*Journal on Interactive Systems, 2023, 14:1,* doi: 10.5753/jis.2023.3161

# Evaluating a New Auto-ML Approach for Sentiment Analysis and Intent Recognition Tasks

**Douglas Nunes de Oliveira** [ **Take Blip - Research & Innovation** | ***douglas.nunes@take.net*** ]
**Milo Noronha Rocha Utsch** [ **Take Blip - Research & Innovation** | ***milo@take.net*** ]
**Diogo Villela Pedro de Almeida Machado** [ **Take Blip - Research & Innovation** | ***diogo@take.net*** ]
**Nina Goulart Pena** [ **Take Blip - Research & Innovation** | ***nina@take.net*** ]
**Ramon Gomes Durães de Oliveira** [ **Take Blip - Research & Innovation** | ***ramon.oliveira@take.net*** ]
**Arthur Iperoyg Rodrigues Carvalho** [ **Take Blip - Research & Innovation** | ***arthurc@take.net*** ]
**Luiz Henrique de Campos Merschmann** [ **Federal University of Lavras** | ***luiz.hcm@ufla.br*** ]

**Abstract**
Automated Machine Learning (AutoML) is a research area that aims to help humans solve Machine Learning (ML) problems by automatically discovering good ML pipelines (algorithms and their hyperparameters for every stage of a machine learning process) for a given dataset. Since we have a combinatorial optimization problem for which it is impossible to evaluate all possible pipelines, most AutoML systems use a Genetic Algorithm (GA) or Bayesian Optimization (BO) to find a good solution. These systems usually evaluate the performance of the pipelines using the K-fold cross-validation method, for which the more pipelines are evaluated, the higher the chance of finding an overfitted solution. To avoid the aforementioned issue, we propose a system named Auto-ML System for Text Classification (ASTeC), that uses the Bootstrap Bias Corrected CV (BBC-CV) method to evaluate the performance of the pipelines. More specifically, the proposed system combines GA, BO, and BBC-CV to find a good ML pipeline for the text classification task. We evaluated our approach by comparing it with state-of-the-art systems: in the the Sentiment Analysis (SA) task, we compared our approach to TPOT (Tree-based Pipeline Optimization Tool) and Google Cloud AutoML service, and for the Intent Recognition (IR) task, we compared with TPOT and MLJAR AutoML. Concerning the data, we analysed seven public datasets from the SA domain and sixteen from the IR domain. Four out of those sixteen are composed by written English text, while all of the others are in Brazilian Portuguese. Statistical tests show that, in 21 out of 23 datasets, our system's performance is equivalent to or better than the others.

***Keywords:*** *automl, bias correction cross-validation, genetic algorithm, bayesian optimization, intent recognition, chatbot*

## 1 Introduction

We are living in an era where data are readily available and accessible. In this way, an increasing number of people with different levels of skills in the Machine Learning (ML) field are trying to apply a great variety of machine learning approaches to extract useful information hidden in data. In this context, the need for off-the-shelf solutions in ML to help different types of users is increasingly evident. Thus, Automated Machine Learning (AutoML) arose aiming at democratizing ML to enable a wider audience, including non-experts, to benefit from its potential (Hutter et al., 2019).

AutoML is the process of automating multiple machine learning tasks (Guyon et al., 2015; de Sá et al., 2017; Hutter et al., 2019), such as preprocessing, model selection and hyperparameters optimization. Therefore, it can be used to find an optimized group of tasks automatically, decreasing the specific knowledge required by its user. Given a dataset, an AutoML system can be used to recommend a pipeline (sequence of tasks) to solve a machine learning problem.

Most existing AutoML solutions (Thornton et al., 2013; Feurer et al., 2015; Olson et al., 2016; de Sá et al., 2017) present a similar way of operating, where several pipelines are created and evaluated. Basically, an optimization technique is used to indicate which pipelines will be created. Then, each of these pipelines is evaluated using a chosen metric (such as accuracy or F1-Measure) and the cross-validation technique. Currently, optimization techniques such as Genetic Algorithms (GAs) and Bayesian Optimization (BO) are the main differential amongst AutoML solutions, while the cross-validation approach remains common to all of them.

The K-fold cross-validation is a technique that can be used to estimate the generalized predictive performance of a pipeline for a supervised machine learning task. In this approach, a labeled dataset is divided into k groups (folds) of approximately equal size. Then, a model is trained and evaluated k times, each time a group is used as a test set and the remaining ones as training set. Finally, the average of the metrics obtained in these k evaluations summarizes the pipeline's performance. This process tends to present a satisfactory estimate for evaluating a single pipeline. However, this may not apply when comparing several pipelines.

When comparing multiple pipelines with K-fold cross-validation, one of the main problems that arise is predictive performance overestimation (Tsamardinos et al., 2015). This is mainly due to the repetitive use of training sets by several pipelines. In this situation, the chance of overfitting the pipeline with the best performance to the data increases with the number of pipelines evaluated. One of the possible solutions to this problem would be using Bootstrap Bias Corrected CV (BBC-CV), proposed by Tsamardinos et al. (2018), to correct the performance evaluation bias.

BBC-CV is a method designed to estimate the performance of a predictive algorithm based on the predictions of its previously evaluated models (with different hyperparameter configurations) without adding too much computational cost. It selects a sample from the predictions, finds the best configuration for that sample and estimates its predictive performance based on the remaining instances. This process is repeated many times and the average of its results is the final estimation. Hence, we proposed in our previous work (de Oliveira and de Campos Merschmann, 2022) a new AutoML approach that combines a Genetic Algorithm (GA) with Bayesian Optimization and the BBC-CV, and evaluated it for the Sentiment Analysis(SA) task. In this work, which is an expansion and revision of that previously published paper, we now focus on evaluating our previously proposed AutoML approach on Intent Recognition (IR) tasks. Furthermore, MLJAR AutoML (Płońska and Płoński, 2021) was introduced in this expansion as an additional AutoML solution to be used as benchmark for the AutoML system proposed in the original paper.

In our previous work (de Oliveira and de Campos Merschmann, 2022) we focused on the Sentiment Analysis task, which is a field that studies techniques capable of automatically extracting information related to opinions and feelings from natural language data. It has become a remarkably active research area in recent years, motivated by the increasing amount of text produced in online social networks. Due to its high applicability, it has been used as a tool in industry and commerce (Ravi and Ravi, 2015; Ribeiro et al., 2016), besides being an object of study in the academy. In the context of SA, only texts written in the Brazilian Portuguese language were considered because, although it is the ninth most spoken language in the world with more than 250 million speakers (Eberhard et al., 2022), the number of papers focused on sentiment analysis for the Portuguese language is relatively small (Souza et al., 2016; Pereira, 2021) when compared to other languages, such as English. Besides, despite multiple language solutions being available, such as the ones proposed by Silva et al. (2011), Narr et al. (2012), and de Oliveira and de Campos Merschmann (2021), their performance may be unstable for different languages.

In this work we expand the evaluation of our Auto-ML technique to the Intent Recognition (IR) domain. IR is a well-established NLP task where the goal is to determine which of a set of classes matches a given written or spoken utterance to best respond to an interaction. This technology is widely used on chatbot systems to improve user experience, and while a rule-based approach to this problem exists, it falls short when faced with new dialogue contexts (Huggins et al., 2021). For the IR context, the proposed approach was evaluated using datasets with text written in English and Brazilian Portuguese. The decision to add English datasets in the evaluation of the IR case was based on the lack of open datasets for Intent Recognition in Portuguese. While being real examples of the usage of IR in chatbots, the Portuguese datasets were smaller than the ones used in the SA case. So, to complement the results and also begin to explore the performance of our approach in another language, we also tested the IR task on English datasets.

The remainder of this paper is organized as follows. Section 2 provides a brief description of the main related works. Background about the BBC-CV is presented in Section 3. Then, the proposed method is presented in Section 4. In Section 5, we present an overview of the baseline methods used in the comparative study carried out here. Section 6 presents the details of the computational experiments and reports on the obtained results for both tasks. Finally, Section 7 concludes this experience and presents guidelines for future work.

## 2 Related Work

Auto-WEKA (Thornton et al., 2013; Kotthoff et al., 2017) and the Auto-Sklearn (Feurer et al., 2015) are open-source automated machine learning toolkits and are quite similar. They both use the same BO algorithm, named Sequential Model-based Algorithm Configuration (SMAC, proposed by Hutter et al. (2011)). The principal difference is the usage of WEKA package by the former, and scikit-learn library by the latter. Besides that, the Auto-Sklearn can use meta-learning and work with ensembles of the best pipelines evaluated. Although the Auto-WEKA tries to use the repeated random subsampling validation (Kohavi, 1995), Auto-WEKA and Auto-Sklearn end up using the k-fold cross-validation method to estimate the evaluated model's predictive performance.

