound of the final test error, therefore if the err_{train} is above the best validation error seen so far, we terminate it.

For more details on the optimization process, the interested reader is referred to [27].

4 Alpine Meadow Runtime

Another aspect what makes Alpine Meadow unique is, that it is fully integrated into Northstar and uses its existing sampling engine. In order to provide interactive speed for data exploration, Northstar's backend IDEA uses sophisticated sampling and progressive query computation techniques. Alpine Meadow uses these capabilities for its AutoML tuning.

Algorithm 1: Adaptive Pipeline Selection (APS)

Input: Pipeline $pipeline$, $\mathcal{D}$
Output: Score (negation of error)

Split $\mathcal{D}$ into $\mathcal{D}_{train}$ and $\mathcal{D}_{validation}$.
Split $\mathcal{D}_{train}$ into equal-sized $\mathcal{D}^1_{train}, \ldots, \mathcal{D}^N_{train}$;
foreach $i \in 1 \ldots N$ **do**
    Train $pipeline$ on $D^{1 \ldots i}_{train}$;
    $err_{validation} \leftarrow$ Test $pipeline$ on $D_{validation}$;
    if $err_{validation} < err_{best}$ **then**
        $err_{best} \leftarrow err_{validation}$;
    yield -$err_{validation}$;
    $err_{train} \leftarrow$ Test $pipeline$ on $D^{1 \ldots i}_{train}$;
    if $err_{train} > err_{best}$ **then**
        return *-inf*
return -$err_{validation}$

Sampling. *Alpine Meadow* uses Northstar's user-defined sampling strategy, which makes the adaptive pipeline selection (discussed in Section 3.3) trivial to implement.

Caching. Lots of ML pipelines share the same primitives, similar to how data exploration queries share a lot of the same operations. Hence, we can also leverage Northstar's caching features. For example, if we apply transfer learning by using a pre-trained neural network to extract features, we cache its results such that pipelines can avoid redundant computation.

Scheduling. Northstar implements several scheduling strategy, which aim to optimize the overall experience for the user (e.g., getting an initial result quickly is more important than improving a result by including more data). These scheduling techniques turned out to be also useful in an environment, which combines data exploration and model building.

Scalability. As we have mentioned, by decoupling the AutoML and runtime, we are able to scale up by adding more runtime workers. We are also actively working on directly executing workloads in other ML systems, e.g., training neural networks in Tensorflow or PyTorch.

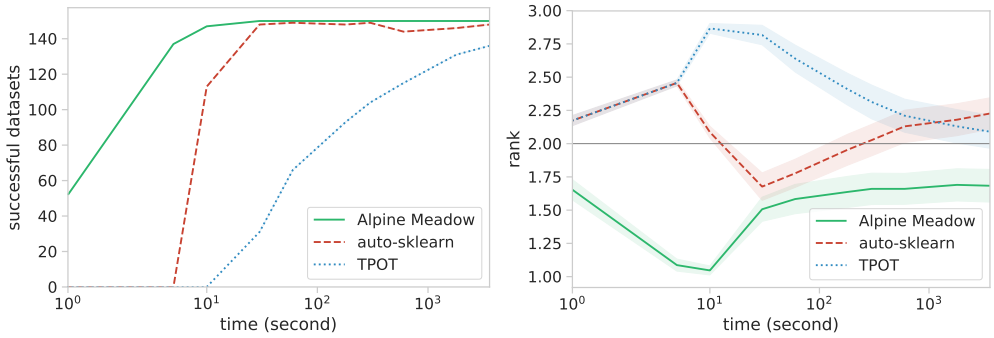
5 Evaluation

Datasets. For the majority of our evaluation, we use the datasets provided by the DARPA D3M competition [1]. DARPA's program on Data Driven Discovery of Models (D3M) has the goal to build tools to automatically build models for a given task with and without human feedback. There are 300 datasets in total: 220 classification datasets, the smallest being 151 records large, the largest being 1025000 records large, and 80 regression datasets, the smallest being 159 and the largest being 89640 records large.

