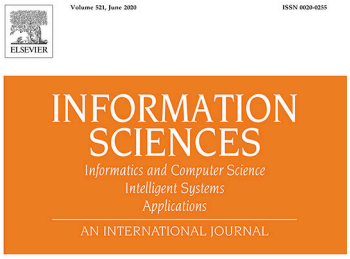
Journal Pre-proofs



Intelligent Ensembling of Auto-ML System Outputs for Solving Classification Problems

Juan Pablo Consuegra-Ayala, Yoan Gutiérrez, Yudivian Almeida-Cruz, Manuel Palomar

PII: S0020-0255(22)00750-2
DOI: https://doi.org/10.1016/j.ins.2022.07.061
Reference: INS 17786

Available online at www.sciencedirect.com
ScienceDirect

To appear in: *Information Sciences*

Received Date: 8 April 2022
Accepted Date: 13 July 2022

Please cite this article as: J.P. Consuegra-Ayala, Y. Gutiérrez, Y. Almeida-Cruz, M. Palomar, Intelligent Ensembling of Auto-ML System Outputs for Solving Classification Problems, *Information Sciences* (2022), doi: https://doi.org/10.1016/j.ins.2022.07.061

Intelligent Ensembling of Auto-ML System Outputs for Solving Classification Problems

Juan Pablo Consuegra-Ayala[a,*], Yoan Gutiérrez[b,c], Yudivian Almeida-Cruz[a], Manuel Palomar[b,c]

[a] *School of Math and Computer Science, University of Havana, La Habana, Cuba, 10200.*
[b] *University Institute for Computing Research (IUII), University of Alicante, Alicante, Spain, 03690.*
[c] *Department of Language and Computing Systems, University of Alicante, Alicante, Spain, 03690.*

Abstract

Automatic Machine Learning (Auto-ML) tools enable the automatic solution of real-world problems through machine learning techniques. These tools tend to be more time consuming than standard machine learning libraries, therefore, exploiting all the available resources to the full is a valuable feature. This paper presents a two-phase optimization system for solving classification problems. The system is designed to produce more robust classifiers by exploiting the different architectures that are generated while solving classification problems with Auto-ML tools, particularly AutoGOAL. In the first phase, the system follows a probabilistic strategy to find the best combination of algorithms and hyperparameters to generate a collection of base models according to certain diversity criteria; and in the second, it follows similar Auto-ML strategies to ensemble those models. The HAHA 2019 challenge corpus and the Adult dataset were used to evaluate the system. The experimental results show that: i) a better solution can be built by ensembling a subset of the already tested models; ii) the performance of ensemble methods depends on the collection of base models used; and, iii) ensuring diversity using the double-fault measure produces better results than the disagreement measure. The source code is available online for the research community.

Keywords: Ensemble Methods, Auto-ML, Grammatical Evolution, Supervised Learning

1. Introduction

In recent years, there has been an increasing interest in applying machine learning techniques to solve several types of complex tasks. Text sentiment analysis, image labeling, and machine translation are just a few well-known examples of these tasks [1, 26, 2, 36]. In general, classification problems are one of the most common problems for which machine learning approaches tend to produce good results. Proof thereof is the fact that machine learning-based systems have been applied even in high-risk decision-making contexts, e.g. hiring processes, applications for bank loans, health insurance, criminal recidivism, among others [8, 17, 24, 37].

*Corresponding author.
Email addresses: jpconsuegra@matcom.uh.cu (Juan Pablo Consuegra-Ayala), ygutierrez@dlsi.ua.es (Yoan Gutiérrez), yudy@matcom.uh.cu (Yudivian Almeida-Cruz), mpalomar@dlsi.ua.es (Manuel Palomar)

Recent advances in Automatic Machine Learning (Auto-ML) have enabled the development of libraries and other tools to effectively find the best combination of algorithms and hyperparameters to solve a problem [19]. Several technologies have been proposed to solve the Auto-ML problem, such as Auto-Weka [35], Auto-sklearn [16], and Auto-Keras [20]. These libraries are alternatives for reducing the time spent by researchers in solving well-studied problems. Even if Auto-ML tools tend to be more time and resource consuming than standard ML libraries, not having to think about which architecture might be the best for solving a problem makes them worth developing. Also, by being applicable to a broad range of problems, learning to use these tools may be easier than learning several independent libraries.

In this context, we find AutoGOAL[1] to be a good example of the potential of Auto-ML libraries. AutoGOAL is a novel Auto-ML system that uses heterogeneous techniques. In contrast with other existing Auto-ML technologies, AutoGOAL can automatically build machine learning pipelines that combine techniques and algorithms from different frameworks, including shallow classifiers, natural language processing tools, and neural networks [13, 14]. AutoGOAL performs a Probabilistic Grammatical Evolution Search to explore the space of available algorithms and their hyperparamenters to build pipelines [15]. At the end of the search, the best performing pipeline found according to the input loss function is given as a solution for the problem.

Even though AutoGOAL is capable of combining algorithms from several different frameworks to build pipelines, it lacks the ability to combine multiple end-to-end pipelines to generate a solution. This means that although multiple hypotheses are tested while AutoGOAL is searching for the best configuration, the final solution comprises a single end-to-end hypothesis and all intermediate pipelines found by AutoGOAL are discarded. However, we consider that these discarded pipelines could be a great source of information, especially if the collection of pipelines is intelligently selected to maximize a desired criteria.

One way of combining several hypothesis to produce a more robust solution, i.e. a solution that generalizes better to unseen inputs, is through ensemble methods. There are several ensemble methods available in the literature, such as *voting* and *weighted-voting* [10], *boosting* [32], and *bagging* [3]. However, depending on the problem, one ensemble method might perform better than the other. This presents the aforementioned challenge of finding the best method to apply to fit a desired criteria, a time-consuming task when done manually. This challenge can be overcome by intelligently exploring the space of ensemble methods, with their respective hyperparameters, to find the best one to apply.

We believe that by ensembling the outputs of several pipelines, the single, biased hypothesis that AutoGOAL originally generates can be improved, and therefore, a more robust solution can be provided. Also, by having a starting collection of pre-trained models, we expect that more complex metrics can be better optimized. Solving problems in two phases to enhance performance has been studied in the literature, providing good results [27, 22]. Additionally, using ensembles to merge multiple snapshots of a single neural network architecture along its optimization path has been proven to be an effective technique referred to as snapshot ensembles [18]. By contrast with snapshot ensembles, this paper studies the ensemble of several Auto-ML generated models instead of using a single neural network architecture.

This paper's objective is to design and validate a system that takes advantage of all the different architectures that are generated while solving classification problems with Auto-ML tools, particularly AutoGOAL, to produce more robust classifiers. We propose a two-phase

[1] autogoal.github.io

optimization system based on Automatic Machine Learning (Auto-ML) for solving classification problems. In the first phase, the system follows a probabilistic strategy to find the best combination of algorithms and hyperparameters to generate a collection of base models according to certain diversity criteria; and in the second, it follows similar Auto-ML strategies to ensemble those models.