In the same way that several Auto-ML approaches, TPOT (Olson et al., 2016) and REsilient ClassifIcation Pipeline Evolution (RECIPE, proposed by de Sá et al. (2017)) also use an Evolutionary Algorithm. The first one uses Genetic Programming (GP), while the second one uses Grammar-based Genetic Programming (GGP). They both have statistically similar predictive performance, but the second one implements methods to prevent the generation of invalid pipelines. Both TPOT and RECIPE use k-fold cross-validation, but the RECIPE resamples the folds after every five generations, trying to avoid overfitting.

To the best of our knowledge, our approach is the first that combines the BO algorithm with the Genetic Algorithm and uses BBC-CV to estimate the predictive performance. Also, none of the previously mentioned methods are ready for SA task, since they focus only on classification task, disregarding the Natural Language Processing tasks involved in the preprocessing step. The only Auto-ML methods that we found ready for this were the AutoML Natural Language available on the Google Cloud AutoML platform (Google Cloud, 2019) and MLJAR AutoML. More details on Google Cloud AutoML and MLJAR AutoML on Section 5.2 and Section 5.3.

## 3 Background

Overfitting is still an open problem in hyperparameter optimization as well as in Auto-ML systems (Feurer and Hutter, 2019). It is well-know that some machine learning algorithms models can be biased towards some data, and this can also happen to its hyperparameters (Cawley and Talbot, 2010). This problem can cause Auto-ML systems to select machine learning models that appear to have good generalized predictive performance, but that actually perform well only on the data used. We could find, for example, the values for the SVM classifier hyperparameters (such as C and kernel) that would make a model perfectly fit the training data, but this model would probably have a low predictive performance for new data because of overfitting.

Several approaches are used to try to solve this problem, such as shuffling the train and test data for each evaluation, using a separate holdout to evaluate the configurations, selecting sub-optimal hyperparameter configurations, or even using ensembles (Feurer and Hutter, 2019). However, most of them ignore one of the main problems of overfitting, which is correctly estimating the generalized predictive performance of the evaluated models.

K-fold cross-validation is commonly used to estimate the predictive performance of machine learning models, but it can be biased accordingly to the number of folds, number of configurations evaluated and the dataset used. The properties of cross-validation and some variants are explored in the hyperparameter optimization context by Tsamardinos et al. (2015). In short, that study shows that the cross-validation overestimates the predictive performance on the hyperparameter optimization context, while the nested cross-validation yields the most precise results with a high computational cost. They also show that the method presented by Tibshirani and Tibshirani (2009) is promising to estimate the bias in most scenarios, but should be used with care, considering the number of folds in the cross-validation method and the number of configurations evaluated.

BBC-CV and the Bootstrap Bias Corrected with Dropping Cross-Validation (BBCD-CV, proposed by Tsamardinos et al. (2018)) are other promising methods for estimating predictive performance, and were developed considering the hyperparameter optimization context. BBC-CV method was developed with the objective of estimating the generalized predictive performance of the evaluated models in an accurate and efficient way. This method works with the predictions obtained by all models evaluated using k-fold cross-validation. Basically, $b$ iterations are made, in each one the best configuration is chosen according to a replacement sampling containing $n$ instances. In each iteration, the performance of the selected configuration is calculated considering the instances that were not chosen in that sample. Finally, the average of these performances obtained is returned as a result of the estimated performance of the best configuration. In addition, it is also possible to calculate the confidence intervals, adding more information to this assessment.

BBC-CV was developed to estimate the performance of a group of configurations already evaluated, however, these configurations are generally evaluated iteratively. Such assessments can be made with the help of the k-fold cross-validation however, the BBCD-CV can be more efficient. In this method, new configurations are evaluated considering the previously evaluated configurations. For this, each fold is evaluated similarly to the one that would be performed using the k-fold cross-validation, however, the predictions of the configurations already evaluated are used to determine if it is advantageous to evaluate this new configuration in all folds. For this, the probability of the new configuration being better than the old ones is calculated after evaluating it in each fold. In this context, the evaluation of this configuration can be terminated early if the value of this probability is less than a value of $p$. This way, it is not necessary to evaluate all folds with the configurations that perform poorly in the first folds.

Thus, the BBC-CV can be used in an Auto-ML system to select better models among the evaluated ones, while the BBCD-CV can be used to reduce the computational resources used.

## 4 Proposed Method

An Auto-ML problem can be defined as a Combined Algorithm Selection and Hyperparameter Optimization (CASH) problem (Thornton et al., 2013). As the name implies, the objective is to select an algorithm and its hyperparameters while optimizing the predictive performance for a dataset. However, this objective can be extended to select an entire pipeline (set of preprocessing methods with their hyperparameters and a classifier with its hyperparameters). Defining the entire pipeline can be helpful in problems where the data preprocessing has a large impact on the classifier's performance.

Previous works, done by Uysal and Günal (2014), Alam and Yao (2019), and de Oliveira and de Campos Merschmann (2021), show that the preprocessing tasks can impact the predictive performance of the classifiers in text classification. Considering this, the proposed method was designed to automatically select an entire pipeline for a text classification problem.

While most of the work treats the algorithm selection and the hyperparameter optimization as a single problem, our approach divides it into two subproblems and combines their results to return the full pipeline. Below, we present the details of the proposed approach that combines the Genetic Algorithm and the Bayesian Optimization (BO) approach.

The purpose of the GA in our approach is to suggest a good set of algorithms with their hyperparameters (pipeline) given a dataset. So, it is used to select a set of preprocessing algorithms and a classifier (represented by an individual of the GA). To complement the pipeline definition, the BO is used to suggest good hyperparameters values (configurations) to the individuals generated by the GA. Last but not least, the BBC-CV (Tsamardinos et al., 2018) is used to select the best individual (pipeline), while the BBCD-CV (Tsamardinos et al., 2018) is used to select the best hyperparameters set for an individual.

Figure 1 shows an overview of the proposed method. It has four steps named Individual Management, Pipeline Management, Pre-evaluation, and Evaluation, each one represented as a rounded rectangle. After each step, it also shows its output.

The proposed approach needs a dataset and a group of methods to work. The dataset is composed of labeled instances, that is, a group of instances where each one is associated with a class label. The group of methods defines what methods can be used to create combinations, and each method must have its specifications defined. These specifications are used to make sure that only valid combinations are

