AUTOMATED MACHINE LEARNING AND TASK SPACE NAVIGATION

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

M.S.

by

Ziyang Wu

May 2021

ABSTRACT

Machine learning has become almost ubiquitous in our lives. Training these models have even entered mainstream with the advent of efficient deep learning frameworks and advancements in hardware innovations. There are, however, two noteworthy limitations that prevent practitioners outside the machine learning community from successfully training models that achieve the desirable performance. The first one is the heavy requirement of expert knowledge. There are many components around building a successful machine learning model that each one could impact its final performance. Selecting these components is not easy: it requires trials and errors and a lot of human knowledge! The second is the inefficiency of training deep learning models, especially in the problem picking a good pretraiend feature extractor for a given task. This work tackles both of these problems by incorporating AutoML techniques. As introduced and detailed in later chapters, these frameworks could effectively automate many processes and complexity within training machine learning models.

BIOGRAPHICAL SKETCH

Ziyang Wu was born in Suzhou, Jiangsu Province in China. Ziyang graduacted with *Summa Cum Laude* from Cornell University and received his B.S. in Computer Science and Operation Research. He continued his study to pursue an M.S. in Computer Science, under the supervision of Prof. Bharath Hariharan and Prof. Madeleine Udell.

To my father and mother.

ACKNOWLEDGEMENTS

There are many people I am grateful for during my journey at Cornell.

I am grateful to have my amazing advisors: Professor Bharath Hariharan and Professor Madeleine Udell. Apart from being my role models, they, with their diligence, enthusiasm, and unbounded knowledge have shown me what a true scholar should be like.

I am grateful to close collaborators: Professor Yi Ma from UC Berkeley. His sharpness and insightfulness have given me completely new research directions and perspectives.

I would like to thank my friends and close collaborators: Chengrun Yang, Bram Wallace, Jerry Chee, Christina Baek, Ryan Chan, Yaodong Yu, Simon Zhai, Lijun Ding, Jicong Fan, and Professor Chris De Sa.

I am grateful to my girlfriend, Xinyi, for her continued support throughout these years of study. It would be an unimaginable time without her.

My final gratefulness goes to my parents. Thank you for being my role models for so many years and for your unbounded support and love for me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In this thesis, I present two of my works during my program. This thesis will mainly compose of my research for the past two years. These research projects revolve around the common theme of extending machine learning towards automated machine learning. Specifically, they explore methods that make machine learning more automated and less reliant on expert knowledge.

1.1 Automated Machine Learning

Automated machine learning (AutoML) tries to alleviate expert knowledge required for ML practices. For a given dataset, an ML pipeline comprises of learning components such as imputation, dimensionality reduction and estimation that together map input data to output predictions. Each component could significantly affect performance of the trained model, thus requiring substantial expert knowledge. It is prohibitively expensive to evaluate on all possible pipelines due to the combinatorial structure of the problem. To avoid costly enumeration, we found that a surrogate model based on low rank tensors was able to automate the pipeline selection process and achieve competitive results. We collected a large number of training datasets (mostly tabular) and fitted different pipelines on them in an offline stage, yielding a tensor of pipeline performance with each dimension corresponding to one learning component. The underlying assumption is that the obtained tensor has low rank structure. By applying factorization models on this tensor, we were able to get concrete embeddings of each component. When encountering a new dataset, we showed that only a modest number of pipelines needs to be evaluated and performance on other pipelines can be extrapolated efficiently from

the embeddings thanks to tensor completion. We have summerized the work described above in Chapter 2 and published in KDD 2020.

1.2 Expert Model Selection

One other related project has also been conducted on the domain of imagery data. In this project, I studied the problem of expert model selection, which aims to obtain optimal performance on a given task by selecting and then fine-tuning a suitable feature extractor(i.e. expert) from a large pool of candidate experts. These experts are trained on different domains and transfer performance is therefore determined by how ”close” they are to the target task. We thus identified the key to an automated selection scheme: a quantitative characterization of any task so that similarity between tasks can be measured concretely and efficiently. Prior works[2] on this topic often relied on a separate, generic expert (e.g. pretrained on ImageNet), also known as a probe network, that contains semantic information about the task at hand. To alleviate the reliance on a probe network, we proposed a new method inspired by the Neural Tangent Kernel (NTK) theory, which proves that a wide enough network simplifies to a linear model with a fixed kernel determined by the *distribution* of model parameters and data. As each kernel value can be interpreted as a similarity measure between two datapoints, I extended the notion of NTK to operate between tasks by treating each task as a datapoint. This formulation naturally leads to a kernel over tasks, named Task Tangent Kernel (TTK), with each entry measuring the distance between two tasks. Apart from removing the dependence on a carefully chosen probe network, extensive experiments have validated that TTK indeed contains rich signals for model selection. When evaluated under symmetric distance (e.g. cosine similarity), TTK surprisingly outperforms previous methods by over 20 percent, a huge increase all with fewer assumptions and less engineering. We

have summerized the work described above in Chapter 3 and published in CVPR 2021.

CHAPTER 2

AUTOMATED MACHINE LEARNING PIPELINE SELECTION

2.1 Introduction

A machine learning pipeline is a directed graph of learning components including imputation, encoding, standardization, dimensionality reduction, and estimation, that together define a function mapping input data to output predictions. Each component may also include hyperparameters, such as the output dimension of PCA, or the number of trees in a random forest. Simple pipelines may consist of sequences of these components; more complex pipelines may combine inputs to form pipelines with more complex topologies. An example pipeline is shown as Figure 2.1.

The job of a data scientist facing a new supervised learning problem is to choose the pipeline that yields a low out-of-sample error from among all possible pipelines. This task is challenging. First, no component dominates all others: there is "no free lunch" [84]. Rather, each performs well on certain data distributions. For example, the PCA dimensionality reducer works well on data points in $\mathbb{R}^d$ that roughly lie in a low rank subspace $\mathbb{R}^k$ with $k < d$; the feature selector that keeps features with large

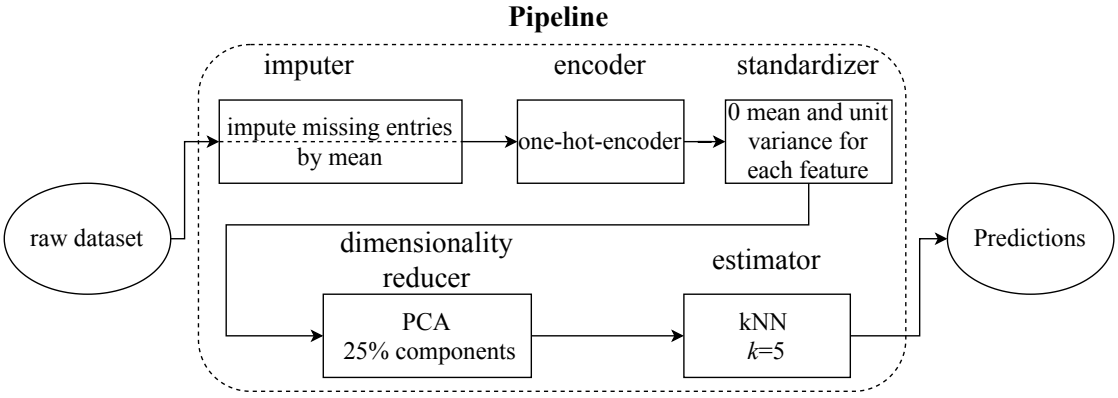


Figure 2.1: An example pipeline.

variances works well on datasets if such features are more informative; the Gaussian naive Bayes classifier works well on features with normally distributed values in each class. However, it is difficult to check these distributional assumptions without running the component on the data: an expensive proposition! The second is the dependence of these choices: for example, standardizing the data may help some estimators, and harm others. Moreover, as the number of possible machine learning components grows, the number of possibilities grows exponentially, defying enumeration. Automating the selection of a pipeline is thus an important problem, which has received attention both from academia and industry [58, 19, 15, 51].

Human experts tackle this difficulty by choosing the right combination according to their domain knowledge. However, finding the right combination takes substantial expertise, and still requires several model fits to find the right combination of components and hyperparameters. An automated pipeline construction system, like a human expert, first forms a *surrogate model* to predict which pipelines are likely to work well. Surrogate models are meta-models that map dataset and machine learning model properties to quantities that characterize performance or informativeness.

A good surrogate model enables efficient search through the pipeline space. "All models are wrong, but some are useful [6]": a good surrogate model makes predictions that guide the search for pipelines without the need for many model fits. Auto-sklearn [19] and Alpine Meadow [66] use meta-learning [71, 3, 45, 76] to choose promising pipelines from those that perform the best on neighboring datasets, and use Bayesian optimization to fine-tune hyperparameters. TPOT [58] uses genetic programming to search over pipeline topologies. Alpine Meadow [66] uses multi-armed bandit to balance the exploration and exploitation of pipeline structures. In this work, we use a low multilinear rank tensor as our surrogate model. This model makes explicit use of the

combinatorial structure of the problem: as a result, the number of pipeline evaluations required to fit the surrogate model on a new dataset is modest, and independent of the number of pipeline components.

Our system learns the surrogate model for a new dataset by fitting a few pipelines on it. The problem of which pipelines to evaluate first, in order to predict the effectiveness of others, is called the *cold-start problem* in the literature on recommender systems. This problem is also of great interest to the AutoML community. Proximity in meta-features, "simple, statistical or landmarking metrics to characterize datasets [86]", are used by many AutoML systems [61, 20, 19, 22, 66] to select models that work well on neighboring datasets, with the belief that models perform similarly on datasets with similar characteristics. Probabilistic matrix factorization has been used to extract dataset latent representations from pipeline performance [22]. Other dataset and pipeline embeddings have also been proposed that use pipeline performance or even textual dataset or algorithm descriptions to build surrogate models [82, 86, 16].

In this work, we build pipeline embeddings by fitting a tensor decomposition to the (incompletely observed) tensor of pipeline performance on a set of training datasets. The tensor model is easy to extend to a new dataset by fitting a constant number of pipelines on it. We describe a simple rule to select which pipelines to observe by solving a constrained version of the classical experiment design [78, 36, 63, 7] problem using a greedy heuristic [52].

We consider the following concrete challenge in this work: select several pipelines that perform the best within a given time limit for a new dataset, in the case that we already know or have time to collect pipeline performance on some existing datasets. We focus on small data and traditional supervised machine learning pipelines in our experiments, although the methodology can be generalized to a wider range of disciplines.

Our main technical contributions are: a new tensor model to exploit the combinatorial pipeline performance structure, and a new pipeline search mechanism that builds on ideas from greedy experiment design. Together, these ideas yield a new state-of-the-art system for AutoML pipeline selection. Since OBOE [86] is an AutoML system that selects machine learning models by matrix factorization, we name our system in this work TENSOROBOE: the AutoML system that uses tensor decomposition to select pipelines.

This chapter is organized as follows. Section 2.2 introduces notation and terminology. Section 2.3 describes the main ideas used efficiently search the pipeline space. Section 2.3.1 gives details on TENSOROBOE. Section 2.4 shows experimental results.

2.2 Notation and Terminology

Meta-learning. Meta-learning, also called "learning to learn", uses results from past tasks to make predictions or decisions on a new task. In our setting, we learn from a corpus of datasets called *meta-training* datasets by fitting pipelines to these datasets in an offline stage; the new dataset, which requires a fast recommendation for a pipeline, is called the *meta-test* dataset.

Model. A *model* $\mathcal{A}$ is a specific combination of algorithm and hyperparameter settings, e.g. k-nearest neighbors with $k = 3$.

Pipeline component. A pipeline component is a model or model type. Examples include missing entry imputers, dimensionality reducers, supervised learners, and data visualizers. We consider the following components in this work:

- *Data imputer*: A preprocessor that fills in missing entries.

- *Encoder*: A transformer that converts categorical features to numerical codes. Here, we consider encoding categoricals as integers or with a one-hot encoder.
- *Standardizer*: A standardizer centers and rescales data.
- *Dimensionality reducer*: A transformer that reduces the dimensionality of the dataset by either creating new features (like PCA) or subsampling features.
- *Estimator*: The supervised learner. For the classification tasks in this work, estimators are classifiers.

Linear algebra. Our work follows the notation of [86] and [41]. We define $[n] = \{1, \ldots, n\}$ for $n \in \mathbb{Z}$, and denote *vector*, *matrix*, and *tensor* variables respectively by lowercase letters (x), capital letters (X) and Euler script letters ($\mathcal{X}$). The order of a tensor is the number of dimensions; matrices are order-two tensors. Each dimension is called a mode. Throughout this work, all vectors are column vectors. To denote a part of matrix or tensor, we use a colon to denote the dimension that is not fixed: given a matrix $A \in \mathbb{R}^{m \times n}$, $A_{i,:}$ and $A_{:,j}$ (or a_j) denote the ith row and jth column of A, respectively. A fiber is a one-dimensional section of a tensor $\mathcal{X}$, defined by fixing every index but one; for example, one fiber of the order-3 tensor $\mathcal{X}$ is $X_{:jk}$. Fibers of a tensor are analogous to rows and columns of a matrix. A slice is an $(N-1)$-dimensional section of an order-N tensor $\mathcal{X}$. The mode-n matricization of $\mathcal{X}$, denoted as $\mathcal{X}^{(n)}$, is a matrix whose columns are the mode-n fibers of $\mathcal{X}$. $\mathcal{X}$ has *multilinear rank* $(r_1, r_2, \ldots)$ if r_n is the rank of $\mathcal{X}^{(n)}$. For example, given an order-3 tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, we have $X^{(1)} \in \mathbb{R}^{I \times (J \times K)}$, and $\mathcal{X}$ has multilinear rank (r_1, r_2, r_3) if $\mathcal{X}^{(n)}$ has rank r_n for $n \in [3]$. We denote the *n-mode product* of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots I_N}$ with a matrix $U \in \mathbb{R}^{J \times I_n}$ by $\mathcal{X} \times_n U \in \mathbb{R}^{I_1 \times \cdots I_{n-1} \times J \times I_{n+1} \times \cdots I_N}$; the $(i_1, i_2, \ldots, i_{n-1}, j, i_{n+1}, \ldots, i_N)$-th entry is $\Sigma_{i_n=1}^{I_n} x_{i_1 i_2 \cdots i_{n-1} i_n i_{n+1} \cdots i_N} u_{j i_n}$. Given two tensors with the same shape, we use $\odot$ to denote their entrywise product. Given an ordered set $\mathcal{S} = \{s_1, \ldots, s_k\}$ where $s_1 < \ldots < s_k \in [n]$, we write $A_{:\mathcal{S}} = [A_{:,s_1}, A_{:,s_2}, \ldots, A_{:,s_k}]$; given

an ordinary set S, we use $A_{:S}$ to denote $A_{:\mathcal{S}}$, in which $\mathcal{S}$ is the ordered version of set S.

Pipeline performance. The performance of a machine learning pipeline is usually characterized by cross-validation error. Given a dataset $\mathcal{D}$ and a pipeline $\mathcal{P}$, we denote the error of $\mathcal{P}$ on $\mathcal{D}$ as $\mathcal{P}(\mathcal{D})$. It is common practice to evaluate this error by cross-validating $\mathcal{P}$ on $\mathcal{D}$ with a certain number of folds (often 3, 5 or 10) and a fixed dataset partition. We use $\mathcal{P}(\mathcal{D})$ to denote the cross-validation error we observe with a certain number of folds and a certain partition.

Error tensor and error matrix. Pipeline errors on training datasets form an *error tensor*, which we denote as $\mathcal{E}$. In our experiments, $\mathcal{E}$ is an order-6 tensor, with 6 modes corresponding to datasets, imputers, encoders, standardizers, dimensionality reducers and estimators, respectively. The $(i_1, i_2, \ldots, i_6)$-th entry of $\mathcal{E}$ is the error of the pipeline formed by composing the i_2-th imputer, i_3-th encoder, i_4-th standardizer, i_5-th dimensionality reducer, and i_6-th estimator and evaluating this pipeline on the i_1-th dataset. If a pipeline-dataset combination has been evaluated, we say the corresponding entry in the error tensor $\mathcal{E}$ is observed. The first unfolding of the error tensor, $\mathcal{E}^{(1)}$, is called the *error matrix* E, whose (i, j)-th entry $E_{ij} = \mathcal{P}_j(\mathcal{D}_i)$ is the error of pipeline j on dataset i.

