Evaluating a New Genetic Algorithm for Automated Machine Learning in Positive-Unlabelled Learning

Jack D. Saunders[1[0000-0002-0801-2909]] and Alex A. Freitas[1[0000-0001-9825-4700]]

[1] School of Computing, University of Kent, Canterbury, CT2 7NF, UK
jds39@kent.ac.uk, A.A.Freitas@kent.ac.uk

Abstract. Positive-Unlabelled (PU) learning is a growing area of machine learning that aims to learn classifiers from data consisting of a set of labelled positive instances and a set of unlabelled instances, where the latter can be either positive or negative instances, but their label is unknown. There are many PU-learning algorithms, so an exhaustive search to find the best algorithm for a given dataset is computationally unfeasible. We recently proposed GA-Auto-PU, the first Genetic Algorithm-based Automated Machine Learning system for PU learning, and reported its preliminary results. This work presents an improved version of this system with an extended search space to include spy-based techniques, and provides an extensive evaluation of the new and previous versions of this system.

Keywords: Positive-Unlabelled Learning, classification, Automated Machine Learning (Auto-ML), Genetic Algorithm

1. Introduction

Positive-Unlabelled (PU) learning is a field of machine learning that aims to learn classifiers from positive and unlabelled data [1]. This differs from binary classification due to the absence of a distinct and accurate negative set. The two instance sets present in PU learning are the *labelled* positive set, consisting of positive instances that have been labelled as positive, and the *unlabelled* set, consisting of instances which can be in reality positive or negative, but whose label is unknown. Thus, the challenge is to learn a classifier that can predict the probability of an instance belonging to the positive class, despite the mixture of negative and positive instances in the *unlabelled* set. When learning such a classifier, a PU learning algorithm is 'aware' of the difference between the concepts of unlabelled and negative instances. This allows a PU learning algorithm to try to infer which unlabelled instances are negative and which unlabelled instances are positive. By contrast, a traditional classification algorithm given PU data is 'not aware' of the difference between unlabelled and negative instances; the algorithm simply discriminates between two classes: labelled positive vs. unlabelled, so the learned classifier simply predicts the probability of an instance being labelled [2].

PU learning naturally occurs in many application domains, such as bioinformatics [2], text-mining [3], pharmacology [4], etc. [1], due to the impracticality of obtaining fully labelled data. E.g., consider datasets where the class variable represents the presence or absence of a disease. Instances with the positive class label (disease) are in

general reliable, indicating that the patient was diagnosed earlier. However, the data often also includes patients who did not undergo detailed medical examination yet, essentially 'lack of evidence for the positive class', instead of 'evidence for the negative class' (not having the disease). Hence, if those patients are labelled with the negative class (as it is usually the case), this leads to unreliable negative-class labels. PU learning avoids this by treating all those patients as 'unlabelled', to reflect their unreliability.

Many PU learning algorithms have been proposed (see [1] for a comprehensive review). Hence, finding the optimal PU learning algorithm for specific classification tasks can prove unfeasibly time consuming and computationally expensive, should one use an exhaustive search. Furthermore, algorithm predictive performance is largely dependent on the input data [1]. Therefore, the PU learning area could benefit greatly from an Automated Machine Learning (Auto-ML) system, which selects the best algorithm for a given input dataset, among a pre-defined set of candidate algorithms [5], [6].

Our previous short paper [7] recently proposed GA-Auto-PU, the first Genetic Algorithm (GA)-based Auto-ML system for PU learning, and reported its preliminary results. This work presents an improved version of this system with an extended search space and provides a much more extensive evaluation of the new and previous versions of this system, comparing their predictive performance to two popular PU learning methods on three distributions of PU data across 20 datasets, i.e., 60 PU learning problems in total; whilst only 20 PU learning problems were used in [7].