The specific contributions of this research are as follows:

- We propose a system based on AutoGOAL and ensemble methods to solve arbitrary classification problems. The improvement in the generalization capability of the solutions is the main advantage of our proposal, which is available online as a *python* project for the research community.
- We study how some diversity measures can impact the performance of an ensemble method when used to ensure diversity between the collection of base classifiers.
- We provide the basis for future studies on unfairness and bias mitigation. Since the optimization process is divided into two phases, a second loss function can be optimized after the first phase by imposing constraints to limit detriment in performance.

The remainder of the paper is organized as follows. Section 2 gives a brief overview of current research in Auto-ML techniques and ensemble methods. Section 3 presents our two-phase optimization system for solving classification problems. Section 4 shows the results of evaluating the system. Section 5 discusses the implications of the results we obtained from the experiments. Finally, Sections 6 presents the conclusions, along with future lines of work.

2. Related Work

This section presents a brief overview of the current research in Auto-ML techniques and ensemble methods. Auto-ML techniques are applied in both phases of our system: first to generate a collection of base models, and then to find the best architecture to ensemble them. Ensemble methods are used to produce a more robust classifier than those originally found with Auto-ML.

2.1. Automatic Machine Learning

Automatic Machine Learning (Auto-ML) is the process of automating the solution of real-world problems through machine learning techniques. The process involves removing the necessity of machine learning human experts to select the appropriate features, workflows, machine learning paradigms, algorithms, and their hyperparameters, to solve a problem [19]. The main advantages of Auto-ML technologies include: (1) reducing the time spent in solving well-studied problems; and, (2) removing the need for expert knowledge. Additionally, these technologies tend to generate simpler solutions that often outperform human-designed solutions.

Several technologies have been proposed to solve the Auto-ML problem. Auto-Weka [35] was one of the first solutions presented. It is based on the Weka software [38], a software made of several visualization tools and algorithms for data analysis and predictive modeling. Auto-Weka solves the Auto-ML problem as a CASH problem as defined next.

Definition 1. *Let* $A = \{A^{(1)}, \ldots, A^{(R)}\}$ *be a set of algorithms, and let the hyperparameters of each algorithm* $A^{(j)}$ *have domain* $\Lambda^{(j)}$*. Further, let* $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ *be a training set which is split into K cross-validation folds* $\{D^{(1)}_{valid}, \ldots, D^{(K)}_{valid}\}$ *and* $\{D^{(1)}_{train}, \ldots, D^{(K)}_{train}\}$ *such that*

$D^{(i)}_{train} = D \backslash D^{(i)}_{valid}$ *for* $i = 1, \ldots, K$. *Finally, let* $\mathcal{L}(A^{(j)}_{\lambda}, D^{(i)}_{train}, D^{(i)}_{valid})$ *denote the loss that algorithm* $A^{(j)}$ *achieves on* $D^{(i)}_{valid}$ *when trained on* $D^{(i)}_{train}$ *with hyperparameters* λ. *Then, the* Combined Algorithm Selection and Hyperparameter Optimization (CASH) *problem is to find the joint algorithm and hyperparameter setting that minimizes this loss:*

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

In the case of cross-validation not being applied, $D^{(*)}_{train}$ *and* $D^{(*)}_{valid}$ *are used to denote a precomputed split of D, such that* $D^{(*)}_{train} = D \backslash D^{(*)}_{valid}$.

Other popular Auto-ML systems are Auto-sklearn [16] and Auto-Keras [20]. These systems are based on the machine learning libraries, Scikit-Learn [29] and Keras [5], respectively. Both systems provide an interface to find the best machine learning architecture to solve a problem. A key difference between them is the way their search spaces are defined. While Auto-sklearn explores a conditional search space, i.e., a space with some hyperparameters conditioned to others, Auto-Keras performs a Neural Architecture Search (NAS [12]), which involves exploring hierarchic spaces of arbitrary complexity.

AutoGOAL [14, 15] is one of the most recent contributions to the field of Auto-ML. AutoGOAL is a system that uses heterogeneous techniques to solve the CASH problem. The core of AutoGOAL is on its customizable space of pipelines and its pool of search algorithms, which are used to find the best configuration to solve a problem. Each pipeline is defined as a set of interconnected algorithms that maps a predefined input to its corresponding output. The space of pipelines comprises not only the set of algorithms but also their hyperparameters.

Several sources of algorithms are included in AutoGOAL's space, such as Scikit-Learn [29], NLTK [25], Keras [5], and Pytorch [28]. However, AutoGOAL lacks the ability to combine multiple end-to-end pipelines to generate a solution. This limitation can be overcome by using ensemble methods.

2.2. Ensemble Methods

Ensemble methods are designed to address the low bias / high variance problem exhibited by most machine learning models, making them suitable for producing more robust classifications [31]. An ensemble model is made of several low bias models whose predictions are combined to produce a final prediction. The main assumption is that the combination of the low-level predictions will produce an output with a lower variance while keeping the bias low. Having a diverse set of low-level models is a key feature to achieve this [39].

The multi-hypothesis nature of ensemble models ensures that, if fine-tuned, they will perform better than any of the individual models in the general case. This also allows them to estimate the degree of confidence or quality of the predictions they output. Classic ensemble techniques include *voting* and *weighted-voting* [10], *boosting* [32], and *bagging* [3].

In a classification problem, *voting* produces as output the label that achieved the majority of votes, treating each low-level model prediction as a vote. *Weighted-voting* works just like voting, but each low-level model is assigned a weight that indicates the importance of its votes. The label with the highest cumulative score is returned as output. *Boosting* runs an iterative process where models are trained sequentially, each one trying to improve its performance in the training examples where the previous models had performed the worst. During this process, each submodel is also assigned a score that weights the importance of its prediction. *Bagging* trains each

sub-model in a different selection (with replacement) of the original training examples. This way, a model with a high variance should produce trained models with high diversity. Alternatively, *feature bagging* works similarly by selecting a subset of the features instead of the training examples, causing correlated features to be analyzed separately in some sub-models.

The ability of ensemble methods to build more robust solutions has made them suitable for several applications. The health domain is one example where these methods have been applied with great success. For example, a two-stage ensemble-based strategy is proposed in the literature for categorizing glioma and non-glioma cases in the first stage, and in the second stage for grade categorization of the identified glioma cases [22]. A multi-model and multi-slice ensemble learning architecture based on 2D convolutional neural networks is proposed for Alzheimer's disease diagnosis [23]. A hybrid application of ensemble methods with multiple neural networks in a reinforcement learning framework is presented to enable COVID-19 infection forecasting with high accuracy [21]. Similarly, an ensemble-based multi-criteria decision making method is proposed for detecting COVID-19 from cough sounds [6]. Finally, ensemble methods are applied in combination with transfer learning techniques for COVID-19 detection from chest CT-scans [33]; for this, various pre-trained models are fine-tuned and an ensemble classifier makes the final prediction.

A simple but powerful technique in the context of ensemble methods is the so-called snapshot ensemble [18]. This technique generates multiple base predictors by training a single neural network while making it repeatedly and rapidly converge to several local minima along its optimization path and saving the model parameters. All the neural networks are then ensembled to produce the final predictor. Snapshot ensembles are more robust and accurate than individual networks given their ensemble nature, at no additional training costs. They have been applied to food image analysis with good results [34].