Ensemble. An ensemble [14, 8, 65, 83] combines a finite set of individual machine learning models into a single prediction model. For simplicity, the combination method we use is majority voting for classification. We define the *candidate learner* to be individual machine learning pipelines that we select from to create the ensemble, and *base learner* to be pipelines that are included in the ensemble. An ensemble of pipelines is itself a pipeline, but not a simple linear pipeline. By creating ensembles of linear pipelines, TENSOROBOE can perform better than any linear pipeline.

2.3 Methodology

2.3.1 Overview

TENSOROBOE has two phases. In the offline phase, we compute the performance of pipelines on meta-training datasets to build a tensor surrogate model. In the online phase, we run a small number of pipelines on the new meta-test dataset to specialize the surrogate model and identify promising pipelines.

Offline Stage. We collect a partially observed error tensor using the approach described in Section 2.3.2 to limit the total runtime of the offline phase. We complete and decompose the error tensor $\mathcal{E}$ using the EM-Tucker algorithm, shown as Algorithm 1, with dataset and estimator ranks empirically chosen to be the ones that give low reconstruction error, described in Section 2.4.2.

Online Stage. Online, given a new dataset $\mathcal{D}$ with $n^{\mathcal{D}}$ data points and $p^{\mathcal{D}}$ features, we first predict the running time of each pipeline by a simple model: order-3 polynomial regression on $n^{\mathcal{D}}$ and $p^{\mathcal{D}}$ and their logarithms. This simple model works well because the time to fit the estimator dominates the time to fit the pipeline, and the theoretical complexities of estimators we use have no higher order terms [34, 86].

The initial dataset and estimator ranks are set to the number of principal components that capture 97% of the energy in the respective tensor matricizations. We double the runtime budget at each iteration and increment the estimator rank if the performance improves. In each iteration, we build ensembles whose base learners are the 5 pipelines with the best cross-validation error. An ensemble can improve on the performance of the best base pipeline. An example is shown as Figure 2.2.

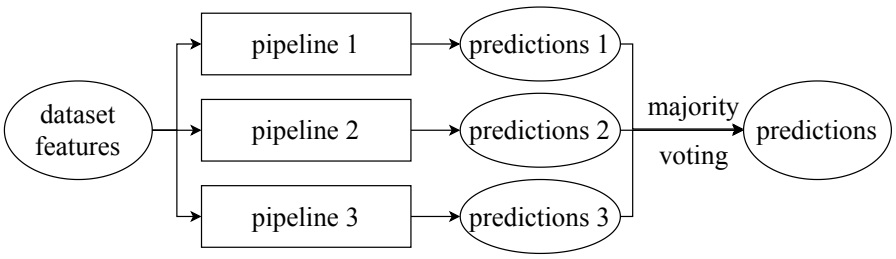


Figure 2.2: A pipeline ensemble with 3 base learners.

2.3.2 Tensor Collection for Meta-Training

In the meta-training phase of meta-learning, meta-training data is generally assumed to be already available or cheap to collect. Given the large number of possible pipeline combinations, though, collecting meta-training data can be prohibitively expensive. As an example, even if it takes one minute on average to evaluate each pipeline on each dataset, evaluating $20,000$ pipelines on 200 meta-training datasets would take more than 7 years of CPU time. This motivates us to use tensor completion to limit the time spent on the collection of meta-training data, while preserving accuracy of our surrogate model.

We collect pipeline performance in a biased way: using 3-fold cross-validation, we only evaluate pipelines that complete within 120 seconds. This rule gives a missing ratio of 3.3%. Notice that the entries are not missing uniformly at random: for example, some datasets are large and expensive to evaluate; our training data systematically lacks data from these large datasets. Nevertheless, we will show how to infer these entries using tensor completion in Section 2.3.3, and demonstrate in Section 2.4.2 that the method performs well despite bias.

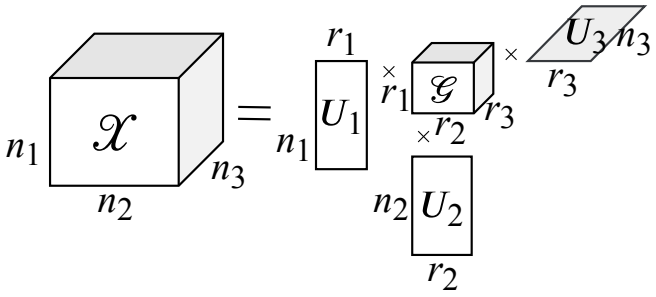


Figure 2.3: Tucker decomposition on an order-3 tensor.

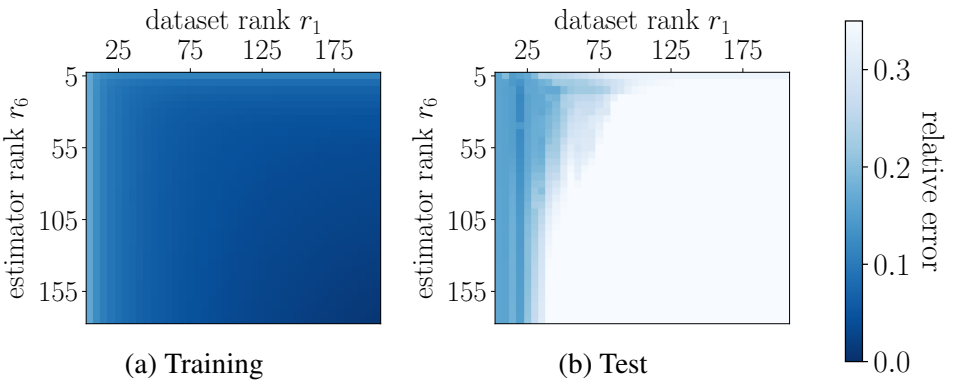
2.3.3 Tensor Decomposition and Rank

The meta-training phase constructs the error tensor $\mathcal{E}$. In the meta-test phase, we see a new dataset, corresponding to a new slice of $\mathcal{E}$. To learn about the slice efficiently, we use a low rank tensor decomposition to predict all the entries in this slice from a subset of its informative entries.

Unlike matrices, there are many incompatible notions of tensor ranks and low rank tensor decompositions, including CANDECOMP/PARAFAC (CP) [9, 30], Tucker [73], and tensor-train [59]. Each emphasizes a different aspect of the tensor low rank property. In this work, we use Tucker decomposition; an illustration on an order-3 tensor is shown as Figure 2.3. As a form of higher-order PCA, Tucker decomposes a tensor into the product of a *core tensor* and several *factor matrices*, one for each mode [41]. A tensor with low multilinear rank has a low rank Tucker decomposition. In our setting of order-6 tensors, Tucker decomposition of $\mathcal{E}$ is

$$\mathcal{E} \approx \hat{\mathcal{E}} = \mathcal{G} \times_1 U_1 \times \cdots \times_6 U_6, \tag{2.1}$$

with core tensor $\mathcal{G} \in \mathbb{R}^{r_1 \times r_2 \times \cdots \times r_6}$ and column-orthonormal factor matrices $U_i \in \mathbb{R}^{n_i \times r_i}$, $i \in \{1, 2, \ldots, 6\}$. $\hat{\mathcal{E}}$ is linear in the factor matrices. Each factor matrix can thus be viewed as embedding the corresponding dataset or pipeline component, with pipeline embeddings as columns of $Y = (\mathcal{G} \times_2 U_2 \times \cdots \times_6 U_6)^{(1)} \in \mathbb{R}^{r_1 \times (\prod_{i=2}^{6} n_i)}$, the mode-1 matricization of the



(a) Training (b) Test

Figure 2.4: Relative error heatmaps when varying ranks in dataset and estimator dimensions. Here, training entries are the ones with runtime less than 90 seconds; the test entries are the ones with runtime between 90 and 120 seconds.

product. We can use this observation to approximately factor the error matrix E, using Equation 2.1, as

$$X^\top Y \approx E \in \mathbb{R}^{n_1 \times (\Pi_{i=2}^{6} n_i)}, \tag{2.2}$$

in which $X \in \mathbb{R}^{r_1 \times n_1}$ and $Y \in \mathbb{R}^{r_1 \times (\Pi_{i=2}^{6} n_i)}$ are dataset and pipeline embeddings, respectively.

Figure 2.4 shows the low rank Tucker decomposition fits the error tensor well.

2.3.4 Tensor Completion

To infer missing entries in the error tensor we collected, namely the entries that take more than the time threshold to evaluate, we use the expectation-maximization (EM) [13, 69] approach together with Tucker decomposition in each step, which we call EM-Tucker and present as Algorithm 1.

In Algorithm 1, $\mathbf{\Omega}$ is a binary tensor that denotes whether each entry of the error tensor $\mathcal{E}$ is observed or not. $\mathbf{\Omega}$ has the same shape as the original error tensor, with the corresponding entry $\mathbf{\Omega}_{i_1,i_2,...,i_n} = 1$ if the $(i_1, i_2, ..., i_n)$-th entry of the error tensor

Algorithm 1 EM-Tucker algorithm for tensor completion

Input: order-n error tensor $\mathcal{E}$ with missing entries, target multilinear rank $[r_1, \ldots, r_n]$
Output: imputed error tensor $\mathcal{E}$

$\mathcal{E}_{\text{obs}} \leftarrow \mathcal{E}$
$\boldsymbol{\Omega} \leftarrow$ observed entries in $\mathcal{E}_{\text{obs}}$
do
    $\mathcal{G}, \{U_i\}_{i=1}^n \leftarrow \text{Tucker}(\mathcal{E}, \text{ranks}=[r_1, \ldots, r_n])$
    $\mathcal{E}_{\text{pred}} \leftarrow \mathcal{G} \times_1 U_1 \times \cdots \times_n U_n$
    $\mathcal{E} \leftarrow \boldsymbol{\Omega} \odot \mathcal{E}_{\text{obs}} + (1 - \boldsymbol{\Omega}) \odot \mathcal{E}_{\text{pred}}$
while not converged

is observed, and 0 otherwise. The algorithm is regarded to have converged when the decrease of relative error is less than 0.1%.

Why bother with tensor completion? To recover the missing entries of a tensor, we can also perform matrix completion after matricization or perform matrix completion on every slice separately. Tensors are more constrained and so provide better fits to sparse and noisy data. Consider a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots I_N}$ with multilinear rank $[r_1, r_2, \ldots, r_n]$, where $I_1 = I_2 = \ldots = I_n = I$ and $r_1 = r_2 = \ldots = r_n = r$. The number of degrees of freedom of $\mathcal{X}$, which is the minimum number of entries required to recover $\mathcal{X}$, is $r^n + n(rI - r^2) =: m_0$. If we unfold $\mathcal{X}$ to $X \in \mathbb{R}^{I \times I^{n-1}}$, the number of degrees of freedom of X is $(I + I^{n-1} - r)r =: m_1$. If we treat every slice of $\mathcal{X}$ separately, the number of degrees of freedom is $I^{n-2}(2rI - r^2) =: m_2$. Therefore, when $r < I$, we have $m_0 < m_1 < m_2$, which means we need fewer parameters to determine $\mathcal{X}$, compared to the matricization and union of slices. Thus, tensor completion may outperform matrix completion on $\mathcal{X}$ with the same number of observed entries.

2.3.5 Fast and Accurate Resource-Constrained Active Learning

Given a new dataset, we first select a subset of pipelines to fit, so that we may estimate the performance of other pipelines. We use ideas from linear experiment design, which picks a subset of low-cost statistical trials to minimize the variance of the resulting estimator, to make this selection.

Concretely, we estimate the embedding x of the new dataset by linear regression. Given the linear model as Equation 2.2, with known performance e_S of a subset $S \subseteq [n]$ of pipelines on the new dataset, we have

$$e_S = (Y_{:S})^\top x + \epsilon, \tag{2.3}$$

in which Y collects the latent embeddings of pipeline performance, and ϵ is the error in this linear model. An example of the source of error is the misspecification of target multilinear rank for the Tucker decomposition. We estimate x by linear regression and denote the result as $\hat{x}$. Then we estimate the performance of pipelines in $[n]\backslash S$ by the corresponding entries in $\hat{e} = Y^\top \hat{x}$.

Now we consider which S to choose to accurately estimate x. We will motivate the use of the experiment design model and its greedy approach by first showing how to constrain the *number* of pipelines sampled in Section 2.3.5, and then develop a *time*-constrained version that we use in practice in Section 2.3.5.

Greedy method for size-constrained experiment design

Suppose the error $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$. Using the linear regression model, Equation 2.3, we want to minimize the expected ℓ_2 error $E_\epsilon \|\hat{x} - x\|^2 = E_\epsilon \|\hat{x} - E_\epsilon \hat{x}\|^2 + \|E_\epsilon \hat{x} - x\|^2$. Here, the second term is 0 since linear regression is unbiased, and the first term is the covariance

$\sigma^2(YY^\top)^{-1}$ of the estimated embedding $\hat{x}$, which is straightforward to compute.

Imagine we have enough time to run at most m pipelines (and all pipelines run equally slowly). Given pipeline embeddings $\{y_j\}_{j=1}^n$ (which we call *design vectors* or *designs*), in which each $y_j \in \mathbb{R}^k$, we minimize a scalarization of the covariance to obtain the (number-constrained) D-optimal experiment design problem

$$\begin{array}{ll} \text{maximize} & \log\det\left(\sum_{j\in S} y_j y_j^\top\right) \\ \text{subject to} & |S| \leq m \\ & S \subseteq [n]. \end{array} \tag{2.4}$$

Here, $\sum_{j\in S} y_j y_j^\top$, the inverse of (scaled) covariance matrix, is called the Fisher information matrix.

Obtaining an exact solution for a mixed-integer nonlinear combinatorial optimization problem like Problem 2.4 is prohibitively expensive. Convexification is commonly used to solve such a problem [7, 63, 86]. However, we have more than 20,000 pipelines to select from, making convex relaxations also too slow. Moreover, we can find better solutions with the greedy heuristic we present next.

Greedy methods form another popular approach to combinatorial optimization problems like Problem 2.4. Importantly, the objective function of Problem 2.4, $f(S) = \log\det\left(\sum_{j\in S} y_j y_j^\top\right)$, is submodular. (Recall a set function $g : 2^V \to \mathbb{R}$ defined on a subset of V is submodular if for every $A \subseteq B \subseteq V$ and every element $s \in V\backslash B$, we have $g(A\cup\{s\}) - g(A) \geq g(B\cup\{s\}) - g(B)$. This characterizes a "diminishing return" property.) Given a size constraint, the submodular function maximization problem

$$\begin{array}{ll} \text{maximize} & g(S) \\ \text{subject to} & S \subseteq V \\ & |S| \leq m \end{array} \tag{2.5}$$

can be solved with a $1 - \frac{1}{e}$ approximation ratio [54] by the greedy approach: in every step, add the single design vector that maximizes the increase in function value. In D-optimal experiment design, we can compute this increase efficiently using Lemma 1.

Lemma 1 (Matrix Determinant Lemma [31, 52])**.** *For any invertible matrix $A \in \mathbb{R}^{k\times k}$ and $a, b \in \mathbb{R}^k$,*

$$\det(A + ab^\top) = \det(A)(1 + b^\top A^{-1}a).$$

At the t-th step in our setting, with an already constructed Fisher information matrix $X_t = \sum_{j\in S} y_j y_j^\top$, we have

$$\operatorname{argmax}_{j\in[n]\setminus S} \det(X_t + y_j y_j^\top) = \operatorname{argmax}_{j\in[n]\setminus S} y_j^\top X_t^{-1} y_j.$$

Here, $y_j^\top X_t^{-1} y_j$ can be seen as the payoff for adding pipeline j. From the t-th to the $(t + 1)$-th step, with the selected design vector at the t-th step as y_t, we update X_t to $X_{t+1} = X_t + y_t y_t^\top$ by Lemma 2:

Lemma 2 (Sherman-Morrison formula [67, 29])**.** *For any invertible matrix $A \in \mathbb{R}^{k\times k}$ and $a, b \in \mathbb{R}^k$,*

$$(A + ab^\top)^{-1} = A^{-1} - \frac{A^{-1}ab^\top A^{-1}}{1 + b^\top A^{-1}a}.$$

Pseudocode for the greedy algorithm to solve Problem 2.4 is shown as Algorithm 2, with per-iteration time complexity $O(k^3 + nk^2)$: it takes $O(k^3)$ (for a naive matrix multiplication algorithm) to update X_t^{-1} and $O(nk^2)$ to choose the best pipeline to add.

