UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

CURRICULUM LEARNING APPLIED TO THE COMBINED ALGORITHM SELECTION AND HYPERPARAMETER OPTIMIZATION PROBLEM

LUCAS NILDAIMON DOS SANTOS SILVA

ORIENTADOR: PROF. DR. DIEGO FURTADO SILVA

São Carlos – SP

Março/2020

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

CURRICULUM LEARNING APPLIED TO THE COMBINED ALGORITHM SELECTION AND HYPERPARAMETER OPTIMIZATION PROBLEM

LUCAS NILDAIMON DOS SANTOS SILVA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Metodologia e técnicas de computação. Linha de pesquisa: Inteligência artificial.
Orientador: Prof. Dr. Diego Furtado Silva

São Carlos – SP

Março/2020

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Defesa de Dissertação de Mestrado do candidato Lucas Nildaimon dos Santos Silva, realizada em 25/05/2021.

Comissão Julgadora:

Prof. Dr. Diego Furtado Silva (UFSCar)

Profa. Dra. Heloisa de Arruda Camargo (UFSCar)

Prof. Dr. Andre Carlos Ponce de Leon Ferreira de Carvalho (USP)

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

O Relatório de Defesa assinado pelos membros da Comissão Julgadora encontra-se arquivado junto ao Programa de Pós-Graduação em Ciência da Computação.

Dedico este trabalho à minha família.

AGRADECIMENTOS

A Deus, que me abençoou durante toda minha caminhada.

Aos meus pais, Têotonio e Leudes, que sempre acreditaram no meu potencial e estavam lá para me apoiar.

A minha melhor amiga e companheira, Mahayra, por estar sempre disposta a me ajudar, por todo carinho e conforto nos momentos difíceis.

Ao orientador deste trabalho de pesquisa, Prof. Dr. Diego Furtado Silva, que através das suas vastas experiências e competências, me auxiliou em todas as dúvidas e questionamentos que apresentei. Sem sua assistência e dedicação a fim de transmitir seus conhecimentos para realização das pesquisas, não seria possível o correto desenvolvimento deste trabalho.

À companhia B2W Digital, por oferecer a possibilidade de trabalhar em um projeto real, proporcionando um aprendizado significativo, além de se colocar como parceira dessa pesquisa.

Aos professores escolhidos para auxiliar no projeto Apache Marvin-AI em conjunto com meu orientador, Profª. Dra. Helena de Medeiros Caseli e Prof. Dr. Daniel Lucrédio, que além de prestar assistências e apoios durante o projeto da empresa, também realizaram diversas contribuições para o projeto de pesquisa do mestrado. Os três professores citados foram muito coerentes e não ajudaram apenas nas relações acadêmicas, mas também foram extremamente compreensíveis nas relações humanas. Os três são ótimas pessoas e quero levar para a vida toda como mestres e amigos.

Aos meus amigos e companheiros de laboratório Lucas Cardoso Silva, Fernando Zagatti e Bruno Sette, pelo apoio mútuo e conhecimento compartilhado. Que nós possamos continuar nessa caminhada juntos e firmes.

Aos membros da banca Prof. Dr. André Carlos Ponce de Leon Ferreira de Carvalho e Prof. Dra. Heloisa de Arruda Camargo, por aceitarem ceder parte de seu tempo para avaliar e enriquecer meu trabalho.

Ao corpo docente do Departamento de Computação da Universidade Federal de São Carlos pelo conhecimento obtido através de suas dedicadas aulas e pelo esforço em auxiliar todos os alunos.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) pelo financiamento parcial deste projeto.

À Universidade Federal de São Carlos, pelo programa e estrutura de excelência, fazendo com que, o prazer por aprender e desenvolver ciência seja ímpar.

Nunca esqueça onde reside sua força
Então volte pras origens, é o colo de quem você ama.

Gustavo Pereira Marques, Djonga

RESUMO

A automatização do aprendizado de máquina (AutoML) tem como objetivo encontrar o melhor pipeline de aprendizado de máquina (ML) em um espaço de pesquisa complexo e de grande dimensionalidade, avaliando várias configurações de algoritmos. O treinamento de vários algoritmos de ML custa tempo e, como as ferramentas de AutoML geralmente obedecem a uma restrição de tempo, a exploração do espaço de pesquisa pode encontrar resultados abaixo do ideal. Neste trabalho, exploramos a aplicação de técnicas de aprendizado de currículo para superar essa limitação. A aprendizagem curricular e a aprendizagem anti-curricular são estratégias para ordenar exemplos durante o treinamento do modelo com base em sua dificuldade. Estes mostraram melhorar o desempenho de modelos e acelerar o processo de treinamento em investigações empíricas anteriores usando modelos baseados em otimização. Aplicamos e comparamos estratégias de currículo em dois otimizadores de um sistema AutoML para melhorar a exploração do espaço de pesquisa e encontrar pipelines de aprendizado de máquina de bom desempenho com eficiência. Os resultados indicam que o AutoML pode se beneficiar de uma estratégia curricular. Na maioria dos cenários avaliados, as estratégias de currículo levaram o algoritmo AutoML a melhores resultados de classificação.

Palavras-chave: AutoML, Aprendizagem curricular, Aprendizagem de máquina

ABSTRACT

AutoML has the goal to find the best Machine Learning (ML) pipeline in a complex and high dimensional search space by evaluating multiple algorithm configurations. Training multiple ML algorithms is time costly, and as AutoML tools usually obey a time constraint, the exploration of the search space may find sub-optimal results. In this work, we explore the application of curriculum learning techniques to overcome this limitation. Curriculum learning and anti-curriculum learning are strategies for ordering examples during model training based on their difficulty. These have shown to improve model performance and accelerate the training process on previous empirical investigations using optimization-based models. We apply and compare curriculum strategies on two optimizers of an AutoML system to accelerate the search space exploration and find good performing machine learning pipelines with efficiency. The results indicate that AutoML can benefit from a curriculum strategy. In most of the evaluated scenarios, the curriculum strategies led the AutoML algorithm to better classification results.

Keywords: AutoML, Curriculum learning, Machine learning

LIST OF FIGURES

GLOSSARY

AntiCL – *Anti-curriculum Learning*

AutoML – *Automated Machine Learning*

BOHB – *Bayesian Optimization and Hyperband*

BOSH – *Bayesian Optimization and Successive Halving*

CASH – *Combined Algorithm Selection and Hyperparameter optimization*

CL – *Curriculum Learning*

GMM – *Gaussian Mixture Models*

HB – *Hyperband*

IH – *Instance Hardness*

ML – *Machine Learning*

SH – *Successive Halving*

SMAC – *Sequential Model-based Algorithm Configuration*

SMBO – *Sequential Model-Based Optimization*

SVM – *Support Vector Machine*

kDN – *k-Disagreeing Neighbors*

CONTENTS

Chapter 1

INTRODUCTION

Due to its relevance for modern society, Machine learning (ML) is receiving increasing attention in both industry and academia. Applications of ML range from specific domains to varied problems, such as automatic speech recognition (YU; DENG, 2014), self-driving cars (BADUE et al., 2020), computer vision (SZELISKI, 2010), and natural language processing tasks (HIRSCHBERG; MANNING, 2015). Therefore, ML is employed from applications in smartphones to modern industrial processes.

Figure 1.1 illustrates a prototypical ML pipeline. A regular pipeline for building a ML model comprises five stages:

1. The input data is prepared (data preparation) through transformation and cleaning techniques.
2. Features can be selected or generated from data (feature engineering).
3. One or more algorithms are selected (algorithm selection).
4. The hyperparameters of the selected algorithms are configured (algorithm configuration).
5. The algorithm is trained and the resultant model is evaluated (evaluation).

Defining a ML pipeline and its components is an iterative process. For example, the evaluation stage results may lead to changes in all or specific previous stages of the pipeline to achieve better performance. Therefore, defining a ML pipeline takes substantial human expertise and usually requires several model training to find the right combination of components and hyperparameters for each dataset (YANG et al., 2020). In general, only trained experts with specialized data preprocessing knowledge, domain-driven meaningful feature engineering skills, and ex-

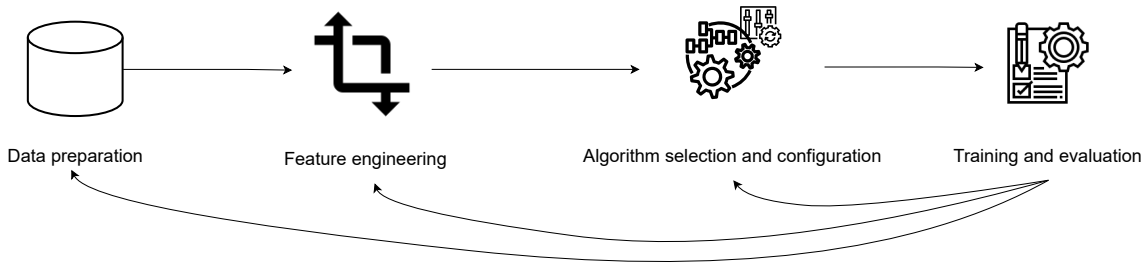

Figure 1.1: Prototypical ML pipeline

pertise in fine-tuned models could build an efficient ML pipeline leading to excellent predictive power.

Given the nonexistence of a single algorithm capable of achieving excellent performance in any application, researchers continuously propose a plethora of new methods or modifications of existing algorithms for specific tasks. With such an increase in algorithms and problem variability, the ML process becomes even more complex, expensive, and dependent on high human expertise and skills.

Motivated by industry needs and ML issues in general, Automated Machine Learning (HUTTER; KOTTHOFF; VANSCHOREN, 2019) (AutoML) has appeared in recent years as a sub-area of ML. The goal of AutoML is to lessen the need for human interference in the ML process and make it increasingly accessible for non-expert users and less costly for the industry. Current AutoML approaches are competitive with humans in some scenarios.

In the automated construction of ML pipelines, the choice between different preprocessing and ML algorithms is modeled as a categorical hyperparameter, a problem known as Combined Algorithm Selection and Hyperparameter Optimization (CASH). Most AutoML approaches rely on robust hyperparameter optimization and algorithm selection techniques to search through a complex, conditional, and high dimensional search space (HUTTER; KOTTHOFF; VANSCHOREN, 2019), making it costly for AutoML to solve.

According to Parmentier et al. (2019), one of the reasons why AutoML is so time-consuming is due to the time taken by training the ML algorithm. Training ML algorithms on small datasets

takes at least a few seconds and up to days for bigger datasets, which is increasingly common with big data.

Inspired by the way that humans learn, Curriculum Learning (CL) adapts the concept of curriculum [1] to ML algorithms. In human education, teachers rely heavily on a highly organized curriculum, starting from easy concepts and gradually presenting more complex ones. CL proposes to train ML algorithms by presenting first a small subset of easier data and gradually increasing both the subset size and the difficulty of data within it (ELMAN, 1993; BENGIO et al., 2009). Previous studies show that CL can improve model performance on target tasks and accelerate the training process (WANG; CHEN; ZHU, 2020). Despite being counter-intuitive, anti-curriculum learning (AntiCL), that is, training a ML model from the hardest to the easiest data, can be as good or better than CL in some scenarios (KOCMI; BOJAR, 2017; ZHANG et al., 2019).

Hence, in this work, we propose and compare different CL strategies applied to CASH solvers of an AutoML system to conduct a more efficient search space exploration and find good performing machine-learned models with efficiency.

Unlike previous work by Guo et al. (2020), which utilizes CL strategies to speed up neural architecture search (NAS), our focus is on AutoML for "traditional" ML. That is, ML pipelines that do not include deep neural networks.

In general, the results show that our approach shows improvements over the baseline, Auto-sklearn, on most of the selected datasets.

1.1 Hypothesis and objectives

This research hypothesizes that the application of CL strategies on CASH solvers of an AutoML system can improve the search space exploration, and consequently find better performing machine-learned models with efficiency.

The main objective of this research is to apply and evaluate CL strategies on a CASH solver for an AutoML system. The general approach for applying CL consists of two main components: (i) a difficulty measurer for estimating the difficulty of each example in the training set, and (ii) a training scheduler that decides the pace and the sequence of data subsets to be presented through the training process (WANG; CHEN; ZHU, 2020).

[1] Taba (1962) defines a curriculum as a plan for learning. In ML, the term generally refers to adequately ordering information by difficulty.

As for difficulty measurers, we used three instance hardness (SMITH; MARTINEZ; GIRAUD-CARRIER, 2014) methods, which estimate instances that are hard to classify correctly. We adapt the Successive Halving and Hyperband optimization techniques present in the Auto-Sklearn system (FEURER; HUTTER, 2018) to sample data subsets based on a curriculum for the training scheduler. Finally, we compare the performance of our Curriculum Learning-based AutoML system, which we dub CurL-AutoML, with the original Auto-sklearn system, on twelve datasets built for classification tasks, utilizing the AutoML Benchmark framework presented by Gijsbers et al. (2019).

1.2 Contributions

The main contributions of this work are:

- The CurL-AutoML, a new approach for incorporating Curriculum Learning in an AutoML tool.
- A comparison of CL and AntiCL strategies in the context of Machine Learning Automation.
- A comparison of distance-based (kDN) and probabilistic (GMM) difficulty measurers for curriculum design.

Chapter 2

THEORETICAL FOUNDATION

In this chapter, we present the theoretical background involving this research. Specifically, we present techniques to automate parts of the ML pipeline and the basic concepts of Curriculum Learn.

2.1 Automated Machine Learning

AutoML is the process of automating any step of the ML pipeline. It emerged from reducing the necessary expertise to apply machine learning, making it more accessible for non-experts. AutoML can be seen as an effort to democratize ML. It makes state-of-the-art ML approaches accessible to domain scientists interested in applying them but who have superficial knowledge about the technologies behind it (HUTTER; KOTTHOFF; VANSCHOREN, 2019). AutoML techniques are usually found in three steps of the traditional ML pipeline: data preparation, feature engineering, and algorithm selection and configuration.

2.1.1 Data preparation

Data preparation is one of the most important steps in the ML pipeline. However, most of the approaches to automate it in the current AutoML tools are simple and not very efficient. The main solutions only include general purpose techniques such as missing data imputation and data normalization, usually hard-coded and not generated based on any metric during optimization (ZÖLLER; HUBER, 2019).

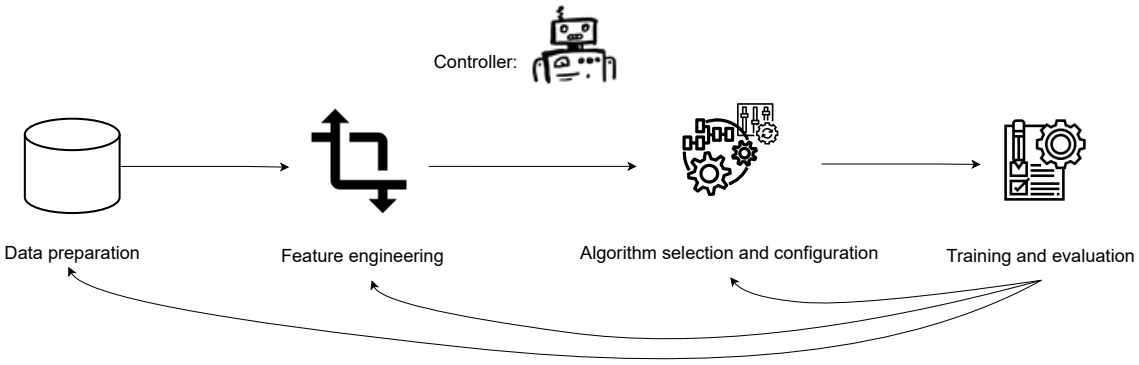

Figure 2.1: Prototypical AutoML pipeline.

2.1.2 Feature engineering

The automation of feature engineering encompasses feature selection and feature generation, two distinct processes that can be employed together in AutoML. Feature selection consists of finding a subset of the original feature set, which is the smallest and most representative considering the original data set. The probability distribution of the reduced data is as close to the original one as possible. Since the number of features that can be acquired through this process is unlimited and one of the goals of AutoML is to automatically solve the problem without considering any previous knowledge about the domain, feature generation is the hardest to automate (ZÖLLER; HUBER, 2019). Based on the initial data set, operations such as discretization, normalization, or simple combinations of features (such as the sum of continuous attributes), can be applied to generate new features.

2.1.3 Algorithm selection and configuration

The algorithm selection and configuration is arguably the most investigated ML pipeline's steps in AutoML. It is easy to understand its motivations, considering that different ML algorithms have numerous potential configurations, each one with its hyperparameters to be tuned. The choice of which one to use becomes a challenging problem, mainly for non-expert users.

To understand how optimization methods are applied in AutoML tools, the main techniques proposed for this task are described in the following sections.

2.1.3.1 Simple search

The simple search refers to a group of simple techniques that do not consider any assumptions about the search space. Therefore, each setting in the search space is evaluated individually. Figure 2.2 illustrates the two most commonly used Simple Search techniques, Grid Search and Random Search.

- *Grid Search*: One of the most simple and widely used HPO techniques. In Grid Search, the user needs to define a finite search scope for each hyperparameter in the Euclidean space (LECUN et al., 1998). Then, the algorithm performs a simple sequential search in space. However, it suffers from low scalability since the size of the search space highly influences grid Search.
- *Random Search*: An alternative to Grid Search is Random Search (ANDERSON, 1953). As the name suggests, hyperparameters are selected randomly until a stop criterion is met, for example, search time. The main advantage of this technique is that it helps to avoid local minimums (BERGSTRA; BENGIO, 2012)

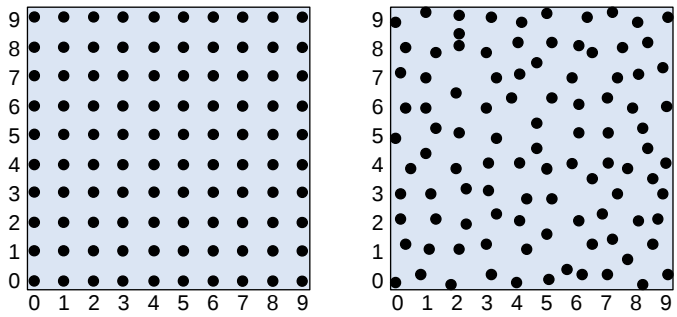

Figure 2.2: Examples of the search space explored by Grid Search (left) and Random Search (right).

2.1.3.2 Sequential Model-based Algorithm Configuration

The Sequential Model-based Algorithm Configuration (SMAC) (HUTTER; HOOS; LEYTON-BROWN, 2011) algorithm is a Sequential Model-Based Optimization (SMBO) (HUTTER; HOOS; LEYTON-BROWN, 2011) framework, that is, an stochastic optimization framework that can explicitly work with both categorical and continuous hyperparameters. It can also exploit conditional spaces such as when a hyperparameter may only be relevant if one or more hyperparameter assumes a certain value (KOTTHOFF et al., 2019).

A common approach to apply SMBO is by Bayesian optimization (BROCHU; CORA; FREITAS, 2010). Bayesian optimization iteratively fits a surrogate model to observations of the

target function. Then, it uses an acquisition function to determine the next candidate points based on the predictive distribution of the surrogate model. With the increase in the number of observations, the surrogate model's predictive distribution improves. Consequently, the algorithm has a better assumption of which regions in the parameter space are worth exploring and which are not. SMAC is a popular tree-based approach for the surrogate model.

2.1.3.3 Successive Halving

Successive Halving (SH) (JAMIESON; TALWALKAR, 2016) is a bandit-based strategy, applied for multi-fidelity algorithm selection that tries to find the best algorithm given a finite set of algorithms and taking into account the low-fidelity approximations of their performance.

Thus, for a given initial budget (e.g number of iterations or data subset), SH queries all available algorithms for that budget removing the worst-performing half, doubling the budget, and iterating this process until only a single algorithm is left.

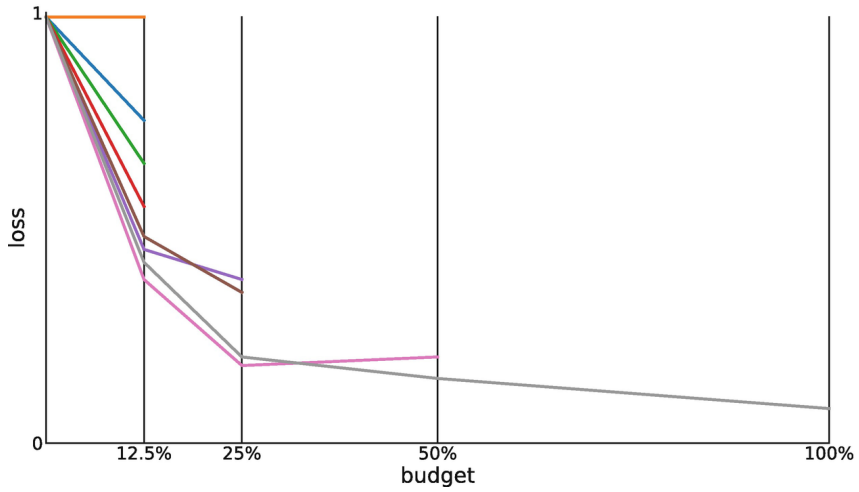

Figure 2.3: Illustration of successive halving for eight algorithms/configurations (HUTTER; KOTTHOFF; VANSCHOREN, 2019).

The primary concern involving SH is the budget-vs-number of configurations trade-off. When assuming a total budget, the user has to decide whether to try many configurations, assign a small budget to each, or try only a few and assign them a larger budget. Small budgets can result in terminating good configurations precipitately, while large budgets can result in a waste of resources by running poor configurations extensively (HUTTER; KOTTHOFF; VANSCHOREN, 2019).

2.1.3.4 HyperBand

The HyperBand (HB) algorithm (LI et al., 2018) is an extension of SH that aims to solve the budget-vs-number of configurations trade-off. For this purpose, HB first divides the total budget into several combinations of the number of configurations and a minimum resource associated with each configuration. Then, it calls SH as a subroutine on each set of random configurations. A higher number of configurations corresponds to a smaller minimum resource, implying a more aggressive early-stopping.

Succinctly, HyperBand removes the need to manually select a number of configurations for a fixed budget, and it can exploit situations in which adaptive allocation works well while protecting itself in situations where more conservative allocations are required (LI et al., 2018)

The success of HyperBand is attributed to its strong anytime performance (yield good configurations with a small budget), robustness, scalability, and flexibility. Yet, it lacks strong final performance, finding the best configurations in a large space, due to being a random search-based method.