2 Background

2.1 Positive-Unlabelled (PU) Learning

PU learning is a field of machine learning that aims to learn models from datasets that consist of only positive-class and unlabelled instances [1]. PU learning shares the goal of binary classification – accurately predicting the class of an unseen instance by learning to distinguish between two classes. However, as a standard binary classifier requires a training set with two class labels, such a classifier built using PU data would have to treat all unlabelled instances as a separate class, and so will predict the probability of an instance being labelled ($\mathbf{Pr}(\boldsymbol{s} = \mathbf{1})$) instead of the probability of an instance belonging to the positive class ($\mathbf{Pr}(\boldsymbol{y} = \mathbf{1})$) [2] – where **s** is a variable taking 1 or 0 to indicate whether or not an instance is labelled, and **y** is the true label of an instance, taking values 1 or 0 to denote the positive or negative class, respectively. In contrast, PU learning models are trained to predict $\mathbf{Pr}(\boldsymbol{y} = \mathbf{1})$ using PU data and have been shown theoretically to improve upon standard binary classifiers when applied to PU datasets [8].

Within the PU learning literature, it is commonly assumed (implicitly or explicitly) that the data adheres to the selected completely at random assumption [2], stating that for the given data, $\mathbf{Pr}(\boldsymbol{s} = \mathbf{1}) = \mathbf{Pr}(\boldsymbol{s} = \mathbf{1}|\boldsymbol{x})$, where $\mathbf{Pr}(\boldsymbol{s} = \mathbf{1})$ is the probability of an instance being labelled, and **x** represents a feature vector. I.e., the labelled examples are selected from the positive distribution irrespective of their features and the labelled set is an independent and identically distributed sample from the positive distribution.

There are several approaches to PU learning, including the two-step approach, biased learning, and incorporation of the class prior [1]. The two-step approach is by far the most popular and is the focus of our proposed Auto-ML system.

The first step of this approach identifies a set of reliable negative (RN) instances among the unlabelled set; i.e., a set of instances substantially different from the labelled positive instances and likely not unlabelled positives. The second step builds a classifier to distinguish the labelled positives from the RN set. These two steps use only the training set [9]. Provided that the RN set is an accurate representation of the negative class, this model will predict $\mathbf{Pr}(\boldsymbol{y} = \mathbf{1})$ rather than $\mathbf{Pr}(\boldsymbol{s} = \mathbf{1})$.

This approach makes two assumptions [1]: (a) data separability, i.e., it is assumed that there is a natural separation between the positive and negative classes; and (b) data smoothness, i.e., it is assumed that instances that are similar to each other have a similar probability of belonging to the positive class.

An example of a two-step technique is the "Deep Forest PU" (DF-PU) method [10]. It involves training a state-of-the-art deep forest classifier on a random sample of 20% of the unlabelled instances (treated as the negative class) and all positive instances. The bottom 1% of instances with the lowest probability of belonging to the positive class are added to the RN set. This process is repeated 5 times. A classifier is then trained to distinguish the RN set and the positive set. Our system is compared against DF-PU as a recent state-of-the-art PU learning method.

We also compare our system against a well-known method: S-EM [9]. S-EM uses a spy-based approach, hiding some of the labelled positive instances in the unlabelled set and using them to determine the threshold under which an instance is considered RN. It uses the Expectation Maximisation algorithm [12] and determines the RN threshold as the predicted probability of being positive under which *l*% of spy instances fall.

Whilst the literature generally refers to two individual steps for two-step methods, this work uses slightly different terminology. We refer to the steps as phases and recognise that "Step 1" often consists of two distinct phases. Hence, when discussing two-step methods, this work references Phase I-A, which extracts an initial RN set; Phase I-B, an optional step using the initial RN set to further extract RN instances from the unlabelled set; and Phase II, "Step 2" in the usual description, which builds a classifier using the positive and RN sets. This notation is advantageous as it recognises that "Step 1" often consists of two distinct phases, and the use of "phase" rather than "step" allows us to reference the individual steps of each phase without confusion.

2.2 Automated Machine Learning (Auto-ML)

Auto-ML is an emerging field of machine learning (ML) that looks to limit the human involvement in ML applications [5], reducing the demand for domain experts and allowing those without extensive ML knowledge to operate complex ML pipelines [6]. As algorithm performance is largely dependent on input data [13], Auto-ML is a useful tool as it searches for the best algorithm specific to the target ML task.

