Infinite Hyperparameter Optimization of Directed Acyclic Graph Machine Learning Pipelines

Chapman Siu

Abstract

In the past decade machine learning has been successfully applied in a wide variety of fields, leading to a growing demand for machine learning systems to be used even by non-experts. In this paper we introduce an approach for optimizing potentially infinitely many hyperparameters over machine learning pipelines. In particular we demonstrate that pipelines that are constructed can discover new novel pipeline and generate synthetic feature constructors. We compare with other existing approaches and demonstrate extensions and suggest future research questions.

Introduction

Machine Learning services have grown in many fields, driving demand for machine learning systems that can be used by both non-experts and experts alike. As a consequence, there has been an increase in number of commercial enterprises offering machine learning system. Each of these services rely on automatically determining the optimal model, this includes the choice of optimal machine learning algorithm, preprocessing, feature selection and all of the associated hyperparameters.

In this paper we will consider the task of automated machine learning (AutoML) to determine the optimal pipeline for the target problem. As features get transformed into many different representations, we must be comfortable in both searching over a large (possibly infinite) parameter space of possible representations, and selecting the feature representation which is ideal for the problem at hand.

This work makes the following contributions:

- Extend previous approaches of optimization using tree-based pipelines to directed acyclic graphs
- Demonstrate approaches for performing search over infinite large space using Determinantal Point Processes and Random Forest algorithms
- Propose particle filtering as an approach to dynamically generate and create new machine learning pipelines

Related Work

The tuning of models has historically been completed through optimizing parts of the pipeline at a time. For example, grid search algrotihms have been used to determine the optimal set of model parameters through brute force search. It has been shown that randomly search is more efficienty that an exhaustive search. Furthermore Bayesian optimization of model parameters has been explored and been more effective than tuning parameters by experts.

In recent work, TPOT has developed an autoML system which uses generic algorithms in order to blend and discover new pipeline combinations, `auto-sklearn` has also been developed and shown to be competitive in machine learning competition which operates

Preprint. Work in progress.

over Bayesian optimization to dsicver ideal combination of pipelines. Genetic programming has also been used (Zutty et al.) to develop better pipelines than humans for supervised classification tasks.

All of these approaches demonstrate intelligent systems being able to design machine learning pipelines, which makes machine learning more accessible and may save practitioners time by automating part of the machine learning process.

Framework

In this section, we describe our approach and constrast it with other attempts at building machine learning pipelines. We begin this section by describing basic pipeline operators, and the computational underpinning in order to ensure they can all be combined for machine learning problem.

Pipeline Operators as a Directed Acyclic Graph

Machine learning pipelines each can be represented as a *directed acyclic graph*, which is a graph with directed edges in which there are no cycles. More specifically every machine learning is an *interfaced dag*[1]. Interfaced dag, or (I, O)-dag is a generalisation of dag, where I represented the input interface and O is the output interface.

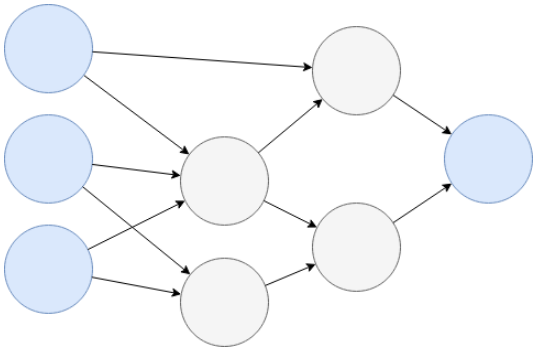

Figure 1: Example of (3-1)-dag

In our setting, the input interface would be our data sources (in this paper we will consider only one input dataset), and our output sources would typically be a single data output. Each node can again be framed as a $(I', 1)$-dag, where it can receive the output of *multiple* operators but will always output a single dataset as the output source. Each of these operators may have a series of hyperparameters which can be tuned or altered. In this scenario, the goal is to generate possible pipelines (combination of operators) in order to create new features or product a machine learning model. As such operators can be thought of as **combinators** and can be parallelized or combined using variety of frameworks and software implementations such as Spark.