2.1.3.5 Bayesian Optimization and HyperBand (BOHB)

A more recent approach denominated BOHB (FALKNER; KLEIN; HUTTER, 2018) combines Bayesian optimization and Hyperband to overcome some of HB's issues. BOHB applies HB to decide how many configurations to evaluate with which budget. However, it replaces the random selection of configurations at the beginning of each HB iteration with a model-based search (FALKNER; KLEIN; HUTTER, 2018).

BOHB intends to achieve the best of both methods: strong anytime performance (using low fidelities in HyperBand) and strong final performance (replacing HyperBand's random search by Bayesian optimization). Moreover, BOHB can use parallel resources effectively and deals with problem domains ranging from a few to many dozen hyperparameters.

2.2 AutoML as a Combined Algorithm Selection and Hyperparameter Optimization problem

The *Combined Algorithm Selection and Hyperparameter Optimization (CASH)* problem, that is, simultaneously selecting an algorithm and tuning its hyperparameters to optimize one or more objectives, was first introduced by Thornton et al. (2013).

Hutter, Kotthoff and Vanschoren (2019) define the CASH problem as follows:

Let $\mathcal{A} = \{A^{(1)}, ..., A^{(R)}\}$ be a set of ML algorithms, and let the hyperparameters of each algorithm $A^{(j)}$ have domain $\Lambda^{(j)}$. Also, let $\mathcal{D}^{(i)}_{train} = \{(x_1, y_1), ..., (x_n, y_n)\}$ be a training set that is split into K cross-validation folds $\{\mathcal{D}^{(1)}_{valid}, ..., \mathcal{D}^{(K)}_{valid}\}$ and $\{\mathcal{D}^{(1)}_{train}, ..., \mathcal{D}^{(K)}_{train}\}$ such that $\mathcal{D}^{(1)}_{train} = \mathcal{D}_{train} \setminus \mathcal{D}^{(i)}_{valid}$ for $i = 1, ..., K$. At last, let $\mathcal{L}(A^{(j)}_{\lambda}, \mathcal{D}^{(i)}_{train}, \mathcal{D}^{(i)}_{valid})$ denote the loss which algorithm $A^(j)$ achieves on $\mathcal{D}^{(i)}_{valid}$ when trained on $\mathcal{D}^{(i)}_{train}$ with hyperparameters λ. Hence, the CASH problem is to find the joint algorithm and hyperparameter setting that minimizes this loss:

$$A^*{}_{\lambda^*} \in \underset{A^{(j)} \in \mathcal{A}, \lambda \in \Lambda^{(j)}}{argmin} \frac{1}{K} \sum_{i=1}^{K} \mathcal{L}(A^{(j)}_{\lambda}, \mathcal{D}^{(i)}_{train}, \mathcal{D}^{(i)}_{valid}) \,. \tag{2.1}$$

Solving CASH for AutoML is time consuming. One way to cope with this problem is through meta-learning, that is, using knowledge previously obtained, such as the performance of learning algorithms across datasets, in order to start the search in a region near the best solution. Also, another alternative is multi-fidelity optimization methods (FERNáNDEZ-GODINO et al., 2019). These methods, such as Successive Halving, speed up optimization by means of low-fidelity approximations of the actual loss function which needs to be minimized. Successive Halving uses adaptive resource allocation and early-stopping, that is, terminating the evaluation of probably bad performing configurations early. Although useful, this early-stopping produces sub-optimal results. This means that a configuration that would yield the best final results may be discarded in favor of others that had a better initial performance on a low-fidelity approximation but ended up yielding worse final results.

In the next section, we describe Auto-sklearn, an AutoML tool that leverages the benefits from both meta-learning and multi-fidelity optimization.

2.3 Auto-sklearn

Based on the python machine learning package Scikit-learn (PEDREGOSA et al., 2011), Auto-sklearn (FEURER; HUTTER, 2018) is a popular AutoML tool that utilizes the SMAC optimizer as the CASH solver for automatically building classification and regression pipelines.

A configuration algorithm, such as SMAC, has a fundamental mechanism called intensification. This intensification mechanism controls how many evaluations to perform with each configuration and decides when a configuration becomes the new current best-known configuration. (HUTTER; HOOS; LEYTON-BROWN, 2011).

One of the capabilities of Auto-sklearn is to use SH as the intensification mechanism for the SMAC optimizer, this combines Bayesian optimization with SH (BOSH), resulting in an adaptation of the BOHB algorithm (FALKNER; KLEIN; HUTTER, 2018). Similar to BOHB, BOSH is a combination of Bayesian optimization and multi-fidelity optimization, resulting in a robust and efficient hyperparameter optimization algorithm, but instead of HyperBand it uses Successive Halving.

BOHB original implementation uses a Parzen Estimator as the surrogate model for Bayesian optimization, while the Auto-sklearn implementation of BOHB uses a Random Forest as the surrogate model.

Auto-sklearn[1] comprises 16 classifiers, 12 regressors, 15 feature preprocessing techniques, and 12 data preparation methods of the scikit-learn library. By default, Auto-sklearn uses meta-learning to warm-start[2] Bayesian optimization, automatically taking into account past performance on similar data sets to suggest initial pipeline configurations. This meta-learning is fueled by an extensive evaluation of different pipelines on 204 distinct data sets.

Auto-sklearn pipeline begins with an hardcoded data preparation step which includes methods such as imputation and categorical encoding. Subsequently, there is a feature engineering step. Next, an ML algorithm is selected and has its hyperparameters optimized by the Sequential Model-based Algorithm Configuration (SMAC) (HUTTER; HOOS; LEYTON-BROWN, 2011) algorithm.

Furthermore, Auto-sklearn applies ensemble learning to make the final result more robust against overfitting. During the optimization process, the tool stores the models evaluated in the search space. Then, it applies ensemble selection (CARUANA et al., 2004), a model post-processing method that starts from an empty ensemble and, following a greedy approach, iteratively adds the model that maximizes ensemble validation performance, constructing an ensemble of the previously trained models.

2.4 Curriculum Learning

Curriculum Learning (CL) consists of training models from the easiest to the hardest data. Its benefits may include: faster model training convergence and better generalization capacity. In contrast, anti-curriculum learning (AntiCL), consists in training a ML model from the hardest to the easiest data.

[1] In this paper, we consider the version 0.12.1 of Auto-sklearn.

[2] Start the optimization algorithm in a region of the search space which is potentially close to the optimum.

For applying a CL strategy, we need to define two components: (i) the difficulty measurer, to generate our curriculum by estimating the difficulty of each training example; and (ii) the training scheduler, to control the pace and sequence of data presented to a model during training.

Wang, Chen and Zhu (2020) categorize CL frameworks considering these two components. A predefined CL has both the difficulty measurer and the training scheduler totally designed by human prior knowledge with no data-driven models or algorithms involved. In contrast, an automatic CL has at least one of the components involving an algorithm.

For a predefined CL, there are different approaches to a difficulty measurer, such as sentence length in NLP tasks (PLATANIOS et al., 2019; SPITKOVSKY; ALSHAWI; JURAFSKY, 2010; TAY et al., 2019), or the number of objects in images in the task of semantic segmentation (Wei et al., 2017). A predefined training scheduler can be discrete or continuous, and is usually task agnostic (WANG; CHEN; ZHU, 2020). A discrete training scheduler adjusts the training data after every fixed number of epochs or convergence on the current data subset. In contrast, continuous schedulers adjust the training data subset at every epoch.

Wang, Chen and Zhu (2020) summarize four of the major strategies for automatic CL:

- Transfer Teacher: A pre-trained model acts as the teacher and measures the difficulty of training examples according to its performance on them, which is later used for training a student algorithm.
- Self-Paced Learning (SPL): The student algorithm also acts as a teacher, measuring the difficulty of training examples according to its losses on them.
- Reinforcement Learning Teacher: A reinforcement learning (RL) model act as the teacher to perform dynamic data selection according to the feedback from the student.
- Other Automatic CL: Methods that include various automatic CL strategies except the above.

Chapter 3

RELATED WORK

In this section, some of the related works available in the literature are briefly described.

Several AutoML systems were developed, such as Auto-Weka (THORNTON et al., 2013), TPOT (OLSON et al., 2016), Hyperopt-sklearn (KOMER; BERGSTRA; ELIASMITH, 2014), ATM (Swearingen et al., 2017), and Auto-Stacker (CHEN et al., 2018), and Recipe (SÁ et al., 2017). However, the AutoML problem is still time-consuming. Therefore, different approaches need to be proposed to efficiently tackle this complex search space and find good performing ML pipelines. Some previous approaches aim to speed up AutoML through methods such as meta-learning and multi-fidelity optimization (FEURER et al., 2015; Parmentier et al., 2019).

Auto-sklearn achieved a boost in search for the best ML pipeline by warm-starting the Bayesian optimization procedure via meta-learning (FEURER et al., 2015). Recently, its authors introduced a new version of the system, dubbed Auto-sklearn 2.0 (FEURER et al., 2020). The previous version used meta-features to select a set of previously-seen datasets similar to the new dataset to be tackled and then started with configurations found to perform well. Auto-sklearn 2.0 utilizes a portfolio of machine learning pipelines that cover as many diverse datasets as possible and minimizes the risk of failure when facing a new task.

TPOT-SH (Parmentier et al., 2019) presents a different version of TPOT, an AutoML system based on Evolutionary Algorithms (EA). EAs are typically slow to converge, which makes TPOT incapable of scaling to large datasets. Therefore, the authors introduced TPOT-SH inspired by the concept of Successive Halving used in Multi-Armed Bandit problems (LATTIMORE; SZEPESVÁRI, 2020). This solution allows TPOT to explore the search space faster and have much better performance on larger datasets. Similarly, Curriculum Learning may help AutoML explore a search space by speeding-up faster the training process of candidate algorithms configurations.

Curriculum Learning was first proposed by Bengio et al. (2009), where the authors conducted an empirical investigation on the use and effects of CL for supervised visual and language tasks. The difficulty of examples was determined by the loss value[1] of a pre-trained model. Similarly, our work approaches the estimation of the difficulty of each example relying on pre-trained models.

Since its proposal, CL has been investigated on a plenty of ML applications, such as reinforcement learning (FLORENSA et al., 2017; NARVEKAR et al., 2020), natural language processing (PLATANIOS et al., 2019; XU et al., 2020), healthcare prediction (EL-BOURI et al., 2020), computer vision (GUO et al., 2018; PENTINA; SHARMANSKA; LAMPERT, 2015; Wang et al., 2019).

However, to the best of our knowledge, Guo et al. (2020) is the only previous work that investigates the use of CL on AutoML domains, more specifically NAS. Neural architecture search methods aim to search for an optimal architecture in a predefined search space that is often extremely large. Due to the limitation of computational resources, searching in the entire space of architectures is unfeasible. The affordable approach is to sample a very small proportion of the architectures from the search space. It causes NAS models to become hard to train, and they often find sub-optimal architectures. Trying to attack this problem, Guo et al. (2020) proposed a curriculum search method that starts from a small search space and gradually incorporates the learned knowledge to guide the search in a large space. Extensive experiments on CIFAR-10 and ImageNet demonstrated significant improvements in the search efficiency and architecture performance over previous NAS methods.

In contrast to the previously described work, in this work, we focus on applying CL strategies on AutoML for ML pipelines that do not include deep neural networks. The previous empirical investigations on CL have shown improvements in multiple ML domains. Multi-fidelity optimization and meta-learning have been helpful to speed up AutoML procedures. In our work, we investigated CL strategies on AutoML tasks while still leveraging the benefits of multi-fidelity optimization and meta-learning by adapting the SH procedure utilized by the CASH solvers implemented in Auto-sklearn.

[1] The loss value indicates how bad the model's prediction was on a single example. A perfect prediction has a loss value of zero, while worse predictions will have greater loss values.

Chapter 4

METHODOLOGY

Since a predefined CL requires human intervention and it is opposite to the primary goal of AutoML, we chose to follow the automatic CL strategy. More specifically, we adopt a Transfer Teacher strategy, following the general principle illustrated in Figure 4.1.

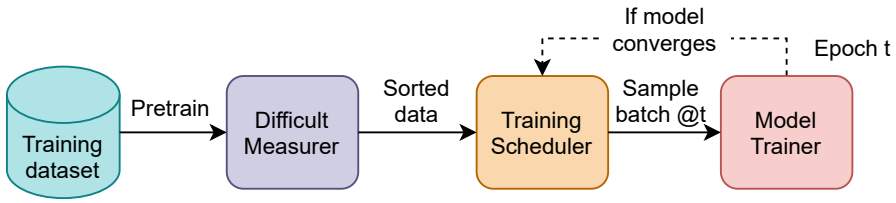

Figure 4.1: General Transfer Teacher curriculum design.

As automatic difficulty measurers, we utilized instance hardness (IH) methods to assign a score of difficulty to each data example. Hardness ordering, that is, the use of IH methods to order examples by difficulty for supervised classification problems, was first introduced by (SMITH; MARTINEZ, 2016). A hardness ordering uses instance hardness to order the instances in a data set based on their likelihood of being misclassified.

We use the k-Disagreeing Neighbors and an ensemble-based approach to estimate IH and generate a curriculum by hardness ordering, and also present a probabilistic-based method for doing so:

- **k-Disagreeing Neighbors (kDN)**: kDN measures the local overlap between an instance and its nearest neighbors. In this method, an instance is considered hard to classify when there is an overlap of different classes in its region of competence. The kDN of an instance is the percentage of the k nearest neighbors (using Euclidean distance) of that instance with a different target class value (SMITH; MARTINEZ; GIRAUD-CARRIER, 2014).

- **Gaussian Mixture Model (GMM)**: In this work, we investigated the use of GMM as an estimator of IH. A GMM is a parametric function of probability density represented as a weighted sum of the densities of Gaussian components (REYNOLDS, 2009). The central idea of these probabilistic models is that all data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. Our approach consists of fitting a GMM to a dataset, then, for every instance, we extract the probabilities of belonging to one the N Gaussian distributions, where N is the number of target classes. Thus, the IH will be defined according to the probability of an instance not being generated by a Gaussian distribution. Considering that the number of Gaussians is defined according to the number of target classes, the instances that are in the decision boundaries between Gaussians will be deemed to have a greater degree of difficulty than those closer to a Gaussian's centroid.

- **An ensemble of algorithms**: We considered an ensemble of different learning algorithms for estimating IH. The ensemble is composed of a Support Vector Machine (SVM) (VAPNIK, 2013), a Random Forest (RF) (BREIMAN, 2001) and a Multi-Layer Perceptron (HORNIK; STINCHCOMBE; WHITE, 1989). In our approach, we fit the ensemble to a dataset and extract prediction scores for every instance. We assume the scores to be inversely proportional to IH, that is, instances with higher scores are easily assigned to the correct target class, therefore they have a lower IH value. Instances with lower prediction scores will be harder to classify and thus have a higher IH value.

After generating a curriculum by hardness ordering, it is necessary only a training scheduler to apply CL. In Auto-sklearn, when using the BOSH (or BOHB) optimizer, subsets of training data can be taken as a budget. By default, these subsets are randomly sampled from the training data. We adapted this sampling strategy to select subsamples of data according to a curriculum strategy, such as CL or AntiCL. Therefore, in our approach, CurL-AutoML, the SH procedure used in both BOSH and BOHB algorithms acts as a predefined training scheduler for Auto-sklearn.

For the CL approach, the relatively easiest examples are sampled first, then SH evaluates algorithms configurations on it, discarding the worst-performing half. In the next iterations, SH will progressively sample harder data and repeat the process, until the budget reaches its limit and one configuration is left. In the AntiCL approach, in turn, the process starts form the hardest examples and goes on decreasing the difficult of the instances until the end. In contrast to the general transfer teacher curriculum design in Figure 4.1, our training scheduler does not

samples at each training epoch, but at each SH iteration. Moreover, it does not only involves the training of a single algorithm, but multiple algorithm configurations.

Figure 4.2 illustrates the design of our proposal. First the selected dataset is utilized to pre-train a difficulty measurer model, which then is used to assign a difficulty score to each instance in the dataset. Next, we select one curriculum learning strategy, which could be the traditional curriculum (easy instances to hard instances) or the anti-curriculum (hard instances to easy instances). Then, we choose between BOSH or BOHB as the CASH solver to be utilized and start the AutoML optimization process. During optimization, the Successive Halving procedure used in both CASH solvers acts as the training scheduler using the previously sorted data to train and evaluate multiple algorithms configurations. Finally, the best ML pipeline constructed during optimization is outputted.

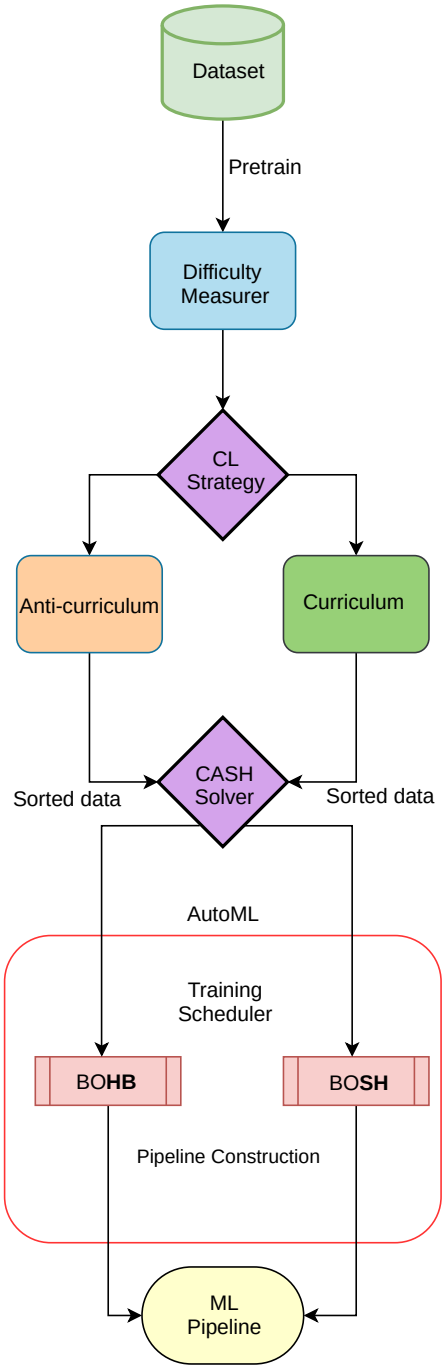

Figure 4.2: Curriculum learning-based AutoML design.

Chapter 5

EXPERIMENTAL EVALUATION

In this chapter, we describe our experiments with the proposed CurL-AutoML. Moreover, we present and discuss the results obtained in our evaluation.

All experiments were performed on a computer with 12 cores Intel Xeon E3 - 12xx v2 (Ivy Bridge), which operated at 2.4 GHz, and operational memory of 64 GB. Code for all experiments is in the GitHub repository at `https://github.com/odahviing-dov/CurL-AutoML`.

5.1 Experimental Setup

In total, we selected twelve datasets for experimentation from the Open ML repository. Table 5.1 presents the datasets. MiniBooNE, Jasmine, Amazon, Australian, Blood transfusion, and Christine, are datasets that represent binary classification problems, while Covertype, Helena, Fashion-MNIST, CNAE-9, Connect-4, and Dilbert, represent multi-class classification problems.

First, we generated two different curriculums for each dataset, one by applying kDN and another via GMM. We noted that the proposed difficulty measurer via an ensemble of algorithms was taking over 12 hours long to produce a result during experimentation. Since our goal in this research is to achieve better time and performance efficiency for AutoML, we consider that taking hours to generate a curriculum is an unfeasible trade-off. Hence, we decided to not proceed with using the proposed ensemble of algorithms as a difficulty measurer.

The execution of the experiments was carried out through the AutoML benchmark framework introduced by (GIJSBERS et al., 2019). We used this framework to run and evaluate the performance of our proposed CurL-AutoML, using the BOSH and the BOHB optimizer (with CL and AntiCL strategies) on the selected datasets. We performed the same experiments for

Table 5.1: Selected OpenML datasets.

Dataset	Instances	Features	Classes	Class balance	OpenML ID
Covertype	581,012	55	7	Imbalanced	1596
Helena	65,196	28	100	Imbalanced	41169
MiniBooNE	130,064	51	2	Imbalanced	41150
Blood transfusion	748	5	2	Imbalanced	1464
Amazon	32,769	10	2	Imbalanced	4135
Connect-4	67,557	43	3	Imbalanced	40668
Australian	690	15	2	Balanced	40981
Fashion-MNIST	70,000	785	10	Balanced	40996
Jasmine	2,984	145	2	Balanced	41143
Christine	5418	1637	2	Balanced	41142
CNAE-9	1080	857	9	Balanced	1468
Dilbert	10,000	2001	5	Balanced	41163

Auto-sklearn and compared the results with those achieved by CurL-AutoML. The benchmarking framework supports a broad range of measures. In this work, we chose log loss for both binary classification and multi-class classification problems. The measures are estimated with five-fold cross-validation.

Since CurL-AutoML is an adaption of Auto-sklearn, both AutoML tools were used with the same search spaces. We fixed the number of 12 processing units, 64GB memory, and a run time of 1 hour per cross-validation fold as hyperparameters that specified available resources.

5.2 Results and Discussion

In this section, we describe the results we obtained in our experimentation. All curriculum approaches were executed by our proposed CurL-AutoML using both BOSH and BOHB optimizers, while the standard Auto-sklearn carried out the "No curriculum" approach.

The total time for generating a curriculum via kDN and GMM for each dataset is presented in Table 5.2. In general, both kDN and GMM difficulty measurers could generate a curriculum in a feasible time, but kDN had the lowest average time.

Table 5.3 illustrates a subsample of the generated curriculum for the MiniBooNE dataset. The index column corresponds to the index of each instance in the dataset default order. The score column indicates the difficulty assigned to an instance, and it ranges from zero (easier instances) to one (harder instances). In this example, the easiest instance is on index seven (0.02), and the hardest one is on index four (0.74). By keeping the index and the score of each

Table 5.2: Time in seconds for generating a curriculum via kDN or GMM, for each dataset.

Dataset	Difficulty measurer	Time in seconds
Covertype	kDN	400
	GMM	129
Helena	kDN	4
	GMM	504
MiniBooNE	kDN	20
	GMM	110
Jasmine	kDN	0.20
	GMM	0.26
Amazon	kDN	1.2
	GMM	0.56
Australian	kDN	0.04
	GMM	0.04
Fashion-MNIST	kDN	49
	GMM	1308
Blood transfusion	kDN	0.01
	GMM	0.07
Christine	kDN	1.7
	GMM	8.9
CNAE-9	kDN	0.06
	GMM	4
Connect-4	kDN	5.9
	GMM	4.9
Dilbert	kDN	3
	GMM	60

instance in the dataset, we can select instances for model training following an order: easiest to hardest for typical CL, or the inverse for AntiCL.