The application of ensemble methods in information extraction problems, such as entity recognition and relation extraction tasks, has been less studied in the literature. An optimization system based on ensemble methods was proposed as a better alternative to classic corpus bootstrapping for corpora extension [7]. That system reduces the bias towards a particular architecture or set of rules and provides richer information on the quality of the final corpus. The system finds the best ensemble configuration E^* by performing a Probabilistic Grammatical Evolution Search.

This work will study the usage of ensemble methods to overcome AutoGOAL's limitations when it comes to combining multiple end-to-end pipelines to generate a solution. It is expected to obtain more robust classifiers in this way.

3. Method

This section presents our two-phase optimization system for solving classification problems. Section 3.1 provides a general overview of the system. Section 3.2 explains how the initial set of base models is obtained. Section 3.3 describes how the system searches for the best ensemble configuration. Finally, Section 3.4 presents some mock ensemble models, which can be used to estimate the best performance obtainable by ensembling the found set of base models.

3.1. Overview

The system takes as input a dataset $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ and a loss functions $\mathcal{L}$. The goal of the system is to produce a classification model whose performance is optimized against $\mathcal{L}$.

The system is divided into two main phases. The first one is responsible for generating a collection of models, each one called a *base model*. This collection is built by optimizing a loss

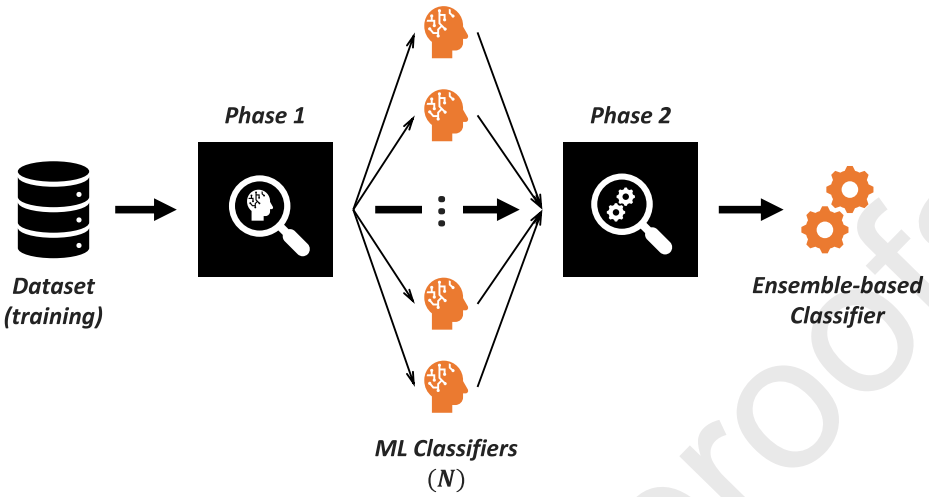


Figure 1: Overview of our two-phase optimization system for solving classification problems. In the first phase, the system generates a collection of base models using Auto-ML strategies. In the second phase, it follows similar Auto-ML strategies to ensemble the collection of base models.

function $\mathcal{L}^1$, while ensuring diversity across the whole population. The second phase is responsible for producing a classification model. This model is generated by ensembling the collection of base models to optimize its performance according to $\mathcal{L}^2$. In the context of this paper, we will take $\mathcal{L} = \mathcal{L}^1 = \mathcal{L}^2$. Therefore, the system is intended to achieve the best performance possible on $\mathcal{L}$ by improving the best performance achieved by the base models. Figure 1 summarizes the architecture of the system. Figure 2 and 3 provide an overview of the first and second phase, respectively.

Sections 3.2 and 3.3 provide further details on the first and second phase, respectively.

3.2. Base Models Generation

In this phase, the system is given the task of generating N models to fit D according to loss $\mathcal{L}$. Figure 2 provides an overview of this phase.

Definition 1 is modified to search for a collection of models instead of a single model, subject to a diversity measure $\mathcal{D}$. That is, we want to find a collection of base models (models that optimize the performance on dataset D according to loss $\mathcal{L}$) while guaranteeing some differences between their hypotheses using measure $\mathcal{D}$. Ensuring diversity in the collection of base models is important because ensemble methods are not able to improve performance if all of the base models have exactly the same hypothesis, i.e., all of them make the same predictions.

The procedure applied to generate the collection of base models is summarized by function GenerateBaseModels. The space of algorithms and hyperparameters is explored using a preselected search strategy. All of this is captured by the **explore** function. After scoring the sampled architectures and estimating the diversity between the currently selected models and the new generation of models, the collection of base models is updated to fit the N capacity. All of this is captured by the **reselect** function.

To intelligently **explore** the space of algorithms and hyperparameters, i.e., to solve the modified CASH problem, we use AutoGOAL's Probabilistic Grammatical Evolution Search (`PESearch`)

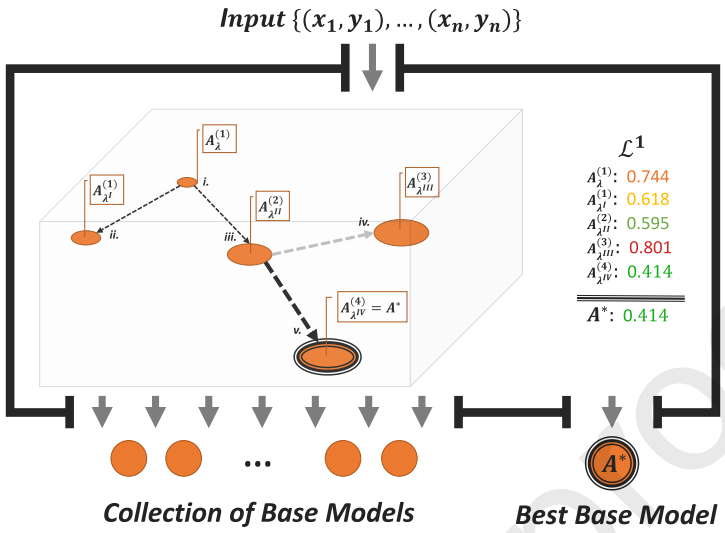


Figure 2: First phase: generation of base models. The goal is not only to find the best model but also to produce a collection of them to be used as base models for the ensembling phase.

Function GenerateBaseModels($N, D, A, \Lambda, \mathcal{L}, \mathcal{D}$)

set base_models $\leftarrow \emptyset$
for generation $\in$ **explore**(A, Λ) **do**
  scores $\leftarrow \emptyset$
  for $A_\lambda^{(j)} \in$ generation **do**
    scores$^{(j)} \leftarrow \frac{1}{K} \sum_{i=1}^{K} \mathcal{L}(A_\lambda^{(j)}, D_{train}^{(i)}, D_{valid}^{(i)})$
  end
  diversity $\leftarrow \mathcal{D}$(base_models $\frown$ generation, D)
  base_models $\leftarrow$ **reselect**(base_models $\frown$ generation, scores, diversity, N)
end
return base_models

implementation. AutoGOAL refers to the models it builds as pipelines, given that each one of them is made of interconnected algorithms. The search starts with a random sampling strategy, but as it evaluates more pipelines, it modifies a probabilistic sampling model so that pipelines similar to the best ones found are more commonly sampled. The space of algorithms and hyperparameters is also set to AutoGOAL's default, which includes several classic `scikit-learn` machine learning algorithms. For more details about AutoGOAL's search implementation, see [15].