There remains the problem of how to select an initial set of designs S to start from, such that $X_0 = \sum_{j\in S} y_j y_j^\top = Y_{:S} Y_{:S}^\top$ is non-singular. This is equivalent to the problem of finding a subset of vectors in $\{y_j\}_{j=1}^n$ that can span $\mathbb{R}^k$. We select this sized-k subset S_0 to be the first k pivot columns from QR factorization with column pivoting [25, 28] on Y, with time complexity $O((n + k)k^2)$.

Algorithm 2 Greedy algorithm for size-constrained D-design

Input: design vectors $\{y_j\}_{j=1}^n$, in which $y_j \in \mathbb{R}^k$; maximum number of selected pipelines m; initial set of designs $S_0 \subseteq [n]$, s.t. $X_0 = \sum_{j \in S_0} y_j y_j^\top$ is non-singular

Output: The selected set of designs $S \subseteq [n]$

```
function GREEDY_ED_NUMBER
    S ← S_0
    do
        i ← argmax_{j∈[n]\S} y_j^⊤ X_t^{-1} y_j
        S ← S ∪ {i}
        X_{t+1} ← X_t + y_i y_i^⊤
    while |S| ≤ m
    return S
```

Greedy method for time-constrained experiment design

We here move on to the realistic case in AutoML pipeline selection: which pipelines should we select to gain an accurate estimate of the entire pipeline space? In this setting, each pipeline is associated with a different cost. We characterize the cost as running time, and form the time-constrained version of experiment design as

$$\begin{aligned} \text{maximize} \quad & \log\det\Big(\sum_{j\in S} y_j y_j^\top\Big) \\ \text{subject to} \quad & \sum_{j\in S} \hat{t}_j \le S \\ & S \subseteq [n], \end{aligned} \tag{2.6}$$

in which $\{\hat{t}_i\}_{i=1}^n$ are the estimated pipeline running times. The payoff of adding design i in the t-th step can thus be formulated as $\frac{y_i^\top X_t^{-1} y_i}{\hat{t}_i}$, giving Algorithm 3 the greedy method to solve Problem 2.6.

The initialization problem is solved similarly by the QR method. Given runtime limit τ, we select among columns with corresponding pipelines predicted to finish within $\frac{\tau}{2k}$. Pseudocode for this initialization algorithm is shown as Algorithm 4.

A corner case of Algorithm 4, shown as Case 1, is that there are not enough pipelines predicted to be able to finish within time limit. This corresponds to the case that the

Algorithm 3 Greedy algorithm for time-constrained D-design

Input: design vectors $\{y_j\}_{j=1}^n$, in which $y_j \in \mathbb{R}^k$; estimated running time of pipelines $\{\hat{t}_i\}_{i=1}^n$; maximum running time τ; initial set of designs $S_0 \subseteq [n]$, s.t. $X_0 = \sum_{j \in S_0} y_j y_j^\top$ is non-singular

Output: The selected set of designs $S \subseteq [n]$

function GREEDY_ED_TIME
  $S \leftarrow S_0$
  do
    $i \leftarrow \text{argmax}_{j \in [n] \setminus S} \frac{y_j^\top X_t^{-1} y_j}{\hat{t}_j}$
    $S \leftarrow S \cup \{i\}$
    $X_{t+1} \leftarrow X_t + y_i y_i^\top$
  while $\sum_{i \in S} \hat{t}_i \leq \tau$
  return S

Algorithm 4 Initialization of the greedy algorithm for time-constrained D-design, by QR factorization with column pivoting

Input: design vectors $\{y_j\}_{j=1}^n$, in which $y_j \in \mathbb{R}^k$; (predicted) running time of all pipelines $\{\hat{t}_i\}_{i=1}^n$; maximum running time τ

Output: A subset of designs $S_0 \subseteq [n]$ for Algorithm 3 initialization

function QR_INITIALIZATION
  $S_{\text{valid}} \leftarrow \{i \in [n] : \hat{t}_i \leq \frac{\tau}{2k}\}$
  $S_0 \leftarrow \emptyset$, $\hat{t}_{\text{sum}} \leftarrow 0$
  if $|S_{\text{valid}}| < k$ **then** ▷ Case 1
    do
      $i \leftarrow \text{argmin}_{j \in [n] \setminus S} \hat{t}_j$
      $S_0 \leftarrow S_0 \cup \{i\}$
      $\hat{t}_{\text{sum}} \leftarrow \hat{t}_{\text{sum}} + \hat{t}_i$
    while $\hat{t}_{\text{sum}} \leq \tau$
  else ▷ Case 2
    $S_0 \leftarrow \text{QR\_with\_column\_pivoting}(Y_{:S_{\text{valid}}})[:k]$
  return S_0

runtime limit is relatively small compared to the time of fitting pipelines on current dataset. In this case we greedily select the fast pipelines and do not run Algorithm 3 afterwards.

As a side note, the assumption that performance of different pipelines are predicted with equal variance is not quite realistic, especially when some components have much more pipelines than others. If the variance is known (but unequal), we obtain a weighted

least squares problem. In the error matrix E, we can estimate the variance of prediction error of each pipeline $j \in [n]$ by the sample variance of $e_j - X^\top y_j$ and select the promising pipelines with the goal of minimizing the rescaled covariance. Practically, however, this rescaled method does not systematically improve on the standard least squares approach in our experiments (shown in Appendix A.2), so we retrench to the simpler approach.

2.4 Experimental Evaluations

Code for all experiments is in the GitHub repository at `https://github.com/udellgroup/oboe`. We use a Linux machine with 128 Intel® Xeon® E7-4850 v4 2.10GHz CPU cores and 1056GB memory. Offline, we collect cross-validated pipeline performance on meta-training datasets: 215 OpenML [77, 21] classification datasets with number of data points between 150 and 10,000, listed in Appendix A.1.1. The 215 datasets are chosen alphabetically. Pipelines are combinations of the machine learning components shown in Appendix A.1.3, Table A.1, which lists 4 data imputers, 2 encoders, 2 standardizers, 8 dimensionality reducers and 183 estimators, resulting in 23,424 linear pipeline candidates in total.

2.4.1 Comparison with Time-Constrained AutoML Pipeline Build Systems

In this section, we demonstrate the performance of TENSOROBOE as an AutoML system for pipeline selection.

A naive approach for pipeline selection is to choose the one that on average performs

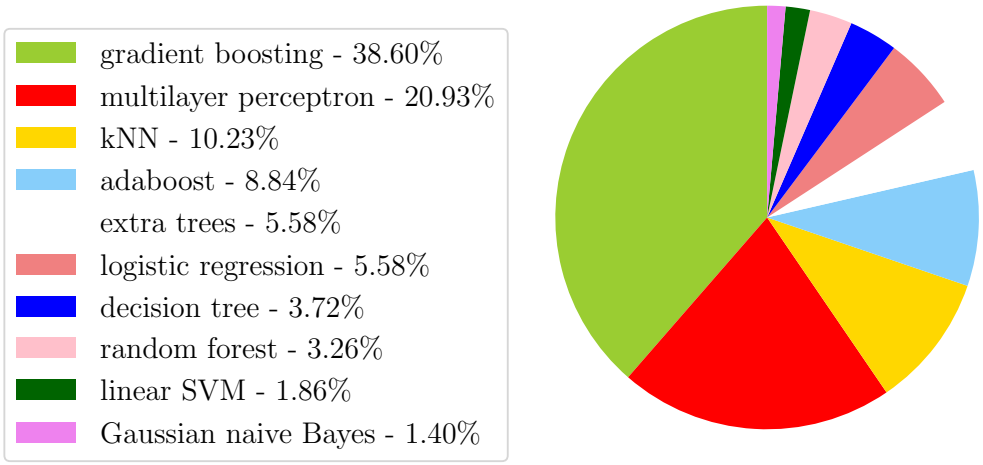
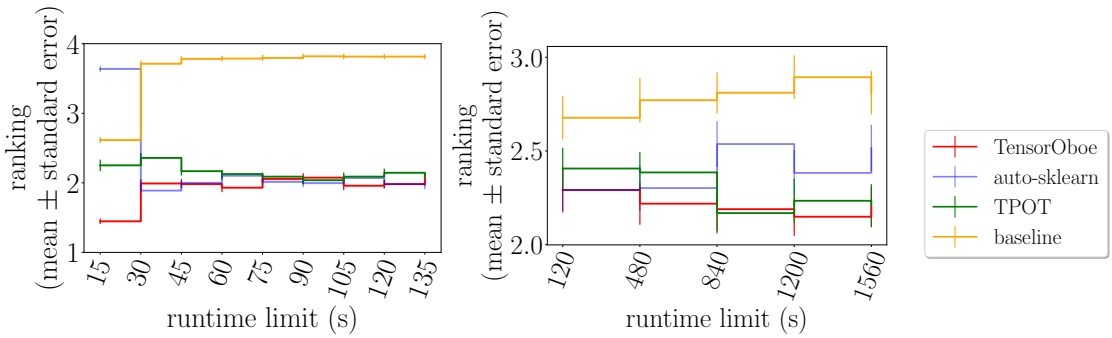


Figure 2.5: Which estimators work best? Distribution of estimator types in best pipelines on meta-training datasets.

the best among all meta-training datasets, which we call the baseline pipeline. Given the pipeline selection problem, it is common for human practitioners to try out the best pipeline at the very beginning. On our meta-training datasets, the baseline pipeline is: impute missing entries with the mode, encode categorical features as integers, standardize each feature, remove features with 0 variance, and classify by gradient boosting with learning rate 0.25 and maximum depth 3. The baseline pipeline has an average ranking of 1568 among all 23,424 pipelines across all 215 meta-training datasets.

Human practitioners may also reduce the number of trials by choosing certain pipeline components to be the type that performs the best on average. Figure 2.5, however, shows that although some estimator types (gradient boosting and multilayer perceptron) are commonly seen among the best pipelines, no estimator type uniformly dominates the rest.

We compare TENSOROBOE with auto-sklearn [19], TPOT [58], and the baseline pipeline in Figure 2.6. To ensure fair comparisons, we use a single CPU core for each AutoML system. We allow each to choose from the same primitives. We can see that:



(a) OpenML (meta-LOOCV) (b) UCI (meta-test)

Figure 2.6: Rankings of AutoML systems for pipeline search in a time-constrained setting, vs the baseline pipeline. We meta-train on OpenML classification datasets and meta-test on UCI classification datasets [17]. Until the first time the systems can produce a pipeline, we classify every data point with the most common class label. Lower ranks are better.

1 All AutoML frameworks are able to construct pipelines that outperform the baseline on average once the method returns a pipeline (for auto-sklearn, this takes 30 seconds).

2 TENSOROBOE on average outperforms the competing methods and produces meaningful pipeline configurations fastest.

3 With the longer running time in Figure 2.6b, TENSOROBOE still outperforms in most cases.

These results show that TENSOROBOE is able to accurately approximate the hyperparameter landscape. We discuss these results in greater detail in Section 2.4.5.

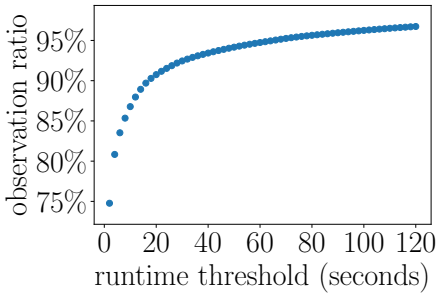


Figure 2.7: CDF of pipeline runtime on meta-training datasets.

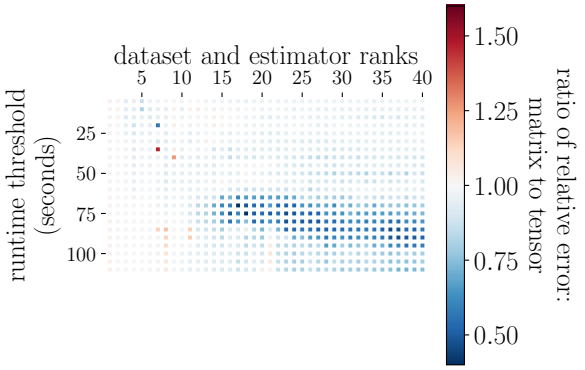
2.4.2 Tensor Completion vs Matrix Completion for Error Tensor Completion

Given meta-training data $\{\mathcal{D}, \mathcal{P}, \mathcal{P}(\mathcal{D})\}$ on a subset of dataset-pipeline combinations, a good surrogate model should accurately predict the performance of new dataset-pipeline combinations.

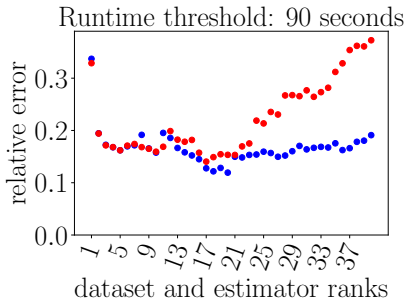
Figure 2.7 shows that most pipelines run quickly on most datasets: for example, over 90% finish in less than 20 seconds and over 95% finish in less than 80 seconds.

Figure 2.8 compares relative errors of predictions by tensor and matrix surrogate models. For each runtime threshold, we treat pipeline-dataset combinations with running time less than the threshold as training data, and those that take longer than threshold and less than 120 seconds as test. We compute relative errors on test data, hence the name "runtime generalization". To ensure a fair comparison, we set the dataset and estimator ranks to be equal in the tensor model, which is required for the matrix model, since column rank equals row rank for a matrix. We can see that:

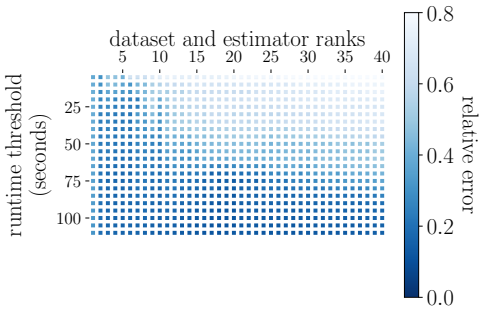
1 The tensor model outperforms the matrix model in nearly all cases, demonstrating that the additional combinatorial structure provided by the tensor model helps recover the combinatorial relationships among different pipeline components.



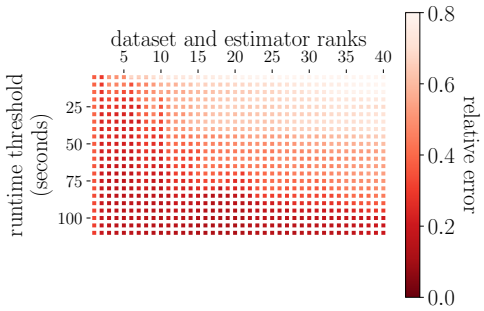
(a) Runtime generalization for tensor vs matrix models. Blue means the tensor model achieves lower error; red means the matrix model does. The tensor model outperforms for longer running times.



(b) Tensor and matrix completion errors when runtime threshold = 90 seconds (3.3% of entries in error tensor missing).



(c) Runtime generalization error by tensor model. Darker colors mean smaller errors.



(d) Runtime generalization error by matrix model. Darker colors mean smaller errors.

Figure 2.8: Tensor completion vs matrix completion for inferring pipeline performance.

2 Figure 2.8b shows the U-shaped error curve as we increase the dataset and estimator ranks for both matrix and tensor models, moving from underfitting (decreasing error) to overfitting (increasing error). Informed by these results, we select both ranks to be 20, the rank in the middle, in the tensor surrogate model.