Table 5.3: Sample of the generated curriculum for the MiniBooNE dataset.

Index	Score
0	0.06
1	0.12
2	0.32
3	0.18
4	0.74
5	0.4
6	0.14
7	0.02
8	0.32
9	0.12

The results for the Covertype dataset using the BOSH optimizer for all curriculum strategies (CurL-AutoML) and No curriculum (Auto-sklearn) can be found in Table 5.4. All results were estimated via log loss using five-fold cross-validation.

Table 5.4: Log loss scores for the multi-class classification task on the Covertype dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	0.30850	0.20547	0.27961	0.26476	0.22762
AntiCL-kDN	0.26832	0.21157	**0.20431**	**0.15877**	0.22368
AntiCL-GMM	0.27007	0.21157	0.27268	0.19952	0.29724
CL-kDN	**0.19271**	**0.19166**	0.28512	0.26678	0.27009
CL-GMM	0.29360	0.21157	0.27670	0.26678	**0.17950**

For this multi-class classification dataset, curriculum strategies achieved better results than No curriculum on all folds. CL was the best in folds one, two, and five. AntiCL has the advantage in folds three and four. Curriculums generated via kDN outperformed those by GMM, yielding better scores in four of the five folds. It demonstrated an improvement when applying curriculum-based strategies. The average scores for each approach on all five folds are illustrated in Figure 5.1.

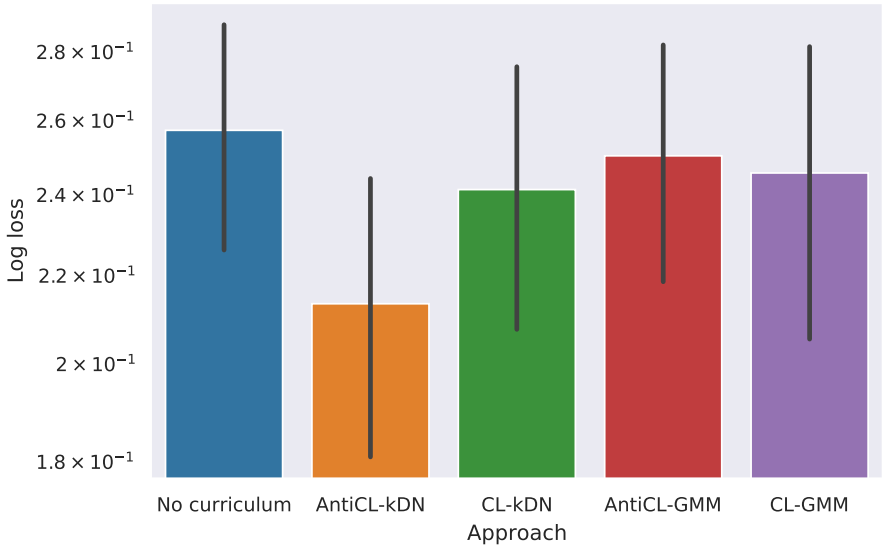

Figure 5.1: Average (and standard deviation) of log loss scores for the Covertype dataset using BOSH.

The AntiCL with kDN as the difficulty measurer achieved the best average score (0.2133). Table 5.5 presents the number of final models that each approach obtained for each cross-validation fold. In average, all curriculum approaches obtained a model count similar to a No curriculum approach, as illustrated in Figure 5.2.

Table 5.5: Number of models for the multi-class classification task on the Covertype dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	3	2	3	3	3
AntiCL-kDN	2	1	3	2	3
AntiCL-GMM	3	1	3	3	2
CL-kDN	4	2	2	2	2
CL-GMM	2	1	3	2	4

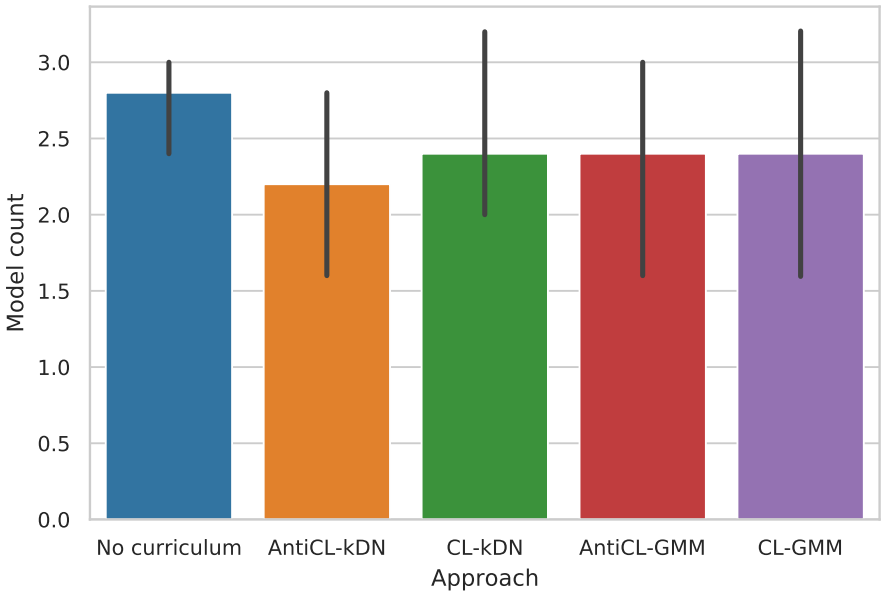

Figure 5.2: Average (and standard deviation) number of models for the Covertype dataset using BOSH.

Table 5.6: Log loss scores for the multi-class classification task on the Covertype dataset for each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	0.3085	**0.2115**	0.2296	0.2203	0.2156
AntiCL-kDN	0.3085	0.2115	0.2464	**0.1010**	0.3124
AntiCL-GMM	0.2657	0.2777	0.2448	0.2667	**0.1811**
CL-kDN	0.3058	0.2115	**0.1486**	0.2667	0.2943
CL-GMM	**0.2413**	0.2115	0.2011	0.2461	0.2292

The results for the Covertype dataset using the BOHB optimizer for all curriculum strategies can be found in Table 5.6. In this case, the use of curriculum strategies exhibited improvements over No curriculum in four out of five folds. CL-GMM achieved the best score in fold 1. All approaches had similar scores in fold 2, with exception to AntiCL-GMM, which had the lowest score in this case. CL-kDN achieved the best score in fold 3, followed by CL-GMM. In fold 4, AntiCL-kDN achieved the best score across folds (0.1010). AntiCL-GMM was the best

approach in fold 5. Again, the results demonstrated an improvement when applying curriculum-based strategies, specially in fold 4, however the best results still vary between these methods. The average score using BOHB for each approach on all five folds are illustrated in Figure 5.3.

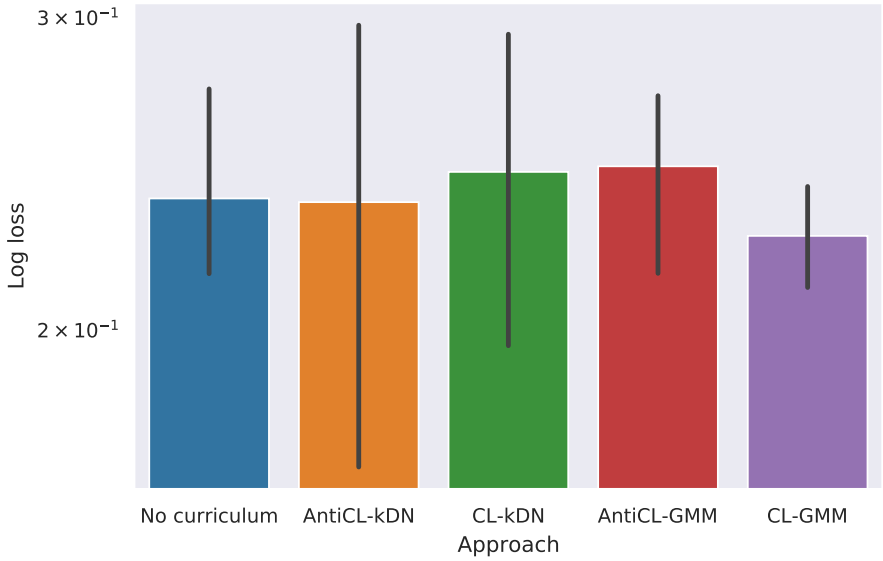

Figure 5.3: Average (and standard deviation) of log loss scores for the Covertype dataset using BOHB.

The CL with GMM as a difficulty measurer has the best average score (0.2260). Table 5.7 presents the number of final models that each approach obtained for each cross-validation fold. In average, all curriculum approaches obtained a model count similar to a No curriculum approach, as illustrated in Figure 5.4.

Table 5.7: Number of models for the multi-class classification task on the Covertype dataset for each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	2	1	2	3	2
AntiCL-kDN	2	1	3	2	1
AntiCL-GMM	3	2	3	2	1
CL-kDN	3	1	3	2	2
CL-GMM	3	1	3	2	3

Table 5.8 displays the results for the Helena dataset with the BOSH optimizer. Mostly, curriculum strategies achieved better scores by slight differences. Again, curriculums generated via kDN outperformed those by generated GMM. In this case, the No curriculum approach had better performance in one of the folds. Figure 5.5 presents the average log loss score for each approach. Overall, the CL-kDN approach achieved the best average score (2.5532).

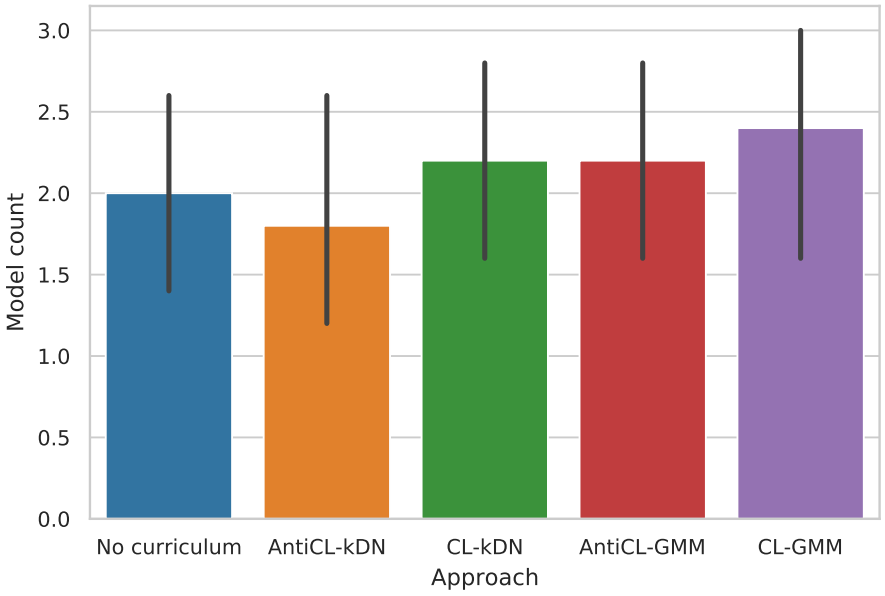

Figure 5.4: Average (and standard deviation) number of models for the Covertype dataset using BOHB.

Table 5.8: Log loss scores for the multi-class classification task on the Helena dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	2.5618	**2.5504**	2.5847	2.5506	2.5469
AntiCL-kDN	2.5615	2.5541	2.5852	**2.5453**	2.5350
AntiCL-GMM	2.5627	2.5563	2.5886	2.5506	2.5440
CL-kDN	**2.5496**	2.5585	**2.5731**	2.5514	**2.5335**
CL-GMM	2.5622	2.5621	2.5881	2.5455	2.5347

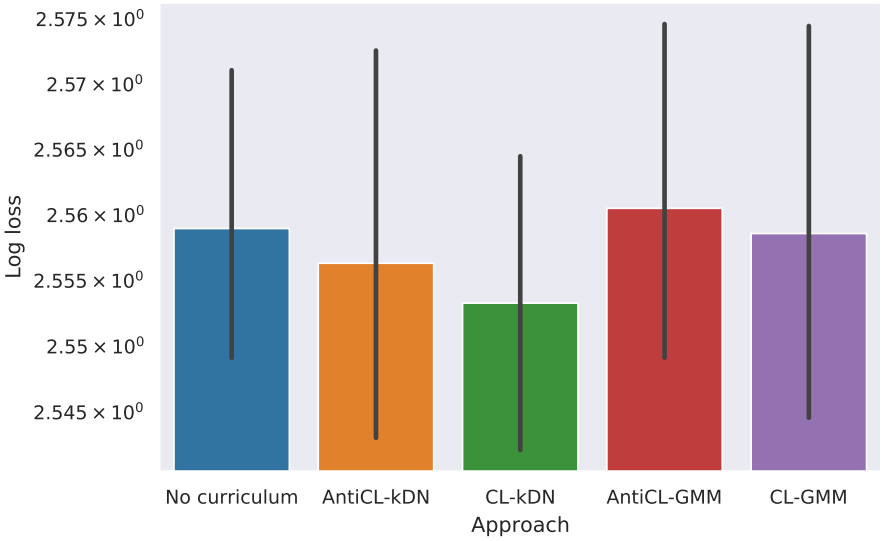

Figure 5.5: Average (and standard deviation) of log loss scores for the Helena dataset using BOSH.

The model count obtained on each cross-validation fold using BOSH is presented in Table 5.9. All approaches have similar model counts, although Figure 5.6 illustrates that CL-GMM had less model count variability among folds.

Table 5.9: Model count for the Helena dataset on each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	7	7	6	5	5
AntiCL-kDN	7	7	6	7	6
AntiCL-GMM	5	6	6	5	6
CL-kDN	6	8	7	6	6
CL-GMM	6	6	6	6	6

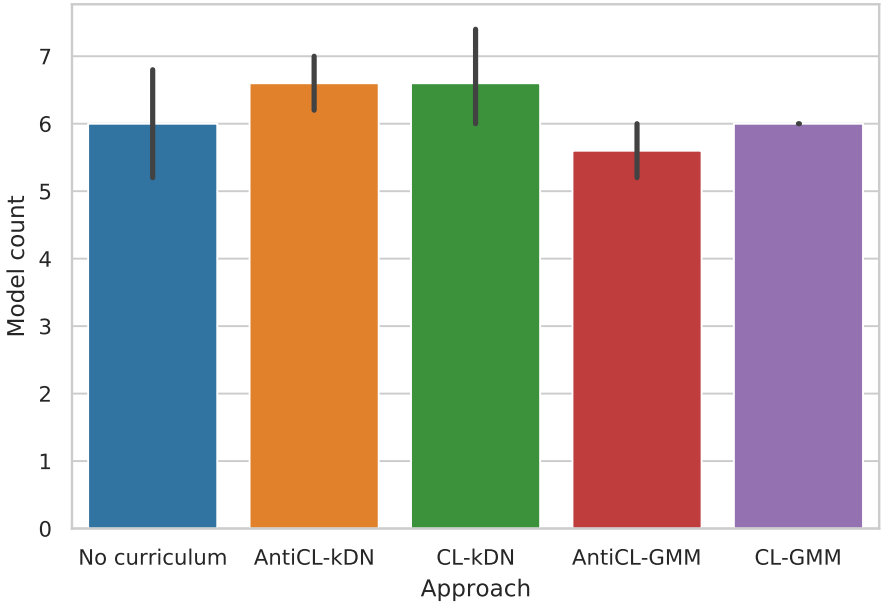

Figure 5.6: Average (and standard deviation) number of models for the Helena dataset using BOSH.

Table 5.10: Log loss scores for the multi-class classification task on the Helena dataset for each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	2.5666	2.5590	2.5879	2.5506	2.5460
AntiCL-kDN	2.5614	2.5594	2.5887	2.5507	2.5365
AntiCL-GMM	2.5572	**2.5516**	**2.5725**	2.5509	**2.5377**
CL-kDN	**2.5560**	2.5586	2.5887	**2.5497**	2.5433
CL-GMM	2.5627	2.5563	2.5887	2.5624	2.5460

The results for the Helena dataset using the BOHB optimizer are presented in Table 5.10. In this case, curriculum strategies also exhibited minor improvements on all folds. The CL and AntiCL approaches using curriculums generated via kDN had the best performances, with

AntiCL-kDN achieving the best scores in folds two, three, and five, while CL-kDN was the best in the remaining folds (one and four). Figure 5.7 presents the average log loss score for each approach. Overall, the AntiCL-kDN approach achieved the best average score (2.5540).

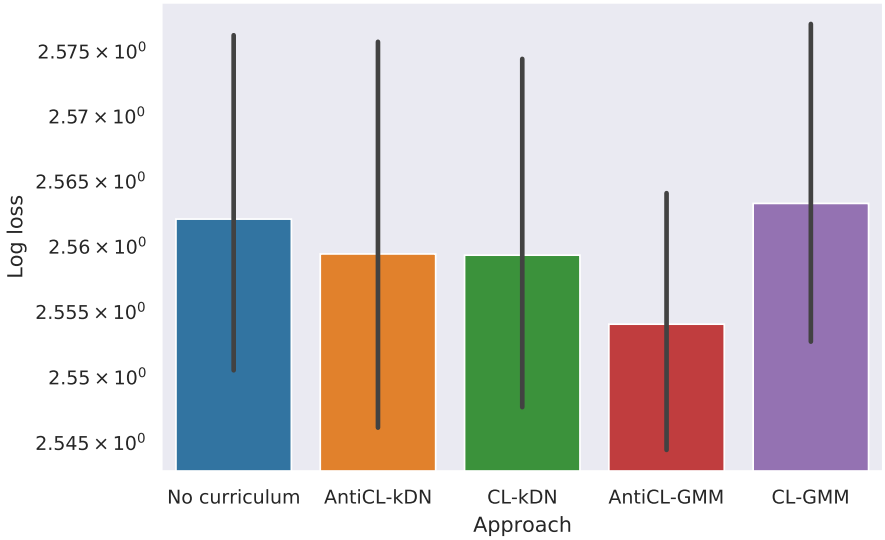

Figure 5.7: Average (and standard deviation) of log loss scores for the Helena dataset using BOHB.

The model count obtained on each cross-validation fold using BOHB is presented in Table 5.11. All approaches have similar model counts, Figure 5.8 shows that AntiCL-GMM had the highest average model count among folds.

Table 5.11: Model count for the Helena dataset on each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	5	6	7	5	5
AntiCL-kDN	7	6	5	6	7
AntiCL-GMM	8	6	7	6	7
CL-kDN	5	7	5	6	6
CL-GMM	5	8	5	4	5

Table 5.12: Log loss scores for the binary classification task on the MiniBooNE dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	0.1405	0.5977	0.2699	**0.1300**	0.1753
AntiCL-kDN	0.1436	0.1388	0.1365	0.1302	0.1473
AntiCL-GMM	0.3341	0.1353	0.2713	0.1324	0.5357
CL-kDN	**0.1399**	**0.1320**	0.1370	0.1323	**0.1269**
CL-GMM	0.1400	0.1340	**0.1340**	0.1340	0.1277

The results for the MiniBooNE dataset using BOSH are presented in Table 5.12. In this binary classification task, the CL approaches outperformed the AntiCL and No curriculum ap-

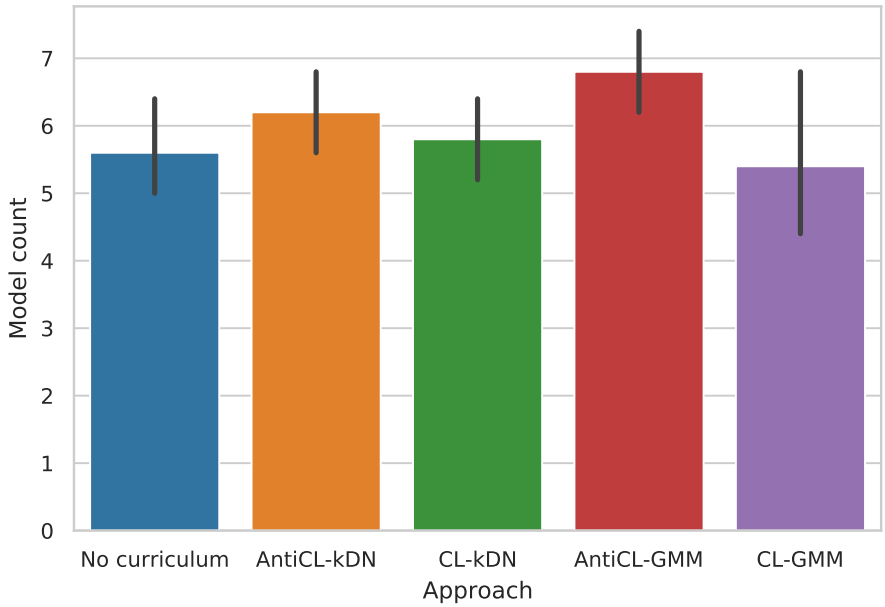

Figure 5.8: Average (and standard deviation) number of models for the Helena dataset using BOHB.

proaches. As seen in Figure 5.9, both CL-kDN and CL-GMM had very similar average scores. However, the CL-kDN approach achieved the best average score (0.1336).

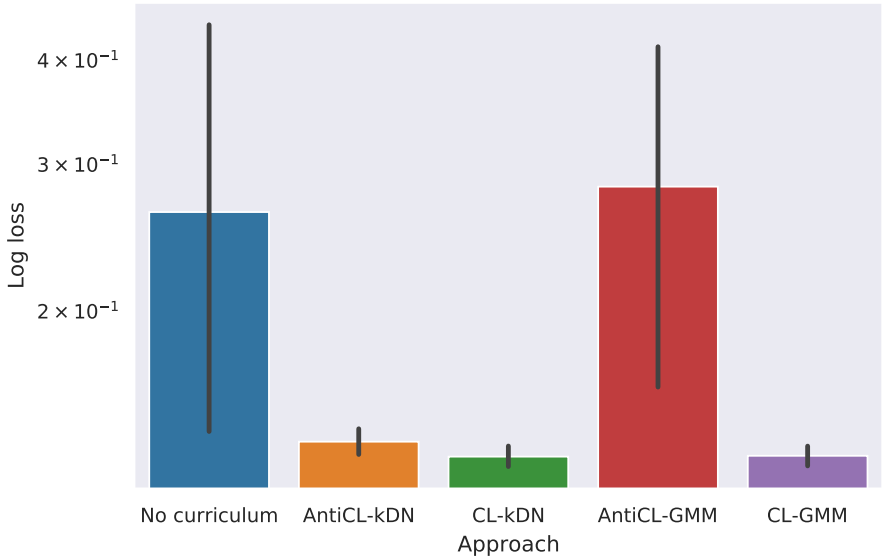

Figure 5.9: Average (and standard deviation) of log loss scores for the MiniBooNE dataset using BOSH.