To **reselect** the collection of base models, i.e. the collection of AutoGOAL's pipelines, a greedy approach is studied in this paper. Function reselect summarizes the proposed strategy. The algorithm always includes the best-performing model according to $\mathcal{L}$ in the selection. Each subsequent iteration adds the unselected model that maximizes diversity with respect to all previously selected models. This greedy approach does not guarantee that the final collection achieves the best possible diversity with respect to $\mathcal{D}$. Accuracy is also not taken into account, except to select the best-performing model. Future improvements to solve this task are discussed

in Section 6.1

Function reselect(M, $scores$, $diversity$, N)

set $R \leftarrow \emptyset$
set $R^{(0)} \leftarrow \underset{m^{(j)} \in M}{argmin}\ scores^{(j)}$
for $r \leftarrow 1$ **to** N **do**
  $R^{(r)} \leftarrow \underset{(m^{(j)} \in M \setminus R)}{argmax} \sum_{(m^{(i)} \in R)} diversity^{(i,j)}$
end
return $R^{(0)} \cdots R^{(N)}$

Three reference **reselect** methods for base models were implemented for comparison with the *double-fault* and *disagreement* diversity measures, which are presented in Section 3.2.1.

- **Shuffle**: The collection of base models is built by randomly shuffling the current selection of base models and the new ones found. The first N models after shuffling are selected for the next generation.
- **Arbitrary**: The collection of base models is built in the same way as the *shuffle* strategy, but the best performing model is always included in the collection of selected base models.
- **Best**: The collection of base models is built by selecting the best performing models among the previously selected base models and the new ones found.

Next, Section 3.2.1 provides some details on the diversity measures studied in this work.

3.2.1. Diversity Measures

Two measures were implemented to estimate the diversity of a given collection of base models. Both of them precompute a misclassification matrix, which is then used to compute pairwise measures between the base models. The misclassification matrix is computed as follows.

$$M_{i,j} = \begin{cases} 1 & \text{if } \textbf{\textit{model j}} \text{ correctly classifies } \textbf{\textit{sample i}}\ (D_{valid}) \\ -1 & \text{otherwise} \end{cases}$$

The following measures are computed between pairs of base models to estimate how different are their hypotheses, and therefore the diversity of collection including both of them at the same time.

Disagreement. This measures the frequency with which one of the models fails when the other does not, and vice versa. The higher this measure, the more different they are.

$$\text{disagreement}(m^{(a)}, m^{(b)}) = \frac{|\{\, M_{i,a} \neq M_{i,b} \mid s^{(i)} \in D^{(*)}_{valid} \,\}|}{|D^{(*)}_{valid}|}$$

Double Fault. This measures how often both models fail at the same time. The higher this measure, the greater their difference.

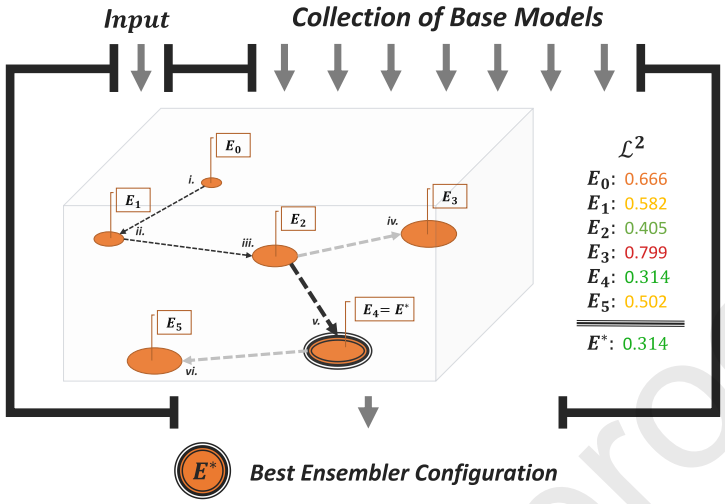


Figure 3: Second phase: search for the best ensemble configuration. The goal is to find the best method to ensemble the collection of base models.

$$\text{double-fault}(m^{(a)}, m^{(b)}) = 1 - \frac{|\{\, M_{i,a} = M_{i,b} = -1 \mid s^{(i)} \in D^{(*)}_{valid} \,\}|}{|D^{(*)}_{valid}|}$$

3.3. *Intelligent Ensembling of Models*

In this phase, the system is given the task of combining the predictions of N models to fit D according to loss $\mathcal{L}$. Figure 3 provides an overview of this phase.

The system solves once again a CASH problem as presented in Definition 1. Instead of working directly on $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, this time the system works on $D^e = \{(y_1^{(*)}, y_1), \ldots, (y_n^{(*)}, y_n)\}$, where $y_i^{(*)} = [y_i^{(0)}, \ldots, y_i^{(N)}]$ and each $y_i^{(j)}$ is the output of base model j for sample i, $i \in [1 \ldots n]$, $j \in [1 \ldots N]$. In other words, the system is asked to find the best combination E^* of algorithms and their hyperparameters to ensemble the outputs of the base models. Particularly, a custom space of algorithms specialized in ensemble strategies and their hyperparameters is used in this phase.

The ensemble strategies adopted are presented next.

Voting Classifier Assigns the most common label among the ones predicted by the base models. In the case of a tie, it selects the label predicted by the most accurate base model among them.

Overfitted Voting Classifier Assigns to each combination of base models' outputs the label that ensures the best performance in D^e_{train}. In prediction time, if a previously unseen combination is found, it selects the label predicted by the most accurate base model (ignoring whether it was the most voted label).

ML Voting Classifier Fits a machine learning model on D^e_{train} to optimize $\mathcal{L}$. The machine learning model's architecture is taken from AutoGOAL's default pool of algorithms.

All of the ensemble strategies share the following core of hyperparameters. The *ML Voting Classifier* includes an additional set of hyperparamenters depending on the selected architecture.

- **n-classifiers**: *how many base classifiers will be used to predict?* This is an integer value that ranges from 2 to N.
- **selection-names**: *which base classifiers will be used to predict?* This is a list of identifiers whose length is equal to **n-classifiers**.

The space of algorithms and hyperparameters is explored using AutoGOAL's Probabilistic Grammatical Evolution Search (`PESearch`) implementation. The search algorithm was modified to allow the specification of constraints on the solution. For example, the search can be restricted to enforce that any model M found in this second phase has to perform at least as good as those found in the first phase, i.e., $\Delta\mathcal{L}^1(M, D^{(i)}_{train}, D^{(i)}_{valid}) \leq \epsilon$ for any given ϵ. Since $\mathcal{L} = \mathcal{L}^1 = \mathcal{L}^2$ in the context of this paper, we will not study the impact of such rules here.

3.4. Oracle Ensembles

To estimate the best performance obtainable by ensembling the found set of base models, two measures are proposed next. These measures estimate the performance achieved by mock ensemble models, i.e., ensemble models that know the correct outcomes in advance, and therefore models that have no real use outside of being used for comparison purposes. The ensemble models are referred to as *Optimistic Oracle* and *Overfitted Oracle*.