Baselines. We compare against four baselines: (1) hand-made solutions from DARPA: while some DARPA solutions are state-of-the-art highly tuned solutions, others only represent reasonable solutions; a solution a relatively experienced data scientist can manually come up with in a few days; (2) `auto-sklearn` (version 0.4): automatically searched solutions from auto-sklearn [10], which is the state-of-art open-source AutoML system; (3) `TPOT` (version 0.9) : an interactive AutoML system using genetic programming [25]; (4) `Azure` (as of March 2019): Microsoft Azure AutoML (based on [24]). The experiments are restricted to AutoML, while feature engineering and other transformation primitives are not evaluated.

Results. We compare the performances of these systems over time in Figure 4(a) and across datasets in Figure 5. For more experiment results, please refer to our full paper [27].

Figure 4(a) shows when the first result was returned by the individual systems. We note that *Alpine Meadow* is able to return solutions for over a third of the datasets within 1 second and for all datasets after 26 seconds with an average time per dataset of 2.76 seconds. This can be largely contributed



(a) Number of Successful Datasets (b) Ranks

Figure 4: (a) Time to produce first result per dataset (more early results implies better interactivity) (b) Relative rank of the solutions averaged over all datasets (with 95% confidence bands) over time (lower is better).

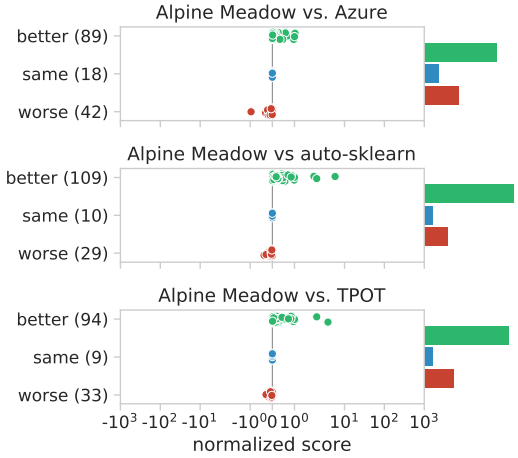


Figure 5: Comparisons of *Alpine Meadow* with different systems across multiple test datasets. Normalized scores are computed as *Alpine Meadow*'s score over the other system's score. Scores are discretized into "better': *Alpine Meadow* outperforms other system, "same": scores are equal, and "worse": *Alpine Meadow* performs worse than other system.

to the adaptive execution strategy. The curve of `auto-sklearn` went down because results were collected from runs of different time limits, so it found a pipeline for some datasets in a run of short time while failed to do so in a run of long time. Second, in Figure 4(b) we show how the relative average rank (over all test datasets) of the three systems evolves over time. Lower rank is better. *Alpine Meadow* consistently holds the best rank throughout the entire time span.

The discretized results of comparison between different systems on all datasets are summarized in Figure 5. Overall, *Alpine Meadow* outperforms or equals `Azure` in 70% of the datasets, 79% for `auto-sklearn` and 74% for `TPOT`.

6 Conclusion

We have discussed a new system for Interactive Automated Machine Learning. This low-latency automatic selection, based on Multi-Armed Bandits, Bayesian Optimization and Meta-Learning theory, efficiently explores the pipeline search space and enables domain experts to bring value to the optimization process. Besides, we decouple the runtime from the AutoML component and employ sampling and caching to execute efficiently for ML workloads, and we introduce interactivity-aware scheduling and apply parallelism extensively to further reduce the response time. We have tested *Alpine Meadow* on datasets with very heterogeneous characteristics, from sample size to feature types. Our experiments show that when compared against state-of-the-art systems or expert-solutions, *Alpine Meadow* generally generates better results in a shorter amount of time. Please refer to `http://www.einblick.ai/assets/northstar_demo.mp4` for a demo of how Alpine Meadow can be used in an interactive visual environment.

References

[1] Data-driven discovery of models (d3m). https://www.darpa.mil/program/data-driven-discovery-of-models.

[2] Weka. https://www.cs.waikato.ac.nz/ml/weka/.

[3] Martín Abadi et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[4] Marcin Andrychowicz et al. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

[5] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[6] James Bergstra, Daniel Yamins, and David Daniel Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. 2013.

[7] Carsten Binnig, Benedetto Buratti, Yeounoh Chung, Cyrus Cousins, Tim Kraska, Zeyuan Shang, Eli Upfal, Robert Zeleznik, and Emanuel Zgraggen. Towards interactive curation & automatic tuning of ml pipelines. In *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, page 1. ACM, 2018.