2.4.3 Cold-Start Performance by Greedy Experiment Design

We compare the performance of different approaches to solve the experiment design problem, so as to choose which pipelines we should sample. Recall that there are two approaches:

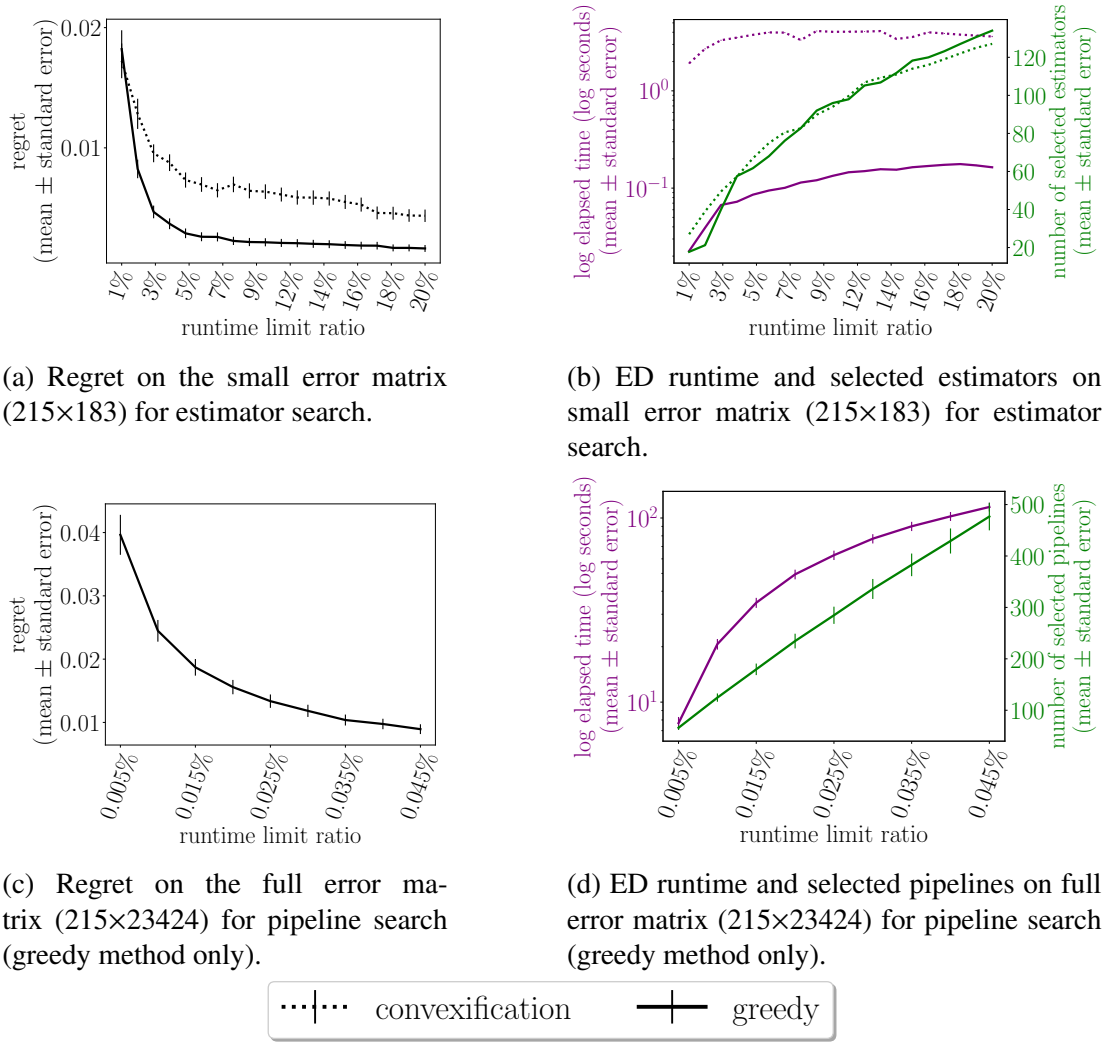
- **Convexification**: Solve the relaxed problem (Equation 2.6 with $v_i \in [0, 1]$, $\forall i \in [n]$) with an SLSQP solver, sort the entries in the optimal solution v^*, and greedily add the pipeline with large v_i^* until the runtime limit is reached.
- **Greedy**: Solve the original integer programming problem (Equation 2.6) by the greedy algorithm (Algorithm 3), initialized by time-constrained QR (Algorithm 4).

For our problem, the greedy approach is superior, since the convexification method is prohibitive on our large 215×23424 error matrix. Hence we compare these methods on a subset of pipelines that only differ by estimators, 183 in total. This setting matches an experiment in [86]. Shown in Figure 2.9, we can see that:

1 The greedy method performs better for cold-start than convexification (Figure 2.9a): it selects informative designs that better predict the high-performing pipelines (Figure 2.9b).

2 The greedy method is more than 30× faster than convexification, which allows TENSOROBOE to devote its runtime budget to fitting pipelines instead of searching for the informative pipelines.

3 Shown in Figure 2.9d, the greedy algorithm still takes a fair amount of time if the number of designs we select is large; however, the dataset ranks we choose



(a) Regret on the small error matrix (215×183) for estimator search.

(b) ED runtime and selected estimators on small error matrix (215×183) for estimator search.

(c) Regret on the full error matrix (215×23424) for pipeline search (greedy method only).

(d) ED runtime and selected pipelines on full error matrix (215×23424) for pipeline search (greedy method only).

Figure 2.9: Comparison of time-constrained experiment design methods across meta-training datasets. The y-axes in 2.9a and 2.9c are regrets: the difference between minimum pipeline error found by each method and the true minimum. The x-axes are runtime limit ratios: ratios of the runtime limit to the total runtime of all pipelines on each dataset.

are less than 50, so it generally takes less than 10 seconds to choose informative pipelines. This time can be further reduced using Lemma 2.

Table 2.1: Runtime prediction accuracy on OpenML datasets

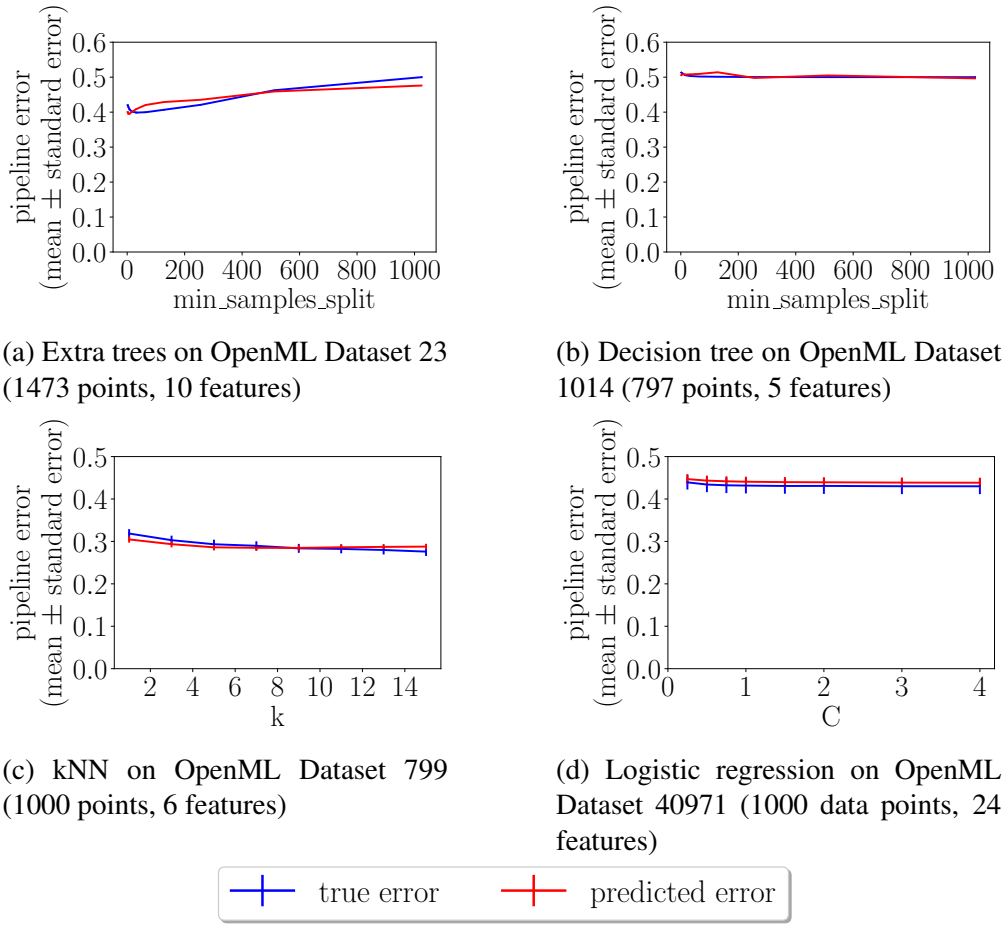
Pipeline estimator type	Runtime prediction accuracy	
	within factor of 2	within factor of 4
Adaboost	73.6%	86.9%
Decision tree	62.7%	78.9%
Extra trees	71.0%	83.8%
Gradient boosting	53.4%	77.5%
Gaussian naive Bayes	67.3%	82.3%
kNN	68.7%	84.4%
Logistic regression	53.6%	76.1%
Multilayer perceptron	74.5%	88.9%
Perceptron	64.5%	82.2%
Random Forest	69.5%	84.9%
Linear SVM	56.8%	79.5%

2.4.4 Pipeline Runtime Prediction Performance

Runtime prediction accuracy is critical for the performance of our time-constrained pipeline selection system. Recall that our predictions use order-3 polynomial regression on $n^{\mathcal{D}}$ and $p^{\mathcal{D}}$, the numbers of data points and features in $\mathcal{D}$, and their logarithms. We shown in Table 2.1 that this runtime predictor performs well.

2.4.5 Learning the Hyperparameter Landscapes

Hyperparameter landscapes plot pipeline performance with respect to hyperparameter values. While parameter landscapes have been extensively studied, especially in the deep learning context (for example, [38, 48, 23]), hyperparameter landscapes are less studied. The previous sections focus on how we can choose among different pipeline component types. In this section, we show that our tensor surrogate model is able to learn hyperparameter landscapes of different estimator types that exhibit qualitatively different behaviors.



(a) Extra trees on OpenML Dataset 23 (1473 points, 10 features)

(b) Decision tree on OpenML Dataset 1014 (797 points, 5 features)

(c) kNN on OpenML Dataset 799 (1000 points, 6 features)

(d) Logistic regression on OpenML Dataset 40971 (1000 data points, 24 features)

Figure 2.10: Hyperparameter landscape prediction examples.

Figure 2.10 shows some examples of both real and predicted hyperparameter landscapes after running our system for 135 seconds. We can see that our predictions match the overall tendencies of the curves. Larger plots (Figure A.3 in Appendix A.3) show our predictions also capture most of the small variations in these landscapes.

Note TENSOROBOE does not use a subroutine for hyperparameter optimization: it chooses the hyperparameter for each estimator from a predefined grid of values instead of optimizing hyperparameters by, for example, Bayesian optimization. The hyperparameter landscapes visualized here give confidence that grid search effectively samples performant hyperparameter settings within the range of hyperparameters: a coarse grid

suffices.

2.5 Overfitting Analysis

Two types of overfitting are of concern in AutoML systems: traditional overfitting (overfitting of models on training folds) and meta-overfitting (overfitting of AutoML surrogate models).

Traditional overfitting may happen in any machine learning system, and is often mitigated by controlling model complexity, cross validation on training set, etc. In TENSOROBOE, we always evaluate pipelines by k-fold cross validation, and build an ensemble since the pipeline with lowest cross-validation error may not be the one with lowest test error.

Meta-overfitting happens when meta-training datasets are biased in some sense, and when the surrogate model is so complex that it captures noise in addition to model performance. We mitigate meta-overfitting in the following ways: The OpenML meta-training datasets we collect have diverse topics ranging among multiple science and sociology disciplines. The surrogate model we use is low rank tensor decomposition, a model with low complexity. It denoises cross-validated pipeline error, as discussed in Section 2.2.

Meta-overfitting still presents many perils. The surrogate model may lack training instances. For example, the perceptron algorithm never performs the best on any meta-training dataset, as shown in Figure 2.5. Hence TENSOROBOE is unlikely ever to choose a perceptron pipeline. To mitigate this problem, we must collect pipeline performance in a larger space, or consider if the perceptron algorithm (for example) is truly dominated.

Another possible source of meta-overfitting is that our meta-training datasets have no more than 10,000 points and smaller number of features. Order-3 polynomial runtime predictors may not generalize well to larger problems.

2.6 Summary

This chapter develops TENSOROBOE, a new structured model based on tensor decomposition for AutoML pipeline selection. The low multilinear rank tensor surrogate model allows us to efficiently learn about new datasets. The greedy experiment design method selects informative pipelines to evaluate. Together, TENSOROBOE tames the combinatorial complexity of the pipeline search space: the time complexity scales linearly in the number of candidates for each pipeline component. Empirically, TENSOROBOE relies on more offline work than competing methods, but such work pays off to improve on the state of the art in AutoML pipeline selection.

This work is the first tensor method for pipeline selection. There are many avenues for improvement and extensions. For example, one could enlarge the pipeline search space, explore nonlinear surrogate models, explore different mechanisms to initialize the greedy method, develop an extension for neural architecture search, and design task-oriented pipeline selection systems that have better performance on domain-specific datasets. Further, the combinatorial space may be better handled by a method that dynamically adapts to the results of finished pipeline runs, thus leveraging its conditional structure.

CHAPTER 3

AUTOMATED TRANSFER LEARNING

3.1 Introduction

Transfer learning is key to the success and popularity of computer vision. The features from a convolutional neural network (CNN) trained on one task can be incredibly useful across a broad variety of tasks[79, 70, 74, 50]. Even larger performance gains can be obtained by *selecting* a pretrained model more specially suited to the task at hand. This behavior has been studied in past work[11], but only recently has attention turned to *how to determine* which specialized model is appropriate for a given task[2].

The key to such model selection is characterizing tasks and their relationships. What does one need to characterize a task? *A priori*, it seems that we need to characterize two things:

1. The *input*, characterizing which requires **a dataset** of images, and **features** to represent them, and
2. The *output*, characterizing which requires **labels** for the dataset.

Current approaches to characterizing tasks synthesize both sources of information. Task2Vec[2], for example, trains a linear head on top of a pretrained feature extractor and uses the Fisher information associated with the resulting model to generate a vectorized embedding. Decisions such as choosing the best pretraining task for a target task can then effectively be made using retrieval-like techniques with this embedding.

But how much of these accurate decisions come from characterizing the *input domain alone*, and how much comes from knowledge of the precise task? This question

has important practical considerations. For example, suppose we want to choose a pretrained representation for analyzing x-ray images. We may not yet know *what* we want to recognize in x-ray images. In fact, we may want a pretrained representation suitable for *any* kind of x-ray image analysis, even those we haven't conceived yet. In such cases we are interested in characterizing only the general problem domain, and do not have particular labels (yet) that we are interested in. This raises the question: **Can we characterize tasks without labels?**

As our first contribution, we answer this question in the affirmative. To address this problem, we introduce PseudoTask: a modification of the Task2Vec algorithm that replaces labels with *pseudolabels* output by an image classifier trained in a generic source domain. While these pseudolabels are definitely incorrect due to domain misalignment, they prove discriminative enough to be quite useful in characterizing tasks. Empirically, we find that PseudoTask embeddings are *as accurate as supervised Task2Vec embeddings*, indicating that one can characterize tasks effectively *even without labels*.

Key to this performance, as also to the performance of Task2Vec, is the inductive bias provided by the pre-trained feature representation. However, this inductive bias may prove harmful as the task domains move farther away from the domain where the feature extractor is pretrained [79], making it risky to rely so heavily on such feature representations. This raises a second question: **can we characterize tasks without features**?

We answer this too in the affirmative. We design a method called Task Tangent Kernel (TTK) that measures task similarity using the gradients of randomly initialized networls. TTK *does not use pretrained probe networks at all*. Despite this lack of inductive bias, it provides useful selections as well, at less than half the error of PseudoTask and Task2Vec with random feature extractors.

In sum, this work introduces two new techniques for characterizing tasks that lift some of the restrictive assumptions of prior work (Figure 3.2):

1. We introduce **PseudoTask**, a new way of characterizing tasks without labels, allowing one to characterize problem domains in general. We find PseudoTask performs almost the same as Task2Vec, indicating that labels are in fact not necessary.
2. To avoid the potentially mismatched inductive bias of pretrained feature extractors, we introduce **Task Tangent Kernels**, which characterizes tasks effectively even without such feature extractors.