Optimistic Oracle. This oracle model returns the correct label if at least one of the base models returned that label. The only way for this model to fail is if none of the base models were able to suggest the correct label.

$$O^{(i)}_{optimistic}(M) = \begin{cases} y_i & \text{if } y_i \in \{\, y_i^{(j)} \mid m^{(j)} \in M \,\} \\ y_i^{(0)} & \text{otherwise} \end{cases}$$

The score achieved by this ensemble reflects the degree of coverage of the base models in the evaluated collection, i.e., how many samples are correctly predicted by at least one base classifier.

Overfitted Oracle. This oracle model computes all combinations of the base models' outputs and assigns to each combination the most frequent label found across the corresponding set of correct labels. This model fails when the same combination of base models' outputs has to match different labels for all sample inputs to be classified correctly.

$$O^{(i)}_{overfitted}(M) = \textbf{max\_count}\left(\left\{ y_k \middle| (x_k, y_k) \in D, \bigwedge_{m^{(j)} \in M} y_k^{(j)} = y_i^{(j)} \right\}\right)$$

Function **max_count** returns the most frequent item from the collection (in the case of a tie, it always returns the first one found).

The score achieved by this ensemble provides an upper bound of the maximum performance that can be obtained by using a **consistent** set of rules for ensembling the collection of base models.

4. Results

In this section, we evaluate the ability of our two-phase optimization system to solve a classification problem and to improve the performance of a collection of base models by ensembling them. The results are compared between the final model E^* generated by our system and the best model A^* found by AutoGOAL in the base model generation phase. Section 4.1 describes the experimental setup developed to test the system. Section 4.2 shows the results obtained by the system on the HAHA 2019 challenge. Additionally, Section 4.3 presents the evaluation of the system on the Adult dataset. A deeper analysis on the reported results is provided in Section 5.

4.1. Experimental Setup

In all the experiments, the system was setup to allow each phase to run for at most 10 000 iterations or one hour. AutoGOAL's search parameters were setup as follows: `popsize=50`, `selection=10`, `cross_validation_steps=3`, and `validation_split=0.3`.

Due to infrastructure limitations, AutoGOAL's deep learning algorithms were excluded from the pool of available algorithms, that is, Keras and BERT-based pipelines were excluded and mainly *ScikitLearn* pipelines were used. AutoGOAL is a tool still in development and the last stable version available at the moment of setting up the experiments (`v0.6.0`) did not fully support deep learning algorithms in all computing architectures.

Not including deep learning algorithms in the experimental setup may have a negative impact on the maximum performance achieved by the system. That is, the scored performance of the system while solving a problem cannot be directly compared to other reported solutions that use deep learning techniques. However, the lack of such algorithms should not affect the capacity of the system to improve the performance of the pipelines found in the first phase, i.e. the base models. Therefore, if the system improves the performance of the base models, then the top performance is expected to further increase once the deep learning architectures become compatible. This is expected to happen not only because the base models will now perform better, but also because more powerful ensemble architectures will be seamlessly added at the same time. AutoGOAL has already be proven to achieve competitive results when set up properly [13]

4.1.1. Library

The system proposed in this paper is part of our in-development `BFair`[2] library. The library is meant to address unfairness issues that arise from training machine learning models in data that exhibit human biases. All resources evaluated in this paper are publicly available at `BFair`.

4.1.2. Hardware

Experiments were run in a machine with the following stats: 8 core Intel Core i9-9900K (-MT-MCP-) CPU, speed/max: 3651/5000 MHz, cache: 16384 KB, and RAM: 64 GB.

4.2. Evaluation in the HAHA Challenge

The system described in this paper was applied and evaluated in the HAHA 2019 (Humor Analysis based on Human Annotation) challenge corpus, from IberLEF 2019 [4]. The corpus contains 30 000 manually classified tweets in Spanish, of which 24 000 are for training and 6 000 for testing. Each tweet is classified as humorous or not. For the experiments reported next, only

[2] https://github.com/bfair-ml/bfair

the training collection is used to tune the system. The testing collection is used to report unbiased scores.

The training collection was divided into three cross-validation folds for the first phase, using a 30% validation split. Additionally, it was also divided into two pre-established training and valid collections using a 30% validation split. These pre-established collections were the ones used to measure diversity in the first phase and the ensembling performance in the second phase. The original testing collection of the corpus was used to report the final testing results. This collection was not used during training.

Tables 1, 2, 3, and 4 summarize the raw results obtained by the system while setting the number of base classifiers to 5, 20, 50, and 100, respectively. The scoring metric used is the F_1 measure as suggested in the challenge description. Five population control strategies were tested for comparison purposes, of which two are the diversification strategies discussed in Section 3.2.1, and the other three are the reference **reselect** methods presented in Section 3.2. Each table shows the F_1 measure achieved by: (1) the optimistic and overfitted oracles (Section 3.4); (2) the model obtained by ensembling (Section 3.3); and, (3) the best base model found (Section 3.3). Additionally, the type of ensemble algorithm used by the best ensemble model found is added at the end of each table.

strategies / evaluations	shuffle	arbitrary	best	disagreement	double fault
optimistic oracle (F_1)	0.996	0.917	0.893	1.000	0.970
overfitted oracle (F_1)	0.715	0.711	0.744	0.748	0.754
A^* (F_1, training)	0.989	0.973	0.945	0.846	0.876
A^* (F_1, testing)	0.690	0.711	0.722	0.749	0.757
E^* (F_1, training)	0.913	0.961	0.917	0.846	0.941
E^* (F_1, testing)	0.731	0.726	0.755	0.749	0.761
ensemble type	voting	learning	learning	voting	voting

Table 1: HAHA 2019. Maximum number of classifiers set to 5. Each column shows the result obtained by using the corresponding base model selection strategy. A^* and E^* stand for the best performing base model and the best ensemble configuration found, respectively. Ensemble types: *voting*, *overfit*, and *learning*, stand for *Voting Classifier*, *Overfitted Voting Classifier*, and *ML Voting Classifier*, respectively.

Section 5 provides some insights on the raw results reported in Section 4.2. The best results were obtained when the maximum number of classifiers was between 20 and 50 and the base model selection strategy was double-fault. Next, Section 4.3 evaluates the performance of that same system configuration in another dataset.

4.3. Evaluation in the Adult dataset

The system presented was also evaluated in the Adult dataset from the UCI repository [11]. This time, the system was only configured to search for a collection with a maximum of 50 base models and using the double-fault strategy to select the base models. That system configuration was identified as the best one according to the experiments presented in Section 4.2, as discussed in Section 5.