[8] Sébastien Bubeck, Nicolo Cesa-Bianchi, et al. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning*, 5(1):1–122, 2012.

[9] Andrew Crotty, Alex Galakatos, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. Vizdom: Interactive analytics through pen and touch. *Proceedings of the VLDB Endowment*, 8(12):2024–2027, 2015.

[10] Matthias Feurer et al. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, pages 2962–2970, 2015.

[11] Matthias Feurer et al. Initializing bayesian hyperparameter optimization via meta-learning. In *AAAI*, pages 1128–1135, 2015.

[12] Daniel Golovin et al. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1487–1495. ACM, 2017.

[13] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

[14] Lilong Jiang, Michael Mandel, and Arnab Nandi. Gesturequery: A multitouch database query interface. *Proceedings of the VLDB Endowment*, 6(12):1342–1345, 2013.

[15] Lars Kotthoff et al. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *The Journal of Machine Learning Research*, 18(1):826–830, 2017.

[16] Tim Kraska. Northstar: an interactive data science system. *Proceedings of the VLDB Endowment*, 11(12):2150–2164, 2018.

[17] Neil Lawrence and Raquel Urtasun. Non-linear matrix factorization with gaussian processes. *Proceedings of the International Conference on Machine Learning*, 2009.

[18] Doris Jung-Lin Lee et al. Accelerating scientific data exploration via visual query systems. *arXiv preprint arXiv:1710.00763*, 2017.

[19] Lisha Li et al. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560*, 2016.

[20] Tian Li et al. Ease. ml: towards multi-tenant resource sharing for machine learning workloads. *Proceedings of the VLDB Endowment*, 11(5):607–620, 2018.

[21] Gang Luo. A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *Network Modeling Analysis in Health Informatics and Bioinformatics*, 5(1):18, 2016.

[22] Hector Mendoza et al. Towards automatically-tuned neural networks. In *Workshop on Automatic Machine Learning*, pages 58–65, 2016.

[23] Arnab Nandi. Querying without keyboards. In *CIDR*, 2013.

[24] Melih Huseyn Elibol Nicolo Fusi, Rishit Sheth. Probabilistic matrix factorization for automated machine learning. *arXiv preprint arXiv:1804.05892*, 2018.

[25] Randal S. Olson et al. *EvoApplications 2016*, chapter Automating Biomedical Data Science Through Tree-Based Pipeline Optimization, pages 123–137. 2016.

[26] F. Pedregosa et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[27] Zeyuan Shang, Emanuel Zgraggen, Benedetto Buratti, Ferdinand Kossmann, Philipp Eichmann, Yeounoh Chung, Carsten Binnig, Eli Upfal, and Tim Kraska. Democratizing data science through interactive curation of ml pipelines. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1171–1188. ACM, 2019.

[28] Tarique Siddiqui et al. Effortless data exploration with zenvisage: an expressive and interactive visual analytics system. *Proceedings of the VLDB Endowment*, 10(4):457–468, 2016.

[29] Evan R. Sparks et al. Automating model search for large scale machine learning. In *Proceedings of the Sixth ACM Symposium on Cloud Computing, SoCC 2015, Kohala Coast, Hawaii, USA, August 27-29, 2015*, pages 368–380, 2015.

[30] Evan R Sparks et al. Automating model search for large scale machine learning. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, pages 368–380. ACM, 2015.

[31] Evan R. Sparks et al. Tupaq: An efficient planner for large-scale predictive analytic queries. *CoRR*, abs/1502.00068, 2015.

[32] Richard S Sutton, Andrew G Barto, Francis Bach, et al. *Reinforcement learning: An introduction*. MIT press, 1998.

[33] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855. ACM, 2013.

[34] Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

[35] Doris Xin et al. Accelerating human-in-the-loop machine learning: Challenges and opportunities. *arXiv preprint arXiv:1804.05892*, 2018.

[36] Emanuel Zgraggen et al. (sl qu) eries: Visual regular expressions for querying and exploring event sequences. 2015.