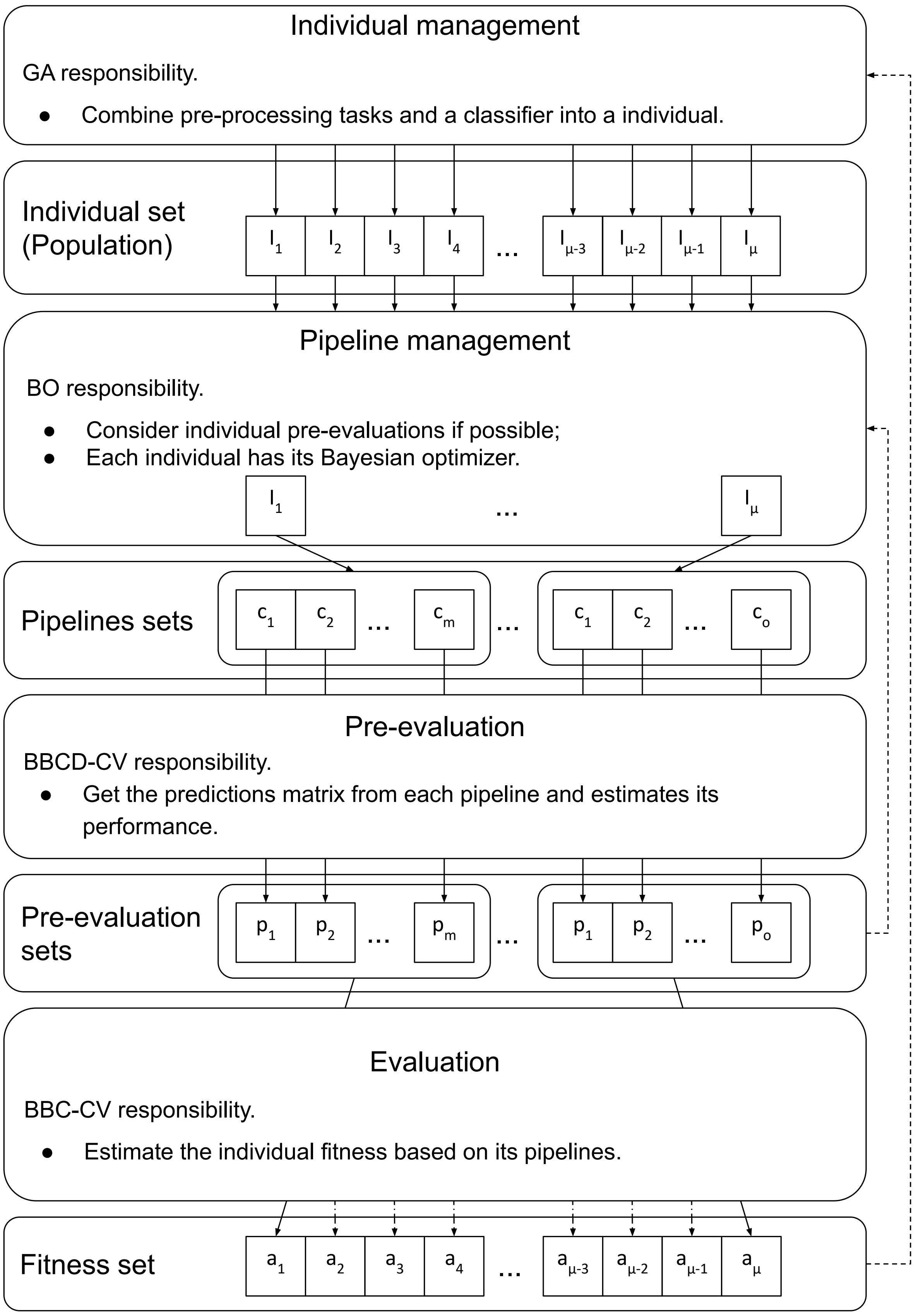
Individual management
GA responsibility.
Combine pre-processing tasks and a classifier into a individual.
Individual set (Population)
$I_1$ $I_2$ $I_3$ $I_4$ ... $I_{\mu-3}$ $I_{\mu-2}$ $I_{\mu-1}$ $I_\mu$
Pipeline management
BO responsibility.
Consider individual pre-evaluations if possible;
Each individual has its Bayesian optimizer.
$I_1$ ... $I_\mu$
Pipelines sets
$c_1$ $c_2$ ... $c_m$ ... $c_1$ $c_2$ ... $c_o$
Pre-evaluation
BBCD-CV responsibility.
Get the predictions matrix from each pipeline and estimates its performance.
Pre-evaluation sets
$p_1$ $p_2$ ... $p_m$ ... $p_1$ $p_2$ ... $p_o$
Evaluation
BBC-CV responsibility.
Estimate the individual fitness based on its pipelines.
Fitness set
$a_1$ $a_2$ $a_3$ $a_4$ ... $a_{\mu-3}$ $a_{\mu-2}$ $a_{\mu-1}$ $a_\mu$

**Figure 1.** Proposed Method Overview

generated, and could be the type specification or the hyperparameter search space specification. The type specification is used to define the method type, e.g., the Gaussian Naive Bayes and the Random Forest are typed as a classifier, and what kind of method types can precede it or succeed it. This is used to avoid different methods to do the same task and prevent the generation of invalid combinations. The hyperparameter search space defines the rules used by the BO method for each method.

A GA is responsible for the Individual management step. The objective of this step is to manage the individuals by choosing which individuals will be evaluated in each generation. The widely known ($\mu$, $\lambda$) evolution strategy (Beyer and Schwefel, 2002) is being used in the GA with a fixed number of generations and some minor changes. The first modification is that the initial population for the GA is created randomly, but tries to use each defined method at least once. The evaluation step is also different, but will be described in more detail later in this section. A tournament without replacement is used to select the combinations that will survive to the next generation. The GA operators, crossover, and mutation were also modified to meet the problem requirements. A single-point crossover technique was adapted to consider the specifications of the methods, so it selects only valid cut points. The mutation can happen in any method of an individual, and it was also modified to use three different operations. The first operation appends a new method after the selected one. The second operation swaps the selected method with a different one. The last operation removes the selected method from the individual. The mutation can select only one operation for each method, and the operations have the same chance of being chosen. After this step, a set of individuals (population) is created.

A BO is responsible for the Pipeline management step. The objective of this step is to choose which configurations will be evaluated for each individual. A BO is used to analyze the results of previously tested pipelines from the same individual and suggests new ones to be evaluated. More precisely, we use a Gaussian Process Regressor (GPR) interchanging in each iteration between the Upper Confidence Bound and the Expected Improvement as its acquisition function. The size of the search in each evaluation is directly proportional to the hyperparameters search space size. A pipeline set for each individual is created after this step.

The Pre-evaluation step has two main objectives, i.e., to estimate the predictive performance and to get the predictions matrix for each pipeline. BBCD-CV is used to accomplish these objectives. The estimated predictive performance is calculated similarly to the traditional k-fold cross-validation, but it considers the previously evaluated pipelines from the same individual to decide if that pipeline should be evaluated in all folds. After this step, the predictive performance and the predictions for each pipeline are stored in pre-evaluation sets for each individual.

The Evaluation step has the objective to estimate the predictive performance of each individual, and the BBC-CV is responsible for this. BBC-CV is used to estimate the predictive performance of each individual by considering the predictions collected from its pipelines in the pre-evaluation step. This estimation is used as the fitness for each individual.

These four steps (Individual Management, Pipeline Management, Pre-evaluation, and Evaluation) are repeated interactively. In the end, the best pipeline found from the best individual obtained is returned by the approach. Although the best pipeline evaluated can be the one chosen in the end, its predictive performance alone does not guarantee this, since it can be from a different individual than the best one found. For example, for one dataset the best evaluated pipeline could be one with the SVM classifier with $C = 10$ and $kernel = Linear$, but the best individual could be one with the MLP classifier. In this case, our method would return the best pipeline with the MLP classifier, since the one with the SVM was less robust than the one with the MLP.

Algorithm 1 presents the proposed approach pseudocode. It aims to return the best pipeline found for a given dataset. For this, this algorithm follows a flow similar to that used by a GA and receives these parameters as input:

- **D**: $D = \{(x_j, y_j)\}_{j=1}^{n}$, is a labeled dataset containing $n$ documents, where $x_j$ represents a text document and $y_j$ represents its label. Tables 1 and 2 show examples of labeled datasets for Sentiment Analysis and Intent Recognition tasks, respectively.

**Table 1.** Labeled dataset example for Sentiment Analysis.

| $j$ | $x$ | $y$ |
|---|---|---|
| 1 | Positive document example. | Positive |
| 2 | Negative document example. | Negative |
| ⋮ | ⋮ | ⋮ |
| $n$ | Neutral document example. | Neutral |

**Table 2.** Labeled dataset example for Intent Recognition.

| $j$ | $x$ | $y$ |
|---|---|---|
| 1 | I would like to buy a bag. | Buy |
| 2 | I need to cancel my plan. | Cancel |
| ⋮ | ⋮ | ⋮ |
| $n$ | Change my login email. | Update profile |

- **M**: $M = \{(\mu_k, t_k, I_k, o_k, H_k)\}_{k=1}^{m}$, is a set that contains the methods available to be used to make an individual. In this context, $\mu_k$ is a preprocessing algorithm or classification algorithm of the type $t_k$; $I_k = \{i_l\}_{l=1}^{q}$ is a set of data types, where $i_l$ represents the accepted input data type by the $\mu_k$ method; $o_k$ represents the accepted output data type by the $\mu_k$ method; and $H = \{(h_g, S_g)\}_{g=1}^{r}$ represents the $\mu_k$ method hyperparameter search space, where $h_g$ is the hyperparameter name and $S_g$ is the possible values set for $h_g$. Table 3 presents an example of a set of methods that can compose an individual.
- **Bbcdcv**: BBC-CV and BBCD-CV parameters set. Includes the number of bootstraps ($Bbcdcv_b$), number of samples to be selected in each bootstrap ($Bbcdcv_n$), evaluation dropping probability ($Bbcdcv_p$), confidence interval ($Bbcdcv_{alpha}$) and the number of partitions to be used internally ($Bbcdcv_{cv}$).