Table 5 summarizes the results obtained by running the system on the Adult dataset. The mean error and standard deviation across 20 independent executions is measured. The scoring metric

evaluations \ strategies	shuffle	arbitrary	best	disagreement	double fault
optimistic oracle (F_1)	1.000	1.000	0.936	1.000	0.998
overfitted oracle (F_1)	0.729	0.771	0.831	0.735	0.889
A^* (F_1, training)	0.876	0.901	0.855	0.875	0.856
A^* (F_1, testing)	0.705	0.721	0.759	0.732	0.755
E^* (F_1, training)	0.870	0.930	0.883	0.875	0.942
E^* (F_1, testing)	0.719	0.740	0.765	0.732	0.767
ensemble type	learning	overfit	voting	learning	voting

Table 2: HAHA 2019. Maximum number of classifiers set to 20. Each column shows the result obtained by using the corresponding base model selection strategy. A^* and E^* stand for the best performing base model and the best ensemble configuration found, respectively. Ensemble types: *voting*, *overfit*, and *learning*, stand for *Voting Classifier*, *Overfitted Voting Classifier*, and *ML Voting Classifier*, respectively.

evaluations \ strategies	shuffle	arbitrary	best	disagreement	double fault
optimistic oracle (F_1)	1.000	1.000	0.978	1.000	1.000
overfitted oracle (F_1)	0.953	0.950	0.927	0.996	0.969
A^* (F_1, training)	1.000	0.928	0.877	0.857	0.913
A^* (F_1, testing)	0.639	0.720	0.720	0.750	0.749
E^* (F_1, training)	1.000	0.924	0.921	1.000	0.923
E^* (F_1, testing)	0.741	0.736	0.731	0.747	0.756
ensemble type	overfit	learning	voting	overfit	voting

Table 3: HAHA 2019. Maximum number of classifiers set to 50. Each column shows the result obtained by using the corresponding base model selection strategy. A^* and E^* stand for the best performing base model and the best ensemble configuration found, respectively. Ensemble types: *voting*, *overfit*, and *learning*, stand for *Voting Classifier*, *Overfitted Voting Classifier*, and *ML Voting Classifier*, respectively.

used was the accuracy score, as commonly adopted in other research. The maximum number of classifiers was set to 50. The base model selection strategy was set to *double-fault*. The rest of the experimental setup was the same as described in Section 4.1.

The results show that the ensemble optimization is capable of producing solutions that generalize better than the base models. The same pre-selected configuration of the system was able to improve the results in all the evaluation scenarios. This fact shows that our system can be used to enhance the solutions found by AutoGOAL. It is worth noting that the ensembling phase was able to achieve high performance even in runs where the base model generation phase found worst performing models than the average. For example, the ensembling phase was able to produce a final model with 0.825 accuracy in the Adult test collection even when the best base model initially found by AutoGOAL achieved 0.789 accuracy for that same collection.

evaluations \ strategies	shuffle	arbitrary	best	disagreement	double fault
optimistic oracle (F_1)	1.000	1.000	0.997	1.000	1.000
overfitted oracle (F_1)	0.988	0.992	0.984	0.998	0.988
A^* (F_1, training)	0.998	0.856	0.853	0.902	0.920
A^* (F_1, testing)	0.683	0.748	0.759	0.745	0.750
E^* (F_1, training)	1.000	1.000	1.000	0.970	0.940
E^* (F_1, testing)	0.756	0.745	0.761	0.728	0.743
ensemble type	overfit	overfit	overfit	learning	voting

Table 4: HAHA 2019. Maximum number of classifiers set to 100. Each column shows the result obtained by using the corresponding base model selection strategy. A^* and E^* stand for the best performing base model and the best ensemble configuration found, respectively. Ensemble types: *voting*, *overfit*, and *learning*, stand for *Voting Classifier*, *Overfitted Voting Classifier*, and *ML Voting Classifier*, respectively.

Accuracy Score	
evaluations	adult
A^* (training)	0.839 (±0.069)
E^* (training)	**0.860** (±0.060)
delta (training)	0.021
A^* (testing)	0.823 (±0.019)
E^* (testing)	**0.831** (±0.013)
delta (testing)	0.008

Table 5: Evaluation in the Adult dataset in terms of mean accuracy score and its standard deviation. Maximum number of classifiers set to 50. The base model selection strategy was set to *double-fault*. A^* and E^* stand for the best performing base model and the best ensemble configuration found, respectively. Delta (training and testing) stands for the difference between the mean accuracy score of E^* and A^* ($\Delta = E^* - A^*$).

5. Discussion

Some interesting patterns can be observed by analysing the performance of the oracle ensembles for different reselection strategies and number of base classifiers. Table 6 summarizes the score of the optimistic and overfitted oracles on the test collection. Some patterns are presented next.

- The disagreement measure ensures the maximum coverage in the trained set regardless of the number of base classifiers selected. This makes sense since the disagreement measure rewards the selection of models which have conflicting predictions. By looking at the performance of the overfitted oracle, it can be noticed that even though it provides the maximum coverage, there is no set of consistent rules that can be applied to exploit that coverage.

- The double-fault measure provides the most consistent score for both the optimistic and overfitted oracles, in particular when the number of base classifiers is low. This suggests that a double-fault-based diversification strategy may provide the best performance for subsequent ensembling, since it provides a higher score while using a consistent set of rules.

- The coverage exhibited by the oracle ensembles increases seamlessly as the number of base classifiers increases independently of the reselection strategy. This makes sense since having a larger pool of voters increases the probability of finding one that correctly classifies the most conflicting samples if there is certain diversity between the voters. This idea is supported by the fact that the reselection strategy that greedily selects only the best performing models is the one that achieves the lowest coverage (according to the optimistic ensemble).

Optimistic Oracle (F_1)					
n-classifiers	shuffle	arbitrary	best	disagreement	double fault
5	0.996	0.917	0.893	1.000	0.970
20	1.000	1.000	0.936	1.000	0.998
50	1.000	1.000	0.978	1.000	1.000
100	1.000	1.000	0.997	1.000	1.000
Overfitted Oracle (F_1)					
n-classifiers	shuffle	arbitrary	best	disagreement	double fault
5	0.715	0.711	0.744	0.748	0.754
20	0.729	0.771	0.831	0.735	0.889
50	0.953	0.950	0.927	0.996	0.969
100	0.988	0.992	0.984	0.998	0.988

Table 6: Summary of the oracle measures for each base model selection strategy as the maximum number of base classifiers increases.

Table 7 summarizes the difference between the performance achieved by the best ensemble found E^* and the best base model used A^*, both in the training and testing collections. It is noticeable that the ensemble optimization is capable of outperforming its base models in the training collection, even though some strategies require a larger number of base models for being able to do so. The double-fault strategy is capable of improving the performance even with a low number of base classifiers. This makes sense since that strategy penalizes the system from building a collection in which all the base models fail in the same samples. Therefore, this results in a collection of base models which fix each other's failures. Such behaviour is different from the one exhibited by the disagreement strategy, which rewards the base models for having different predictions independently of whether they were right or wrong. This may lead to situations in which it is difficult for the ensemble to decide which base model to trust, especially if the number of base models is low.

More importantly, Table 7 also shows that the ensemble optimization is capable of producing ensembles that generalize better than their base models, i.e., the ensembles perform better in the testing collection. This is particularly true when the number of base classifiers is not too large. As

the number of base models increases, so does the number of different combinations, and therefore, the ensemble is more prone to overfitting the training set.