Table 5.13 shows the model count for the MiniBooNE dataset. The AntiCL-GMM approach on fold 2 obtained the highest model count (16). However, AntiCL-GMM also has the lowest model count across folds (2), in fold 1. Figure 5.10 illustrates the average model count for each approach. The CL-kDN has an average model count of 10 and the lowest variability.

Since a curriculum strategy can enable faster model training, we suppose that the curriculum approaches may lead AutoML to a more efficient exploration of the search space by speeding up the evaluation of configurations. Consequently, we expect to observe two primary outcomes: (i) a higher number of evaluated models; and (ii) a more accurate result.

Table 5.13: Model count for the MiniBooNE dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	9	7	5	9	13
AntiCL-kDN	13	10	14	11	11
AntiCL-GMM	2	16	4	12	7
CL-kDN	10	10	11	9	11
CL-GMM	13	9	13	12	11

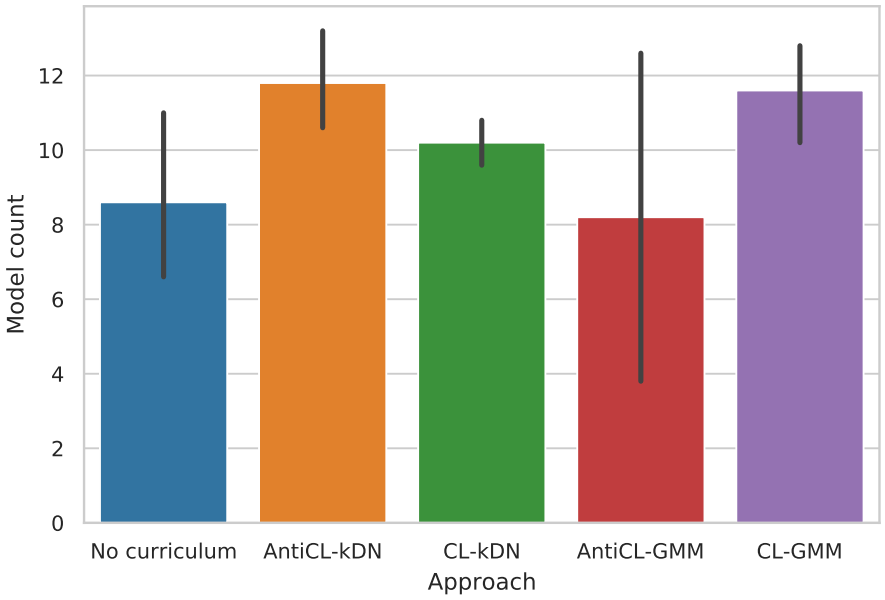

Figure 5.10: Average (and standard deviation) number of models for the MiniBooNE dataset using BOSH.

Table 5.14: Log loss scores for the binary classification task on the MiniBooNE dataset for each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	**0.1454**	0.1376	0.1370	0.1341	0.1296
AntiCL-kDN	0.4836	**0.1359**	**0.1369**	**0.1325**	0.1306
AntiCL-GMM	0.1501	0.1540	0.1435	0.1345	0.1369
CL-kDN	0.6127	0.5161	0.4537	0.1365	0.1284
CL-GMM	0.1457	0.1370	0.1388	0.1397	**0.1242**

Table 5.14 shows the results for the MiniBooNE dataset using the BOHB optimizer. For this dataset, the AntiCL-kDN was the best approach in most folds. It slightly surpassed the

No curriculum baseline in folds two, three, and four. None of the curriculum approaches could surpass the No curriculum baseline approach in fold 1. Notably, the kDN-based approaches achieved significantly worse results in this fold, demonstrating that kDN faced some difficulties in this case. The CL-GMM had the best score across folds in fold five. However, Figure5.11 illustrates that the No curriculum average score was not surpassed in this case.

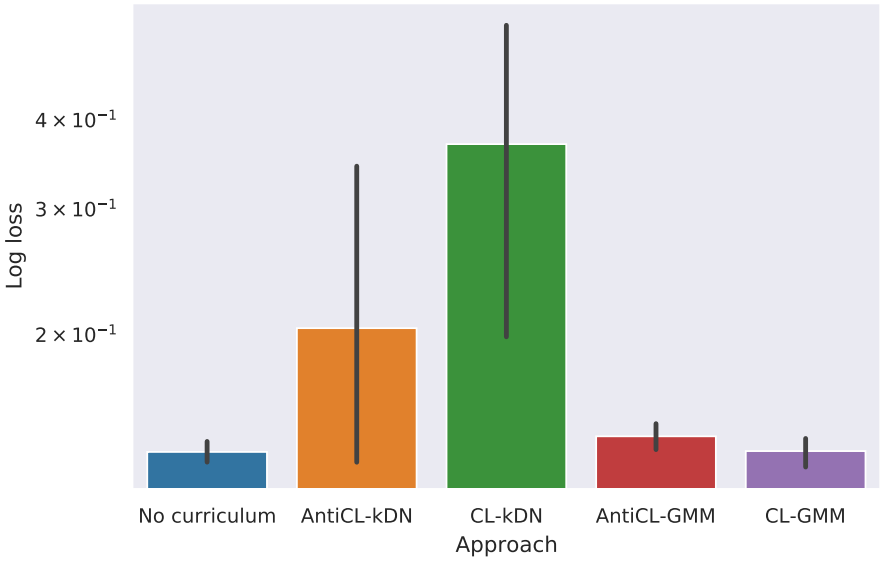

Figure 5.11: Average (and standard deviation) of log loss scores for the MiniBooNE dataset using BOHB.

The average model count achieved by each approach using the BOHB optimizer is displayed in Figure 5.12. The No curriculum, AntiCL-GMM, and CL-GMM had a similar average model count (10), while CL-kDN achieved the lowest average model count, followed by AntiCL-kDN. As illustrated in Figure 5.11 the CL-kDN approach also had the highest average log loss score.

Table 5.15: Log loss scores for the binary classification task on the Jasmine dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	0.3963	0.3982	0.3320	0.4115	0.4632
AntiCL-kDN	0.4070	**0.3790**	**0.3295**	**0.3929**	0.4439
AntiCL-GMM	**0.3913**	0.3827	0.3578	0.4179	0.4324
CL-kDN	0.3973	0.3935	0.3323	0.4145	0.4333
CL-GMM	0.4014	0.3917	0.3497	0.3944	**0.4216**

Finally, Table 5.15 contains the log loss scores for the Jasmine dataset obtained using BOSH.

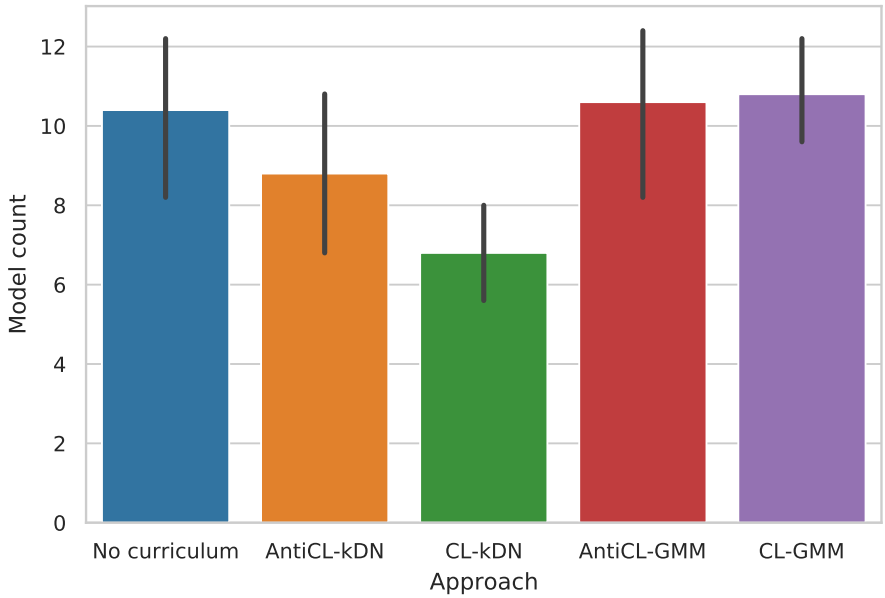

Figure 5.12: Average (and standard deviation) number of models for the MiniBooNE dataset using BOHB.

Table 5.16: Model count for the Jasmine dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	11	10	7	11	12
AntiCL-kDN	6	15	10	8	12
AntiCL-GMM	10	14	8	8	13
CL-kDN	15	11	11	7	11
CL-GMM	15	9	11	7	12

The AntiCL-kDN approach was the best in folds two, three, and four, while CL-GMM scored better in fold 5. The average scores for each approach on all five folds are displayed in Figure 5.13. In this case, the best average score was achieved by the AntiCL-kDN approach (0.390519), followed by both CL approaches. In general, all curriculum approaches had better average scores than a No curriculum approach.

The model count for this dataset is presented in Table 5.16. On average, all approaches obtained the same model count (10), with the exception to CL-kDN (11), as illustrated in Figure 5.14.

Table 5.17 presents the log loss scores for the Jasmine dataset using the BOHB optimizer. In this experiment, the AntiCL-GMM and CL-kDN approaches achieved the best scores in fol 2 and fold 3, respectively. Yet, the No curriculum baseline was not surpassed by any of the curriculum approaches in the remaining three folds. The No curriculum baseline had the best average score, as exhibited in Figure 5.15.

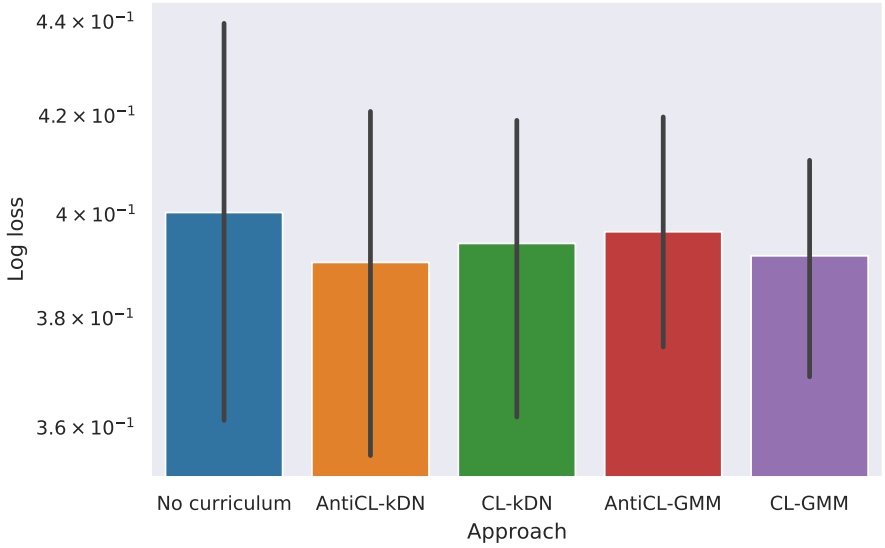

Figure 5.13: Average (and standard deviation) number of log loss scores for the Jasmine dataset using BOSH.

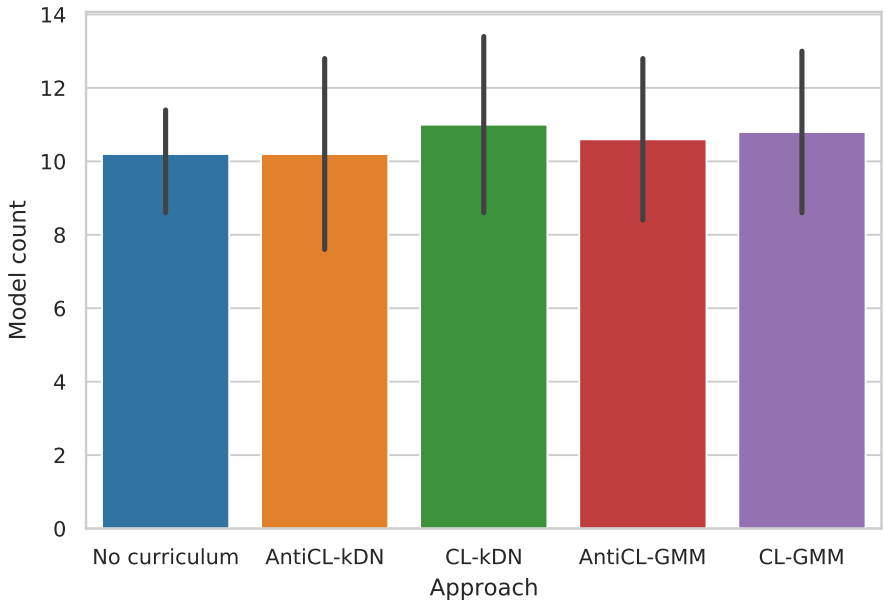

Figure 5.14: Average (and standard deviation) number of models for the Jasmine dataset using BOSH.

Table 5.18 displays the model count obtained by each approach using BOHB as the optimizer. Again, the average model count is similar among the approaches, as displayed in Figure 5.16.

The results for the Amazon employee access dataset using BOSH are presented in Table 5.19. For this dataset, curriculum approaches achieved improvements in four folds. AntiCL-kDN was the best in folds one and four, while AntiCL-GMM and CL-kDN achieved the best scores in fold two and five, respectively. However, none of the approaches could surpass the

Table 5.17: Log loss scores for the binary classification task on the Jasmine dataset for each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	**0.3995**	0.3755	0.3748	**0.3968**	**0.4255**
AntiCL-kDN	0.4005	0.3871	0.4425	0.4622	0.4374
AntiCL-GMM	0.4895	**0.3702**	0.3452	0.4605	0.4284
CL-kDN	0.4002	0.3847	**0.3377**	0.4000	0.4868
CL-GMM	0.4464	0.3911	0.4787	0.4143	0.5320

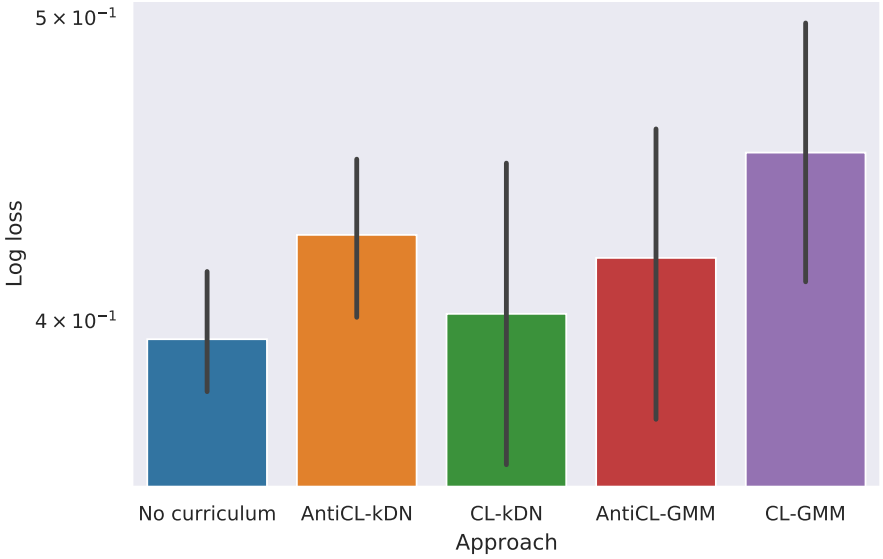

Figure 5.15: Average (and standard deviation) number of log loss scores for the Jasmine dataset using BOHB.

Table 5.18: Model count for the Jasmine dataset for each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	11	10	7	11	12
AntiCL-kDN	6	15	10	8	12
AntiCL-GMM	10	14	8	8	13
CL-kDN	15	11	11	7	11
CL-GMM	15	9	11	7	12

Table 5.19: Log loss scores for the binary classification task on the Amazon employee access dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	0.1603	0.1598	**0.1563**	0.1700	0.1511
AntiCL-kDN	**0.1500**	0.1567	0.1763	**0.1578**	0.1560
AntiCL-GMM	0.2671	**0.1554**	0.1718	0.1589	0.1752
CL-kDN	0.1512	0.1653	0.1586	0.1819	**0.1484**
CL-GMM	0.1721	0.1582	0.1679	0.1609	0.1532

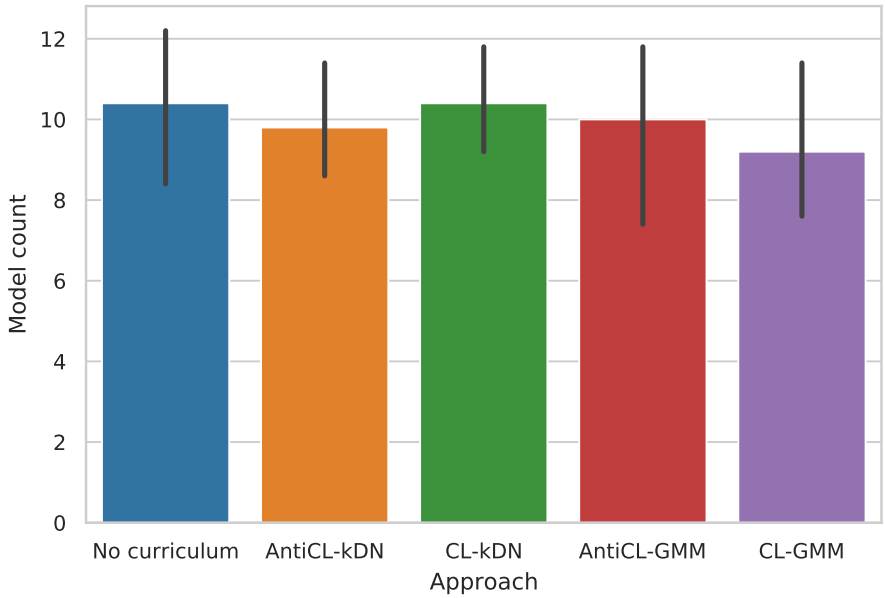

Figure 5.16: Average (and standard deviation) number of models for the Jasmine dataset using BOHB.

baseline result in fold three. Figure 5.17 demonstrates that, in average, most of the CL and AntiCL approaches achieved worse results than the baseline. Although, the AntiCL-kDN had an average score (0.1594) slightly better than the baseline (0.1595). Furthermore, Table 5.20 shows the model count obtained by the approaches on each cross-validation fold. In average, the AntiCL-kDN has the lowest average model count among approaches, as displayed in Figure 5.18

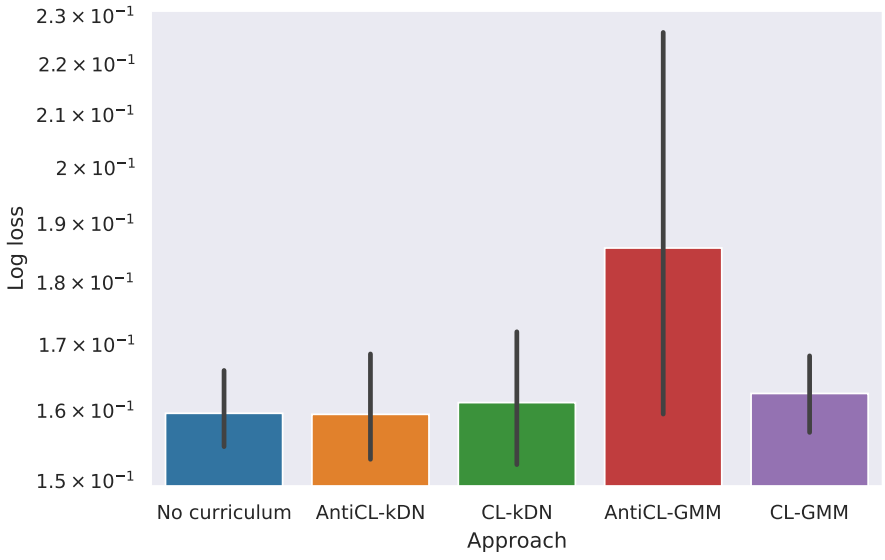

Figure 5.17: Average (and standard deviation) number of log loss scores for the Amazon employee access dataset using BOSH.

Table 5.20: Model count for the Amazon employee access dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	9	12	11	10	15
AntiCL-kDN	11	8	8	9	11
AntiCL-GMM	13	11	15	10	8
CL-kDN	14	9	16	13	12
CL-GMM	12	13	10	8	7

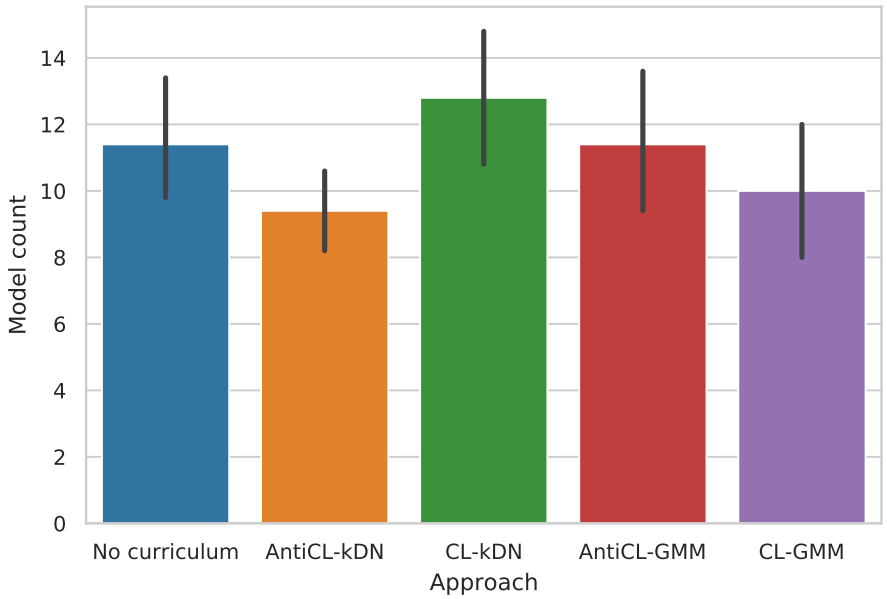

Figure 5.18: Average (and standard deviation) number of models for the Amazon employee access dataset using BOSH.