**Table 3.** Methods set example.

| $k$ | $\mu$ | $t$ | $I$ | $o$ | $H$ |
|---|---|---|---|---|---|
| 1 | Simple Tokenizer | Tokenizer | {Text} | Tokens | ∅ |
| 2 | Simple N-gram | N-Gram | {Tokens} | Tokens | See Table 4 |
| 3 | Simple converter | Lowercase converter | {Text, Tokens} | <Same as input> | ∅ |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| $m$ | SVM Classifier | Classifier | {Features} | Predictions | See Table 5 |

**Table 4.** Simple N-gram hyperparameter search space example.

| $g$ | $h$ | $S$ |
|---|---|---|
| 1 | N | {2, 3, 4} |

**Table 5.** SVM classifier hyperparameter search space example.

| $g$ | $h$ | $S$ |
|---|---|---|
| 1 | c | {0.01, 0.1, 1, 10, 100} |
| 2 | kernel | {Linear, RBF, Sigmoid} |

- **BO**: Bayesian optimization parameters set. Includes the minimum number of pipelines to be evaluated ($Bo_{min}$), maximum number of pipelines to be evaluated ($Bo_{max}$), and a generation step value to execute deep searches ($Bo_{step}$).
- **Gen**: Set composed of the parameters used by the GA. Includes the probability of combining 2 individuals ($Gen_{p\_comb}$), the probability of mutating an individual ($Gen_{p\_mut}$), the probability of mutating each gene (method representation) of an individual ($Gen_{p\_ind\_mut}$), the number of generations ($Gen_{qty\_ger}$), the number of individuals selected for the next generation ($Gen_{mu}$), the number of individuals in each generation ($Gen_{lambda}$) and the number of individuals selected in each tournament ($Gen_{tor\_size}$).

$FirstPopulation$ function (line 3 of Algorithm 1) is used to create the first individuals. For that, it receives a set of methods ($M$) and a set of parameters ($Gen$). Its objective is to return a set of $Gen_{lambda}$ individuals created with the methods of the set $M$. It tries to create a diverse population by creating individuals with methods least used.

$MakePartitions$ function (line 4 of Algorithm 1) receives as input a labeled dataset $D$ and a set of parameters $Bbcdcv$. From this input, the algorithm generates $Bbcdcv_{cv}$ stratified partitions and a set of tuples formed by sets of instances for training and testing is returned.

$PreEval$ function (lines 7 and 15 of Algorithm 1) receives as parameters the sets with all the previous pipelines evaluations ($Evals$), the partitions used for cross-validation ($Partitions$), the individual to be optimized ($individual$), the pipelines quantity to be evaluated ($qty$), in addition with the $Bbcdcv$ and $Bo$ parameter sets. In the end, it returns all the individual's ($individual$) pipelines evaluations obtained.

$GenOps$ function (line 11 of the Algorithm 1), is used to generate new individuals from the current ones. For that, this algorithm receives as input a set of individuals ($Population$) and the parameters used by the genetic algorithm ($Gen$). In the end, this algorithm returns a set containing the new individuals by applying the mutation and crossover operations.

$CalcSearchSize$ function (line 12 of Algorithm 1), is used to define the number of pipelines that must be evaluated for each individual. To perform this task, this function receives as input a set of individuals ($Children$), a set with the executed evaluations ($Evals$), the ordinal number of the current generation ($generation$), and the hyperparameters used by the Bayesian optimization method ($Bo$). In the end, this function returns a set with the number of pipelines that must be evaluated for each individual.

$Select$ function (line 18 of Algorithm 1) is used to choose which individuals will survive to the next generation. To achieve this goal, this algorithm receives a set of individuals ($Children$), a set of its evaluations ($Evals$), the hyperparameters set used by the genetic algorithm ($Gen$), the hyperparameters set used by the BBC-CV method ($Bbcdcv$), and the partitions sets ($Partitions$). In the end, this algorithm selects a subset from $Children$ using a tournament selection method.

$GetBestPipeline$ function (line 20 of Algorithm 1) aims to create a pipeline from the best individual with the best configuration found. For this, this algorithm receives as input the set with all executed evaluations ($Evals$) and the used partitions ($Partitions$).

To implement and evaluate the ASTeC we used these open-source tools:

- Chocolate (AIworx, 2017): used in the BO.
- DEAP (Fortin et al., 2012): used in the GA.
- Nlpnet (Fonseca and Rosa, 2013): used in the PoS-tagger.
- NLTK (Bird et al., 2009): used in the n-gram, stemmer, and tokenizers.
- Scikit-learn (Pedregosa et al., 2011): used in the classifier, TF-IDF vectorizer, and Feature selector.

## 5 Baseline methods

In this section, we present an overview of the baseline methods used on the two study cases detailed in this research, Sentiment Analysis and Intent Recognition. Section 5.1 provides a brief description of the TPOT. The Google Cloud AutoML service is discussed in Section 5.2. Then, in Section 5.3 we describe the MLJAR method.

```
Input: $D, M, Bbcdcv, Bo, Gen.$
Output: Best pipeline found for the dataset $D$.
1  begin
2      $Evals \leftarrow \varnothing;$
3      $Population \leftarrow FirstPopulation(M, Gen);$
4      $Partitions \leftarrow MakePartitions(D, Bbcdcv);$
5      $qty \leftarrow Bo_{max};$
6      foreach $ind \subset Population$ do
7          $NewEvals \leftarrow PreEval(Evals, Partitions, ind, qty, Bbcdcv, Bo);$
8          $Evals \leftarrow Evals \cup NewEvaluations;$
9      end
10     for $i = 2$ to $Gen_{qty\_ger}$ do
11         $Children \leftarrow GenOps(Population, Gen);$
12         $SearchSize \leftarrow CalcSearchSize(Children, Evals, i, Bo);$
13         foreach $ind \subset Children$ do
14             $qty \leftarrow SearchSize_{ind};$
15             $NewEvaluations \leftarrow PreEval(Evals, Partitions, ind, qty, Bbcdcv, Bo);$
16             $Evals \leftarrow Evals \cup NewEvaluations;$
17         end
18         $Population \leftarrow Select(Children, Evals, Gen, Bbcdcv, Partitions);$
19     end
20     $BestPipeline \leftarrow GetBestPipeline(Evals, D);$
21     return $BestPipeline;$
22 end
```

**Algorithm 1:** ASTeC

## 5.1 TPOT

TPOT is an AutoML method that automates the steps of feature preprocessing, feature selection, feature construction, model selection, and parameter optimization. It uses genetic programming to automatically optimize data transformations and machine learning model training. Each pipeline is represented as a tree composed of four operators (preprocessors, decomposition, feature selection, and models) and they are evolved using Genetic Programming (GP) while trying to maximize the classification accuracy.

The current TPOT version doesn't natively work with text datasets. To be able to do this, it needs to implement text preprocessing techniques to, at least, transform the text data into dataset features. And since each language has its own rules, many techniques are language-dependent (i.e. POS-tagger, stemmer, etc.).

To be able to use text data with TPOT one needs to transform it into dataset features. But choosing the best combination of preprocessing tasks isn't an easy task, since they are directly related to the generated features. But once we turn the text data into dataset features, we can use the TPOT to train a machine learning model.

## 5.2 Google Cloud AutoML

Google Cloud AutoML [1] is a paid platform provided by Google that allows users to create custom machine learning models based on images, videos, text, and tables from custom datasets. AutoML Natural Language [2] allows users to create supervised models for text classification, entity extraction, and sentiment analysis. For that, it uses the Google Vizier (Golovin et al., 2017) black-box optimization system to search, select and tune different models. This system implements multiple search methods and can swap between each one according. Besides that, it also has the ability to transfer knowledge between different runs (meta-learning) and to stop early accordingly to the search progress. The user doesn't have direct access to the trained model and the training information, but he can interact with the system through an Application Programming Interface (API).

From the user perspective, the service takes a labeled dataset and offers an API that accesses the model and can be used to predict new documents. This simplicity can be seen as an advantage because it allows non-experts to easily use machine learning techniques, but it has its drawbacks. The user doesn't know which methods are used to process the text, train the classifier model, and neither the hyperparameter's optimization details. These pieces of information are private to Google, but the Google Vizier (Golovin et al., 2017) is probably used to tune the hyperparameters.

[1]https://cloud.google.com/automl/

[2]https://cloud.google.com/natural-language/automl/docs

## 5.3 MLJAR

MLJAR AutoML is a package for automating algorithm selection, preprocessing techniques, model training, explanation, and evaluation. The mode used in the experiments was 'Perform', in which the MLJAR uses a random search method in combination with hill climbing. Every checked model in this mode is saved and used for building an ensemble at the end of each step. MLJAR accepts textual data as input, so it is not necessary to perform any data transformation or preprocessing to use it.