Reference [14] proposed the Tree-based Pipeline Optimisation Tool (TPOT), an Auto-ML system using genetic programming (GP). The GP uses tree-based encoding

such that the individuals in the population are ML pipelines. The functions are pipeline operators and hyperparameters, e.g., specifying the number of trees in a random forest or the number of features selected by filter feature selection methods. The original version of TPOT uses a multi-objective optimization approach, where each individual is evaluated by both the classification accuracy and the complexity of the pipeline produced, based on the Non-dominated Sorting Genetic Algorithm II [15], drawing on the well-known concept of Pareto dominance [16], [17]. A drawback of the original version is that it can produce individuals that represent invalid pipelines, with a large computational cost in terms of evaluation [18]. This issue has been addressed by other EA-based Auto-ML systems, such as the Resilient Classification Pipeline Evolution system (RECIPE). Like TPOT, RECIPE, proposed by [18], is a genetic programming system that evolves ML pipelines. However, RECIPE uses a grammar to ensure that all generated individuals are valid, so that it does not waste resources on invalid individuals. Furthermore, RECIPE evaluates a larger search space than TPOT which, whilst making for a more complex search space, allows for a greater variety of solutions [18].

The systems described in this section are for standard binary classification and are not suitable for PU learning. Hence, we have not compared our system against any of these. Instead, we have compared against the two PU methods outlined in Section 2.1.

3 The GA-Auto-PU System

The next three Subsections define individual representation and genetic operators used by the GA. These subsections are based on the description of GA-Auto-PU in [7], with two main differences. First, the current GA version uses an extended search space of PU learning methods which includes two spy-based methods, not used in [7]. This involves an extended individual representation and a new procedure for handling instances used as "spies" (Pseudocode 3, described later). Second, in [7] the GA used a lexicographic multi-objective fitness function for optimising F-measure with higher priority and recall as lower priority (these measures are defined below). In contrast, in this current paper the GA uses a simpler fitness function, optimising the F-measure only. This simplification was made because further experiments (performed after the writing of [7]) have shown that the simpler fitness function (F-measure only) produces results qualitatively similar to the more complex lexicographic fitness function, hence the former is now used in this work. The F-measure is defined as follows:

$$\text{F-Measure} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \qquad \text{Precision} = \frac{\text{TP}}{\text{TP}+\text{FP}} \qquad \text{Recall} = \frac{\text{TP}}{\text{TP}+\text{FN}}$$

TP is true positive count, FP is false positive count, and FN is false negative count.

3.1 Individual Representation

An individual of the GA represents a candidate solution, i.e., a two-step PU learning method and its hyperparameter configuration. In both versions of the system, each two-step technique is composed of three components: phase I-A, phase I-B, and phase II. Phase I-A of the original system [7] consists of three parameters: iteration_count_1A,

threshold_1A, classifier_1A. The new proposed version of the system introduces three new parameters for phase I-A: spy_flag, spy_rate, and spy_tolerance.

The iteration count determines the number of subsets to split the unlabelled set into when learning a classifier to distinguish between the positive and the unlabelled set. E.g., if the iteration count is 5, each subset will contain 20% of the unlabelled data. This helps to handle the imbalance present in many PU learning datasets. The threshold determines the predicted probability of belonging to the positive class that an instance must fall under to be considered a reliable negative (RN) instance. In the literature, the iteration count and threshold are either set heuristically [9] or arbitrarily [10]. The previous version of the system did not generate PU learning methods with heuristic initialisation, but this has been added with the inclusion of spy-based methods. The classifier is the classifier used to predict the RN instances. Spy_flag is a Boolean value used to indicate whether or not to use a spy-based method in Phase I-A. Spy_rate determines the percentage of positive instances to use as spies. Spy_tolerance determines what percentage of spies can remain in the unlabelled set when the threshold is calculated.