The results for the Amazon employee access dataset using BOHB are presented in Table 5.21. This time, the CL approach could not surpass the No curriculum baseline in most folds. However, the CL-kDN exhibited improvements over the baseline in folds one and three. Moreover, as shown in Figure 5.19, CL-kDN achieved the best average score. The model count obtained by each approach is displayed in Table 5.22. As displayed in Figure 5.20, the CL-kDN has a higher average model count than the No curriculum baseline, while AntiCL-GMM has the lowest among approaches.

Table 5.23 shows the results for the Australian dataset with the BOSH optimizer. Again, curriculum strategies exhibited improvements in most folds. CL-GMM achieved the best scores in folds one and two, while AntiCL-kDN and CL-kDN were better in folds four and five, respectively. While none of the approaches were able to surpass the baseline in fold three, Figure 5.21 demonstrates that, on average, all curriculum strategies showed improvement over the No curriculum baseline, and CL-kDN achieved the best average score. The model count obtained

Table 5.21: Log loss scores for the binary classification task on the Amazon employee access dataset for each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	0.2024	**0.1531**	0.1770	**0.1569**	**0.1450**
AntiCL-kDN	0.1746	0.1671	0.1611	0.1573	0.1577
AntiCL-GMM	0.2988	0.2645	0.1645	0.1617	0.1505
CL-kDN	**0.1553**	0.1564	**0.1603**	0.1582	0.1597
CL-GMM	0.2561	0.1933	0.1699	0.1656	0.1536

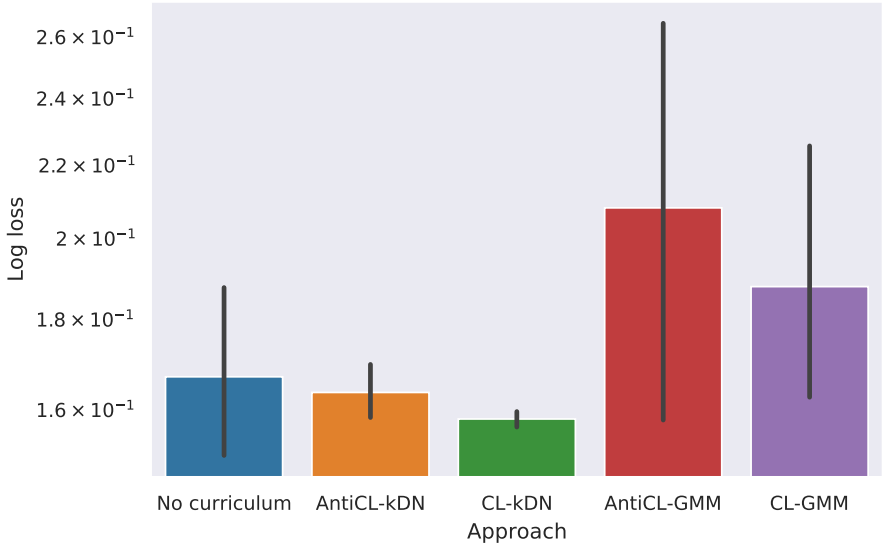

Figure 5.19: Average (and standard deviation) number of log loss scores for the Amazon employee access dataset using BOHB.

Table 5.22: Model count for the Amazon employee access dataset for each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	15	11	11	8	6
AntiCL-kDN	15	14	17	14	9
AntiCL-GMM	15	14	7	13	14
CL-kDN	13	14	8	11	9
CL-GMM	10	13	12	11	15

Table 5.23: Log loss scores for the binary classification task on the Australian dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	0.2930	0.7719	**0.3977**	0.3458	0.4145
AntiCL-kDN	0.4918	0.3973	0.4028	**0.3440**	0.5180
AntiCL-GMM	0.3401	0.3724	0.4836	0.3882	0.4497
CL-kDN	0.4105	0.3657	0.4150	0.4055	**0.3685**
CL-GMM	**0.2825**	**0.3525**	0.4574	0.4213	0.5676

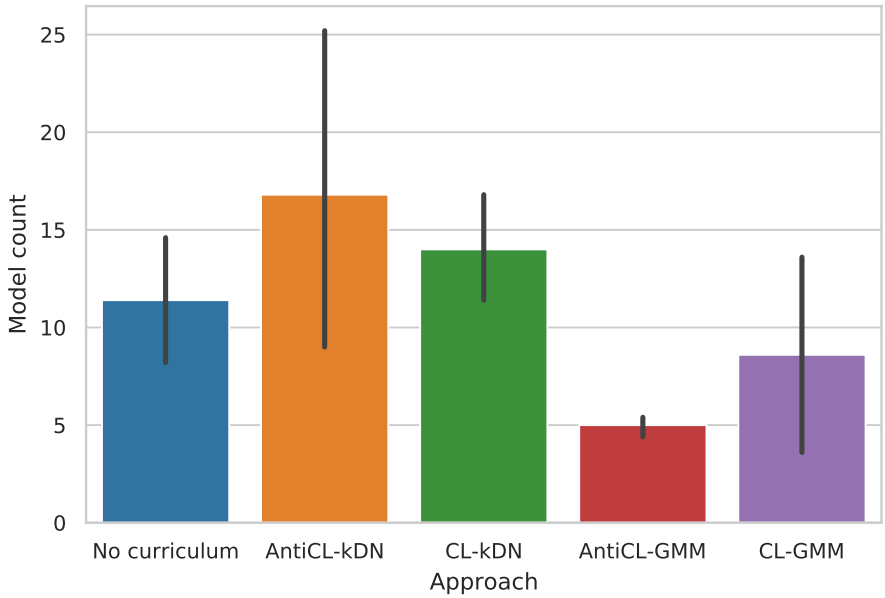

Figure 5.20: Average (and standard deviation) number of models for the Amazon employee access dataset using BOHB.

on each cross-validation fold is presented in Table 5.24. In average, the No curriculum baseline obtained the highest model count, followed by CL-GMM and CL-kDN, as shown in Figure 5.22.

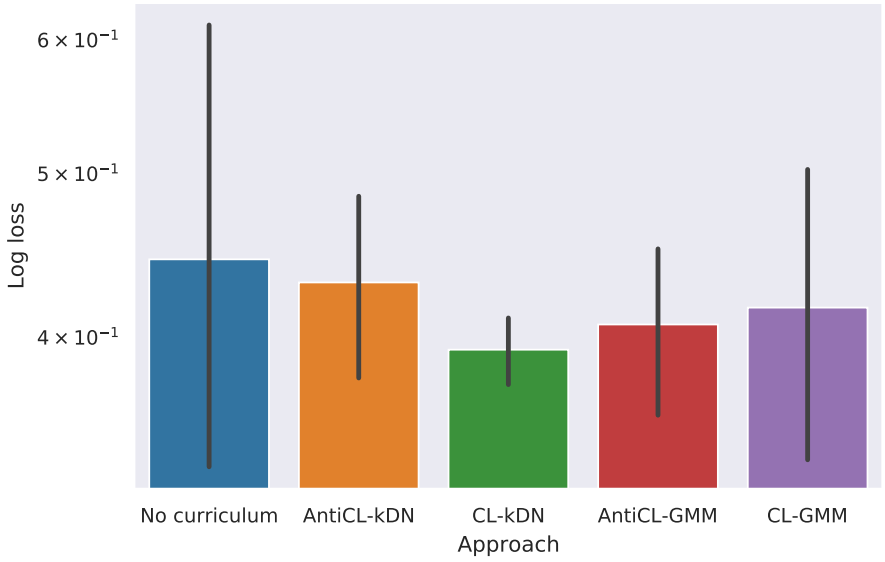

Figure 5.21: Average (and standard deviation) number of log loss scores for the Australian dataset using BOSH.

The results for the Australian dataset using BOHB are displayed in Table 5.25. In this experiment, curriculum strategies showed improvement in most folds, with exception to fold one. AntiCL-kDN achieved the best scores in folds three and five, while CL-GMM and AntiCL-

Table 5.24: Model count for the Australian dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	19	12	21	18	16
AntiCL-kDN	16	17	14	11	10
AntiCL-GMM	11	10	10	17	18
CL-kDN	16	15	10	14	17
CL-GMM	20	17	14	10	15

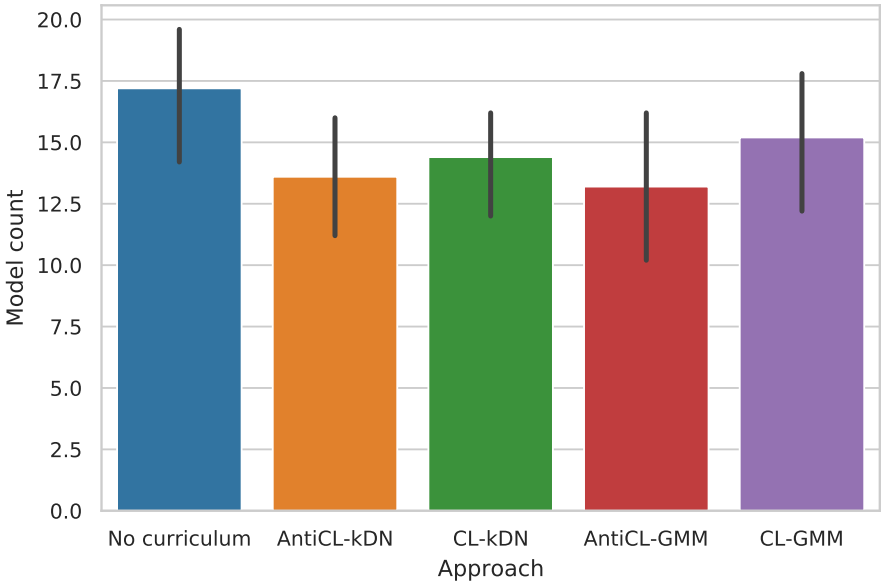

Figure 5.22: Average (and standard deviation) number of models for the Australian dataset using BOSH.

GMM were the best in folds two and four, respectively. Overall, the AntiCL-kDN had the best average score, as shown in Figure 5.23, it surpasses the other curriculum strategies and the baseline. Table 5.26 presents the model count obtained by each approach. Figure 5.24 shows that, in average, the approaches obtained similar model count values, yet AntiCL-GMM has the lowest average model count.

Table 5.25: Log loss scores for the binary classification task on the Australian dataset for each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	**0.3148**	0.4918	0.5515	0.6686	0.5367
AntiCL-kDN	0.3863	0.3762	**0.3965**	0.4171	**0.4519**
AntiCL-GMM	0.5653	0.4579	0.4413	**0.3568**	0.6820
CL-kDN	0.3684	0.3872	0.4000	0.3944	1.0419
CL-GMM	0.3842	**0.3690**	0.5164	0.3790	0.5604

The results for the Fashion-MNIST dataset using BOSH are presented in Table 5.27. Curriculum strategies show improvements in most folds, with the exception of fold three. CL-GMM

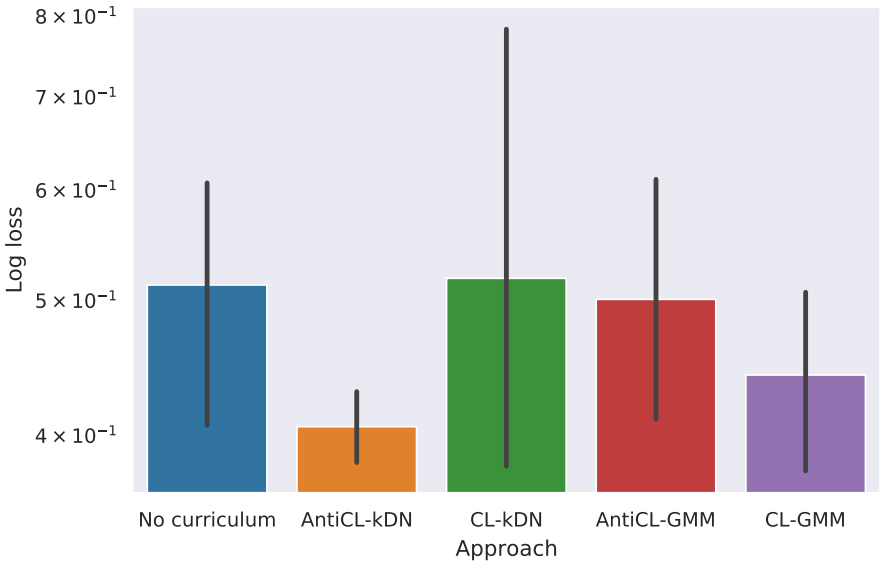

Figure 5.23: Average (and standard deviation) number of log loss scores for the Australian dataset using BOHB.

Table 5.26: Model count for the Australian dataset for each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	19	13	7	18	14
AntiCL-kDN	15	10	19	10	15
AntiCL-GMM	16	9	9	11	10
CL-kDN	15	14	17	12	15
CL-GMM	17	14	10	10	22

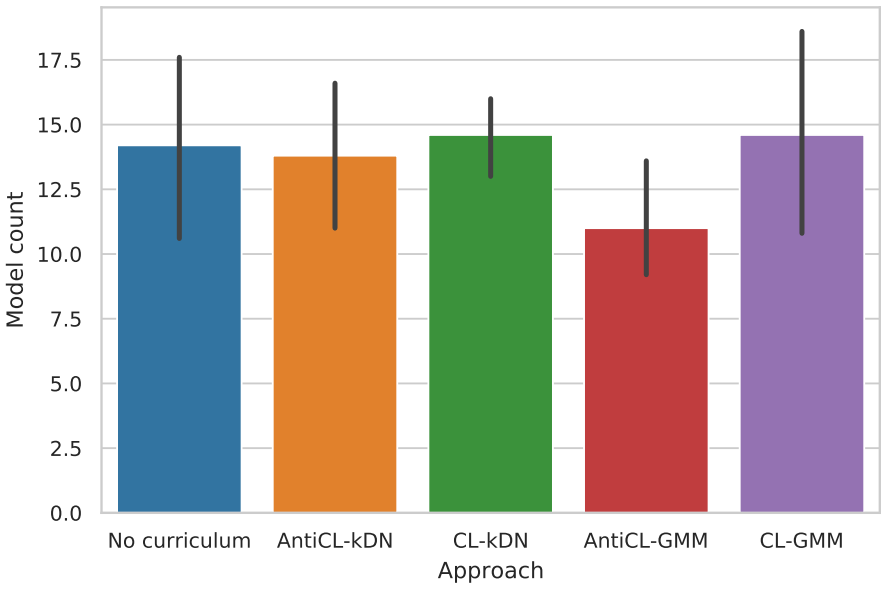

Figure 5.24: Average (and standard deviation) number of models for the Australian dataset using BOHB.

Table 5.27: Log loss scores for the multi-class classification task on the Fashion-MNIST dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	0.2797	0.2739	**0.2716**	0.2681	0.2625
AntiCL-kDN	0.2758	0.2677	0.2725	0.2703	**0.2616**
AntiCL-GMM	0.2742	0.2672	0.2761	0.2682	0.2664
CL-kDN	0.2826	**0.2669**	0.2771	0.2698	0.2640
CL-GMM	**0.2739**	0.2696	0.2735	**0.2641**	0.2679

obtained the best scores in folds one and four, while CL-kDN and AntiCL-kDN were better than the baseline in folds two and five, respectively. Figure 5.25 illustrates that, in average, only the CL-kDN approach could not surpass the No curriculum baseline, and AntiCL-kDN achieved the best score. Table 5.28 presents the model count obtained by the approaches on each cross-validation fold. Once again, all approaches had very similar model count values, also illustrated in Figure 5.26.

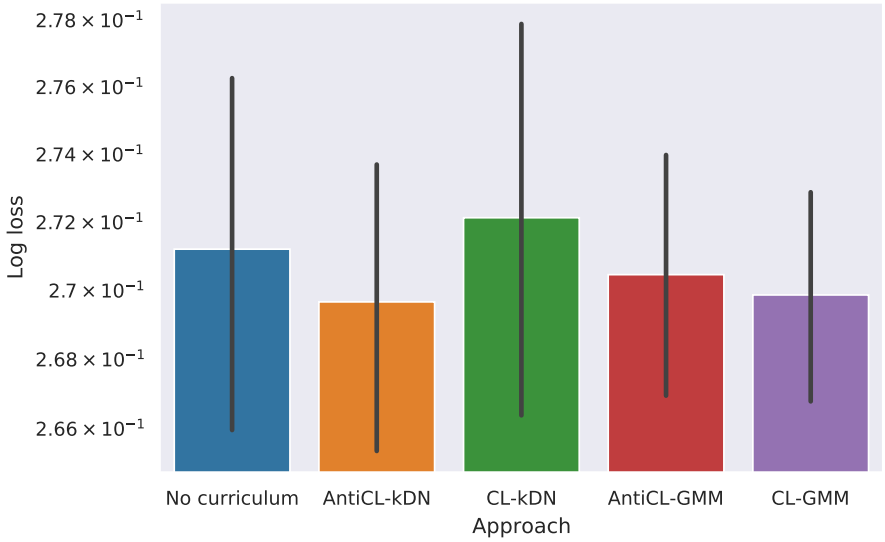

Figure 5.25: Average (and standard deviation) number of log loss scores for the Fashion-MNIST dataset using BOSH.

Table 5.28: Model count for the Fashion-MNIST dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	5	4	4	5	5
AntiCL-kDN	5	4	4	5	5
AntiCL-GMM	7	7	3	5	4
CL-kDN	4	5	5	6	5
CL-GMM	4	6	5	5	5

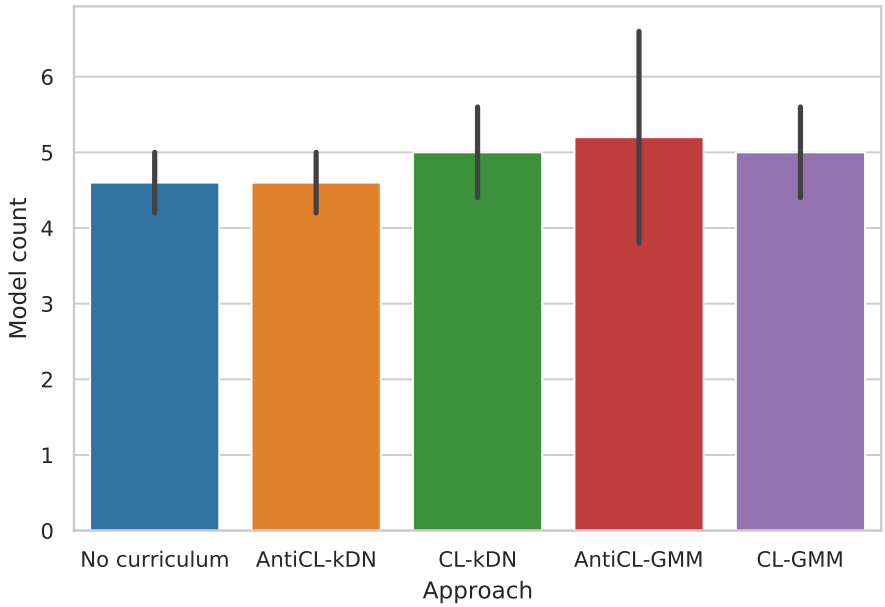

Figure 5.26: Average (and standard deviation) number of models for the Fashion-MNIST dataset using BOSH.

Table 5.29 presents the results for the Fashion-MNIST dataset using the BOHB optimizer. This time, none of the curriculum strategies were able to surpass the baseline in folds one and two. Yet, AntiCL-GMM demonstrated improvements in fold four and five. CL-kDN also exhibited a better score in fold three. In average, AntiCL-kDN and AntiCL-GMM showed minor improvements over the baseline, as illustrated in Figure 5.27. The model count obtained by each approach is presented in Table 5.30. As displayed in Figure 5.28, the average model count is similar to all approaches.

Table 5.29: Log loss scores for the multi-class classification task on the Fashion-MNIST dataset for each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	**0.2704**	**0.2659**	0.2741	0.2706	0.2708
AntiCL-kDN	0.2744	0.2711	0.2700	0.2702	0.2641
AntiCL-GMM	0.2784	0.2716	0.2719	**0.2674**	**0.2581**
CL-kDN	0.2752	0.2716	**0.2694**	0.2680	0.2678
CL-GMM	0.2736	0.2736	0.2810	0.2682	0.2672

Table 5.31 displays the results for the Blood transfusion dataset. Once again, curriculum strategies exhibited improvements over the baseline. AntiCL-kDN was the best approach in fold one, while CL-kDN and AntiCL-GMM achieved the best scores in folds three and four, respectively. Yet, the baseline could not be surpassed by any approach in folds two and five. Moreover, Figure 5.29 illustrates that, in average, none of the approaches were better than the

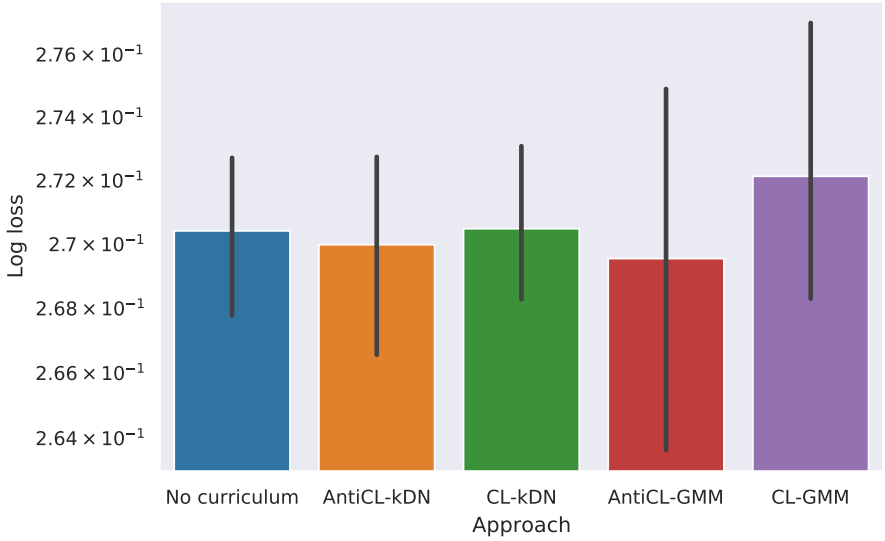

Figure 5.27: Average (and standard deviation) number of log loss scores for the Fashion-MNIST dataset using BOHB.