$(E^* - A^*)$ @ training					
n-classifiers	shuffle	arbitrary	best	disagreement	double fault
5	-0.076	-0.012	-0.027	0.000	0.065
20	-0.006	0.030	0.029	0.000	0.086
50	0.000	-0.004	0.044	0.143	0.010
100	0.002	0.144	0.147	0.069	0.020
$(E^* - A^*)$ @ testing					
n-classifiers	shuffle	arbitrary	best	disagreement	double fault
5	0.040	0.015	0.033	0.000	0.004
20	0.014	0.018	0.005	0.000	0.012
50	0.102	0.015	0.012	-0.003	0.006
100	0.073	-0.003	0.002	-0.017	-0.007

Table 7: Summary of the difference in performance between the best base model found (A^*) and the corresponding ensemble (E^*), for each base model selection strategy as the maximum number of classifiers increases.

Some other highlights from Table 7 are summarized next.

- Diversity measures, i.e. disagreement and double-fault measures, have a greater impact on the performance of the ensemble in the training set when the number of base classifiers is low. This makes sense since the fewer the number of models in the collection of base models, the harder it is to find a "good" selection by chance. However, diversity measures do not seem to have an impact on the generalization capabilities.

- The disagreement strategy struggles to improve performance, both in the training and testing collections. This is particularly true when the number of base classifiers is low. This strategy does not penalize models for making a bad prediction and instead rewards them if their predictions conflict with those of other classifiers. Thus, it is usual for the ensemble to merely decide to trust the prediction of a single base classifier, and thereby the ensemble produces the same results as its best base model. This shows that achieving the maximum optimistic oracle coverage, as the disagreement strategy does, is not necessarily useful.

Table 8 summarizes the performance obtained by the ensemble E^* as the number of base classifiers and reselection strategies change. The double-fault strategy always achieves the best results, except in the last scenario in which the number of base classifiers is the highest. In that case, even when the ensemble outperformed the base model in the training collection, the high number of voters may have affected its generalization capabilities since the space of possible combination of votes is larger.

As expected, the lack of deep learning algorithms in the pool of available methods had a negative impact on the top performance achieved. Our best performing architecture achieves 0.767 F_1 while the best solution reported by AutoGOAL in the HAHA 2019 challenge is 0.789 [13], which uses deep learning algorithms —a pre-processing strategy using a BERT transformer [9]

E^* @ training					
n-classifiers	shuffle	arbitrary	best	disagreement	double fault
5	0.913	**0.961**	0.917	0.846	0.941
20	0.870	0.930	0.883	0.875	**0.942**
50	**1.000**	0.924	0.921	**1.000**	0.923
100	**1.000**	**1.000**	**1.000**	0.970	0.940
E^* @ testing					
n-classifiers	shuffle	arbitrary	best	disagreement	double fault
5	0.731	0.726	0.755	0.749	**0.761**
20	0.719	0.740	0.765	0.732	**0.767**
50	0.741	0.736	0.731	0.747	**0.756**
100	0.756	0.745	**0.761**	0.728	0.743

Table 8: Summary of the performance obtained by the ensemble (E^*) in the training and testing collections, for each base model selection strategy as the maximum number of classifiers increases.

and a neural network with 2 recurrent nodes (one BiLSTM layer and one LSTM layer) followed by 2 time-distributed dense layers—. In comparison, our best performing architecture consists of a voting classifier that ensembles a set of simple *Scikit-Learn* models — a pre-processing strategy using tokenization (*BlanklineTokenizer*, *TreebankWordTokenizer*, etc.) and vectorization algorithms (*TfidfVectorizer*, *CountVectorizer*, etc.) followed by a classification layer (*Perceptron*, *LinearSVC*, *NearestCentroid*, *MultinomialNB*, etc.) —. Despite using a simpler architecture, competitive results are achieved. Moreover, as shown in Table 7, the ensemble optimization is capable of producing ensembles that generalize better than their base models. Therefore, we expect the performance of the model output by our system to improve once more powerful algorithms become compatible.

Tables 9 and 10 summarize the different types of ensembles that were found as optimal in each experiment. The ensemble technique "ML Voting Classifier" does not seem to be used by any reselection strategy consistently. The ensemble technique "Voting Classifier" is being used consistently in double-fault based strategies, and also seems to be used more frequently when the number of base classifiers is low. Moreover, the ensemble technique "Overfitted Voting Classifier" seems to be used more when the number of base classifiers is high.

n-classifiers	shuffle	arbitrary	best	disagreement	double fault
5	voting	learning	learning	voting	voting
20	learning	overfit	voting	learning	voting
50	overfit	learning	voting	overfit	voting
100	overfit	overfit	overfit	learning	voting

Table 9: Summary of the ensemble strategy used by the best ensemble configuration found (E^*) for each base model selection strategy as the maximum number of classifiers increases. Ensemble types: *voting*, *overfit*, and *learning*, stand for *Voting Classifier*, *Overfitted Voting Classifier*, and *ML Voting Classifier*, respectively.

n-classifiers	voting	overfitted	learning
5	3	0	2
20	2	1	2
50	2	2	1
100	1	3	1

Table 10: Frequency of use of each ensemble technique. Columns *voting*, *overfit*, and *learning* stand for *Voting Classifier*, *Overfitted Voting Classifier*, and *ML Voting Classifier*, respectively.

To conclude, the system was shown to provide the most promising results when configured to search for a collection with a maximum of 20 or 50 base models and using the double-fault strategy to select the base models.

6. Conclusion

This paper presents a two-phase optimization system for solving classification problems. The goal of the system is to take advantage of all the different architectures that are tested while solving an Auto-ML problem to produce a more robust solution. The evaluation in the HAHA 2019 challenge corpus and the Adult dataset proved that by setting the system to use the double-fault diversification strategy and a maximum of 50 base classifiers, the performance on the test collection improves. Hence, our work confirms the hypothesis that a better solution can be built by ensembling a subset of the models that are generated while solving the Auto-ML problem.

Two diversity measures taken from the literature were studied: the disagreement and double-fault measures. To quantify the quality of the collections built by using those measures, two additional measures, based on oracle ensembles, were presented in this paper. The results provide some insights on how the selection of base models influences the performance of ensemble techniques. The disagreement measure ensures the maximum coverage over the samples, however, the collections built using this strategy do not necessarily provide consistent outputs. The double-fault measure provides the best results in general.

The main findings of this paper are summarized next.

- By including an additional layer to ensemble some of the models previously found while solving a problem with AutoGOAL, the generalization capabilities of the final model was shown to improve even further.
- The performance of ensemble methods was shown to depend on the collection of base models used.
- Ensuring diversity through disagreement was shown to have a negative impact. On the other hand, the double-fault measure produced the best results, especially in contexts where a small number of base models is required.

6.1. Future Work

The next step of this research will be to apply our proposed method to the present day crucial area of unfairness and bias mitigation. We will study how to take advantage of the two-phase nature of the system to optimize two metrics instead of one. This would open the possibility of

producing models that fit multiple criteria, e.g., fairness metrics on top of the accuracy or precision of the output. One first approach would be to study if the current constraint-based strategy is enough to allow $\mathcal{L}^2$ to be different to $\mathcal{L}^1$, that is, searching for the best configuration to ensemble the collection of base models according to $\mathcal{L}^2$, while ensuring a detriment of $\mathcal{L}^1$ lower than a certain ϵ. Some studies have already shown that by changing the hyperparameters of a model, the fairness in classification can be boosted [30]. Exploiting the pool of available algorithms and hyperparameters in AutoGOAL to find a fair solution to an arbitrary classification problem is the main goal of our `BFair` library .