Phase I-B consist of three parameters: threshold_1B, classifier_1B, and phase_1B_flag. The threshold and the classifier are analogous to those used in phase I-A. The phase_1B_flag parameter indicates whether to skip phase I-B or not. Phase I-B is not always utilised in PU learning techniques, and therefore the system will also generate individuals that are able to skip this phase. Phase II simply consists of a single parameter: classifier_2. This classifier will be trained to distinguish the positive set and the RN set. The list of the 10 genes encoded into an individual is shown in Figure 1.

iteration_ count_1A	thresh. _1A	classifier _1A	spy_ flag	spy_ rate	spy_ toler.	thresh. _1B	classifier _1B	flag _1B	classifier _1B

Fig. 1. Individual representation of the GA

The genes Classifier_1A, Classifier_1B, and Classifier_2 can take the same set of values, representing 18 different candidate classification algorithms: Gaussian naïve Bayes, Random forest, Decision tree, Multilayer perceptron, Support vector machine, Stochastic gradient descent classifier, Logistic regression, K-nearest neighbour, Deep forest, AdaBoost, Gradient boosting classifier, Linear discriminant analysis, Extra tree classifier, Extra trees classifier (an ensemble of Extra trees), Bagging classifier, Bernoulli naïve Bayes, Gaussian process classifier, and Histogram-based gradient boosting classification tree. These values are henceforth referred to as “Candidate_classifiers”.

The candidate values of each gene in the individual representation (defining the search space of PU learning algorithms and their configuration) are as follows:

- *Iteration_count_1A*: { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }
- *Threshold_1A*: { 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5 }
- *Classifier_1A*: { Candidate_classifiers }
- *Spy_flag*: { True, False }
- *Spy_rate*: { 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35 }

- *Spy_tolerance*: { 0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1 }
- *Threshold_1B*: { 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5 }
- *Classifier_1B*: { Candidate_classifiers}
- *Phase_1B_flag*: { True, False }
- *Classifier_2*: { Candidate_classifiers }

The size of the search space of the GA is thus calculated as follows:

$$10 \times 10 \times 18 \times 2 \times 7 \times 11 \times 10 \times 18 \times 2 \times 18 = 1{,}796{,}256{,}000 \text{ possible candidate solutions.}$$

PU learning solutions that do not identify any RN instances get a fitness of 0.

3.2 Outline of the underlying Genetic Algorithm (GA)

Pseudocode 1 outlines the procedure that the GA follows to evolve a PU learning algorithm. Initially, a *Population* of *Population_size* individuals is generated (step 1). The configuration (genome) of the individual is checked against a list of previously assessed configurations, and if it has not already been assessed, the *Fitness* of *Individual* is calculated (step 2.a.i) as described in Section 3.3, including Pseudocodes 2, 3, 4 and 5. If the configuration has already been assessed, the fitness values of the previous assessment are assigned to the individual (step 2.a.ii). Once all individuals have been evaluated, the fittest *Individual* is saved for the following generation (step 2.b). A new population is created from *Population* after undergoing tournament selection (step 2.c), and *New_pop* then undergoes crossover (step 2.d) and mutation (step 2.e). Finally, *Population* is set as *New_pop* and *Fittest_individual* (step 2.f). This process of fitness calculation, selection, crossover, mutation, and elitism is repeated *#generations* times. The fitness of an individual is assigned as the F-measure achieved over the 5 folds of the internal cross-validation (applied to the training set).

Pseudocode 1 Outline of the GA Procedure

1. *Population* = Generate population();
2. **Repeat** *#generations* times:
   - **a. For** each *Individual* in *Population*:
     - **i. If** *Individual* configuration has not already been assessed, **then** Assess fitness(*Individual*, *Training set*); // see Pseudocodes 2, 3, 4, 5
     - **ii. Else** *Individual Fitness* value is assigned as the output of the previous assessment;
   - **b.** *Fittest_individual* = Get fittest individual(*Population*);
   - **c.** *New_pop* = 100 individuals selected from *Population* using tournament selection;
   - **d.** Individuals in *New_pop* undergo crossover;
   - **e.** Individuals in *New_pop* undergo mutation;
   - **f.** *Population* = *New_pop* ∪ *Fittest_individual*;

Both versions of the system utilize standard uniform crossover and mutation (replacing a gene's value by a randomly sampled candidate value) as search operators.