3.2 Related Work

Previous work has studied why pretrained models transfer so well[90, 43], the tradeoffs between specialization and scale in pretraining[11] and how concurrent multitask learning can benefit performance[88]. Task2Vec[2], the work that this work builds off, studies the problem of automated model selection, as does [37]. [68, 1, 47] recommend *algorithms* for various problems using "Active Testing", intelligently adapting exploration based on results. Such types of approaches are inherently more limited computationally than embedding-based methods. [80] predicts per-image performance for single models, while [18, 55] characterize performance per-dataset. [53, 89] deal with the problem of model recommendation for action recognition and object detection respectively, but require some degree of performance evaluation on the new task in order to provide a recommendation.

Transfer learning has been increasingly optimized, with recent advances detailed in [42]. The problem of expert selection has analogs to image retrieval, examples of which include[64, 5]. Other works measuring distances between domains include evaluating

the biases between semantically similar datasets[72] and measuring temporal domain shifts in datastreams[40]

Our PseudoTask framework draws heavily on pseudolabeling for semi-supervised learning, such as in [46]. Parts of our training setup are very similar to that of self-training, such as for few-shot transfer or semi-supervised learning[62, 85]. PseudoTask can be considered a form of self-supervised training such as [24, 56, 10, 27, 32]. The Task Tangent Kernel is inspired by Neural Tangent Kernel literature, originating with [35] and developed in [49, 4, 57].

3.3 Problem setup

A **task** $T = (X_T, Y_T)$ consists of a **domain** of images $X_T = \{x_i\}_{i=1}^{n_T}$ and corresponding **labels** $Y = \{y_i\}_{i=1}^{n_T}$. An **expert** $\Theta_T = (\phi_T, h_T)$ is a dedicated neural network, consisting of a feature extractor ϕ_T and a head h_T that is trained on a task T. Suppose that we have a bank of tasks $T_1, \ldots, T_N$, and have already trained a corresponding bank of experts $\Theta_{T_1}, \ldots, \Theta_{T_N}$. Then, when we encounter a new task T', instead of training a model from scratch, we may want to choose a pretrained expert Θ_{T_i}, use its fixed feature extractor ϕ_{T_i} and train a linear head to solve the new task; see Figure 3.1. *The goal of expert selection is to pick the best pretrained expert for the target task.*

3.4 Background: Task2Vec

Our work builds on Task2Vec[2], a recent approach that tries to characterize tasks and embed them in a useful way. Task2Vec tries to characterize both the input domain,

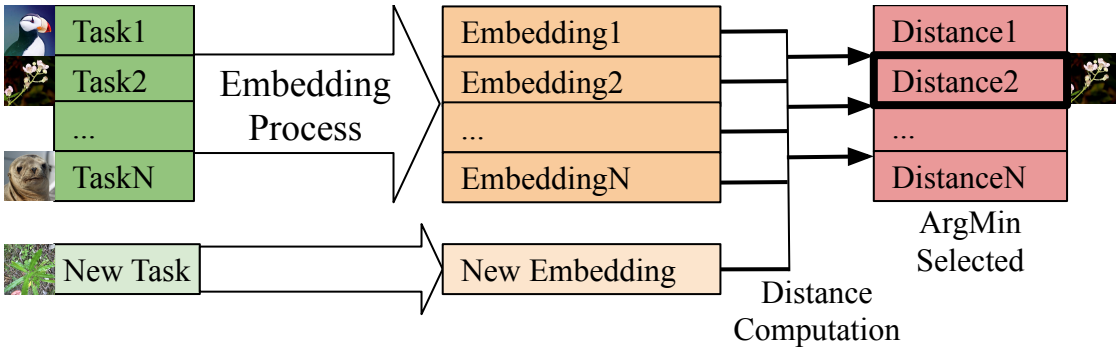


Figure 3.1: Tasks consisting of images and possibly labels are embedded into a vector space. When a new task is introduced, a new embedding is calculated and compared (using a modified cosine distance) to the bank of previous embeddings. The closest embedding is selected to use as pretraining for the new task.

as well as the semantic information carried by the labels. The key intuition behind Task2Vec is that we can try to solve the task with a moderately effective but generic off-the-shelf feature extractor (e.g., trained on Imagenet) and then see which parameters of the feature extractor most impact the performance. Task2Vec posits that tasks which are sensitive to the same set of feature extractor parameters are likely to be "similar" to each other, especially in terms of what they demand out of pretrained features.

Concretely, Task2Vec trains a linear layer on top of the off-the-shelf feature extractor (called a "probe") for the task in question. It then computes the *Fisher Information Matrix* (FIM), which is known to measure the sensitivity of the loss to the parameters of the model. Denoting by $p_w(y|x)$ the output distribution of trained model p_w with weights w, and by $\hat{p}(x)$ the data distribution, the FIM is defined as:

$$F = \mathbb{E}_{x,y\sim\hat{p}(x)p_w(y|x)}[\nabla_w \log p_w(y|x) \nabla_w \log p_w(y|x)^T] \tag{3.1}$$

Task2Vec estimates F using a variational approach that amounts finding the optimal weights $\hat{w}$ and precision matrix Λ that minimize the following objective:

$$\begin{aligned} L(\hat{w}, \Lambda) =& \mathbb{E}_{w\sim\mathcal{N}(\hat{w},\Lambda)}[L_{CE}(X_T, Y_T, p_w)] + \\ & \beta KL(\mathcal{N}(0,\Lambda)||\mathcal{N}(0,\lambda^2 I) \end{aligned} \tag{3.2}$$

Here L_{CE} is the cross entropy loss over the dataset and λ is a hyperparameter. Achille et al. prove that the solution to this is $\Lambda = F + \frac{\beta\lambda^2}{2N}I$. Task2Vec finally takes the diagonal of F and averages together the values for different parameters of the same filter to produce the embedding.

When using this embedding, symmetric distances such as cosine distance, denoted d_{sym}, do not yield satisfactory pefrformance when retrieving experts. This is because there is an inherent asymmetry to the expert selection problem: a task with a large dataset and thousands of classes will yield a good expert for a similar task with only two classes and a small dataset, but not vice versa. Therefore, there is a large benefit to making the distance function *asymmetric* to account for the complexity of the task. The Asymmetric Task2Vec distance (d_{asym}) is defined as:

$$d_{asym}(t_A \implies t_B) = d_{sym}(t_A, t_B) - \alpha d_{sym}(t_A, t_0) \tag{3.3}$$

Here t_0 is the "trivial" task of ImageNet classification. This formulation makes complex tasks that are very different from ImageNet relatively closer to everything else. The intuition is that a more complex task has a higher chance of being a relevant expert given the same degree of symmetric similarity. α's value varies by architecture. In the original work, $\alpha = 0.3$ is reported as optimal when training with a ResNet-34[33]. In our experiments with ResNet-18s we found $\alpha = 0.15$ to yield best performance. See Supplementary for further discussion of α.

Limitations: The Task2Vec formulation requires a fully specified task to have a labeled dataset, as well as a pretrained probe network to be available. We next address these limitations using our proposed alternatives below.

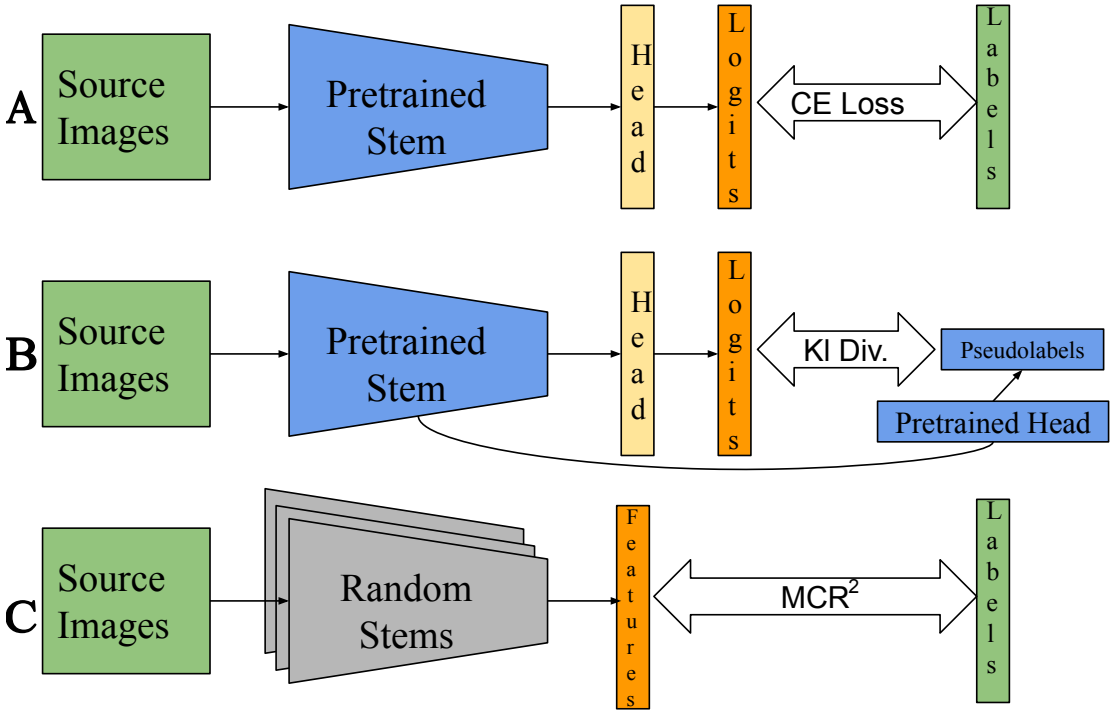


Figure 3.2: **A**: The original Task2Vec[2] framework. A pretrained model is available as are image labels, which a linear head is trained to predict. **B**: Our proposed PseudoTask framework. Pretrained models are available, but labels are not. A zero-initialized head is trained to match the predictions of the full pretrained network. **C**: Our proposed Task Tangent Kernel framework. Labels are available, but pretrained models are not. No training is done. Gradients are calculated from the features and labels using Maximal Coding Rate Reduction (MCR^2)[87] across randomly initialized networks (see Sec. 3.6).

3.5 Characterization Without Labels: PseudoTask

Motivation: The first step in Task2Vec is to train a linear classifier with a probe network's features for the task in question. This uses labels, which in turn provide the semantics of the task. But often we may want to characterize entire problem domains (e.g., x-ray images) without having a specific task in mind. In such cases, labeled datasets may be unavailable.

Even for well-specified tasks, the amount of labeled data required to characterize the task using the Task2Vec approach can also be substantial, precluding applications like

few-shot learning where one may want to choose experts and take decisions with only one or two labels per class. To alleviate this issue, we present a self-supervised task characterization, which we dub "PseudoTask".

3.5.1 Method

What information can we use to characterize a problem domain with no labels? Self-training approaches have recently shown the usefulness of training a network in one domain to match the predictions of a teacher from a source domain *even when the two domains share no classes at all*[62, 85]. Furthermore, unsupervised contrastive learning approaches such as MoCo and SimCLR[32, 10], demonstrate the emergence of semantics simply by learning to distinguish individual images.

Both approach types point to the striking power of *pseudolabels*, and motivate us to create PseudoTask, where pseudolabels are used as a substitute for ground-truth labeling. Concretely, PseudoTask follows the originally presented framework of Task2Vec with a key modification: instead of real labels, we use soft labels from a pretrained classifier (ImageNet or Places365).

Given a domain (task) $T = (X_T)$ consisting of unlabeled images $X_T = \{x_i\}_{i=1}^{n_T}$ and a pre-trained classifier Θ, we follow Algorithm 5 to compute our embedding. Note that when computing the Task2Vec embedding (second for loop), the soft predictions are computed dynamically on *randomly augmented* versions (x') of the images (x).

Asymmetric PseudoTask: The final adjustment we found necessary in the self-supervised algorithm was the measure of asymmetery. In the original work, a bias term of $-\alpha d_{sym}(t_{source}, t_0)$ was used where t_0 is the "trivial" ImageNet task. While the benefit

Algorithm 5 PseudoTask

Compute soft labels $\ell_i = \Theta(x_i)$ for each image
Zero out the linear classification layer (γ)
for 2 epochs **do**
    Fit the linear head γ to $\{\ell_i\}$
for 10 epochs **do**
    Minimize $L(\hat{w}, \Lambda) = \mathbb{E}_{w\sim\mathcal{N}(\hat{w},\Lambda)}[L_{CE}(x', \Theta(x'), p_w)] + \beta KL(\mathcal{N}(0, \Lambda)||\mathcal{N}(0, \lambda^2 I))$

of this term for Task2Vec was replicated in our baseline experiments, we found that it was not appropriate for PseudoTask. For PseudoTask, the distance bias term $d_{sym}(t_a, t_0)$ is fairly homogeneous across tasks compared to the variation for embeddings trained with Task2Vec. We hypothesize that this behavior stems from the training objective of PseudoTask being consistent across domains (matching soft labels from the same pre-trained classifier) while Task2Vec varies more significantly in label distribution.

To define an alternative asymmetric distance, we leverage the observation that a large norm of the PseudoTask (and Task2Vec) embedding is correlated with task hardness: for a complex task, linear classifiers on the probe feature extractor will not work well, yielding large-valued Fisher Information Matrix (see Sec. 2.2 in [2]). As such, to bias the expert selection towards experts trained on more complex tasks, we add a bias term based on the norm of the embedding and define PseudoTask's asymmetric distance between two task embeddings, t_a and t_b as:

$$d'_{asym}(t_a \implies t_b) = d_{sym}(t_a, t_b) - \alpha||t_a|| \tag{3.4}$$

The intuition behind this definition is similar to the Task2Vec asymmetric distance, but uses a slightly different formulation.

3.6 Characterization Without Features: Task Tangent Kernel

Motivation: Task2Vec and PseudoTask both use pretrained feature representations. Even for Task2Vec, which does not need pseudolabels, the availability of this feature representation is critical. Task2Vec relies on the Fisher information matrix, which is typically used to characterize how sensitive the *optimum* parameter setting is. If the feature extractor is far from optimal, using the Fisher information does not make sense. Unfortunately in practice one may operate in such drastically different domains that a given pretrained feature extractor is no longer optimal. Indeed, we find that if the feature extractor is far from optimal, Task2Vec fails at effective characterization (Sec.3.7.4). We therefore need an alternative approach that does not rely so heavily on a suitable feature representation.

The derivatives of random neural networks We want to declare two tasks to be similar if and only if a model trained on one produces a good feature extractor for the other, or alternatively, the optimal feature extractors for the two tasks are close to each other. A brute force approach to measuring task distance might thus be to separately train models for each task from the same initialization and look at how far the optima are in parameter space.

Of course, this is prohibitively expensive and obviously defeats the point, since we wanted to avoid training a separate model for the target task anyway. But what if we don't train these models the whole way? Can we instead just train these models for very few epochs or steps and then evaluate how far they are? Concretely, imagine we start the training for both tasks using the same initialization, and take a single step. If the optimal models for the two tasks are close to each other, one might imagine that the very first update will also be close. If we repeat this for multiple initializations and find

that the first update for the two tasks are always close, then one might conclude that the tasks are "similar". Standard optimization procedures rely on gradient descent, so this first update corresponds to the gradient of the randomly initialized model. Thus, we hypothesize that a measure of task similarity could be computed by calculating the *expected similarity between the gradients of the tasks at the same random initialization*.

The Neural Tangent Kernel, which was first proposed in [35], describes the convergence behavior of neural networks in the limit of infinite width. A side-effect of this analysis is a *kernel function* between data points that comes close to mimicking the behavior of trained neural networks, but *itself requires no training*. This kernel takes the form:

$$k(x, x') = E_\theta \langle \frac{\partial f(\theta, x)}{\partial \theta}, \frac{\partial f(\theta, x')}{\partial \theta} \rangle \tag{3.5}$$

where x and x' are data points (e.g., images), and θ is the parameters of a randomly initialized neural network drawn from a fixed distribution (typically Gaussian).

Our work essentially adapts the above kernel to operate on *tasks* instead of *points*. Based on the results with NTK, we reason that computing the NTK kernel over tasks instead of individual data points (by simply averaging the gradients of points in a task) thus provides a measure of similarity between the tasks.