## 6 Computational Experiments

In this section, we examine two study cases. In the first subsection, we compare our approach with that used in the Google Cloud AutoML and TPOT for the Sentiment Analysis task. After, in the second subsection, we compare our approach with TPOT and MLJAR for the Intent Recognition task.

The datasets used for both study cases are detailed in Section 6.1 for the SA task and in Section 6.2 for the IR task. Then, the entire experimental setup is presented in Section 6.3.

### 6.1 Sentiment Analysis Datasets

For the computational experiments on the sentiment analysis task, we selected seven datasets composed by texts in Portuguese. These datasets were selected because, to the best of our knowledge, they were all of the public sentiment analysis datasets in Brazilian Portuguese. The datasets receive the following names: Application Comments (AC) (Junior and de Campos Merschmann, 2016), Financial Market News (FMN) (Martins et al., 2015), Automobiles *Tweets* (AT) (Martins et al., 2015), Prodemge MG *Tweets* (PMGT) (Ferreira, 2017), *Tweets* in Portuguese (TP) (Narr et al., 2012; Araújo et al., 2016), Traffic Related *Tweets* (TRT) (Xavier, 2018), and TweetSentBR (TSBR) (Brum and das Graças Volpe Nunes, 2018).

The AC dataset consists of app comments taken from the Google Play virtual store. It was originally developed and presented by dos Santos and Ladeira (2014) and revised by Junior and de Campos Merschmann (2016).

The AT dataset was presented by Martins et al. (2015) and contains *tweets* published in 2012 related to the "Fiat" brand.

The FMN dataset was made with financial market news extracted from different news websites in 2014. This dataset was created by a team from the INWeb project (Martins et al., 2015). Its documents were labeled by experts taking into consideration the impact of each news on the market, considering an investor's point of view.

The PMGT dataset was presented by Ferreira (2017) and, according to information provided by its author, it was collected by the IT staff of Prodemge MG and manually labeled by the analysts of that sector.

The TP dataset was initially presented by Narr et al. (2012) and also used by Araújo et al. (2016). It is composed of *tweets* related to eleven popular brands ("Microsoft", "Adidas", "Audi", etc.), and its documents were manually labeled through the Amazon Mechanical Turk service.

The TRT dataset is composed of *tweets* about the traffic in a city and was presented by Xavier (2018). Its documents were manually labeled as positive if they reported a good situation, as negative if they reported a problem, or as neutral if neither.

The TSBR dataset is composed of *tweets* about TV shows and was presented by Brum and das Graças Volpe Nunes (2018). It was labeled by seven annotators. Each dataset instance is available with the individual label given by each annotator, in addition to the number of annotators that had problems to decide between the labels. Considering all this information, we decided to work with only a fraction of the TSBR dataset. So, we selected only the instances that were labeled by at least three annotators with at least $2/3$ of agreement and without any problem.

Table 6 presents the characteristics of each dataset, that is, the number of instances for each class (positive, neutral, and negative), and the average number of characters and words per instance.

### 6.2 Intent Recognition Datasets

We selected two groups of datasets for the task of Intent Recognition. The first group, composed of texts in English, comprises the following datasets: BANKING77 (Casanueva et al., 2020), CLINC150 (Larson et al., 2019), HWU64 (Liu et al., 2019) and SNIPS (Coucke et al., 2018). They were selected from a repository of Intent Detection datasets that was made available by Zhang et al. (2021). We decided to work with them because they are publicly available and are similar in structure to the datasets of the second group. These datasets also allowed us to evaluate our approach with texts written in English. Table 7 presents the following datasets' characteristics of this first group: number of domains it spans, number of instances, training set size, and testing set size.

The second group, composed of texts in Brazilian Portuguese, comprises the following original datasets: BANKING19, CONSTR25, UTILITY21, RETAIL32, DEBTRELIEF13, BANKING3, BANKING10, TELCO33, CREDIT60, HEALTH32, BANKING51 and TELCO8. They contain data from real chatbot intent recognition cases from different domains. Table 8 shows their characteristics, including text domain, number of intents, training set size, and test set size. They are datasets used to set up intent recognition models used on the customer communication channels of client companies. They were anonymized and granted to the study after the client companies consented to their use.

### 6.3 Experimental configuration

In the current stage, ASTeC has 15 hyperparameters. We used the same configuration for both study cases. A brief explanation and the value used for each one are presented below:

- $Bbcdcv_{alpha}$: Alpha parameter used inside the BBCD-CV method. Set to 0.05.
- $Bbcdcv_{b}$: Number of bootstraps to be executed. Set to 1000.
- $Bbcdcv_{cv}$: Number of partitions to use in the internal cross-validation. Set to 5.
- $Bbcdcv_{n}$: Number of samples to select in each bootstrap. Set to 500.
- $Bbcdcv_{p}$: Probability parameter used inside the BBCD-CV. Set to 0.1.
- $Bo_{max}$: Maximum number of pipelines evaluated for a combination in each generation. Set to 10.
- $Bo_{min}$: Minimal number of pipelines evaluated for a combination in each generation. Set to 5.

**Table 6.** Sentiment Analysis datasets characteristics

| Dataset | # Negative | # Neutral | # Positive | # Total | # Avg characters | # Avg words |
|---|---|---|---|---|---|---|
| AC | 815 | - | 815 | 1630 | 73.17 | 13.25 |
| AT | 6054 | - | 5997 | 12051 | 102.67 | 16.74 |
| FMN | 751 | 453 | 928 | 2132 | 2195.02 | 282.27 |
| PMGT | 2446 | 2453 | 3300 | 8199 | 116.46 | 16.14 |
| TP | 213 | 264 | 297 | 774 | 78.36 | 14.08 |
| TRT | 1757 | 1442 | 752 | 3951 | 111.24 | 16.48 |
| TSBR | 398 | 325 | 687 | 1410 | 73.76 | 11.58 |

**Table 7.** English IR datasets characteristics

| Dataset | # Domain | # Intents | # Training | # Testing |
|---|---|---|---|---|
| BANKING77 (Casanueva et al., 2020) | 1 | 77 | 10162 | 3080 |
| CLINC150 (Larson et al., 2019) | 10 | 150 | 18000 | 4500 |
| HWU64 (Liu et al., 2019) | 21 | 64 | 10030 | 1076 |
| SNIPS (Coucke et al., 2018) | 1 | 7 | 13784 | 700 |

- $Bo_{step}$: Number of generations that it should use the $Bo_{min}$ after using the $Bo_{max}$. Set to 3.
- $Gen_{p\_comb}$: Crossover probability between combinations. Set to 0.35.
- $Gen_{p\_mut}$: Mutation probability for the combinations. Set to 0.35.
- $Gen_{p\_ind\_mut}$: Method mutation probability for a combination selected for mutation. Set to 0.1.
- $Gen_{qty\_ger}$: Number of generations to execute. Set to 5.
- $Gen_{mu}$: Number of combinations to be kept from one generation to the next. Set to 6.
- $Gen_{lambda}$: Number of combinations to be evaluated in each generation. Set to 12.
- $Gen_{tor\_size}$: Number of the combinations to be selected for each tournament. Set to 4.

These hyperparameters were obtained through empirical tests and were used in all experiments. More specifically, for the BBCD-CV we used the same values adopted by Tsamardinos et al. (2018). For the selection of the BO and GA hyperparameters, we used a small sample (100 instances) from the smallest dataset (TP). This calibration procedure was made only to speed up the search for a good combination instead of optimizing the general predictive performance of our method.

### 6.3.1 Configuration for Sentiment Analysis

Our approach generates the pipelines based on preprocessing tasks and classifiers commonly used in the problems under consideration. To optimize processing time, we used different sets of preprocessing tasks and classifiers for Sentiment Analysis and Intent Recognition problems. The options for the Sentiment Analysis task are presented below:

- Feature selector.
- Lowercase converter.
- MLP classifier.
- Nlpnet PoS-tagger.
- RF classifier.
- RSLP stemmer.
- SVM classifier.
- Simple N-gram.
- Simple tokenizer.
- TF-IDF vectorizer.
- *Tweets* tokenizer.

Using these tasks and classifiers, a simplified example of a resulting pipeline could be a Lowercase converter, a *Tweets* tokenizer, an RSLP stemmer, a Simple N-gram using $n = 2$, a TF-IDF vectorizer, and the SVM classifier with $C = 10$ and $kernel = Linear$.