Table 5.30: Model count for the Fashion-MNIST dataset for each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	5	6	5	5	4
AntiCL-kDN	5	5	5	5	4
AntiCL-GMM	6	4	5	5	5
CL-kDN	6	4	5	6	4
CL-GMM	5	5	4	4	5

baseline in this case. Table 5.32 shows the model count for each approach. The average model count across cross-validation folds is illustrated in Figure 5.30.

Table 5.31: Log loss scores for the binary classification task on the Blood transfusion dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	0.5993	**0.4902**	0.5349	0.6931	**0.4426**
AntiCL-kDN	**0.4827**	0.7352	0.5373	0.5376	0.6571
AntiCL-GMM	0.5558	0.6335	0.5463	**0.4968**	0.5485
CL-kDN	0.5498	0.4961	**0.5407**	0.5824	0.6810
CL-GMM	0.6381	0.6673	0.5192	0.6821	0.4963

The results for the Blood transfusion dataset using the BOHB optimizer are presented in Table 5.33. In this experiment, AntiCL-GMM achieved the best scores in folds one and four. The CL-kDN approach was the best in fold two, while AntiCL-kDN had the best result in fold three. None of the approaches were able to surpass the baseline in fold five. Figure 5.31 shows that, in average, the best approach was AntiCL-GMM. The CL-GMM and CL-kDN approaches

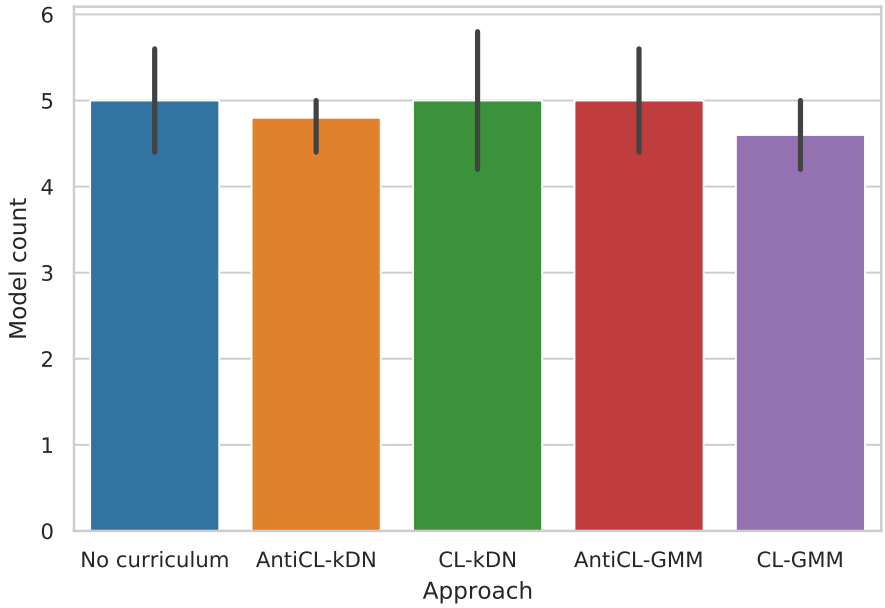

Figure 5.28: Average (and standard deviation) number of models for the Fashion-MNIST dataset using BOHB.

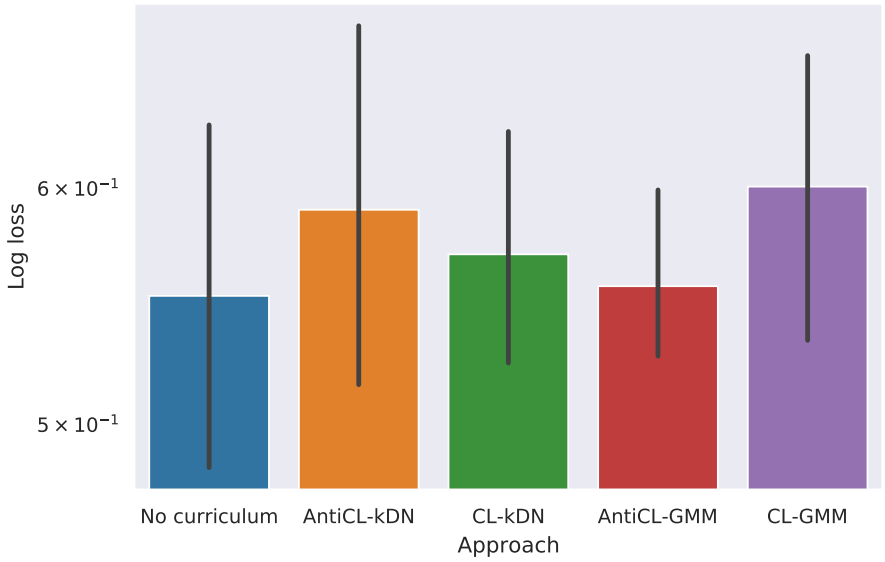

Figure 5.29: Average (and standard deviation) number of log loss scores for the Blood transfusion dataset using BOSH.

also achieved better average results than the baseline. Table 5.34 presents the model count results for this experiment. In average, the AntiCL-kDN obtained the highest model count among approaches, as illustrated in Figure 5.32.

Table 5.35 shows the results for the Christine dataset using the BOSH optimizer. In this case, only the AntiCL-GMM was able to achieve better results than the baseline in folds three, four, and five. The baseline was not surpassed in folds one and three. Although, Figure 5.33

Table 5.32: Model count for the Blood transfusion dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	11	17	18	2	15
AntiCL-kDN	7	5	16	9	7
AntiCL-GMM	9	7	17	20	4
CL-kDN	14	17	24	24	4
CL-GMM	5	8	21	5	22

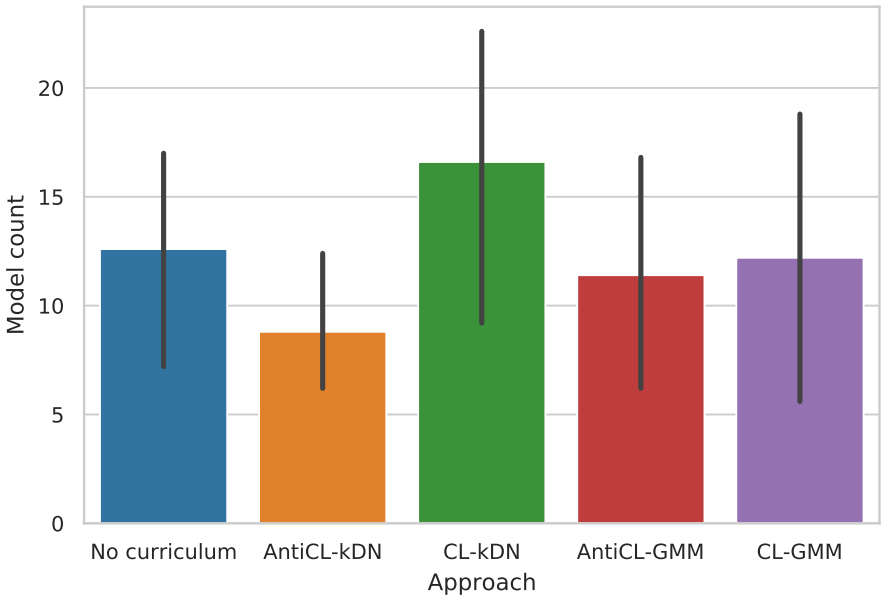

Figure 5.30: Average (and standard deviation) number of models for the Blood transfusion dataset using BOSH.

Table 5.33: Log loss scores for the binary classification task on the Blood transfusion dataset for each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	0.5613	0.4919	0.6009	0.6287	**0.5316**
AntiCL-kDN	0.5670	0.4643	**0.5237**	0.6262	0.6812
AntiCL-GMM	**0.5090**	0.5167	0.5379	**0.4874**	0.5371
CL-kDN	0.5825	**0.4331**	0.5955	0.5967	0.5442
CL-GMM	0.5712	0.4447	0.5535	0.5552	0.5422

illustrates that, in average, both AntiCL-GMM and AntiCL-kDN achieved better scores than the baseline. The model count that each approach obtained is displayed in Table 5.36. The model count is, in average, similar among the approaches, as can be observed in Figure 5.34.

Table 5.37 shows the results for the Christine dataset using BOHB. In this experiment, the baseline was not surpassed only in fold three. CL-kDN exhibited superior results in folds four and five, while AntiCL-GMM and CL-GMM achieved the best results in folds one and two,

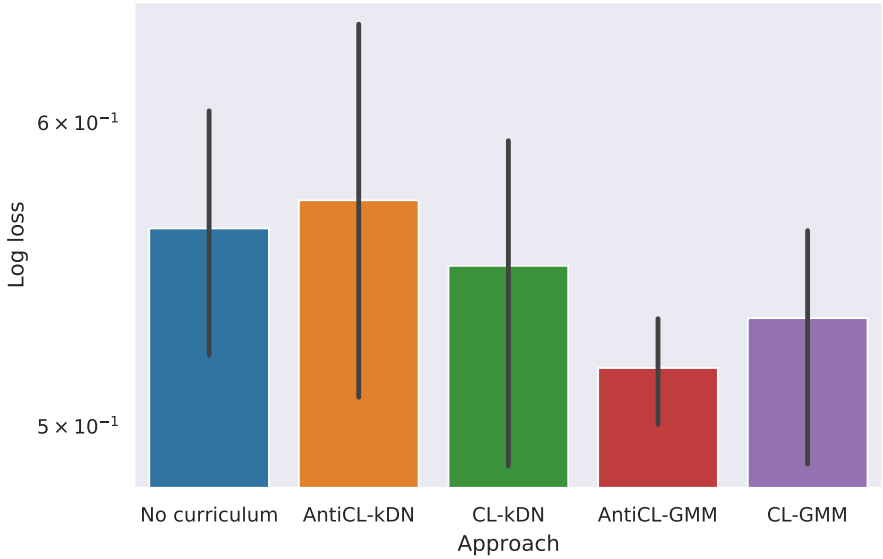

Figure 5.31: Average (and standard deviation) number of log loss scores for the Blood transfusion dataset using BOHB.

Table 5.34: Model count for the Blood transfusion dataset for each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	13	17	13	5	9
AntiCL-kDN	8	16	24	5	31
AntiCL-GMM	16	14	11	16	13
CL-kDN	19	10	11	14	16
CL-GMM	16	2	15	6	4

Table 5.35: Log loss scores for the binary classification task on the Christine dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	**0.5162**	0.5191	**0.5094**	0.5228	0.6147
AntiCL-kDN	0.5287	0.5152	0.5546	0.5209	0.5485
AntiCL-GMM	0.5372	**0.4923**	0.5423	**0.5201**	**0.5424**
CL-kDN	0.6442	0.5757	0.5537	0.5666	0.5899
CL-GMM	0.5179	0.5042	0.5461	0.5338	0.6663

Table 5.36: Model count for the Christine dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	13	16	10	9	11
AntiCL-kDN	18	12	11	12	10
AntiCL-GMM	14	11	14	16	16
CL-kDN	9	12	10	8	11
CL-GMM	13	12	16	12	7

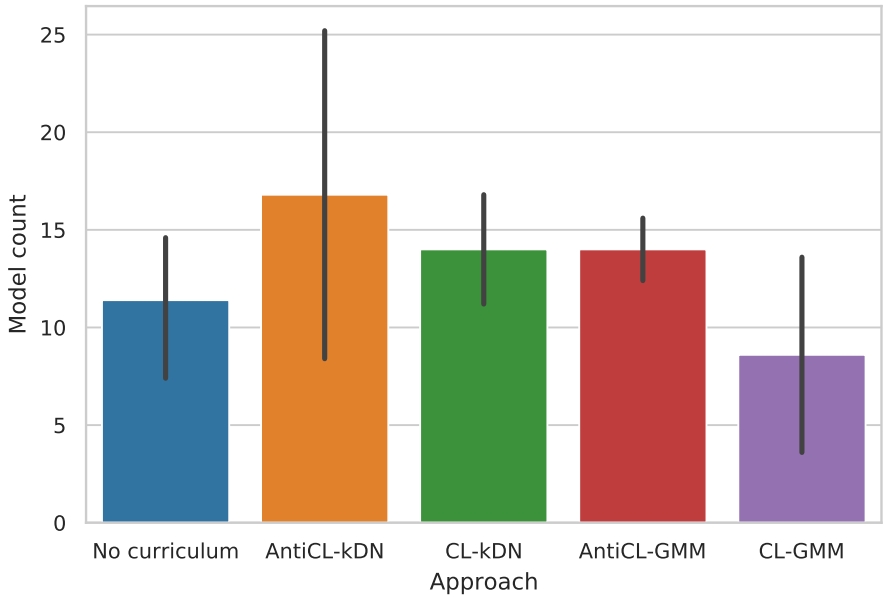

Figure 5.32: Average (and standard deviation) number of models for the Blood transfusion dataset using BOHB.

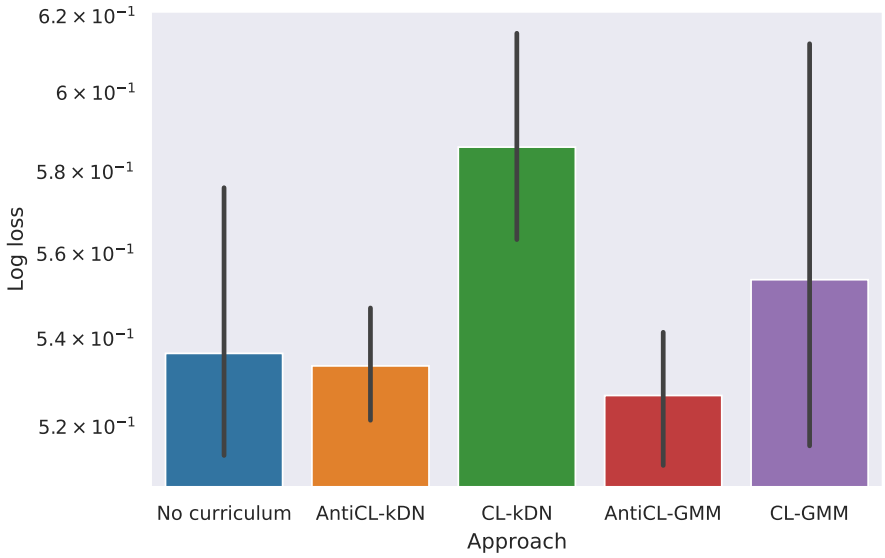

Figure 5.33: Average (and standard deviation) number of log loss scores for the Christine dataset using BOSH.

respectively. This three approaches were, in average, better than the baseline, as displayed in Figure 5.35. The best average result was obtained by the AntiCL-GMM approach. Table 5.38 displays the model count that each approach obtained on each cross-validation fold. Figure 5.36 illustrates that the average model count is again similar among approaches.

The results from the experiment on the CNAE-9 dataset, using BOSH, are presented in Table 5.39. Curriculum strategies show improvements in three folds. CL-GMM achieved achieved the best results in fold one and four, while AntiCL-GMM had the best score in fold two. The No

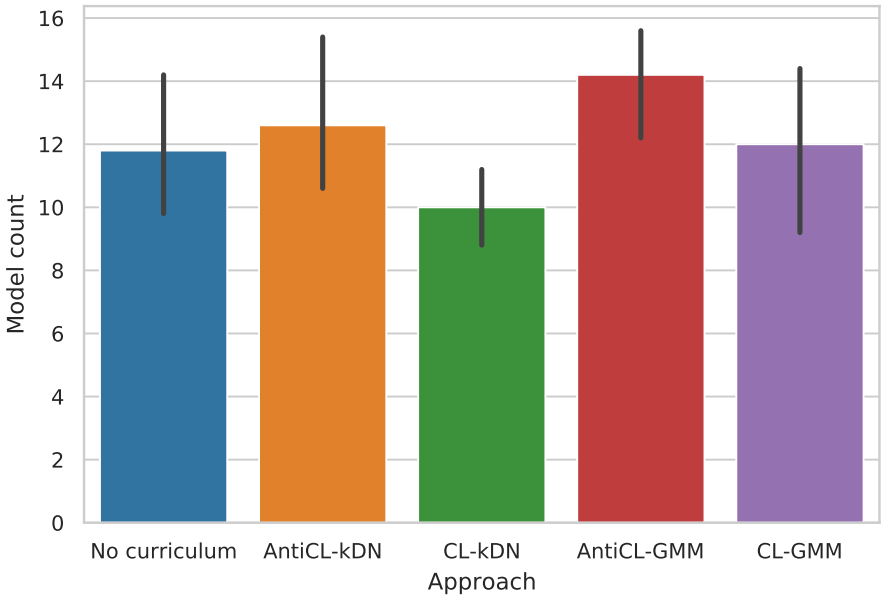

Figure 5.34: Average (and standard deviation) number of models for the Christine dataset using BOSH.

Table 5.37: Log loss scores for the binary classification task on the Christine dataset for each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	0.5232	0.5659	**0.5044**	0.5437	0.5654
AntiCL-kDN	0.5193	0.5527	0.5219	0.5388	0.5593
AntiCL-GMM	**0.5014**	0.5142	0.5180	0.5354	0.5359
CL-kDN	0.5396	0.5158	0.5485	**0.5160**	**0.5321**
CL-GMM	0.6406	**0.5101**	0.5821	0.5516	0.5329

Table 5.38: Model count for the Christine dataset for each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	13	17	11	10	12
AntiCL-kDN	10	10	10	10	14
AntiCL-GMM	14	11	10	12	14
CL-kDN	12	15	12	11	16
CL-GMM	9	14	12	12	12

curriculum baseline was not surpassed by any approach in fold three. Furthermore, Figure 5.37 illustrates that, in average, none of the approaches results surpassed the baseline. Table 5.40 presents the model count for each approach. Figure 5.38 shows that the average model count for the AntiCL-GMM was the highest, followed directly by CL-GMM. The other approaches had similar average model count values.

Table 5.41 presents the results for the CNAE-9 dataset using BOHB. In this case, the AntiCL-GMM and CL-kDN presented improvements over the baseline in folds two and five,

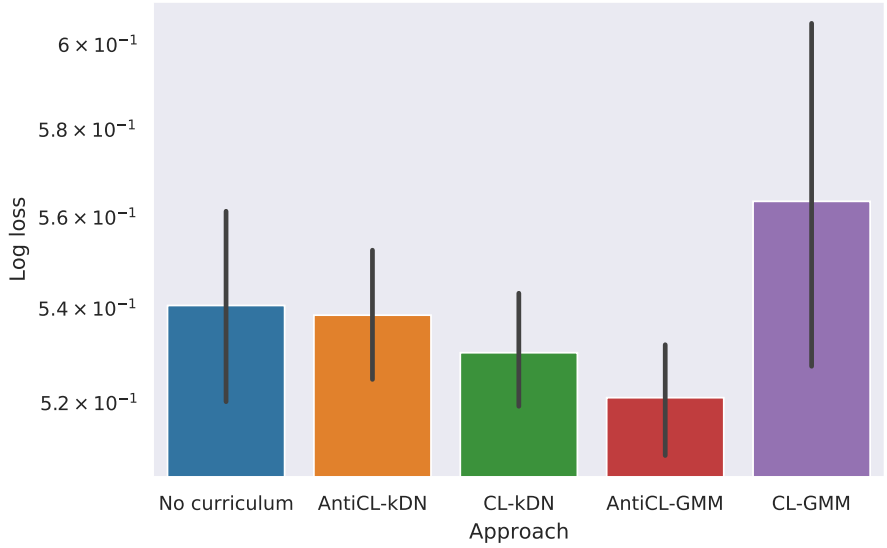

Figure 5.35: Average (and standard deviation) number of log loss scores for the Christine dataset using BOHB.

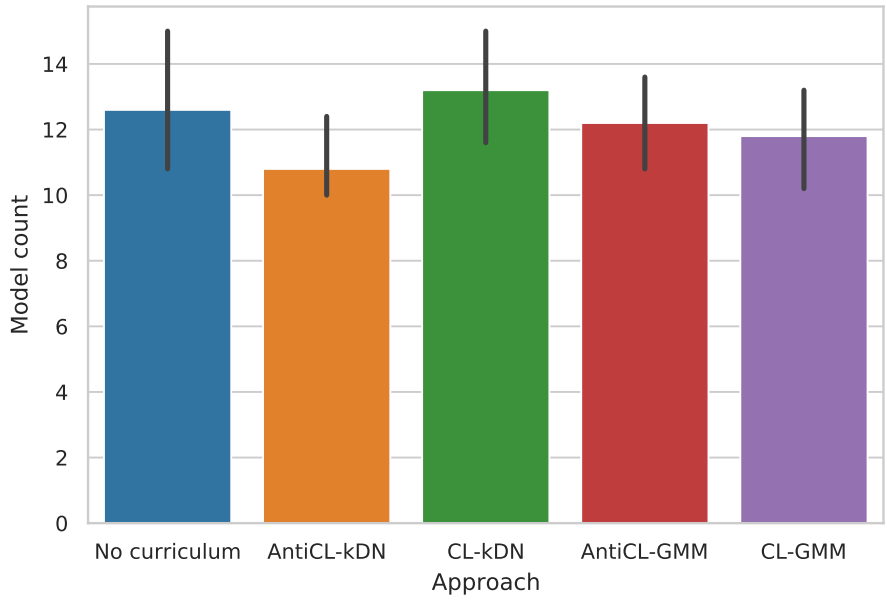

Figure 5.36: Average (and standard deviation) number of models for the Christine dataset using BOHB.

respectively. Yet, Figure 5.39 shows that none of the curriculum strategies surpassed the baseline average results. Table 5.42 displays the model count obtained for this dataset. Figure 5.40 illustrates the average model count for each approach.