We concretize this intuition as follows. Suppose we are given two tasks $T_1 = (X_1, Y_1)$ and $T_2 = (X_2, Y_2)$ consisting of images $X_k = \{x_i^{(k)}\}_{i=1}^{n_k}, k = 1, 2$ and corresponding labels $Y_k = \{y_i^{(k)}\}_{i=1}^{n_k}, k = 1, 2$.

Suppose L is a loss function such that given a feature extractor ϕ, $L(\phi, X, Y)$ measures how well the feature extractor is able to separate out the classes in the task (X, Y). We randomly initialize N feature extractors $\{\phi_i\}_{i=1}^N$ with parameters $\{\theta_i\}_{i=1}^N$. For each feature extractor, for each task, we compute the gradient of the loss with respect to the feature

vector parameters:

$$g_i^{(k)} = \frac{\partial L(\phi_i, X_k, Y_k)}{\partial \theta_i} \tag{3.6}$$

We then define a *kernel* between the two tasks as the average cosine distance between the two gradients :

$$k(T_1, T_2) = \frac{1}{N} \sum_{i=1}^{N} \frac{\langle g_i^{(1)}, g_i^{(2)} \rangle}{||g_i^{(1)}|| ||g_i^{(2)}||} \tag{3.7}$$

Because we are computing this task kernel using the gradients, we call this kernel the *task tangent kernel*. We only use the last residual block of the ResNet to compute the embedding as it contains the most channels.

Loss function:A key component here is the loss function L. Typical loss functions such as cross entropy operate on predictions rather than features, which presents difficulty given the permutation-variant nature of a randomly initialized classifier head. We use the Maximal Coding Rate Reduction loss(MCR^2)[87]. This loss measures how close same-class features are, and how spread out the dataset is in feature space. Specifically, we embed the images X using the feature extractor ϕ yielding embeddings $Z \in \mathbb{R}^{d \times m}$ (d being the feature dimensionality, m the total number of data points). Let the set of embeddings of class j be denoted by Z_j. Then the loss is defined as:

$$L(\phi, X, Y) = -R(Z) + R_{class}(Z) \tag{3.8}$$

$$R(Z) = \frac{1}{2} \log \det \left(I + \frac{d}{m\epsilon^2} Z Z^\top \right) \tag{3.9}$$

$$R_{class}(Z) = \sum_j \frac{m_j}{2m} \log \det \left(I + \frac{d}{m_j \epsilon^2} Z_j Z_j^\top \right). \tag{3.10}$$

With m_j as the number of data points with membership in class j and ϵ a "prescribed precision" constant ($\epsilon^2 = 0.5$ in our work, see [87] for details). The functions R and R_{class} describe the whole-dataset and per-class *coding rates*, measuring the compactness

of all or subsets of features. This loss encourages the dataset as a whole to be non-compact (discriminable) while the class subsets should be highly compact (clustered), bearing resemblance to the supervised contrastive learning objective such as in[39].

3.7 Experiments

3.7.1 Meta-Task and Baselines

We perform experiments on the **CUB+iNat** meta-task of the Task2Vec paper [2], denoted as $\mathcal{T}$. This set consists of 25 species classification tasks from Caltech-UCSD Birds[81] and 25 from iNaturalist[75]. Tasks are sets of species grouped at either the Order or Family level. For each individual task $T \in \mathcal{T}$, the benchmark requires us to choose an expert from $\mathcal{T} - \{T\}$. We measure the error obtained with this choice, relative to the error of the optimal choice, reporting the average relative error across all tasks.

Baselines: The **Random** baseline selects a task from $\mathcal{T} - \{T'\}$ uniformly at random, and uses the corresponding expert. **ImageNet Initialization** does not use any of the experts available in $\mathcal{T}$, but instead uses a pretrained ImageNet network every time (an option not available to selection algorithms). **Average Features** uses an ImageNet-pretrained feature extractor to compute the average feature vector of images in the task, which is used as a task embedding (under cosine distance). Other metrics based solely on ImageNet-pretrained features, such as the H-Divergence[40] (accuracy of a linear classifier separating domains) between tasks produced similar or worse results. **RSA** and **LEEP** are more sophisticated techniques that analyze the features produced by each expert-target pair. While these techniques prove to be quite effective, the computational cost of running inference using each expert quickly can become prohibitive as the num-

Method	Error Increase	Labels?	Pretraining?
LEEP [55]	20.8%	✓	✓
RSA [18]	8.8%	✓	✓
Random Selection	59.5%		
EMD [12]	51.3%	✓	✓
Average Features	39.2%		✓
ImageNet Init.*	30.2%		✓
Task2Vec (Rand. Init)	48.7%	✓	
Task2Vec (Orig)	8.9%	✓	✓
Task Tangent Kernel	21.4%	✓	
PseudoTask (ImageNet)	20.4%		✓
PseudoTask (Places365)	10.0%		✓

Table 3.1: Metric reported is the mean increase of relative error between a method's choice and the optimal, averaged across the 50 tasks of **Cub+iNat** from [2]. Gray indicates methods which require running inference with each proposed expert on the target dataset which quickly becomes computationally prohibitive. Both TTK and PseudoTask outperform all baselines that do not require running inference from each expert on the new dataset. Furthermore, PseudoTask using Places365 initialization almost equals supervised performance.

ber of considered tasks increases.

3.7.2 Main Results

We present our main findings in Table 3.1 and Figure 3.3. Both proposed methods outperform baselines, with PseudoTask achieving performance on par with Task2Vec. Note that LEEP and RSA require computing predictions for each expert/task pair; in the case of RSA an entire model must be *trained on the target task* before the expert

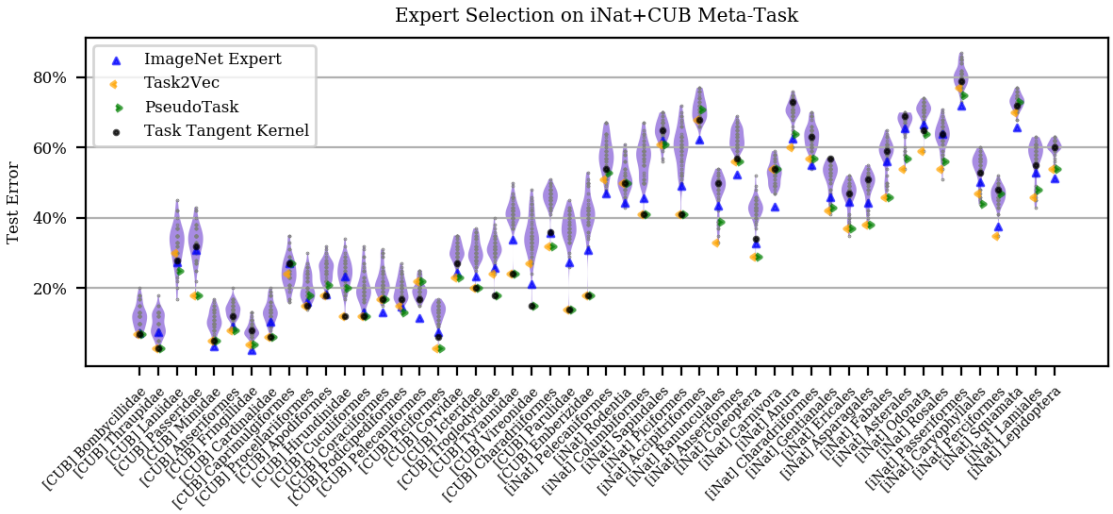


Figure 3.3: Violin plot of the error on each target task (x-axis) obtained by training a linear head on top of a model from each of the other 49 tasks. Markers indicate selection algorithm choices. Both of our methods (PseudoTask and Task Tangent Kernel) reliably outperform using an ImageNet feature extractor. The experimental setting is the same as Figure 3 of [2].

initialization is chosen. In contrast, embedding methods only require per-pair vector arithmetic and representations are persistent.

Task Tangent Kernel TTK has the most significant handicap, operating solely on the provided task dataset as opposed to other methods which employ networks pretrained on over a million images. Despite this, TTK is still useful, beating the strong baseline of initializing from ImageNet every time (ImageNet is not a permitted expert to select in the benchmark). We see in Section 3.7.4 that the performance by TTK is *far* superior to any other method operating off of random initialization and that this benefit stems from using *multiple randomized* networks, in accordance with theoretical work involving the Neural Tangent Kernel.

PseudoTask PseudoTask outperforms the baselines by even more significant margins than TTK. Furthermore, by using Places365 as the initialization for the probe network

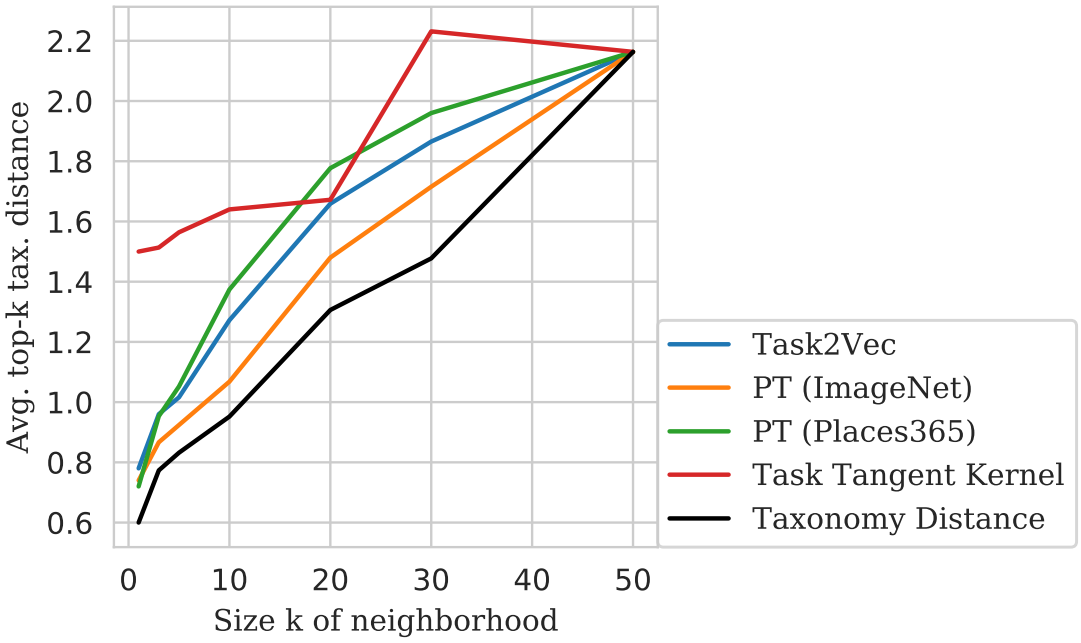


Figure 3.4: Average taxonomical distance between tasks in neighborhoods of varying sizes. Taxonomical distance is the how far up the phylogenetic tree (Order, Class, Phylum, Kingdom) the common root of the two tasks is. Black line represents the ground truth average distance in neighborhoods of a given size (calculated at each task in the meta-task). Preservation of taxonomical distance is desirable, demonstrating capture of the semantics of a task.

instead of ImageNet, we are able to halve the mean relative error increase. In doing so *we achieve results almost equal to the original supervised method despite no available labels*. We present possible reasons for the success of Places365 initialization in Section 3.7.5.

In Figure 3.4, we compare the taxonomical distance between tasks to the induced symmetric distances of our methods. PseudoTask (ImageNet) has an even higher correlation with the ground truth taxonomy than the original Task2Vec.

Example choices: Figure 3.5 demonstrates PseudoTask's decision-making ability. As an example, the 2nd column "Piciformes" is the task of classifying the 7 different types

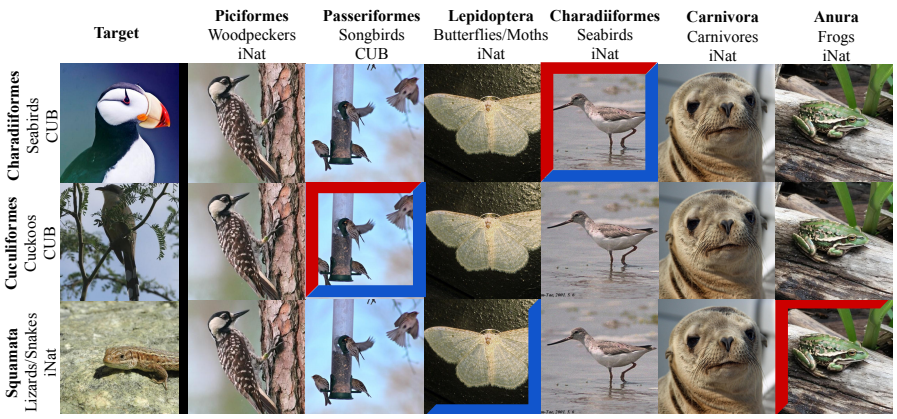


Figure 3.5: Examples of optimal selections vs. those made by PseudoTask. Source tasks (columns) are selected to transfer to target tasks (rows). Representative images of each dataset are shown. A blue bracket indicates the optimal choice, a red bracket the algorithm's choice. PseudoTask selections are made without the use of labels.

of woodpeckers in CUB. Walking through the selections:

1. Charadiiformes (CUB) is properly matched to its counterpart from iNat.
2. Cuculiformes (CUB) is properly matched to Passeriformes (iNat), a very large songbird dataset.
3. Squamata (iNat, lizards and snakes) is "incorrectly" matched to Anura (iNat, frogs). This demonstrates how transferability and semantics are not perfectly correlated: taxonomically, Squamata is much closer to Anura than Lepidoptera.

3.7.3 Other Datasets

CUB Attributes One possible concern with PseudoTask is that it does not take into account the labels of the task. As such, it might end up choosing experts relevant to the domain, but not necessarily to the task at hand.We test this scenario by forming a variant of the **CUB+iNat** benchmark, called **CUB-bp+iNat**, where the CUB labels are the *breast pattern* of the birds instead of species classes. Per-task embeddings are

Method	CUB-bp	Cars
Random	12.6%	28.2%
ImageNet Initialization	11.4%	-5.9%
PseudoTask (ImageNet)	9.3%	18.2%
Task Tangent Kernel	13.4%	14.0%
Task2Vec (orig)	12.6%	14.2%

Table 3.2: (L) Relative errors on the CUB half of the **CUBbp+iNat**. We see that, despite no knowledge of the label shift, PseudoTask performs substantially better than alternatives. (R) Relative errors on the **Cars** meta-task. Denominators in relative error calculation are buffered by 1 due to presence of zeros (perfect accuracies). We note that ImageNet Init. is a much stronger baseline than in **CUB+iNat** due to the strength of self-selection in **Cars**. For 80% of the tasks incorporating any extra data actually hurts performance.

re-calculated for Task2Vec and TTK, while experts are trained/transferred to obtain per-selection accuracies. Results are shown in Table 3.2 (L) for selections on the CUB half of the meta-task. This concern does not prove to be a detriment, on the contrary, PseudoTask performs better relative to other methods than in the purely class-based setting. Intriguingly, this suggests that the more important factor in transfer is the domain, rather than the precise semantics of the labels themselves.

Cars To validate our models on varied types of imagery, we create a new benchmark **Cars** from the Stanford Cars dataset[44] [1]. The tasks are manufacturers (e.g. Audi) that have more than one model of car in the dataset. In Table 3.2 (R), we observe that PseudoTask performs nearly equally to Task2Vec and notably TTK's performance exceeds both of them. This confirms our success of adapting the Task2Vec algorithm to function well without labels or initialization.

[1] We create the **Cars** benchmark as the **Mixed** benchmark of [2] requires attribute labels that are not publicly available.