TPOT lacks native text classification support. To be able to execute it, we have applied the following text preprocessing steps *Tweets* tokenizer, Nlpnet PoS-tagger, RSLP stemmer, Lowercase converter, Simple N-gram (with $n = 2$), and the TF-IDF vectorizer. Then, the resulting datasets were used as input for the TPOT method considering the same classifiers used in ASTeC (MLP, RF, and SVM). For each dataset, we limited the TPOT run time to the average time observed in the proposed method (not considering the text preprocessing steps). Google Cloud AutoML supports text, but since it is a black box, we don't have access to the preprocessing methods used.

### 6.3.2 Configuration for Intent Recognition

When compared to the configuration for Sentiment Analysis, in the case of Intent Recognition, we incorporated a sentence transformer vectorizer and removed the SVM classifier to reduce the search space. The set of options is the following:

- Feature selector.
- Lowercase converter.
- MLP classifier.
- Nlpnet PoS-tagger.
- RF classifier.
- RSLP stemmer.
- Simple N-gram.
- Simple tokenizer.
- TF-IDF vectorizer.

**Table 8.** Portuguese IR datasets characteristics

| Dataset | Domain Area | # Intents | # Training | # Testing |
|---|---|---|---|---|
| BANKING19 | Banking | 19 | 216 | 157 |
| CONSTR25 | Construction | 25 | 227 | 339 |
| UTILITY21 | Utility Provider | 21 | 759 | 362 |
| RETAIL32 | Retail | 32 | 320 | 315 |
| DEBTRELIEF13 | Debt Relief | 13 | 290 | 210 |
| BANKING3 | Banking | 3 | 30 | 28 |
| BANKING10 | Banking | 10 | 391 | 1698 |
| TELCO33 | Telecommunication | 33 | 683 | 1698 |
| CREDIT60 | Credit | 60 | 683 | 2181 |
| HEALTH32 | Health | 32 | 393 | 601 |
| BANKING51 | Banking | 51 | 1020 | 497 |
| TELCO8 | Telecommunication | 8 | 504 | 196 |

- Sentence Transformer vectorizer.
- *Tweets* tokenizer.

Considering this set of options, a pipeline could be, for example, composed of a Lowercase converter, an RSLP stemmer, a Sentence Transformer vectorizer $model\_name = sentence - transformers/all - MiniLM - L12 - v2$, and the MLP classifier with $activation = relu$ and $alpha = 0.01$.

In the case of Intent Recognition, we used only a Sentence Transformer $model\_name = sentence - transformers/all - MiniLM - L12 - v2$ (Reimers and Gurevych, 2019; Wang et al., 2020) before passing the resulting datasets to TPOT. Since MLJAR can handle textual data, we did not perform any preprocessing tasks in the datasets before running it.

## 6.4 Results

In the case of SA, the experiments performed in this work were conducted using the stratified 10-fold cross-validation technique. We created 10 pairs (train and test) for each dataset using this technique. We trained each approach with the same train partitions and evaluated it with the same test partitions to make sure all evaluations were fair. As for the IR problem, we have pairs of train and test data for each datasets.

### 6.4.1 Sentiment Analysis Result

For the Sentiment Analysis task we choose to work with the F1-Measure as the predictive performance metric since some datasets are binary class and others are multi-class. The experiments conducted with the proposed approach were executed in an Intel Xeon E-2136 (12 x 3.30GHz) with 64 gigabytes RAM computer. The experiments carried out with the Google service were performed on its platform, and its computational configuration is not public, which makes it unfeasible execution time comparison. The detailed results can be found on https://github.com/dgspai/ASTeC-detailed-results .

Table 9 presents the average and the standard deviation of the obtained results (F1-Measure). We also compare each approach with ASTeC using Wilcoxon's Signed-Rank Test (Wilcoxon, 1945) with 5% significance as suggested by Rodríguez-Fdez et al. (2015). The highlighted results indicate the best method considering its absolute performance, and the results statistically different are marked with the $*$ symbol.

From the Table 9 it is possible to see that considering only the average results obtained by ASTeC and Google for each dataset, the proposed approach was superior to Google's approach in four datasets (FMN, TA, PMGT, and TRT) and inferior in three (AC, TP, and TSBR). However, considering the statistical test results, the proposed approach was superior to Google's approach in the AT dataset and equivalent in the remaining.

Table 9 also shows that considering only the average results, the ASTeC method has better predictive performance than the TPOT method in six datasets (AC, AT, PMGT, TP, TRT, and TSBR) and worse in one (FMN). However, considering the statistical test results, the proposed approach was superior to TPOT's method in five datasets (AC, AT, TP, TRT, and TSBR) and equivalent in the remaining.

### 6.4.2 Intent Recognition Result

In the case of the Intent Recognition task, we worked with the Weighted F1-Measure as the predictive performance metric since all of the datasets are multi-class and imbalanced. The experiments conducted with the proposed approach were executed on an Azure NCasT4_v3-series virtual machine (8 vCPUs, 56 GB memory), with Nvidia Tesla T4 GPUs with 16 GB of memory each.

Intent Recognition solutions are often used in commercial chatbots. For this application, it is usual for the predicted intent of a given sentence to go through another validation before it is outputted to the user. This validation considers the confidence score of the prediction (usually the probability score returned by the model for each prediction), comparing it to the confidence threshold set for the chatbot. If the prediction probability score is equal to or higher than the chatbot threshold, the label predicted by the model is used. Otherwise, the label "none" is returned, which means that the model did not recognize any label for that sentence. This is done because when users build a dataset for their chatbot model, the most common practice is to use only texts and classes in a given text domain (i.e., a financial chatbot can-

**Table 9.** Comparison between the F1-Measures of our approach, TPOT and the Google Cloud AutoML.

| Dataset | ASTeC | Google | TPOT |
|---|---|---|---|
| AC | 0.963 (0.024) | **0.979 (0.012)** | $*$ 0.917 (0.017) |
| AT | **0.845 (0.035)** | $*$ 0.817 (0.023) | $*$ 0.784 (0.032) |
| FMN | 0.700 (0.023) | 0.657 (0.063) | **0.701 (0.027)** |
| PMGT | **0.891 (0.108)** | 0.890 (0.125) | 0.857 (0.131) |
| TP | 0.670 (0.047) | **0.692 (0.062)** | $*$ 0.628 (0.046) |
| TRT | **0.934 (0.036)** | 0.923 (0.026) | $*$ 0.922 (0.033) |
| TSBR | 0.650 (0.107) | **0.660 (0.084)** | $*$ 0.582 (0.054) |
| Average | **0.808 (0.137)** | 0.803 (0.140) | $*$ 0.770 (0.140) |

not understand texts outside financial context). Therefore, using threshold is essential to filter out-of-domain sentences for which the chatbot is not supposed to recognize any label.

It is usually up to the user to set the confidence threshold for its chatbot. Then, to evaluate the predictive performance of each analyzed method, we calculated the Weighted F1-Measure for each model considering different threshold values, i.e., from 0 to 100 in steps of 1 percent as seen in Table 10. So, for each dataset, we have a weighted F1-Measure file, where each line contains the F1 scores for different methods considering a given threshold value. By calculating the F1 score for distinct threshold values and using the complete set of observations in the experimental comparisons, we hope to evaluate the robustness of each method against user set thresholds.

**Table 10.** F1 Score table for one dataset per Confidence Threshold example.

| method_x | method_y | method_z | threshold |
|---|---|---|---|
| 0.929 | 0.898 | 0.719 | 0.0 |
| 0.929 | 0.898 | 0.719 | 0.01 |
| ⋮ | ⋮ | ⋮ | ⋮ |
| 0.696 | 0.033 | 0.682 | 0.98 |
| 0.697 | 0.033 | 0.686 | 0.99 |
| 0.512 | 0.033 | 0.623 | 1.0 |

The results of this case study are detailed in two parts, the first concerning the Portuguese datasets and the second for the English datasets. Tables 11 and 12 present the average and the standard deviation of the obtained results (Weighted F1-Measure) for the Portuguese and English datasets, respectively. We have compared each pair of approaches using Friedman's Aligned-Rank Test (Hodges Jr and Lehmann, 1962) with 5% significance as suggested by Rodríguez-Fdez et al. (2015). The highlighted results indicate the best ranked method considering the result of the Friedman's Aligned-Rank Test, and the results statistically different from AsTeC are marked with the $*$ symbol.