3.3 Fitness Evaluation

Recall that each individual encodes three classification algorithms, which are used in phases I-A, I-B and II of the PU learning system. Fitness evaluation involves applying these algorithms (with the possible exception of the algorithm for the optional Phase I-B) to the training set. To describe the fitness evaluation process, we use this notation:

RN: The set of reliable negative instances.

P: The set of labelled positive instances.

U: The set of unlabelled instances.

P+RN: The combined set of labelled positive and reliable negative instances.

P(y=1): The probability of an instance being positive, as calculated by the classifier.

Pseudocode 2 Assess Fitness (Individual, Training set)

1. Split Training set into 5 Learning and Validation sets;
2. **For** each Learning set and corresponding Validation set:
   a. P = all labelled positive instances in Learning set;
   b. U = all unlabelled instances in Learning set;
   c. **If** *spy_flag* **then** RN, U = Phase I-A-Spies(P, U) // **call Pseudocode 3**, **else** RN, U = Phase I-A(P, U); **// call Pseudocode 4**
   d. **If** Phase_1B_flag **then** RN, U = Phase I-B(P+RN, U); **// call Pseudocode 5**
   e. Train Classifier_2 to distinguish P and RN;
   f. Classify Validation set;
3. Individual's Fitness Value = F-measure over the 5 Validation sets;

Pseudocode 3 Phase I-A-Spies(*P*, *U*)

1. *RN* = { };
2. *Sets* = split *U* into *Iteration_count_1A* subsets;
3. **For** every *Set* in *Sets:*
   a. *Spies* = *spy_rate%* instances, randomly selected from *P*;
   b. *Spy_set* = *Set* ∪ *Spies*
   c. Run EM(*Classifier_1A*, *P*, *Spy_set*)*;*
   d. Classify all instances in *Spy_set;*
   e. Set *threshold* to a value such that *spy_tolerance*% spies have *P(y=1)* less than *threshold*;
   f. **For** each unlabelled *Instance* in *Spy_set*:
      i. **If** *P(y=1)* < *threshold* **then** *RN* = *RN* ∪ *Instance*, *U* = *U* – *Instance;*
4. **Return** *RN, U*;

The fitness of each individual is computed as specified in Pseudocode 2. The Training set is split into 5 folds for internal cross-validation, creating 5 pairs of Learning and Validation sets (step 1). For each pair of Learning and Validation sets, all labelled positive instances are added to P (step 2.a) and all unlabelled instances are added to U (step 2.b). The RN set is determined with either the phase I-A-Spies(P, U) or phase I-A(P, U) algorithm, depending on the *spy_flag* parameter, which returns a refined U set (step 2.c, executing Pseudocode 3/4). If the flag for running phase I-B is set to true, RN and U sets are further defined with the phase I-B(P+RN, U) algorithm (step 2.d, executing

Pseudocode 5). Classifier_2 is then trained to distinguish P and RN (step 2.e), and then used to classify the Validation set (step 2.f). The Fitness Value of the Individual is assigned as the F-measure over the 5 Validation set classifications (step 3).

Pseudocode 3 describes Phase I-A of the two-phase PU learning method, executed when *spy_flag* is True. The RN set is initialised empty (step 1). The set U of unlabelled instances is split into Iteration_count_1A subsets (step 2). For each Set in the list of subsets, *Spies* is initialised with *spy_rate*% of instances of *P*, randomly selected (step 3.a) and *Set* and *Spies* are combined to form *Spy_set* (step 3.b). Next, the Expectation Maximisation (EM) algorithm is run [9], tuning Classifier_1A on *P* and *Spy_set* (step 3.c). All instances in *Spy_set* are classified and the *threshold* is set so that *spy_tolerance*% of spies have *P(y=1)* less than *threshold* (step 3.d-e). For each unlabelled *Instance* in *Spy_set* (excluding the spies), if *P(y=1)* is less than *threshold*, they are added to *RN* and removed from *U* (step 3.f). The resulting RN and U sets are then returned.