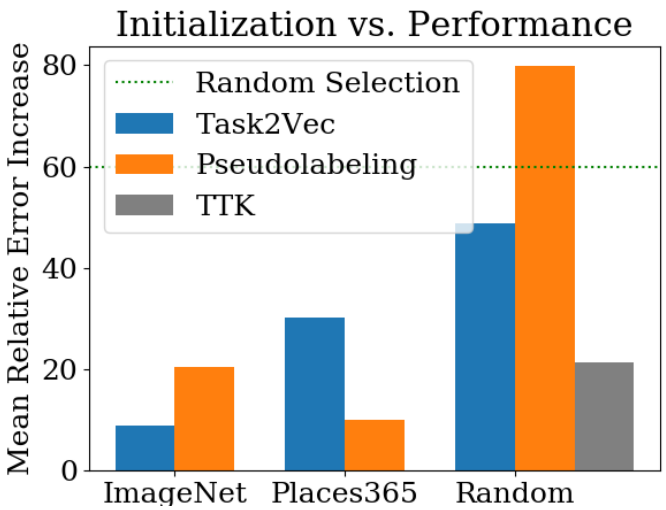


Figure 3.6: Error increase percentage for method-initialization pairs (lower is better). With random initialization, Task Tangent Kernel dramatically outperforms the other methods, more than halving the error. Hyperparameters are constant across initializations.

3.7.4 Varied Initializations

The significance of initialization is demonstrated in Figure 3.6. As previously noted, PseudoTask performs significantly better with Places365 pretraining. Hypothesized reasons for this improvement are presented in Section 3.7.5. Task2Vec performs markedly worse when using Places365, but in the Supplementary we show that this is solely due to hyperparameter tuning.

Both Task2Vec and PseudoTask suffer dramatically with random initialization. Task2Vec still is significantly better than random choice, while PseudoTask is worse. We attribute this difference to PseudoTask having the same dependencies on the probe network as Task2Vec while additionally relying on the induced pseudolabels.

By design, TTK does not rely on a probe network and has vastly superior performance compared to other methods without network initialization, where the error is over a factor of 2 lower. We confirm in Figure 3.7 that the performance of TTK stems

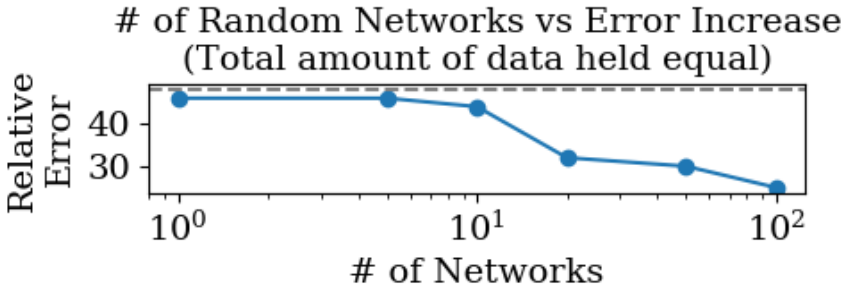


Figure 3.7: Each data point is a TTK experiment using x networks with $100/x$ batches of size 128 per network. The dashed horizontal line is the performance of Task2Vec on a random initialization. We see that TTK with a single random network performs comparably to Task2Vec, and that a diversity of models is more beneficial than repeated gradient computations on a single network.

from the diversity of models used. On the far left of the figure, only a single network is trained with 100 batches of data; this method is fairly equivalent to Task2Vec in both nature and performance. Performance improves with the number of networks, validating the motivation of the TTK: the expectation over random models can substitute for a powerful feature extractor.

3.7.5 ImageNet vs. Places365

In both Task2Vec and PseudoTask, Places365 initialization ultimately yields superior performance than the ImageNet counterparts. This is quite unintuitive, as generally ImageNet transfers better than Places365[26, 91] which has made it the standard in transfer learning. Better pseudo-classification performance is not the reason; the Adjusted Mutual Information between the pseudolabels and ground truth is quite small, ImageNet has the larger score at 0.05 demonstrating a lack of label consistency despite the strong selection performance for both models.

This analysis focuses on PseudoTask. The average prediction confidence (post-softmax) of ImageNet and Places365 are 0.19 and 0.18 respectively, despite the former

having nearly triple the available classes. These correspond to $19,000\times$ and $6,600\times$ higher than a uniform random guess. We theorize that the difference in performance stems from ImageNet making higher confidence predictions, even when incorrect, resulting in less informative embeddings.

We record the class predictions from both models for a batch from each task of size $min(100, n_{dataset})$. ImageNet averages 84 different predictions per class (8% of 1000 total possibilities) while Places365 uses 77 different predictions (21% of 365 total possibilities). Across all tasks, ImageNet predicts 857 classes (85.7% of maximum possible) and Places365 354 (97% of maximum possible).

We hypothesize that this relative softness of prediction from the Places365 model is beneficial for PseudoTask because the objective becomes more akin to contrastive learning (e.g., MoCo or SimCLR[32, 10]) instead of a one-hot classification problem. Consistently, PseudoTask with hard pseudolabels performs far worse. By softening the classification problem, the network parameters might equalize in discriminative power (and thus gradient), preventing a small subset of terms from dominating the embedding calculation.

We visualize the values of all embeddings in Figure 3.8. ImageNet has a longer tail of values while Places365 has a concentrated peak at relatively low values for both algorithms. Thus ImageNet yields "spikier" embeddings whose large values will dominate the distance calculations.

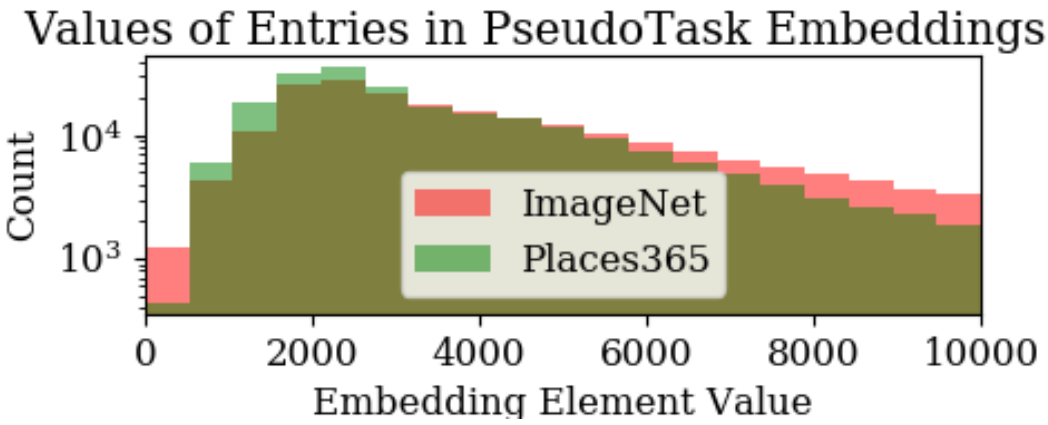


Figure 3.8: Histograms of the values contained in the PseudoTask embeddings. ImageNet has more extreme values, both high and low, than Places365. We attribute this difference to the softness of Places365 labels relative to ImageNet.

3.8 Conclusion

In this work, we designed methods to overcome current limitations of expert selection algorithms: namely performance without labels or model initializations. Our zero-label approach, PseudoTask, achieved nearly equal performance to the previous supervised version, Task2Vec. We demonstrated that both of these methods fail without pretrained model initialization and created the Task Tangent Kernel which doubles the performance of other methods when pretraining is not available. Promising lines of future work include semi-supervised expert selection, zero-label zero-initialization methods, and extension to other data forms such as shapes or natural language.

APPENDIX A

APPENDIX

For reproducibility, refer to Section A.1 for datasets and the pipeline search space. All the code is in the GitHub repository at `https://github.com/udellgroup/oboe`.

A.1 Reproducibility for Meta-training

A.1.1 Meta-training OpenML Datasets

Indices of the OpenML datasets we use for meta-training: 2, 3, 5, 7, 9, 11, 12, 13, 14, 15, 16, 18, 20, 22, 23, 24, 25, 27, 28, 29, 30, 31, 35, 36, 37, 38, 39, 40, 41, 42, 44, 46, 48, 50, 53, 54, 59, 60, 181, 182, 183, 187, 285, 307, 313, 316, 329, 336, 337, 338, 375, 377, 389, 446, 450, 458, 463, 469, 475, 694, 715, 717, 718, 720, 721, 723, 725, 728, 730, 732, 733, 735, 737, 740, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 753, 763, 769, 773, 776, 778, 779, 788, 792, 794, 796, 797, 799, 803, 805, 806, 807, 813, 818, 819, 820, 824, 825, 826, 830, 832, 837, 838, 847, 853, 855, 863, 866, 869, 870, 871, 873, 877, 880, 884, 888, 896, 900, 903, 904, 906, 907, 908, 909, 910, 911, 912, 913, 915, 917, 920, 923, 925, 926, 933, 934, 935, 936, 937, 941, 943, 952, 953, 954, 955, 958, 962, 970, 971, 973, 976, 978, 979, 980, 983, 987, 991, 994, 995, 996, 997, 1005, 1011, 1012, 1014, 1016, 1020, 1021, 1022, 1025, 1026, 1038, 1039, 1041, 1042, 1048, 1049, 1050, 1054, 1056, 1063, 1065, 1067, 1068, 1069, 1071, 1073, 1100, 1115, 1116, 1121, 4134, 40966, 40971, 40975, 40978, 40979, 40981, 40982, 40983, 40984, 40994, 40997, 41000, 41004, 41005.

A.1.2 Meta-test UCI Datasets

”acute-inflammations-1”, ”acute-inflammations-2”, ”arrhythmia”, ”balance-scale”, ”balloons-a”, ”balloons-b”, ”balloons-c”, ”balloons-d”, ”banknote-authentication”, ”blood-transfusion-service-center”, ”breast-cancer-wisconsin-diagnostic”, ”breast-cancer-wisconsin-original”, ”breast-cancer-wisconsin-prognostic”, ”breast-cancer”, ”car-evaluation”, ”chess-king-rook-vs-king-pawn”, ”chess-king-rook-vs-king”, ”climate-model-simulation-crashes”, ”cnae-9”, ”congressional-voting-records”, ”connectionist-bench-sonar”, ”connectionist-bench”, ”contraceptive-method-choice”, ”credit-approval”, ”cylinder-bands”, ”dermatology”, ”echocardiogram”, ”ecoli”, ”fertility”, ”flags”, ”glass-identification”, ”haberman-survival”, ”hayes-roth”, ”heart-disease-cleveland”, ”heart-disease-hungarian”, ”heart-disease-switzerland”, ”heart-disease-va”, ”hepatitis”, ”hill-valley-noise”, ”hill-valley”, ”horse-colic”, ”image-segmentation”, ”indian-liver-patient”, ”ionosphere”, ”iris”, ”lenses”, ”letter-recognition”, ”libras-movement”, ”lung-cancer”, ”magic-gamma-telescope”, ”mammographic-mass”, ”monks-problems-1”, ”monks-problems-2”, ”monks-problems-3”, ”mushroom”, ”nursery”, ”optical-recognition-handwritten-digits”, ”ozone-level-detection-eight”, ”ozone-level-detection-one”, ”parkinsons”, ”pen-based-recognition-handwritten-digits”, ”planning-relax”, ”poker-hand”, ”post-operative-patient”, ”qsar-biodegradation”, ”seeds”, ”seismic-bumps”, ”shuttle-landing-control”, ”skin-segmentation”, ”soybean-large”, ”soybean-small”, ”spambase”, ”spect-heart”, ”spectf-heart”, ”statlog-project-german-credit”, ”statlog-project-landsat-satellite”, ”teaching-assistant-evaluation”, ”thoracic-surgery”, ”thyroid-disease-allbp”, ”thyroid-disease-allhyper”, ”thyroid-disease-allhypo”, ”thyroid-disease-allrep”, ”thyroid-disease-ann-thyroid”, ”thyroid-disease-dis”, ”thyroid-disease-new-thyroid”, ”thyroid-disease-sick-euthyroid”, ”thyroid-disease-sick”, ”thyroid-disease-thyroid-0387”, ”tic-tac-toe-endgame”, ”trains”, ”wall-following-robot-navigation-2”, ”wall-following-robot-navigation-24”, ”wall-following-robot-navigation-4”, ”wine”, ”yeast”, ”zoo”.

Table A.1: Pipeline search space

Component	Algorithm type	Hyperparameter names (values)
Data imputer	Simple imputer	strategy (mean, median, most_frequent, co
Encoder	null	-
	OneHotEncoder	handle_unknown (ignore), sparse (0)
Standardizer	null	-
	StandardScaler	-
Dimensionality reducer	null	-
	PCA	n_components (25%, 50%, 75%)
	VarianceThreshold	-
	SelectKBest	k (25%, 50%, 75%)
Estimator	Adaboost	n_estimators (50,100), learning_rate (1.0,
	Decision tree	min_samples_split (2,4,8,16,32,64,128,256,
	Extra trees	min_samples_split (2,4,8,16,32,64,128,256,512,1024,0.01,0.0 criterion (gini,entropy)
	Gradient boosting	learning_rate (0.001,0.01,0.025,0.05,0.1,(max_depth (3, 6), max_features (null,log2)
	Gaussian naive Bayes	-
	Perceptron	-
	kNN	n_neighbors (1,3,5,7,9,11,13,15), p (1,2)
	Logistic regression	C (0.25,0.5,0.75,1,1.5,2,3,4), solver (li
	Multilayer percep-tron	learning_rate_init (1e-4,0.001,0.01), learning_rate (adaptive), solver (sgd,adam), alpha (1e-4, 0.01)
	Random forest	min_samples_split (2,4,8,16,32,64,128,256,512,1024,0.01,0.0 criterion (gini,entropy)
	Linear SVM	C (0.125,0.25,0.5,0.75,1,2,4,8,16)

A.1.3 Pipeline Search Space

We build pipelines using scikit-learn [60] primitives. The available components are listed in Table A.1. "null" denotes a pass-through.

A.2 Experiment Design for Weighted Least Squares

When factorizing the error matrix by SVD, we approximate performance of different pipelines to different accuracies. Different accuracies can be characterized by different variances in the linear regression model, thus the weighted least squares (WLS) model that would theoretically give the best linear unbiased estimate to the new dataset embedding may perform better.

In detail, recall that the constrained D-optimal experiment design formulation relies on the assumption that given a low rank matrix multiplication model $X^\top Y = E$, the error term in linear regression $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$, which means each pipeline is predicted to the same accuracy. In the WLS version of our pipeline performance estimation setting, the pipeline performance vector of the new dataset can be written as $e = Y^\top x + \epsilon$, in which $\epsilon \sim \mathcal{N}(0, \Sigma)$. $\Sigma = \text{diag}(\sigma_1^2, \sigma_2^2, \ldots, \sigma_n^2)$ is a covariance matrix; diagonal in the weighted least squares setting. For each pipeline $j \in [n]$, we estimate the variance by the sample variance of $e_j - X^\top y_j$, and show a histogram in Figure A.1.

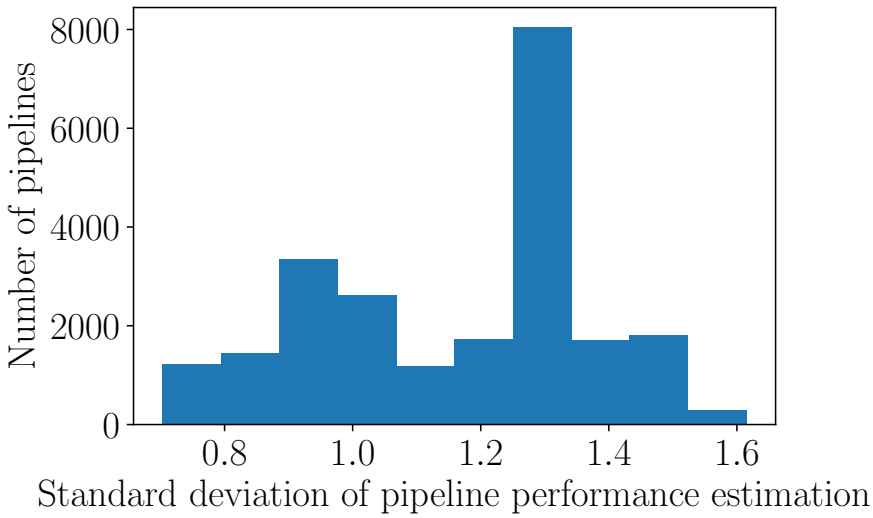


Figure A.1: Standard deviation of prediction accuracy of each pipeline, across meta-training datasets.