As it can be observed in Table 11, regarding the weighted F1 score for the Portuguese datasets, when compared to TPOT method, ASTeC had a superior performance in 8 datasets, in which in 6 of them there is difference with statistical significance. For the remaining four datasets in which TPOT outperforms our approach, there is a significant statistical difference for only one of them. Besides, ASTeC performed better than MLJAR with statistical significance in 10 datasets, having an equivalent performance in only two datasets. We also used Friedman's Aligned-Rank Test on the Weighted F1 measures for all Portuguese datasets to found that the proposed method is placed on the first rank with a significative statistical difference.

When we move to the English datasets, the results presented in Table 12 show that, although ASTeC beats the MLJAR in all evaluated datasets, it overcomes TPOT in only one of them. According to Friedman's Aligned-Rank Test, while TPOT is the best method, there is no significant statistical difference between ASTeC's and TPOT's performance.

In our analysis the F1 measures on the result tables represent the mean of the Weighted F1-measure for each method for each dataset. Which is why in some cases, (for example, the dataset CLINK on Table 12), even thought ASTeC's mean in greater than TPOT's, TPOT turns out to be superior in the statistical test.

## 7 Conclusion

The Auto-ML area gained significant attention recently. Although most recent approaches have different optimization techniques, they usually rely on the classical k-fold cross-validation technique to evaluate the trained models.

In this work, we showed a new approach that combines GA, BO, and the bias correcting method BBC-CV. This proposition was evaluated in two scopes: in a SA task with Portuguese written texts and compared with the Google Cloud AutoML and TPOT; in an IR task using two groups of datasets, one in Portuguese and one in English, and compared with TPOT and MLJAR. For the SA task, were used seven public datasets in Portuguese, while for the IR task were chosen four public datasets in English and twelve in portuguese. These last were obtained from real chatbot IR cases from several different domains.

Regarding the Sentiment Analysis case, statistical tests were used to compare the predictive performance applied by the three approaches. Through these tests, it was possible to show that the presented method has an equivalent or superior predictive performance to Google Cloud AutoML and TPOT in the evaluated datasets. In addition, it is noteworthy to highlight the contribution of the BBC-CV and BBCD-CV methods to estimate the pipelines' predictive performances.

Considering the Intent Recognition Case, when observing

**Table 11.** Comparison between the F1-Measures for PT-BR datasets using confidence threshold.

| Dataset | ASTeC | TPOT | MLJAR |
|---|---|---|---|
| BANKING19 | **0.857 (0.144)** | ∗ 0.776 (0.191) | ∗ 0.483 (0.248) |
| BANKING10 | **0.393 (0.111)** | ∗ 0.315 (0.091) | 0.399 (0.130) |
| TELCO33 | **0.440 (0.142)** | 0.430 (0.146) | 0.447 (0.083) |
| CREDIT60 | **0.648 (0.216)** | ∗ 0.143 (0.045) | ∗ 0.636 (0.176) |
| CONSTR25 | **0.877 (0.091)** | ∗ 0.711 (0.064) | ∗ 0.531 (0.245) |
| UTILITY21 | **0.663 (0.074)** | 0.647 (0.015) | ∗ 0.180 (0.181) |
| HEALTH32 | **0.653 (0.184)** | ∗ 0.462 (0.234) | ∗ 0.561 (0.201) |
| RETAIL32 | 0.783 (0.179) | **0.845 (0.093)** | ∗ 0.501 (0.201) |
| DEBTRELIEF13 | **0.390 (0.064)** | ∗ 0.264 (0.075) | ∗ 0.164 (0.095) |
| BANKING3 | 0.310 (0.310) | **0.860 (0.120)** | ∗ 0.172 (0.181) |
| BANKING51 | 0.422 (0.096) | **0.436 (0.115)** | ∗ 0.353 (0.076) |
| TELCO8 | 0.720 (0.021) | **0.728 (0.072)** | ∗ 0.529 (0.297) |
| Average | **0.596 (0.243)** | ∗ 0.551 (0.260) | ∗ 0.413 (0.243) |

**Table 12.** Comparison between the F1-Measures for English datasets using confidence threshold.

| Dataset | ASTeC | TPOT | MLJAR |
|---|---|---|---|
| BANKING77 | 0.878 (0.086) | **0.927 (0.004)** | ∗ 0.362 (0.201) |
| CLINK | 0.925 (0.097) | **0.915 (0.116)** | ∗ 0.308 (0.137) |
| HWU64 | 0.871 (0.054) | **0.905 (0.018)** | ∗ 0.412 (0.156) |
| SNIPS | **0.957 (0.096)** | ∗ 0.964 (0.003) | ∗ 0.894 (0.103) |
| Average | 0.908 (0.092) | **0.928 (0.063)** | ∗ 0.494 (0.280) |

the full range of confidence thresholds, we found that the proposed approach poses a significant improvement from the other two analysed methods, TPOT and MLJAR, but only when working with Portuguese data. For English data, ASTeC performs better than MLJAR and equally well to TPOT. Since the text preprocessing task options in ASTeC were only available for Portuguese, there is an indication that the optimization of text preprocessing tasks are part of ASTeC's strengths and that with English options of text preprocessing the performance could increase. However the English datasets were also much larger than the ones used in Portuguese so to analyse this difference in performance it would be necessary to evaluate this adaptation of the method in a future work.

Overall, it may be said that the number of hyperparameters required to execute the proposed approach can be considered a weakness. These hyperparameters are necessary for the execution of the BBCD-CV, the GA, and the BO methods. Thus, a future work suggestion is the substitution of these methods for equivalent ones with fewer hyperparameters. For example, one can consider replacing the GA with the Covariance Matrix Adaptation Evolution Strategy (Hansen and Ostermeier, 2001). Besides that, the number of hyperparameters required by the used techniques (classifiers, feature selectors, tokenizers, etc.) greatly exceeds the number used on the ASTeC.

The adaption of the proposed approach to the distributed computing scenario could be another possible future work. This adjustment could increase the computational performance by reducing the time spent to evaluate a solution and, consequently, increasing the number of solutions evaluated in a given time interval. Finally, the proposed approach can also be evaluated for sentiment analysis in other languages, for intent recognition with more text preprocessing options for English and test the approach for other tasks involving text mining or Auto-ML.

## Acknowledgements

This work was partially funded by projects CNPq/Universal 406411/2021-2, FAPEMIG/Universal APQ-02176-21, and Partnership Agreement 006/2020 - Stilingue Inteligência Artificial Ltda - UFLA. The expansion of this study was funded by Stilingue (2023) and TakeBlip (2023) Research & Innovation department.

## References

AIworx (2017). Chocolate: A fully decentralized hyperparameter optimization framework. `https://github.com/AIworx-Labs/chocolate`. Acessado em 17 de março de 2019.

Alam, S. and Yao, N. (2019). The impact of preprocessing steps on the accuracy of machine learning algorithms in sentiment analysis. *Comput. Math. Organ. Theory*, 25(3):319–335.

Araújo, M., dos Reis, J. C., Pereira, A. C. M., and Benevenuto, F. (2016). An evaluation of machine translation for multilingual sentence-level sentiment analysis. In *Proceedings of the Annual ACM Symposium on Applied Computing*, pages 1140–1145, Pisa, Italy. ACM.

Beyer, H. and Schwefel, H. (2002). Evolution strategies - A comprehensive introduction. *Natural Computing*, 1(1):3–52.

Bird, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media, Inc., Sebastopol, Estados Unidos.

Brum, H. B. and das Graças Volpe Nunes, M. (2018). Building a Sentiment Corpus of Tweets in Brazilian Portuguese. In *Proceedings of the International Conference on Language Resources and Evaluation*, Miyazaki, Japan. ELRA.

Casanueva, I., Temčinas, T., Gerz, D., Henderson, M., and Vulić, I. (2020). Efficient intent detection with dual sentence encoders. *arXiv preprint arXiv:2003.04807*.

Cawley, G. C. and Talbot, N. L. C. (2010). On over-fitting in model selection and subsequent selection bias in performance evaluation. *J. Mach. Learn. Res.*, 11:2079–2107.