Example of operators

Operators can be split between $(1, 1)$-dag and $(N, 1)$-dag. If our pipeline consists of only $(1, 1)$-dag, then our pipeline setup would be equivalent to tree based pipelines[2]. A non-exhaustive list of $(1, 1)$-dag operators could be:

- a machine learning classifier, such a logistic regression model, where the input would be our underlying data, and output would be the associated predictions.
- feature selection algorithm, such a univariate chi-square test, where the input would be our underlying data, and output would be a subset of the underlying data containing only the columns which pass the appropriate statistical test
- feature decomposition algorithm, such as PCA, where the input would be our underlying data, and output would be the PCA representation

On the other hand $(N, 1)$-dag could be operators defined in the Data Science Machine[3] or through ensemble methods:

- Data Science Machine: input is two entity sets and their relationship(s). The output would be a single denormalised dataset, where the parameters are related to how operator is to join and aggregate the two tables
- TPOT Combine Features[2]: input is a series of feature selection or decomposed pipelines, which is combined to a single data set
- Ensemble methods: the output of a series of machine learning classifiers, which is to be placed into another machine learning classifier in variety of ways such as stacking, bagging or blending

Evolving Pipelines

To automatically evolve pipelines from one generation to the next, the particle filter algorithm is used.

Particle Filtering

This problem can also be framed as a particle filtering problem[4].

Algorithm 1 Particle Filtering

Input: $\tilde{S}_t = S_t = \emptyset$, T iterations, M samples per iterations
for $t = 1, ..., T$ **do**
  for $m = 1, ..., M$ **do**
    Sample $x_t^{(m)} \sim P(x_t | u_t, x_{t-1}^{(m)})$
    Assign weights $\tilde{w}_t^{(m)} = P(z_t | x_t^{(m)})$
    $\tilde{S}_t = \tilde{S}_t + \langle x_t^{(m)}, \tilde{w}_t^{(m)} \rangle$
  end for
  for $m = 1, ..., M$ **do**
    Resample by drawing i with probability $\propto w_t^{(i)}$
    add x_t^m to S_t
  end for
end for
Output: S_t

In our formulation at iteration t, u_t refers to the *Intensify* action as described by SMBO framework, and the signal z_t, being the metric of choice for our machine learning algorithm to maximize. The usage of DPP can be applied in the *intensify* action to provide suitable diverse samples to be shown and perturbed. This can be used in conjunction with Bayesian Optimization approaches such as Random Forest Regressors to ensure suitable samples are produced.

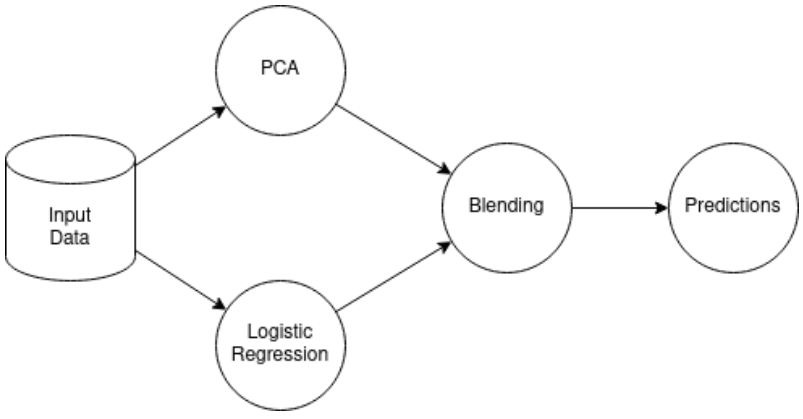

Figure 2: Example of pipeline (to do show how they might evolve, and spawn new ones)

When additional hyperparameters are increased in number through the intensify action, they are further ordered and actioned through either:

- Exploration mode: using random sampling or through determinantal point processes
- Bayesian optimization: using tree based models which have been demonstrated to scale better than gaussian processes and perform well

Learning over Infinite Feature Size

As new pipelines are generated, the size of the hyperparameter space and feature set (with synthetic features) grows as well. In order to efficiently learn over this space, we discuss several naive extensions to algorithms to assist with the problem of learning over an infinite feature size.