In this case, the time-constrained D-experiment design problem to solve becomes

$$\begin{aligned} \text{minimize} \quad & \log\det\Big(\textstyle\sum_{j=1}^{n} v_j \frac{y_j y_j^\top}{\sigma_j^2}\Big)^{-1} \\ \text{subject to} \quad & \sum_{j=1}^{n} v_j \hat{t}_j \le \tau \\ & v_j \in \{0,1\}, \forall j \in [n]. \end{aligned} \tag{A.1}$$

The corresponding greedy approach, which we call *weighted-greedy*, is shown as Algorithm 6. It differs from the ordinary greedy approach in that each y_j is scaled by $1/\sigma_j$. Figure A.2 shows its performance compared to convexification and greedy. We can see the weighted-greedy approach performs similarly to the ordinary greedy approach in our experiments.

Algorithm 6 Greedy algorithm for time-constrained D-design in WLS setting, with QR initialization

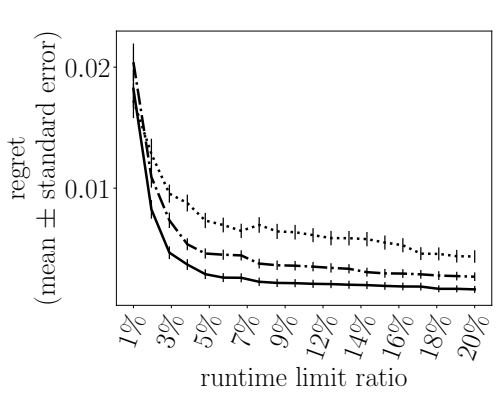
Input: design vectors $\{y_j\}_{j=1}^n$, in which $y_j \in \mathbb{R}^k$; pipeline estimation variances $\{\sigma_j^2\}_{j=1}^n$, (predicted) running time of all pipelines $\{\hat{t}_i\}_{i=1}^n$; maximum running time τ

Output: The selected set of designs $S \subseteq [n]$

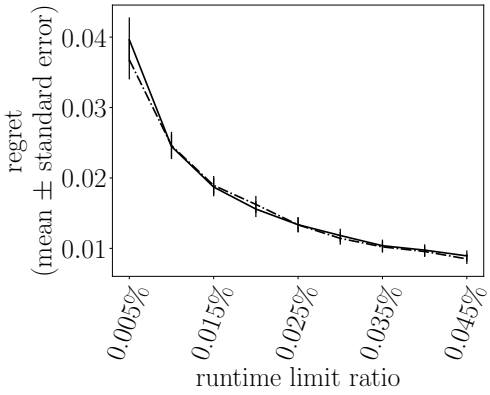
1 $y_j \leftarrow y_j/\sigma_j, \forall j \in [n]$

2 $S_0 \leftarrow$ `QR_initialization`$(\{y_j\}_{j=1}^n, \{\hat{t}_i\}_{i=1}^n, \tau)$

3 $S \leftarrow$ `Greedy_without_repetition`$(\{y_j\}_{j=1}^n, \{\hat{t}_i\}_{i=1}^n, \tau, S_0)$



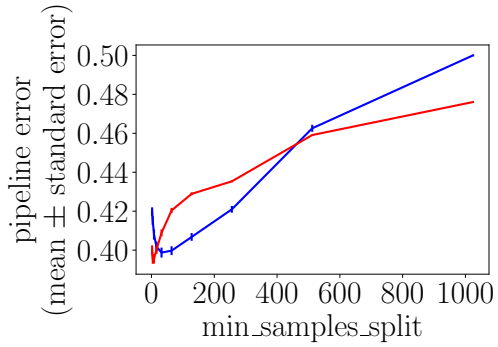
(a) Regret on the subsampled error matrix (215-by-183) for estimator search, including the weighted-greedy method.



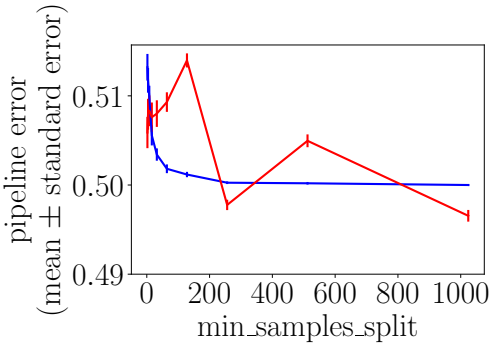
(b) Regret on the full error matrix (215-by-23424) for pipeline search, including the weighted-greedy method.

Figure A.2: Comparison of time-constrained experiment design methods, including the weighted-greedy method.

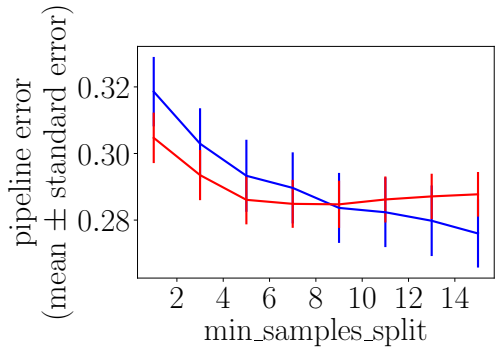
A.3 Zoomed-in Hyperparameter Landscapes



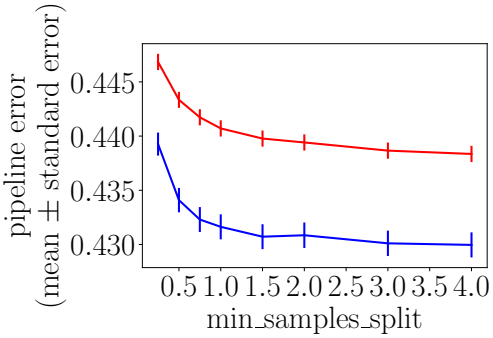
(a) Extra trees on Dataset 23 (1473 points, 10 features)



(b) Decision tree on Dataset 1014 (797 points, 5 features)



(c) kNN on Dataset 799 (1000 points, 6 features)



(d) Logistic regression on Dataset 40971 (1000 data points, 24 features)

Figure A.3: Zoomed-in hyperparameter landscapes in Figure 2.10. The y-axes here do not start from 0.

BIBLIOGRAPHY

[1] Salisu Mamman Abdulrahman, Pavel Brazdil, Jan N van Rijn, and Joaquin Vanschoren. Speeding up algorithm selection using average ranking and active testing by introducing runtime. *Machine learning*, 107(1):79–108, 2018.

[2] Alessandro Achille, Michael Lam, Rahul Tewari, Avinash Ravichandran, Subhransu Maji, Charless C Fowlkes, Stefano Soatto, and Pietro Perona. Task2vec: Task embedding for meta-learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6430–6439, 2019.

[3] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pages 3981–3989, 2016.

[4] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In *Advances in Neural Information Processing Systems*, pages 8141–8150, 2019.

[5] Artem Babenko, Anton Slesarev, Alexandr Chigorin, and Victor Lempitsky. Neural codes for image retrieval. In *European conference on computer vision*, pages 584–599. Springer, 2014.

[6] George EP Box. Science and statistics. *Journal of the American Statistical Association*, 71(356):791–799, 1976.

[7] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[8] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[9] J Douglas Carroll and Jih-Jie Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition. *Psychometrika*, 35(3):283–319, 1970.

[10] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020.

[11] Yin Cui, Yang Song, Chen Sun, Andrew Howard, and Serge Belongie. Large scale fine-grained categorization and domain-specific transfer learning. In *Proceedings*

of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2018.

[12] Yin Cui, Yang Song, Chen Sun, Andrew Howard, and Serge Belongie. Large scale fine-grained categorization and domain-specific transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4109–4118, 2018.

[13] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[14] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[15] Iddo Drori, Yamuna Krishnamurthy, Remi Rampin, Raoni Lourenço, J One, Kyunghyun Cho, Claudio Silva, and Juliana Freire. Alphad3m: Machine learning pipeline synthesis. In *AutoML Workshop at ICML*, 2018.

[16] Iddo Drori, Lu Liu, Yi Nian, Sharath C Koorathota, Jie S Li, Antonio Khalil Moretti, Juliana Freire, and Madeleine Udell. Automl using metadata language embeddings. *arXiv preprint arXiv:1910.03698*, 2019.

[17] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[18] Kshitij Dwivedi and Gemma Roig. Representation similarity analysis for efficient task taxonomy and transfer learning, 2019.

[19] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in neural information processing systems*, pages 2962–2970, 2015.

[20] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Using meta-learning to initialize Bayesian optimization of hyperparameters. In *International Conference on Meta-learning and Algorithm Selection*, pages 3–10. Citeseer, 2014.

[21] Matthias Feurer, Jan N van Rijn, Arlind Kadra, Pieter Gijsbers, Neeratyoy Mallik, Sahithya Ravi, Andreas Müller, Joaquin Vanschoren, and Frank Hutter. Openml-python: an extensible python api for openml. *arXiv preprint arXiv:1911.02490*, 2019.

[22] Nicolo Fusi, Rishit Sheth, and Melih Elibol. Probabilistic matrix factorization

for automated machine learning. In *Advances in Neural Information Processing Systems*, pages 3348–3357, 2018.

[23] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Advances in Neural Information Processing Systems*, pages 8789–8798, 2018.

[24] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. In *International Conference on Learning Representations*, 2018.

[25] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU press, 2012.

[26] Priya Goyal, Dhruv Mahajan, Abhinav Gupta, and Ishan Misra. Scaling and benchmarking self-supervised visual representation learning. *arXiv preprint arXiv:1905.01235*, 2019.

[27] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in Neural Information Processing Systems*, 33, 2020.

[28] Ming Gu and Stanley C Eisenstat. Efficient algorithms for computing a strong rank-revealing qr factorization. *SIAM Journal on Scientific Computing*, 17(4):848–869, 1996.

[29] William W Hager. Updating the inverse of a matrix. *SIAM review*, 31(2):221–239, 1989.

[30] Richard A Harshman et al. Foundations of the parafac procedure: Models and conditions for an" explanatory" multimodal factor analysis. 1970.

[31] David A Harville. Matrix algebra from a statistician's perspective, 1998.

[32] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. *arXiv preprint arXiv:1911.05722*, 2019.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning

for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[34] Frank Hutter, Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014.

[35] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

[36] RC St John and Norman R Draper. D-optimality for regression designs: a review. *Technometrics*, 17(1):15–23, 1975.

[37] Hadi S Jomaa, Lars Schmidt-Thieme, and Josif Grabocka. Dataset2vec: Learning dataset meta-features. *arXiv preprint arXiv:1905.11063*, 2019.

[38] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *ICLR*, 2017.

[39] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning, 2021.

[40] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting change in data streams. 2004.

[41] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[42] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Large scale learning of general visual representations for transfer. *arXiv preprint arXiv:1912.11370*, 2019.

[43] Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better imagenet models transfer better? In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2661–2671, 2019.

[44] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization.

[45] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and brain sciences*, 40, 2017.

[46] Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks.

[47] Rui Leite, Pavel Brazdil, and Joaquin Vanschoren. Selecting classification algorithms with active testing. In *International workshop on machine learning and data mining in pattern recognition*, pages 117–131. Springer, 2012.

[48] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pages 6389–6399, 2018.

[49] Zhiyuan Li, Ruosong Wang, Dingli Yu, Simon S. Du, Wei Hu, Ruslan Salakhutdinov, and Sanjeev Arora. Enhanced convolutional neural tangent kernels, 2019.

[50] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[51] Sijia Liu, Parikshit Ram, Deepak Vijaykeerthy, Djallel Bouneffouf, Gregory Bramble, Horst Samulowitz, Dakuo Wang, Andrew Conn, and Alexander Gray. An admm based framework for automl pipeline configuration. *arXiv preprint arXiv:1905.00424*, 2019.

[52] Vivek Madan, Mohit Singh, Uthaipon Tantipongpipat, and Weijun Xie. Combinatorial algorithms for optimal design. In *Conference on Learning Theory*, pages 2210–2258, 2019.

[53] Pyry Matikainen, Rahul Sukthankar, and Martial Hebert. Model recommendation for action recognition. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2256–2263. IEEE, 2012.

[54] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14(1):265–294, 1978.

[55] Cuong V. Nguyen, Tal Hassner, Matthias Seeger, and Cedric Archambeau. Leep: A new measure to evaluate transferability of learned representations, 2020.

[56] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representions by solving jigsaw puzzles. In *ECCV*, 2016.

[57] Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A. Alemi, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. Neural tangents: Fast and easy infinite neural networks in python. In *International Conference on Learning Representations*, 2020.

[58] Randal S Olson and Jason H Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Automated Machine Learning*, pages 151–160. Springer, 2019.

[59] Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[60] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[61] Bernhard Pfahringer, Hilan Bensusan, and Christophe G Giraud-Carrier. Meta-Learning by Landmarking Various Learning Algorithms. In *ICML*, pages 743–750, 2000.

[62] Cheng Perng Phoo and Bharath Hariharan. Self-training for few-shot transfer across extreme task differences. *arXiv preprint arXiv:2010.07734*, 2020.

[63] Friedrich Pukelsheim. *Optimal design of experiments*, volume 50. SIAM, 1993.

[64] Yong Rui, Thomas S Huang, and Shih-Fu Chang. Image retrieval: Current techniques, promising directions, and open issues. *Journal of visual communication and image representation*, 10(1):39–62, 1999.

[65] Robert E Schapire. The boosting approach to machine learning: An overview. In *Nonlinear estimation and classification*, pages 149–171. Springer, 2003.

[66] 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, 2019.

[67] Jack Sherman and Winifred J Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127, 1950.

[68] Michael R Smith, Logan Mitchell, Christophe Giraud-Carrier, and Tony Martinez. Recommending learning algorithms and their associated hyperparameters. *arXiv preprint arXiv:1407.1890*, 2014.

[69] Qingquan Song, Hancheng Ge, James Caverlee, and Xia Hu. Tensor completion algorithms in big data analytics. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 13(1):1–48, 2019.

[70] Gencer Sumbul, Marcela Charfuelan, Begüm Demir, and Volker Markl. Bigearthnet: A large-scale benchmark archive for remote sensing image understanding.

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

[72] Antonio Torralba and Alexei A Efros. Unbiased look at dataset bias. In *CVPR 2011*, pages 1521–1528. IEEE, 2011.

[73] Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[74] Paul Upchurch, Jacob Gardner, Geoff Pleiss, Robert Pless, Noah Snavely, Kavita Bala, and Kilian Weinberger. Deep feature interpolation for image content changes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7064–7073, 2017.

[75] Grant Van Horn, Oisin Mac Aodha, Yang Song, Yin Cui, Chen Sun, Alex Shepard, Hartwig Adam, Pietro Perona, and Serge Belongie. The inaturalist species classification and detection dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[76] Joaquin Vanschoren. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*, 2018.

[77] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

[78] Abraham Wald. On the efficient design of statistical investigations. *The Annals of Mathematical Statistics*, 14(2):134–140, 1943.

[79] Bram Wallace and Bharath Hariharan. Extending and analyzing self-supervised learning across domains, 2020.

[80] Yu-Xiong Wang and Martial Hebert. Model recommendation: Generating object detectors from few samples. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1619–1628, 2015.

[81] Peter Welinder, Steve Branson, Takeshi Mita, Catherine Wah, Florian Schroff, Serge Belongie, and Pietro Perona. Caltech-ucsd birds 200. 2010.

[82] M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Learning hyperparameter optimization initializations. In *IEEE International Conference on Data Science and Advanced Analytics*, pages 1–10, Oct 2015.

[83] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[84] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[85] Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V. Le. Self-training with noisy student improves imagenet classification, 2020.

[86] Chengrun Yang, Yuji Akimoto, Dae Won Kim, and Madeleine Udell. Oboe: Collaborative filtering for automl model selection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1173–1183, 2019.

[87] Yaodong Yu, Kwan Ho Ryan Chan, Chong You, Chaobing Song, and Yi Ma. Learning diverse and discriminative representations via the principle of maximal coding rate reduction, 2020.

[88] Amir R. Zamir, Alexander Sax, William B. Shen, Leonidas J. Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018.

[89] Peng Zhang, Jiuling Wang, Ali Farhadi, Martial Hebert, and Devi Parikh. Predicting failures of vision systems. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3566–3573, 2014.

[90] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Pro-*

ceedings of the IEEE conference on computer vision and pattern recognition, pages 586–595, 2018.

[91] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.