Coucke, A., Saade, A., Ball, A., Bluche, T., Caulier, A., Leroy, D., Doumouro, C., Gisselbrecht, T., Caltagirone, F., Lavril, T., et al. (2018). Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. *arXiv preprint arXiv:1805.10190*.

de Oliveira, D. N. and de Campos Merschmann, L. H. (2021). Joint evaluation of preprocessing tasks with classifiers for sentiment analysis in brazilian portuguese language. *Multimedia Tools and Applications*, 80(10):15391–15412.

de Oliveira, D. N. and de Campos Merschmann, L. H. (2022). An auto-ml approach applied to text classification. In *WebMedia '22: Brazilian Symposium on Multimedia and the Web, Curitiba, Parań, Brazil, November 7-11, 2022*. ACM.

de Sá, A. G. C., Pinto, W. J. G. S., Oliveira, L. O. V. B., and Pappa, G. L. (2017). Recipe: A grammar-based framework for automatically evolving classification pipelines. In *Proceedings of the European Conference on Genetic Programming*, pages 246–261, Amsterdam, Netherlands. Springer International Publishing.

dos Santos, F. L. and Ladeira, M. (2014). The role of text preprocessing in opinion mining on a social media language dataset. In *Proceedings of the Brazilian Conference on Intelligent Systems*, pages 50–54, São Paulo, Brazil. IEEE.

Eberhard, D. M., Simons, G. F., and Fennig, C. D., editors (2022). *Ethnologue: Languages of the World*. SIL International, Dallas, TX, USA, 25 edition.

Ferreira, R. S. (2017). Análise de sentimentos - Aprenda de uma vez por todas como funciona utilizando dados do twitter. http://minerandodados.com.br/index.php/2017/03/15/analise-de-sentimentos-twitter-como-fazer/. (Accessed on 2019 Mar 3).

Feurer, M. and Hutter, F. (2019). *Hyperparameter Optimization*, pages 3–33. Springer International Publishing, Cham.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. (2015). Efficient and robust automated machine learning. In *Proceedings of the Neural Information Processing Systems Conference*, pages 2962–2970. Curran Associates, Inc., Montreal, Canada.

Fonseca, E. R. and Rosa, J. L. G. (2013). Mac-morpho revisited: Towards robust part-of-speech tagging. In *9th Brazilian Symposium in Information and Human Language Technology (STIL)*, Fortaleza, Brasil. SBC.

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A. G., Parizeau, M., and Gagné, C. (2012). DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13.

Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., and Sculley, D. (2017). Google vizier: A service for black-box optimization. In *Conference on Knowledge Discovery and Data Mining*, pages 1487–1495, Halifax, Canada. ACM.

Google Cloud (2019). Custom machine learning models. https://cloud.google.com/automl/. (Accessed on 2019 Jun 3).

Guyon, I., Bennett, K. P., Cawley, G. C., Escalante, H. J., Escalera, S., Ho, T. K., Macià, N., Ray, B., Saeed, M., Statnikov, A. R., and Viegas, E. (2015). Design of the 2015 chalearn automl challenge. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1–8, Killarney, Ireland. IEEE.

Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9.

Hodges Jr, J. L. and Lehmann, E. L. (1962). Rank methods for combination of independent experiments in analysis of variance. *The Annals of Mathematical Statistics*, 33(2):482–497.

Huggins, M., Alghowinem, S., Jeong, S., Colon-Hernandez, P., Breazeal, C., and Park, H. W. (2021). Practical guidelines for intent recognition: Bert with minimal training data evaluated in real-world hri application. In *Proceedings of the 2021 ACM/IEEE International Conference on Human-Robot Interaction*, pages 341–350.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In Coello, C. A. C., editor, *Proceedings of the Learning and Intelligent Optimization International Conference*, volume 6683 of *Lecture Notes in Computer Science*, pages 507–523, Rome, Italy. Springer.

Hutter, F., Kotthoff, L., and Vanschoren, J., editors (2019). *Automated Machine Learning - Methods, Systems, Challenges*. Springer.

Junior, M. S. and de Campos Merschmann, L. H. (2016). A methodology to handle social media posts in brazilian portuguese for text mining applications. In *Proceedings of the Brazilian Symposium on Multimedia and the Web*, pages 239–246, Teresina, Brazil. ACM.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, page 1137–1143, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F., and Leyton-Brown, K. (2017). Auto-weka 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research*, 18.

Larson, S., Mahendran, A., Peper, J. J., Clarke, C., Lee, A., Hill, P., Kummerfeld, J. K., Leach, K., Laurenzano, M. A., Tang, L., and Mars, J. (2019). An evaluation dataset for intent classification and out-of-scope prediction. In *Proceed-*

*ings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1311–1316, Hong Kong, China. Association for Computational Linguistics.

Liu, X., Eshghi, A., Swietojanski, P., and Rieser, V. (2019). Benchmarking natural language understanding services for building conversational agents. *arXiv preprint arXiv:1903.05566*.

Martins, R. F., Pereira, A. C. M., and Benevenuto, F. (2015). An approach to sentiment analysis of web applications in portuguese. In *Proceedings of the Brazilian Symposium on Multimedia and the Web*, pages 105–112, Manaus, Brazil. ACM.

Narr, S., Hülfenhaus, M., and Albayrak, S. (2012). Language-independent twitter sentiment analysis. In *Proceedings of the Workshop on Knowledge Discovery, Data Mining and Machine Learning*, pages 12–14, Dortmund, Germany.

Olson, R. S., Bartley, N., Urbanowicz, R. J., and Moore, J. H. (2016). Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Genetic and Evolutionary Computation Conference*, New York, USA. ACM.

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

Pereira, D. A. (2021). A survey of sentiment analysis in the portuguese language. *Artificial Intelligence Review*, 54:1087–1115.

Płońska, A. and Płoński, P. (2021). Mljar: State-of-the-art automated machine learning framework for tabular data. version 0.10.3.

Ravi, K. and Ravi, V. (2015). A survey on opinion mining and sentiment analysis: Tasks, approaches and applications. *Knowledge-Based Systems*, 89:14–46.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Ribeiro, F. N., Araújo, M., Gonçalves, P., Gonçalves, M. A., and Benevenuto, F. (2016). SentiBench - a benchmark comparison of state-of-the-practice sentiment analysis methods. *EPJ Data Science*, 5(1):1–29.

Rodríguez-Fdez, I., Canosa, A., Mucientes, M., and Bugarín, A. (2015). STAC: a web platform for the comparison of algorithms using statistical tests. In *Proceedings of the 2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*.

Silva, I. S., Gomide, J., Veloso, A., Jr., W. M., and Ferreira, R. (2011). Effective sentiment stream analysis with self-augmenting training and demand-driven projection. In *Proceedings of the International Conference on Research and Development in Information Retrieval*, pages 475–484, Beijing, China. ACM.

Souza, E., Vitório, D., Castro, D., Oliveira, A. L. I., and Gusmão, C. (2016). Characterizing opinion mining: A systematic mapping study of the portuguese language. In *Proceedings of the Computational Processing of the Portuguese Language*, volume 9727 of *Lecture Notes in Computer Science*, pages 122–127, Tomar, Portugal. Springer.

Stilingue (2023). Curupira s.a. – stilingue. https://stilingue.com.br/. (Accessed on 2023 Fev 03).

TakeBlip (2023). Takenet llc. https://www.take.net/. (Accessed on 2023 Fev 03).

Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2013). Auto-weka: combined selection and hyperparameter optimization of classification algorithms. In *Conference on Knowledge Discovery and Data Mining*, pages 847–855, Chicago, USA. ACM.

Tibshirani, R. J. and Tibshirani, R. (2009). A bias correction for the minimum error rate in cross-validation. *The Annals of Applied Statistics*, 3(2):822–829.

Tsamardinos, I., Greasidou, E., and Borboudakis, G. (2018). Bootstrapping the out-of-sample predictions for efficient and accurate cross-validation. *Machine Learning*, 107(12):1895–1922.

Tsamardinos, I., Rakhshani, A., and Lagani, V. (2015). Performance-estimation properties of cross-validation-based protocols with simultaneous hyper-parameter optimization. *International Journal on Artificial Intelligence Tools*, 24(5):1–29.

Uysal, A. K. and Günal, S. (2014). The impact of preprocessing on text classification. *Information Processing and Management*, 50(1):104–112.

Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., and Zhou, M. (2020). Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83.

Xavier, C. (2018). Polarity classification of traffic related tweets. In *Proceedings of Encontro Nacional de Inteligência Artificial e Computacional*, São Paulo, Brazil.

Zhang, J.-G., Hashimoto, K., Wan, Y., Liu, Y., Xiong, C., and Yu, P. S. (2021). Are pretrained transformers robust in intent classification? a missing ingredient in evaluation of out-of-scope intent detection. *arXiv preprint arXiv:2106.04564*.