Random Forest Regression Trees

An integral part of autoML algorithms has been the use of random forest regression as the basis for hyperparameter optimization[5]. In this setting as new pipelines are generated, naturally the hyperparamter space would grow as well. In order to cater for this, we introduce a simple extension to Random Forest algorithm to allow for this change.

Algorithm 2 Random Forest over growing Feature Sets (RandomForestInfinite)

Input: $\mathbb{D} = \{D_0, D_1, ..., D_n\}$ our datasets, which can be growing in feature size, $ntrees$ number of trees per each base random forest that is grown, $maxtrees$ maximum number of trees $RandomForestInfinite$ can hold, where $maxtrees >= ntrees$.

1: Set $RandomForestInfinite$ to be our model $\mathbb{M}$
2: Train our base random forest M_0 over dataset D_0.
3: Append all trees in M_0 to $\mathbb{M}$
4: **for** $t = 1, ..., n$ **do**
5: Train our base random forest M_t over dataset D_t.
6: Append all trees in M_t to $\mathbb{M}$, taking only the latest $maxtrees$
7: **end for**

Output: S_t

This approach works, as many random forest algorithms are reliant on sampling a subset of the features when fitting each tree. As such, appending trees would be equivalent to previous trees not being able to "sample" the undiscovered hyperparameters of the (then) not generated pipelines. This approach is prone to drift, but retains knowledge and can be used as an alternative to instantiating a new random forest on every new tree evaluation.

Determinantal Point Process

Whereas Random Forest regression trees is used for Bayesian Optimization, Determinantal Point Process can be viewed as a pure exploration method[6], that has demonstrated effectiveness in early iterations. Determinantal point process approach can naturally scale to an unknown parameter size through MCMC approaches, though may suffer when the number of features that are sampled is extremely high.

Sampling k-DPP using MCMC approaches was proposed by [7] with the proof corrected by [8].

Ensemble Methods

Ensemble methods can be easily extended as long as the underlying algorithm can be expanded dynamically. This suggests that bagging can be easily extended through reweighting the predictions, stacking will be supported through superlearners such as grafting[9], and blending approaches can be generated off dynamic neural networks such as DEN framework[10]

Experiments

Here, we run several experiments, involving text and image data to demonstrate ability to learn appropriate representations. We compare our model-free approach with ROAR which is also a model free approach to model based optimization.

Data Sets

Data sets chosen are from the openML repository. This includes datasets which are open as well as simulated datasets (GAMETES)

Algorithm 3 Sampling from a k-DPP using MCMC over infinite dataset

Input: size k

Randomly initialize $Y \subset S$
while not mixed **do**
  Uniformly sample $u \in Y, v \in S \setminus Y$
  set $Y' := Y \cup \{v\} \setminus \{u\}$
  Dynamically create kernel K based on $Y \cup Y'$
  Compute L ensemble via $L = K(I - K)^{-1}$
  if $u \sim U[0,1] < \frac{1}{2} \min\{1, \frac{\det(L_{Y'})}{\det(L_Y)}\}$ **then**
    set $Y := Y'$
  end if
end while

Output: Y

Table 1: My caption

model	KDDCup09_appetency	MagicTelescope	OVA_Breast	adult
daft	0.8228	0.8897	0.9714	0.9020
daft_bayes	0.8226	0.9210	0.9711	0.8946
daft_random	0.7808	0.9225	0.9805	0.8982
daft_stack	0.8001	0.9170	0.9753	0.8938
daft_stack_bayes	0.6605	0.9232	0.9741	0.8777
daft_stack_random	0.8036	0.9194	0.9744	0.8564
logistic	0.5007	0.7723	0.9154	0.5000
randomforst	0.7730	0.9313	0.9795	0.8833
svm	0.5010	0.6977	0.9116	0.5000
tpot	0.7611	0.9223	0.9681	0.8884

Results

To assess the results, we make comparisons using the TPOT library[11], the default random forest with 500 trees, logistic regressoin, svm models and other proposed pipelines such as PCA with logistic regression and our approach, with and without stacking . For comparisons with TPOT, we shall restrict running time and number of evaluations to be the same as our model, using two different configurations, one configuraton which runs for a very short time, and another which runs for longer, to demonstrate the efficacy of our approach.