The results for the Connect-4 dataset are displayed in Table 5.43. In this experiment, curriculum strategies were able to surpass the baseline results in all folds. The AntiCL-GMM approach had the best score in fold one. AntiCL-kDN surpassed all other approaches in folds two, three and four, while CL-GMM was the best in fold five. The average results displayed

Table 5.39: Log loss scores for the multi-class classification task on the CNAE-9 dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	0.1228	0.2657	**0.1255**	0.1849	**0.1641**
AntiCL-kDN	0.1046	0.2854	0.1351	0.2287	0.1741
AntiCL-GMM	0.1116	**0.2836**	0.1436	0.1897	0.2042
CL-kDN	0.1467	0.2918	0.2188	0.1964	0.1941
CL-GMM	**0.0857**	0.3518	0.1994	**0.1494**	0.1786

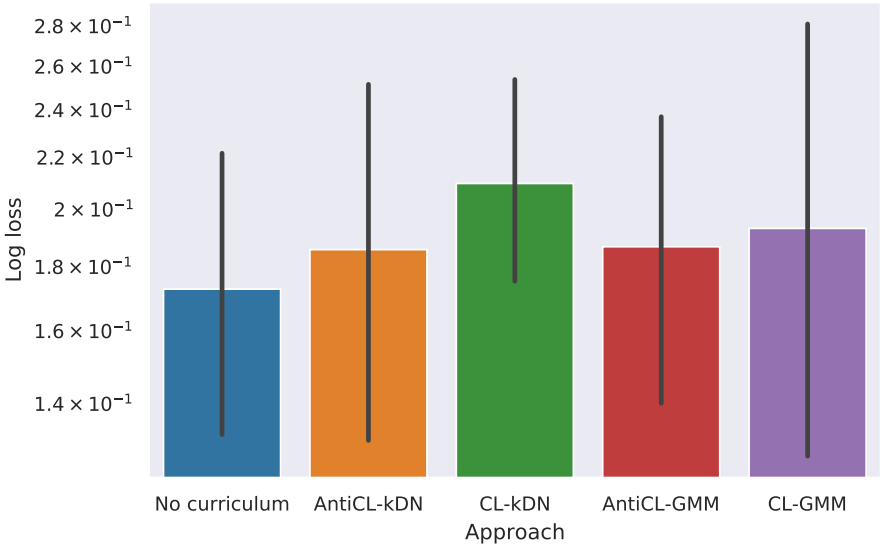

Figure 5.37: Average (and standard deviation) number of log loss scores for the CNAE-9 dataset using BOSH.

Table 5.40: Model count for the CNAE-9 dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	5	6	5	4	7
AntiCL-kDN	7	5	6	6	4
AntiCL-GMM	7	6	7	8	4
CL-kDN	7	5	6	3	4
CL-GMM	6	7	5	4	8

Table 5.41: Log loss scores for the multi-class classification task on the CNAE-9 dataset for each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	**0.0737**	0.2953	**0.1343**	**0.1402**	0.1345
AntiCL-kDN	0.0837	0.3144	0.1794	0.1513	0.1760
AntiCL-GMM	0.0904	**0.2923**	0.1688	0.1608	0.1497
CL-kDN	0.0754	0.2962	0.1395	0.1735	**0.1022**
CL-GMM	0.0795	0.3316	0.2010	0.2018	0.1404

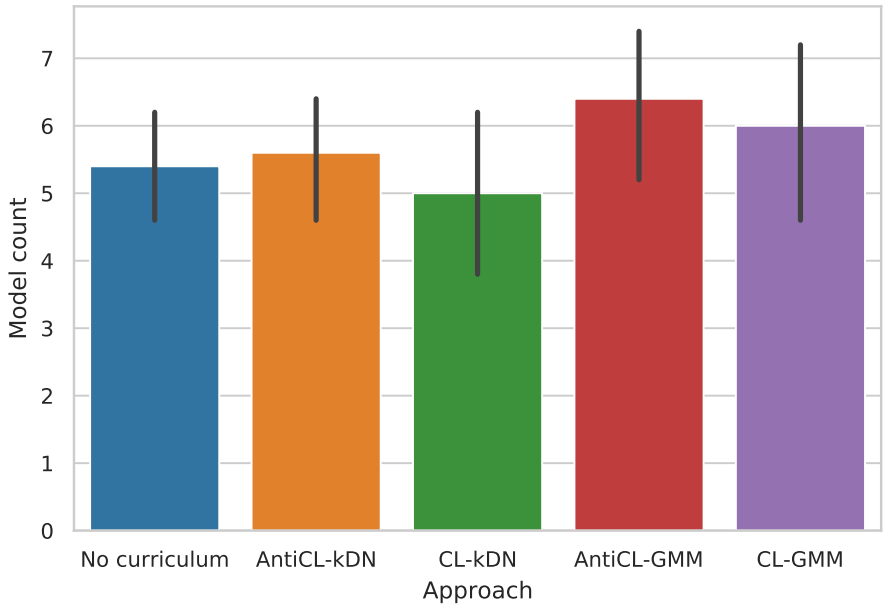

Figure 5.38: Average (and standard deviation) number of models for the CNAE-9 dataset using BOSH.

Table 5.42: Model count for the CNAE-9 dataset for each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	4	6	5	5	3
AntiCL-kDN	6	6	4	6	5
AntiCL-GMM	6	4	4	5	4
CL-kDN	6	5	8	7	3
CL-GMM	7	6	6	6	6

in Figure 5.41 show that all proposed curriculum strategies surpassed the baseline. Table 5.44 presents the model count obtained by each approach. As shown in Figure 5.42, in average, the AntiCL-GMM approach had the most models, while AntiCL-kDN had the least.

Table 5.43: Log loss scores for the multi-class classification task on the Connect-4 dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	0.3811	0.3662	0.3681	0.3675	0.3578
AntiCL-kDN	0.3678	**0.3599**	0.3317	0.3788	0.3728
AntiCL-GMM	0.3562	0.3619	**0.3314**	**0.3649**	0.3479
CL-kDN	0.3602	0.3632	0.3384	0.3579	0.3671
CL-GMM	**0.3503**	0.3669	0.3400	0.3859	0.3588

Table 5.45 presents the results for the Connect-4 dataset using BOHB. The baseline result was not surpassed. Yet, curriculum strategies exhibited improvements in every other fold. The CL-GMM approach achieved the best scores in folds two and four, while AntiCL-kDN and

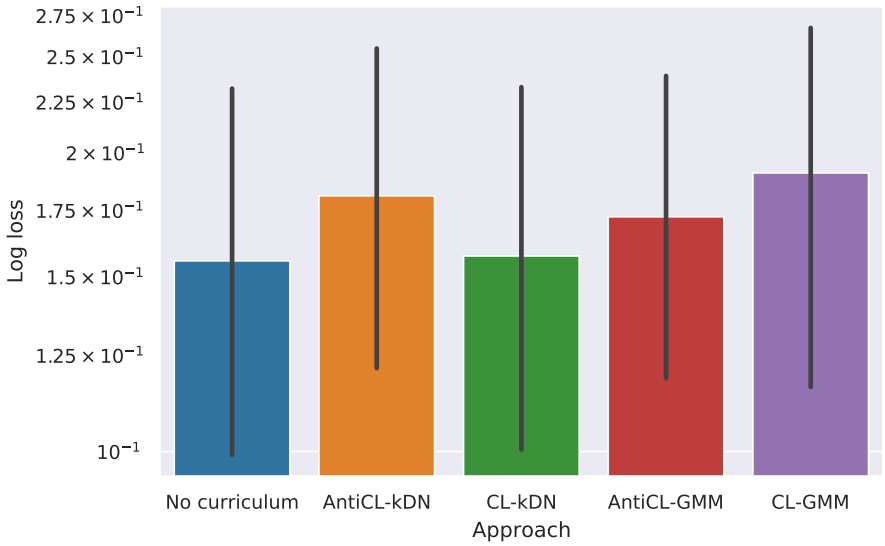

Figure 5.39: Average (and standard deviation) number of log loss scores for the CNAE-9 dataset using BOHB.

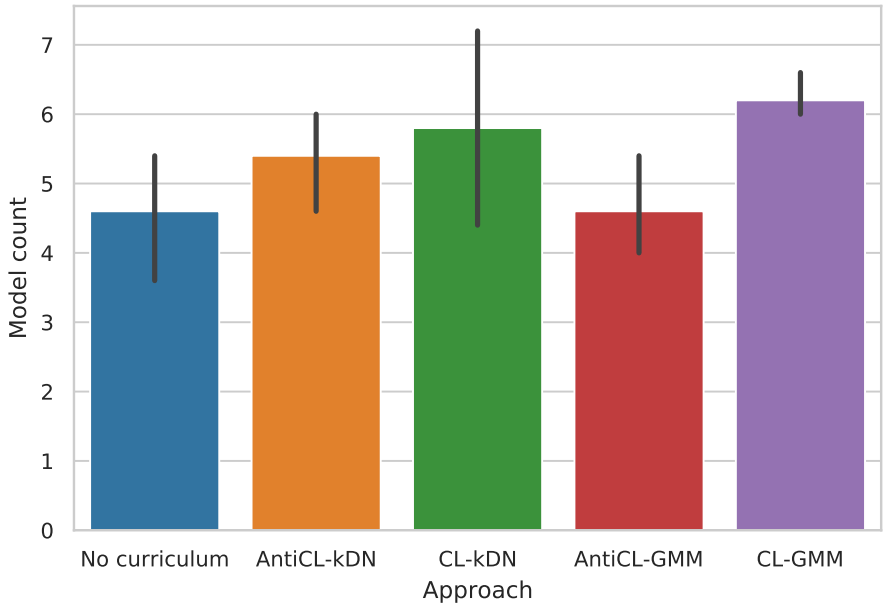

Figure 5.40: Average (and standard deviation) number of models for the CNAE-9 dataset using BOHB.

Table 5.44: Model count for the Connect-4 dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	3	4	2	3	5
AntiCL-kDN	2	4	6	2	2
AntiCL-GMM	6	4	5	6	6
CL-kDN	4	4	5	6	4
CL-GMM	5	4	3	3	6

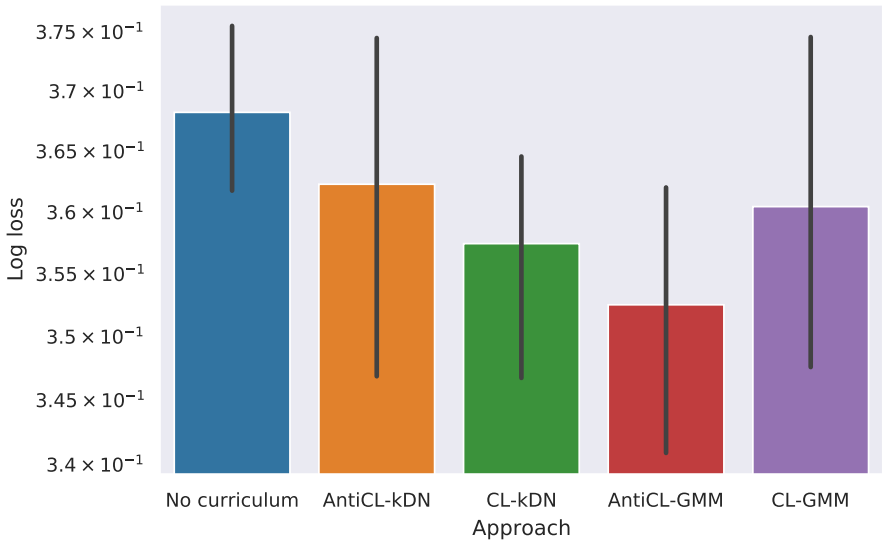

Figure 5.41: Average (and standard deviation) number of log loss scores for the Connect-4 dataset using BOSH.

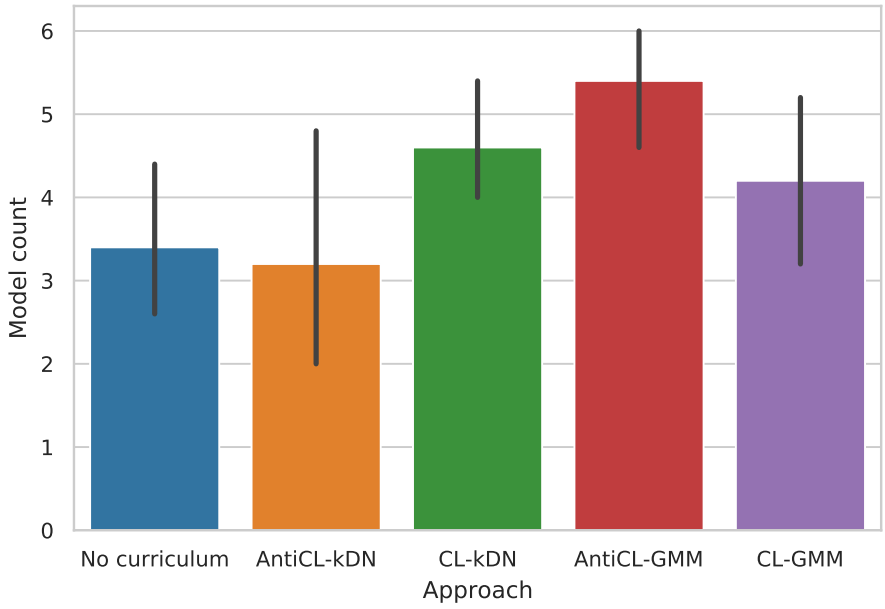

Figure 5.42: Average (and standard deviation) number of models for the Connect-4 dataset using BOSH.

CL-kDN had the best scores, respectively, in folds one and five. Figure 5.43 illustrates that, in average, most curriculum strategies surpassed the baseline results, with the exception of AntiCL-GMM. The model count obtained by each approach on each cross-validation fold is displayed in Table 5.46. The average model count is similar among the approaches, as can be observed in Figure 5.44.

Table 5.47 displays the results for the Dilbert dataset using BOSH. The use of curriculum strategies showed improvements in four folds. The CL-GMM approach had the best scores

Table 5.45: Log loss scores for the multi-class classification task on the Connect-4 dataset for each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	0.3726	0.3697	**0.3283**	0.3740	0.3733
AntiCL-kDN	**0.3520**	0.3642	0.3323	0.3809	0.3642
AntiCL-GMM	0.3561	0.3800	0.3685	0.3483	0.3644
CL-kDN	0.3683	0.3692	0.3287	0.3644	**0.3513**
CL-GMM	0.3746	**0.3571**	0.3379	**0.3584**	0.3740

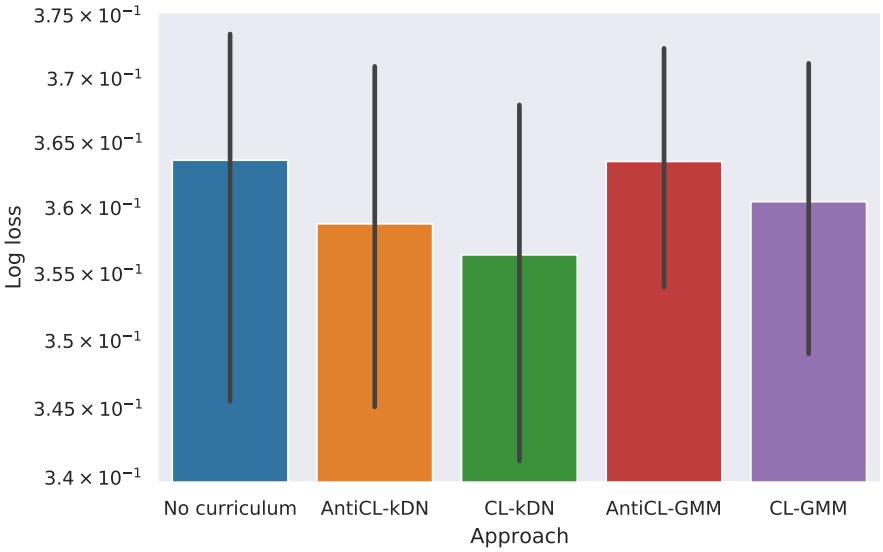

Figure 5.43: Average (and standard deviation) number of log loss scores for the Connect-4 dataset using BOHB.

Table 5.46: Model count for the Connect-4 dataset for each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	4	6	6	6	3
AntiCL-kDN	5	3	6	4	3
AntiCL-GMM	6	2	4	7	4
CL-kDN	3	7	6	4	4
CL-GMM	2	5	5	4	3

in folds two and four, while AntiCL-kDN and AntiCL-GMM surpassed the baseline results, respectively, in folds three and five. None of the approaches could surpass the baseline in fold one. However, the average results displayed in Figure 5.45, show that, all approaches achieved slightly better average results than the baseline. Table 5.48 presents the model count obtained in this case. In average, the model count among approaches is quite similar, as demonstrated in Figure 5.46.

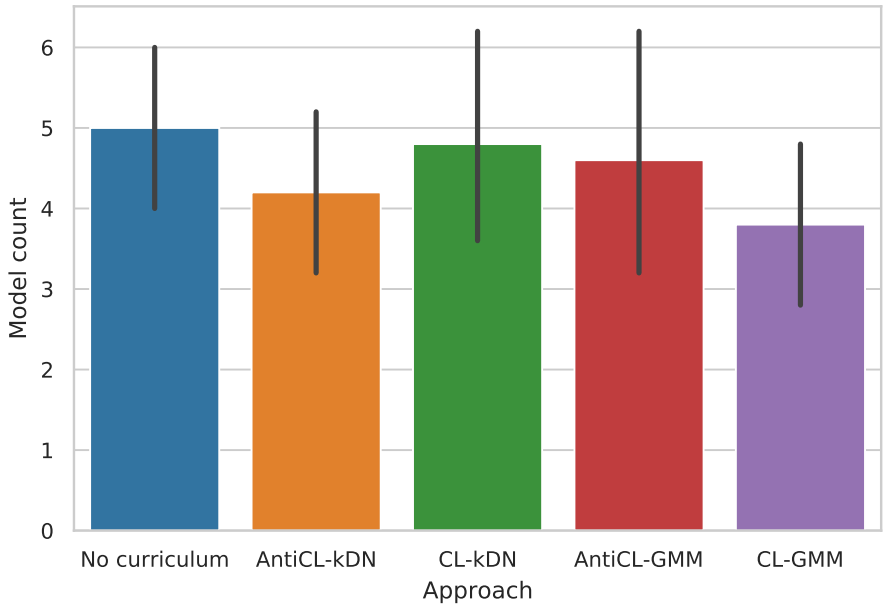

Figure 5.44: Average (and standard deviation) number of models for the Connect-4 dataset using BOHB.

Table 5.47: Log loss scores for the multi-class classification task on the Dilbert dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	**0.0390**	0.0297	0.0476	0.0231	0.0480
AntiCL-kDN	0.0445	0.0226	**0.0471**	0.0239	0.0436
AntiCL-GMM	0.0510	0.0262	0.0498	0.0222	**0.0371**
CL-kDN	0.0414	0.0232	0.0547	0.0202	0.0391
CL-GMM	0.0423	**0.0198**	0.0564	**0.0197**	0.0430

Table 5.48: Model count for the Dilbert dataset for each cross-validation fold using BOSH.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	2	3	5	4	6
AntiCL-kDN	4	3	3	3	5
AntiCL-GMM	5	7	4	3	5
CL-kDN	3	5	4	5	4
CL-GMM	4	4	5	3	4

Table 5.49 shows the results for the Dilbert dataset using BOHB. The CL-GMM approach achieved the best scores in fold two and five, while AntiCL-GMM and CL-kDN had the best results, respectively, in folds one and three. None of the approaches were able to surpass the No curriculum baseline in fold four. Although, as Figure 5.47 illustrates, all approaches had better average results than the baseline, and CL-GMM was the best approach. Table 5.50 shows the model count for this experiment. Once again, the average model count seems similar to all approaches, as shown in Figure 5.48.

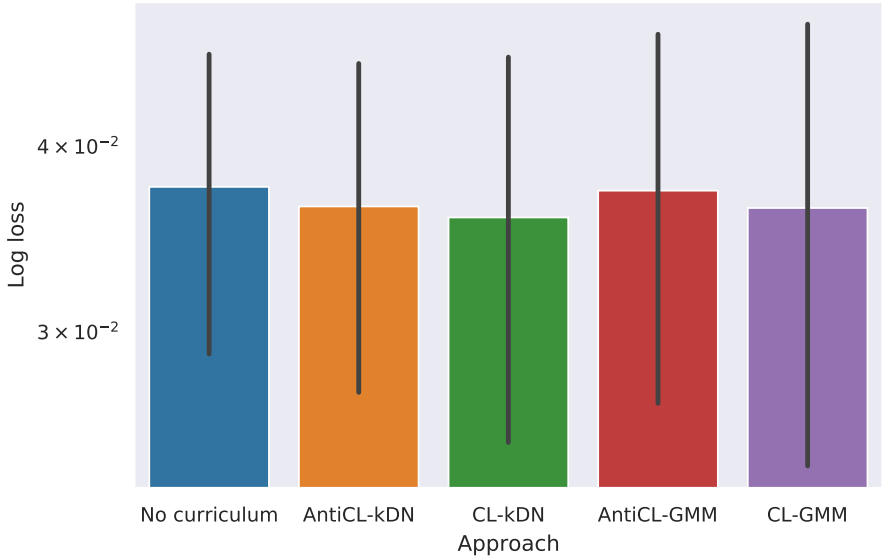

Figure 5.45: Average (and standard deviation) number of log loss scores for the Dilbert dataset using BOSH.

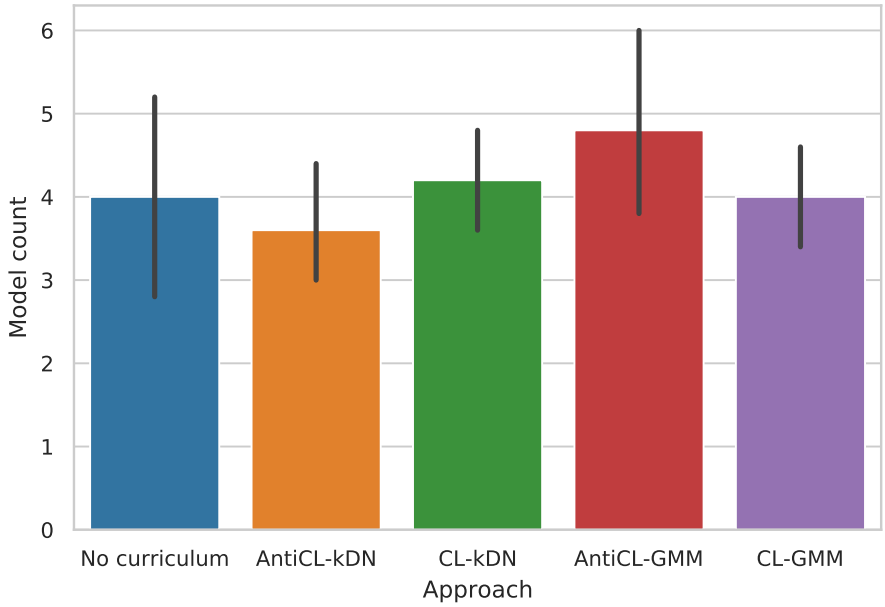

Figure 5.46: Average (and standard deviation) number of models for the Dilbert dataset using BOSH.