[37] Emanuel Zgraggen et al. How progressive visualizations affect exploratory analysis. *IEEE Transactions on Visualization & Computer Graphics*, (8):1977–1987, 2017.

[38] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

A Related Work

AutoML Systems: Most automated ML systems focus on automated learning algorithm selection and hyper-parameter tuning [30, 5, 21, 20, 38, 12, 7] to make machine learning curation fully automated for non-ML experts. Spark TuPAQ [30] and Hyperband [19] use variations of multi-armed bandit (MAB) algorithm to better allocate computational resources for hyper-parameter tuning. However, their search space is limited to hyper-parameter sets for a few (often, user specified) learning algorithms. The output ML pipeline is not practical in that the real-world problems require end-to-end pipeline curation with careful feature engineering/selection and data transformation. One major drawback of MAB-based approach is that the number of arms (a unique configuration/pipeline) explodes with the size of the search space, and the total number of arms can easily exceed the memory size for a full search space with models, hyper-parameters and pre-processors.

Auto-WEKA [33, 15] or its sister package Auto-sklearn [10] solves the problem of learning algorithm selection and their associated hyper-parameter optimization in a combined search space. They also consider various feature selection and data transformation methods to generate end-to-end ML pipelines. Auto-WEKA uses Sequential Model-based Algorithm Configuration (SMAC) to explore the large search space, which is partly discrete and conditional as each selected algorithm has a different set of associated hyper-parameters. The idea is that, instead keeping track of all the possible configurations, the search moves towards a more promising region based on the previous search and evaluation results. Unfortunately, standalone SMAC optimization for the large search space can still run for hours if not days. In addition, Auto-WEKA and its search space construction is limited to classification and regression problems only.

TPOT [25] is a tree-based pipeline optimization tool using genetic programming while requiring little to no input nor prior knowledge from the user. Microsoft has recently introduced an AutoML tool via Azure, based on the work of [24]. They build predictive ML pipelines combining collaborative filtering and Bayesian Optimization (BO). In particular they model the *search space* as probability distribution defined by a Probabilistic Matrix Factorization [17] and than use expected improvement as acquisition function to choose the most promising pipelines.

In *Alpine Meadow*, we combine BO with MAB to construct more compact (and dense) search space for Bayesian Optimization, which results in more accurate and efficient search. Additionally, the current implementation can work with existing (WEKA [2] and Scikit-learn [26]) ML libraries as well as custom ML primitives for more complex problems. As a result, *Alpine Meadow* can support more complex problem types (e.g., graph matching, image and audio classification, etc.), and more importantly, *Alpine Meadow* finds a comparable ML pipeline much more efficiently and can progressively improve the quality of the pipeline.

The interactivity aspect differentiates *Alpine Meadow* from other systems: we design time-based cost models preferring fast pipelines early on, incremental training pipelines, and pipeline early termination to provide better interactivity.

Human-In-The-Loop Data Analytics: There are tools and systems that focus on the human-in-the-loop aspect of data science. Hellix aims to accelerate the iterative ML model training with responsive user feedback [35]. Vizdom [9] provides a unique pen-and-touch interface for the user to easily construct ML workflows and interactively refine the analytics/ML pipelines. Most industry cloud ML services, such as TensorFlow [3], Amazon SageMaker and Azure Machine Learning [24], fall into this category, in that they provide fully-managed environment for ML applications. Unlike systems, the focus is not automated end-to-end pipeline curation; the services provide programmable APIs or web-based interface for ML workflow construction and managed computing resources for the deployment. *Alpine Meadow* targets domain experts or users without ML expertise, and instead of requiring the user to construct a working ML workflow with selected algorithms and pre-processors, the system generates one based on the problem description and the data.

Neural Architecture Search: Also related to our work is neural architecture search, in that we consider deep neural networks as one of the learning algorithms. *Alpine Meadow* currently uses transfer learning [34], a general framework for re-using models leaned in one task for other tasks, in order to quickly train a neural architecture model. This limits the search space to a fixed architectures (e.g., the depth and width of hidden layers, skip connections). Neural networks are hard to design from scratch, and there are many proposed solution using similar Bayesian Optimization [22, 6] or

Reinforcement Learning techniques [38]. In the future, we will integrate some of the automated neural architecture design techniques for the tasks where deep learning is known to perform best.