Additionally, in the future we will study some alternatives to enhance the system's ability to boost the performance of the base models. Improving the way in which the models are ranked according to the diversity measure may result in a better collection of base models. Using a clustering algorithm instead of a greedy method might have a positive impact in that sense. Increasing the set of features provided to the ensembling phase about the base models might be another alternative to consider. For example, including as a feature the performance of each base model on each target class may enable the ensemble to make a better decision regarding which group of base models to trust most in each situation.

Acknowledgments

Funding: This research has been partially funded by the University of Alicante and the University of Havana, the Generalitat Valenciana (*Conselleria d'Educació, Investigació, Cultura i Esport*) and the Spanish Government through the projects LIVING-LANG (RTI2018-094653-B-C22), INTEGER (RTI2018-094649-B-I00) and SIIA (PROMETEO/2018/089, PROMETEU/2018/089). Moreover, it has been backed by the work of both COST Actions: CA19134 - "Distributed Knowledge Graphs" and CA19142 - "Leading Platform for European Citizens, Industries, Academia and Policymakers in Media Accessibility".

References

[1] B. Agarwal and N. Mittal. Machine learning approach for sentiment analysis. In *Prominent feature extraction for sentiment analysis*, pages 21–45. Springer, 2016.

[2] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

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

[4] L. Chiruzzo, S. Castro, M. Etcheverry, D. Garat, J. J. Prada, and A. Rosá. Overview of haha at iberlef 2019: Humor analysis based on human annotation. In *IberLEF@ SEPLN*, pages 132–144, 2019.

[5] F. Chollet et al. Keras, 2015. URL `https://github.com/fchollet/keras`.

[6] N. K. Chowdhury, M. A. Kabir, M. M. Rahman, and S. M. S. Islam. Machine learning for detecting covid-19 from cough sounds: An ensemble-based mcdm method. *Computers in Biology and Medicine*, 145:105405, 2022.

[7] J. P. Consuegra-Ayala, Y. Gutiérrez, A. Piad-Morffis, Y. Almeida-Cruz, and M. Palomar. Automatic extension of corpora from the intelligent ensembling of ehealth knowledge discovery systems outputs. *Journal of Biomedical Informatics*, 116:103716, 2021.

[8] J. Dastin. Amazon scraps secret ai recruiting tool that showed bias against women. In *Ethics of Data and Analytics*, pages 296–299. Auerbach Publications, 2018.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

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

[11] D. Dua and C. Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[12] T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

[13] S. Estevez-Velarde, Y. Gutiérrez, A. Montoyo, and Y. Almeida-Cruz. Automatic discovery of heterogeneous machine learning pipelines: An application to natural language processing. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 3558–3568, 2020.

[14] S. Estevez-Velarde, A. Piad-Morffis, Y. Gutiérrez, A. Montoyo, R. Munoz, and Y. Almeida-Cruz. Solving heterogeneous automl problems with autogoal. In *ICML Workshop on Automated Machine Learning (AutoML@ ICML)*, 2020.

[15] S. Estevez-Velarde, Y. Gutiérrez, Y. Almeida-Cruz, and A. Montoyo. General-purpose hierarchical optimisation of machine learning pipelines with grammatical evolution. *Information Sciences*, 543:58–71, 2021. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2020.07.035. URL `https://www.sciencedirect.com/science/article/pii/S0020025520306988`.

[16] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. *Advances in neural information processing systems*, 28, 2015.

[17] A. J. Hamid and T. M. Ahmed. Developing prediction model of loan risk in banks using data mining. *Machine Learning and Applications: An International Journal*, 3(1):1–9, 2016.

[18] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*, 2017.

[19] F. Hutter, H. Larochelle, B. Kégl, I. Guyon, M. Bilenko, B. Rémi, and R. Caruana. Automl workshop @ icml'14. URL `http://icml2014.automl.org/`.

[20] H. Jin, Q. Song, and X. Hu. Auto-keras: Efficient neural architecture search with network morphism. *arXiv preprint arXiv:1806.10282*, 5, 2018.

[21] W. Jin, S. Dong, C. Yu, and Q. Luo. A data-driven hybrid ensemble ai model for covid-19 infection forecast using multiple neural networks and reinforced learning. *Computers in Biology and Medicine*, page 105560, 2022.

[22] R. C. Joshi, R. Mishra, P. Gandhi, V. K. Pathak, R. Burget, and M. K. Dutta. Ensemble based machine learning approach for prediction of glioma and multi-grade classification. *Computers in Biology and Medicine*, 137:104829, 2021.

[23] W. Kang, L. Lin, B. Zhang, X. Shen, S. Wu, A. D. N. Initiative, et al. Multi-model and multi-slice ensemble learning architecture based on 2d convolutional neural networks for alzheimer's disease diagnosis. *Computers in Biology and Medicine*, 136:104678, 2021.

[24] M. Kumar, R. Ghani, and Z.-S. Mei. Data mining to predict and prevent errors in health insurance claims processing. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 65–74, 2010.

[25] E. Loper and S. Bird. Nltk: The natural language toolkit. *arXiv preprint cs/0205028*, 2002.

[26] V. Mnih. *Machine learning for aerial image labeling*. University of Toronto (Canada), 2013.

[27] S. Ozturk and Y. Unal. A two-stage whale optimization method for classification of parkinson's disease voice recordings. *International Journal of Intelligent Systems and Applications in Engineering*, 8(2):84–93, 2020.

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

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

[30] V. Perrone, M. Donini, M. B. Zafar, R. Schmucker, K. Kenthapadi, and C. Archambeau. Fair bayesian optimization. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, pages 854–863, 2021.

[31] R. Polikar. Ensemble based systems in decision making. *IEEE Circuits and systems magazine*, 6(3):21–45, 2006.

[32] R. E. Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[33] N. S. Shaik and T. K. Cherukuri. Transfer learning based novel ensemble classifier for covid-19 detection from chest ct-scans. *Computers in Biology and Medicine*, 141:105127, 2022.

[34] G. A. Tahir and C. K. Loo. Explainable deep learning ensemble for food image analysis on edge devices. *Computers in Biology and Medicine*, 139:104972, 2021.

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

[36] Y. Ünal, Ş. Öztürk, M. N. Dudak, and M. Ekici. Comparison of current convolutional neural network architectures for classification of damaged and undamaged cars. In *Advances in Deep Learning, Artificial Intelligence and Robotics*, pages 141–149. Springer, 2022.

[37] C. Wang, B. Han, B. Patel, F. Mohideen, and C. Rudin. In pursuit of interpretable, fair and accurate machine learning for criminal recidivism prediction. *arXiv preprint arXiv:2005.04176*, 2020.

[38] I. H. Witten, E. Frank, M. A. Hall, C. J. Pal, and M. DATA. Practical machine learning tools and techniques. In *DATA MINING*, volume 2, page 4, 2005.

[39] P. Yang, Y. Hwa Yang, B. B Zhou, and A. Y Zomaya. A review of ensemble methods in bioinformatics. *Current Bioinformatics*, 5(4):296–308, 2010.