Finally, Tables 5.51 and 5.52 presents the average log loss scores and standard deviation on every selected dataset for each curriculum approach and optimizer. While using the BOSH optimizer (Table 5.51), curriculum approaches achieved minor improvements over the No curriculum baseline on ten of the twelve selected datasets, the AntiCL and CL approaches with a curriculum generated via kDN achieved improvements in seven datasets, while those generated via GMM were better in the remaining three. The AntiCL-kDN is the approach with the most number of improvements when using BOSH, followed by CL-kDN. The baseline was not

Table 5.49: Log loss scores for the multi-class classification task on the Dilbert dataset for each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	0.0524	0.0257	0.0570	**0.0196**	0.0444
AntiCL-kDN	0.0522	0.0291	0.0527	0.0198	0.0453
AntiCL-GMM	**0.0414**	0.0248	0.0488	0.0214	0.0438
CL-kDN	0.0563	0.0268	**0.0442**	0.0208	0.0505
CL-GMM	0.0505	**0.0231**	0.0452	0.0212	**0.0346**

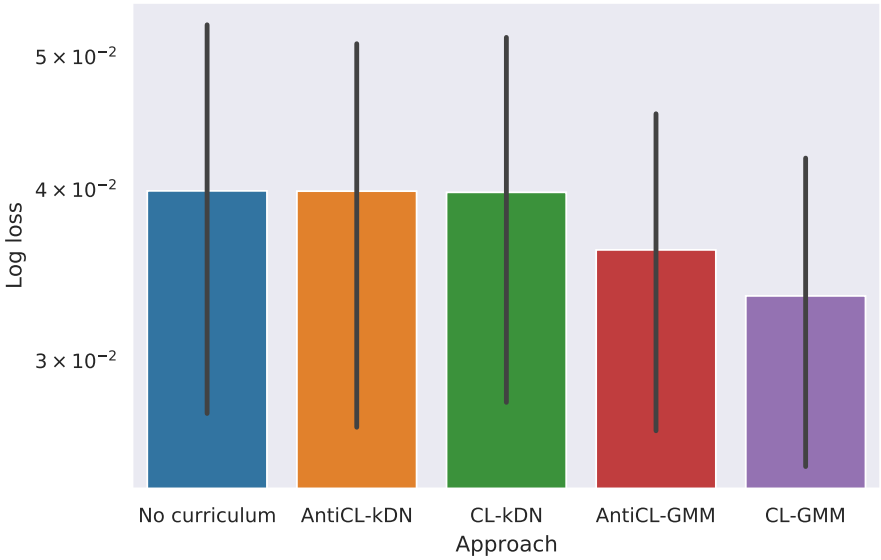

Figure 5.47: Average (and standard deviation) number of log loss scores for the Dilbert dataset using BOHB.

Table 5.50: Model count for the Dilbert dataset for each cross-validation fold using BOHB.

Approach	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
No Curriculum	4	5	3	3	6
AntiCL-kDN	5	3	4	6	6
AntiCL-GMM	4	5	3	4	5
CL-kDN	3	5	4	7	8
CL-GMM	4	4	4	4	4

surpassed on the Blood transfusion (binary classification) dataset, and the CNAE-9 (multiclass classification) dataset.

For the BOHB optimizer(Table 5.52), curriculum approaches showed improvements on nine of the twelve selected datasets. The AntiCL and CL approaches based on GMM surpassed the baseline average score on six datasets, while the kDN based approaches achieved improvements in three datasets. None of the curriculum approaches achieved better results than the No curriculum baseline on the MiniBooNE (binary classification), Jasmine (binary classification),

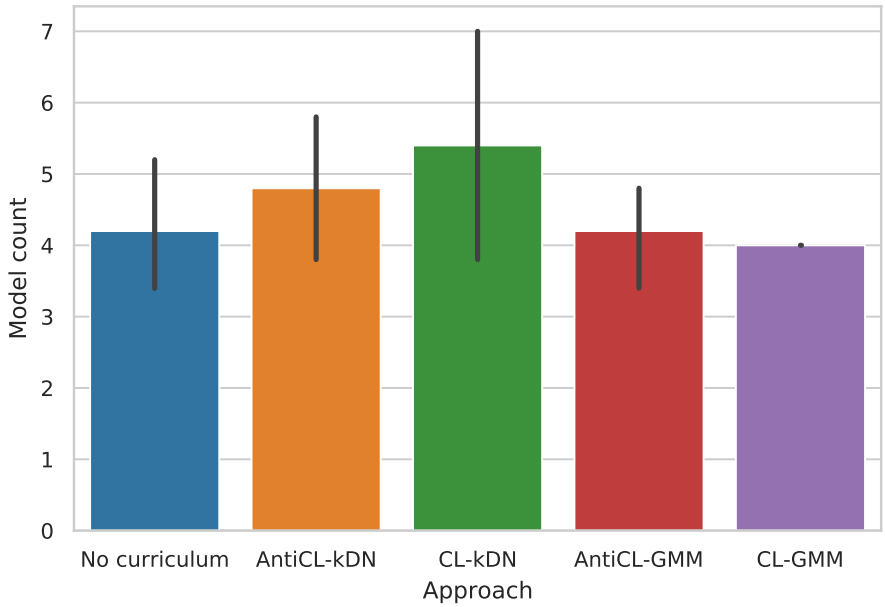

Figure 5.48: Average (and standard deviation) number of models for the Dilbert dataset using BOHB.

and CNAE-9 (multiclass classification) datasets. When using BOHB, the AntiCL-GMM was the most successful approach, achieving the best results on four datasets.

Table 5.51: Average log loss scores and standard deviation on all selected datasets for every curriculum strategy using BOSH.

Dataset	No CL	AntiCL-kDN	AntiCL-GMM	CL-kDN	CL-GMM
Covertype	0.2572 (0.04)	**0.2133 (0.03)**	0.2502 (0.04)	0.2413 (0.04)	0.2456 (0.04)
Helena	2.5589 (0.01)	2.5563 (0.01)	2.5605 (0.01)	**2.5532 (0.01)**	2.5585 (0.02)
MiniBooNE	0.2627 (0.19)	0.1393 (0.006)	0.2818 (0.16)	**0.1337 (0.005)**	0.1340 (0.004)
Jasmine	0.4003 (0.04)	**0.3905 (0.04)**	0.3965 (0.02)	0.3942 (0.04)	0.3918 (0.02)
Amazon	0.1595 (0.06)	**0.1594 (0.09)**	0.1857 (0.04)	0.1611 (0.01)	0.1624 (0.007)
Australian	0.4446 (0.18)	0.4308 (0.07)	0.4068 (0.05)	**0.3930 (0.02)**	0.4163 (0.1)
Fashion-MNIST	0.2711 (0.006)	**0.2696 (0.005)**	0.2704 (0.004)	0.2721 (0.007)	0.2698 (0.004)
Blood transfusion	**0.5520 (0.09)**	0.5900 (0.1)	0.5562 (0.04)	0.5700 (0.06)	0.6006 (0.08)
Christine	0.5364 (0.04)	0.5336 (0.01)	**0.5269 (0.02)**	0.5860 (0.03)	0.5537 (0.06)
CNAE-9	**0.1726 (0.05)**	0.1856 (0.07)	0.1866 (0.06)	0.2095 (0.05)	0.1930 (0.09)
Connect-4	0.3681 (0.08)	0.3622 (0.01)	**0.3524 (0.01)**	0.3574 (0.01)	0.3604 (0.01)
Dilbert	0.0375 (0.01)	0.0364 (0.01)	0.0373 (0.01)	0.0357 (0.01)	**0.0363 (0.01)**

Table 5.52: Average log loss scores and standard deviation on all selected datasets for every curriculum strategy using BOHB.

Dataset	No CL	AntiCL-kDN	AntiCL-GMM	CL-kDN	CL-GMM
Covertype	0.2371 (0.04)	0.2360 (0.08)	0.2472 (0.03)	0.2454 (0.06)	**0.2260 (0.01)**
Helena	2.5621 (0.01)	2.5594 (0.01)	**2.5540 (0.01)**	2.5593 (0.01)	2.5633 (0.01)
MiniBooNE	**0.1368 (0.005)**	0.2040 (0.15)	0.1438 (0.008)	0.3695 (0.22)	0.1371 (0.007)
Jasmine	**0.3945 (0.02)**	0.4260 (0.03)	0.4188 (0.06)	0.4019 (0.05)	0.4526 (0.05)
Amazon	0.1669 (0.02)	0.1635 (0.007)	0.2080 (0.06)	**0.1580 (0.002)**	0.1877 (0.04)
Australian	0.5127 (0.1)	**0.4056 (0.02)**	0.5007 (0.1)	0.5184 (0.29)	0.4418 (0.08)
Fashion-MNIST	0.2704 (0.002)	0.2699 (0.003)	**0.2695 (0.007)**	0.2704 (0.003)	0.2721 (0.005)
Blood transfusion	0.5629 (0.05)	0.5725 (0.08)	**0.5176 (0.02)**	0.5504 (0.06)	0.5334 (0.05)
Christine	0.5405 (0.02)	0.5384 (0.01)	**0.5210 (0.01)**	0.5304 (0.01)	0.5635 (0.05)
CNAE-9	**0.1556 (0.08)**	0.1810 (0.08)	0.1724 (0.07)	0.1574 (0.08)	0.1908 (0.09)
Connect-4	0.3636 (0.01)	0.3587 (0.01)	0.3635 (0.01)	**0.3564 (0.01)**	0.3604 (0.01)
Dilbert	0.0398 (0.01)	0.0398 (0.01)	0.0361 (0.01)	0.0397 (0.01)	**0.0334 (0.01)**

Chapter 6

FINAL REMARKS

We proposed and compared different CL strategies on the CASH solvers of an AutoML system to improve the search space exploration and find good performing machine-learned models. Our proposal, CurL-AutoML, has the design of an automatic CL framework. More specifically, we adopted a Teacher Transfer strategy, using instance hardness methods as our automatic difficulty measurers, and adapted the SH procedure implemented in BOSH and BOHB, the CASH solvers for the AutoML system Auto-sklearn, to act as our predefined training scheduler during the AutoML optimization.

The experiments show that using a curriculum strategy on CASH solvers can guide AutoML to better results in most evaluated scenarios. When analyzing the results, a clear relationship between the model count and the final performance of an approach does not seem to exist. Overall, in our experiments, the AntiCL and CL approaches based on kDN were better when using BOSH as the optimizer, while those based on GMM were better with BOHB. The most successful approaches utilized the anti-curriculum strategy. We assume that by evaluating configuration first on the most difficult data, the SH procedure can select more robust algorithm configurations, which otherwise would be discarded.

In some cases, the AntiCL approaches achieved worse results, indicating that they may not be adequate to be used in specific scenarios. Moreover, the approaches based on kDN were better when using BOSH, while those based on GMM had the advantage when utilizing BOHB. For this reason, we intend to study the relations between difficult measurers and the data space to understand this phenomenon better and propose techniques to avoid these pitfalls.

Since the curriculum generation time for each dataset was not much costly, it is feasible to add a pre-training step in the AutoML pipeline to benefit from a CL strategy.

6.1 Future work

Even after our broad experimental evaluation, there still a need to investigate the effects of CL strategies for AutoML in a broader range of datasets and ML tasks. Using a Transfer Teacher strategy, we use a teacher algorithm, such as kDN or GMM, with its hyperparameters configurations to estimate example difficulty. Thus, future work may investigate the effects of the proposed difficulty measurers hyperparameters on the final generated curriculum.

Besides, we intend to investigate the mechanisms that make one approach achieve better results than the others in each case. Depending on the results of this exploration, we may propose a technique to automatically select or ensemble different strategies.

Moreover, there is still room for improvement regarding the proposed training schedulers, concerning its pace (halving factor) and budget size.

REFERENCES

ANDERSON, R. L. Recent advances in finding best operating conditions. *Journal of the American Statistical Association*, Taylor & Francis, v. 48, n. 264, p. 789–798, 1953.

BADUE, C. et al. Self-driving cars: A survey. *Expert Systems with Applications*, Elsevier, p. 113816, 2020.

BENGIO, Y. et al. Curriculum learning. In: *Proceedings of the 26th Annual International Conference on Machine Learning*. New York, NY, USA: Association for Computing Machinery, 2009. (ICML '09), p. 41–48. ISBN 9781605585161. Available at: <https://doi.org/10.1145/1553374.1553380>.

BERGSTRA, J.; BENGIO, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, v. 13, n. 10, p. 281–305, 2012.

BREIMAN, L. Random forests. *Machine Learning*, Kluwer Academic Publishers, USA, v. 45, n. 1, p. 5–32, Oct. 2001. ISSN 0885-6125. Available at: <https://doi.org/10.1023/A:1010933404324>.

BROCHU, E.; CORA, V. M.; FREITAS, N. de. *A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning*. 2010.

CARUANA, R. et al. Ensemble selection from libraries of models. In: *Proceedings of the Twenty-First International Conference on Machine Learning*. New York, NY, USA: Association for Computing Machinery, 2004. (ICML '04), p. 18. ISBN 1581138385. Available at: <https://doi.org/10.1145/1015330.1015432>.

CHEN, B. et al. Autostacker: A compositional evolutionary learning system. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. New York, NY, USA: Association for Computing Machinery, 2018. (GECCO '18), p. 402–409. ISBN 9781450356183. Available at: <https://doi.org/10.1145/3205455.3205586>.

EL-BOURI, R. et al. Student-teacher curriculum learning via reinforcement learning: Predicting hospital inpatient admission location. In: III, H. D.; SINGH, A. (Ed.). *Proceedings of the 37th International Conference on Machine Learning*. [S.l.]: PMLR, 2020. (Proceedings of Machine Learning Research, v. 119), p. 2848–2857.

ELMAN, J. L. Learning and development in neural networks: the importance of starting small. *Cognition*, v. 48, n. 1, p. 71–99, 1993. ISSN 0010-0277. Available at: <https://www.sciencedirect.com/science/article/pii/0010027793900584>.

FALKNER, S.; KLEIN, A.; HUTTER, F. BOHB: Robust and efficient hyperparameter optimization at scale. In: DY, J.; KRAUSE, A. (Ed.). *Proceedings of the 35th International Conference on Machine Learning*. Stockholmsmässan, Stockholm Sweden: PMLR, 2018. (Proceedings of Machine Learning Research, v. 80), p. 1437–1446.

FERNáNDEZ-GODINO, M. G. et al. Issues in deciding whether to use multifidelity surrogates. *AIAA Journal*, American Institute of Aeronautics and Astronautics (AIAA), v. 57, n. 5, p. 2039–2054, May 2019. ISSN 1533-385X. Available at: <http://dx.doi.org/10.2514/1.J057750>.

FEURER, M. et al. *Auto-Sklearn 2.0: The Next Generation*. 2020.

FEURER, M.; HUTTER, F. Towards further automation in automl. In: *ICML AutoML workshop*. [S.l.: s.n.], 2018.

FEURER, M. et al. Efficient and robust automated machine learning. In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*. Cambridge, MA, USA: MIT Press, 2015. (NIPS'15), p. 2755–2763.

FLORENSA, C. et al. Reverse curriculum generation for reinforcement learning. In: LEVINE, S.; VANHOUCKE, V.; GOLDBERG, K. (Ed.). *Proceedings of the 1st Annual Conference on Robot Learning*. PMLR, 2017. (Proceedings of Machine Learning Research, v. 78), p. 482–495. Available at: <http://proceedings.mlr.press/v78/florensa17a.html>.

GIJSBERS, P. et al. An open source automl benchmark. In: *6th ICML Workshop on Automated Machine Learning*. [S.l.: s.n.], 2019.

GUO, S. et al. Curriculumnet: Weakly supervised learning from large-scale web images. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. [S.l.: s.n.], 2018. p. 135–150.

GUO, Y. et al. Breaking the curse of space explosion: Towards efficient nas with curriculum search. In: PMLR. *International Conference on Machine Learning*. [S.l.], 2020. p. 3822–3831.

HIRSCHBERG, J.; MANNING, C. D. Advances in natural language processing. *Science*, American Association for the Advancement of Science, v. 349, n. 6245, p. 261–266, 2015.

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. *Neural networks*, Elsevier, v. 2, n. 5, p. 359–366, 1989.

HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K. Sequential model-based optimization for general algorithm configuration. In: *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*. Berlin, Heidelberg: Springer-Verlag, 2011. (LION'05), p. 507–523. ISBN 9783642255656.

HUTTER, F.; KOTTHOFF, L.; VANSCHOREN, J. *Automated machine learning: methods, systems, challenges*. [S.l.]: Springer Nature, 2019.

JAMIESON, K.; TALWALKAR, A. Non-stochastic best arm identification and hyperparameter optimization. In: GRETTON, A.; ROBERT, C. C. (Ed.). *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. Cadiz, Spain: PMLR, 2016. (Proceedings of Machine Learning Research, v. 51), p. 240–248. Available at: <http://proceedings.mlr.press/v51/jamieson16.html>.

KOCMI, T.; BOJAR, O. Curriculum learning and minibatch bucketing in neural machine translation. In: *Proceedings of the International Conference Recent Advances in Natural Language Processing, RANLP 2017*. [S.l.: s.n.], 2017. p. 379–386.

KOMER, B.; BERGSTRA, J.; ELIASMITH, C. Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In: *ICML workshop on AutoML*. [S.l.: s.n.], 2014. v. 9, p. 50.

KOTTHOFF, L. et al. Auto-weka: Automatic model selection and hyperparameter optimization in weka. In: HUTTER, F.; KOTTHOFF, L.; VANSCHOREN, J. (Ed.). *Automated Machine Learning: Methods, Systems, Challenges*. Cham, Switzerland: Springer International Publishing, 2019. p. 81–95.

LATTIMORE, T.; SZEPESVÁRI, C. *Bandit algorithms*. [S.l.]: Cambridge University Press, 2020.

LECUN, Y. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, v. 86, n. 11, p. 2278–2324, 1998.

LI, L. et al. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, v. 18, n. 185, p. 1–52, 2018.

NARVEKAR, S. et al. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, v. 21, n. 181, p. 1–50, 2020.

OLSON, R. S. et al. Evaluation of a tree-based pipeline optimization tool for automating data science. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. New York, NY, USA: Association for Computing Machinery, 2016. (GECCO '16), p. 485–492. ISBN 9781450342063. Available at: <https://doi.org/10.1145/2908812.2908918>.

Parmentier, L. et al. Tpot-sh: A faster optimization algorithm to solve the automl problem on large datasets. In: *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. [S.l.: s.n.], 2019. p. 471–478.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, v. 12, n. 10, p. 2825–2830, 2011.

PENTINA, A.; SHARMANSKA, V.; LAMPERT, C. H. Curriculum learning of multiple tasks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2015. p. 5492–5500.

PLATANIOS, E. A. et al. Competence-based curriculum learning for neural machine translation. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. [S.l.: s.n.], 2019. p. 1162–1172.

REYNOLDS, D. Gaussian mixture models. In: ____. *Encyclopedia of Biometrics*. Boston, MA: Springer US, 2009. chap. 1, p. 659–663. ISBN 978-0-387-73003-5.

SÁ, A. G. C. de et al. Recipe: A grammar-based framework for automatically evolving classification pipelines. In: MCDERMOTT, J. et al. (Ed.). *Genetic Programming*. Cham: Springer International Publishing, 2017. p. 246–261. ISBN 978-3-319-55696-3.

SMITH, M. R.; MARTINEZ, T. A comparative evaluation of curriculum learning with filtering and boosting in supervised classification problems. *Computational Intelligence*, v. 32, n. 2, p. 167–195, 2016.

SMITH, M. R.; MARTINEZ, T.; GIRAUD-CARRIER, C. An instance level analysis of data complexity. *Mach. Learn.*, Kluwer Academic Publishers, USA, v. 95, n. 2, p. 225–256, May 2014. ISSN 0885-6125.

SPITKOVSKY, V. I.; ALSHAWI, H.; JURAFSKY, D. From baby steps to leapfrog: How "less is more" in unsupervised dependency parsing. In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Los Angeles, California: Association for Computational Linguistics, 2010. p. 751–759.

Swearingen, T. et al. Atm: A distributed, collaborative, scalable system for automated machine learning. In: *2017 IEEE International Conference on Big Data (Big Data)*. [S.l.: s.n.], 2017. p. 151–162.

SZELISKI, R. *Computer vision: algorithms and applications*. [S.l.]: Springer Science & Business Media, 2010.

TABA, H. *Curriculum development: Theory and practice*. [S.l.]: New York: Harcourt, Brace & World., 1962.

TAY, Y. et al. Simple and effective curriculum pointer-generator networks for reading comprehension over long narratives. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. [S.l.: s.n.], 2019. p. 4922–4931.

THORNTON, C. et al. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2013. (KDD '13), p. 847–855. ISBN 9781450321747. Available at: <https://doi.org/10.1145/2487575.2487629>.

VAPNIK, V. *The nature of statistical learning theory*. [S.l.]: Springer science & business media, 2013.

WANG, X.; CHEN, Y.; ZHU, W. A comprehensive survey on curriculum learning. *arXiv preprint arXiv:2010.13166*, 2020.

Wang, Y. et al. Dynamic curriculum learning for imbalanced data classification. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. [S.l.: s.n.], 2019. p. 5016–5025.

Wei, Y. et al. Stc: A simple to complex framework for weakly-supervised semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 39, n. 11, p. 2314–2320, 2017.

XU, C. et al. Dynamic curriculum learning for low-resource neural machine translation. In: *Proceedings of the 28th International Conference on Computational Linguistics*. Barcelona, Spain (Online): International Committee on Computational Linguistics, 2020. p. 3977–3989. Available at: <https://www.aclweb.org/anthology/2020.coling-main.352>.

YANG, C. et al. Automl pipeline selection: Efficiently navigating the combinatorial space. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2020. (KDD '20), p. 1446–1456. ISBN 9781450379984. Available at: <https://doi.org/10.1145/3394486.3403197>.

YU, D.; DENG, L. *Automatic Speech Recognition: A Deep Learning Approach*. [S.l.]: Springer Publishing Company, Incorporated, 2014. ISBN 1447157788.

ZHANG, X. et al. Curriculum learning for domain adaptation in neural machine translation. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, 2019. p. 1903–1915.

ZÖLLER, M.-A.; HUBER, M. F. Benchmark and survey of automated machine learning frameworks. *arXiv preprint arXiv:1904.12054*, 2019.