Again all datasets are stratified into 70% training and 30% testing, where training only occurs on the training set and testing set results are shown here.

Conclusion

In this paper, we have presented a new algorithm for model optimization leveraging particle filtering and dynamic expandable neural networks. This machine learning pipeline can learn

Table 2: My caption

model	covertype	fri_c1_1000_25	pc4	quake	sick
daft	0.7347	0.9464	0.9289	0.5321	0.9918
daft_bayes	0.8306	0.9656	0.9370	0.5147	0.9943
daft_random	0.9944	0.9489	0.9214	0.5178	0.9964
daft_stack	0.8565	0.9324	0.9368	0.5183	0.9894
daft_stack_bayes	0.8497	0.9653	0.9132	0.4969	0.9948
daft_stack_random	0.9332	0.9656	0.9125	0.5282	0.9917
logistic	0.8246	0.7490	0.5756	0.4982	0.9302
randomforst	0.9935	0.9669	0.9287	0.5238	0.9972
svm	0.7620	0.6472	0.5592	0.5070	0.7112
tpot	0.8184	0.9635	0.9282	0.5001	0.9885

dynamically over a large range of possible pipelines and have demonstrated to be competitive to other approaches.

References

[1] M. Fiore and M. D. Campos, "The algebra of directed acyclic graphs," in *Computation, logic, games, and quantum foundations. the many facets of samson abramsky*, Springer, 2013, pp. 37–51.

[2] R. S. Olson, R. J. Urbanowicz, P. C. Andrews, N. A. Lavender, L. C. Kidd, and J. H. Moore, "Applications of evolutionary computation: 19th european conference, evoapplications 2016, porto, portugal, march 30 – april 1, 2016, proceedings, part i," G. Squillero and P. Burelli, Eds. Springer International Publishing, 2016, pp. 123–137.

[3] J. M. Kanter and K. Veeramachaneni, "Deep feature synthesis: Towards automating data science endeavors," in *Data science and advanced analytics (dsaa), 2015. 36678 2015. ieee international conference on*, 2015, pp. 1–10.

[4] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics (intelligent robotics and autonomous agents)*. The MIT Press, 2005.

[5] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in neural information processing systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2962–2970.

[6] T. Kathuria, A. Deshpande, and P. Kohli, "Batched gaussian process bandit optimization via determinantal point processes," in *Advances in neural information processing systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 4206–4214.

[7] B. Kang, "Fast determinantal point process sampling with application to clustering," in *Advances in neural information processing systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 2319–2327.

[8] N. Anari, S. O. Gharan, and A. Rezaei, "Monte carlo markov chain algorithms for sampling strongly rayleigh distributions and determinantal point processes," in *29th annual conference on learning theory*, 2016, vol. 49, pp. 103–115.

[9] S. Perkins, K. Lacker, and J. Theiler, "Grafting: fast, incremental feature selection by gradient descent in function space," *J. Mach. Learn. Res.*, vol. 3, pp. 1333–1356, 2003.

[10] J. Yoon, E. Yang, J. Lee, and S. J. Hwang, "Lifelong learning with dynamically expandable networks," in *International conference on learning representations*, 2018.

[11] R. S. Olson, N. Bartley, R. J. Urbanowicz, and J. H. Moore, "Evaluation of a tree-based pipeline optimization tool for automating data science," in *Proceedings of the genetic and evolutionary computation conference 2016*, 2016, pp. 485–492.