Pseudocode 4 Phase I-A(*P, U*)

1. *RN* = { };
2. *Sets* = split *U* into *Iteration_count_1A* subsets;
3. **For** every *Set* in *Sets:*
  a. Train *Classifier_1A* on *P* and *Set;*
  b. Classify all unlabelled instances in *Set;*
  c. **For** each unlabelled *Instance* in *Set*:
    i. **If** *P(y=1)* < *Threshold_1A* **then** *RN = RN* ∪ *Instance*, *U = U – Instance;*
4. **Return** *RN, U*;

Pseudocode 5 Phase I-B(*P+RN, U*)

1. Train *Classifier_1B* on *P+RN*;
2. Classify *U*;
3. **For** each *Instance* in *U*:
  a. **If** P(y=1) < *Threshold_1B*
    then *RN = RN* ∪ *Instance*, *U = U – Instance*
4. **Return** *RN, U*;

Phase I-A of the two-phase PU learning method, executed when *Spy_flag* is False, is described in Pseudocode 4. The *RN* set is initialised as an empty set (step 1). The set *U* of unlabelled instances is split into *Iteration_count_1A* subsets (step 2). For each *Set* in the list of subsets, *Classifier_1A* is trained to distinguish *P* and *Set* (step 3.a) and used to classify all unlabelled instances in *Set* (instances previously treated as the negative set during training) (step 3.b). For each unlabelled *Instance*, if the instance's calculated *P(y=1)* is less than *Threshold_1A* then *Instance* is added to *RN* and removed from *U* (step 3.c.i). The resulting *RN* and *U* sets are then returned.

Phase I-B of the two-phase learning method is described in Pseudocode 5. *Classifier_1B* is trained to distinguish the positive and reliable negative instances in *P+RN* (step 1) and the resulting classifier is then used to classify *U* (step 2). For each *Instance*

in *U*, if the *Instance's* calculated *P(y=1)* is less than *Threshold_1B*, *Instance* is added to *RN* and removed from *U* (step 3). The resulting *RN* and *U* sets are returned (step 4).

4 Datasets and Experimental Methodology

Experiments were conducted with a stratified 5-fold cross-validation procedure. Each dataset is split into 5 folds of about the same size with about the same class distribution, and then the methods are run 5 times. Each time, a different fold is used as the test set, and the other 4 folds are used as the training set. For each training set, we run the Auto-ML systems to evolve the best individual configuration. Then, a PU learning classifier is built from the training set with the configuration defined by that best individual. The classifier is then used to predict all instances in the test set. Precision, recall, and F-measure are calculated. This process is repeated for the 5 pairs of training and test sets in the 5-fold cross-validation, and the reported results are the average over the 5 test set results. Each method is evaluated using the same procedure, with the same 5 folds.

We compare the predictive performance of the two versions of the Auto-ML system, GA-Auto-PU [7] (called GA-1) and GA-Auto-PU with the extended search space (GA-2), and two well-established PU learning methods: DF-PU [10] and S-EM [9]. Performance is measured mainly by the F-measure, the most used measure in PU learning [11], but we also report a summary of precision and recall results, for completeness.

The two versions of the GA used the following hyperparameter settings. *#Generations* (number of generations) set to 50. *Population_size* (number of individuals in the population) set to 101 (100 individuals generated using crossover, 1 saved with elitism). *Crossover_prob*, the probability that two individuals will undergo crossover, set to 0.9, *Gene_crossover_prob*, the probability that a specific gene will be swapped when the individuals undergo uniform crossover, set to 0.5. *Mutation_prob*, the probability that an individual's gene will undergo mutation, set to 0.1. *Tournament_size* set to 2. Code for both versions of the GA can be found at https://github.com/jds39/GA-Auto-PU.

Regarding statistical analysis, for each performance measure (F-measure, recall, and precision), we compare the performance of the new GA-2 against GA-1 and the above two PU learning methods using the non-parametric Wilcoxon Signed-Rank test [19]. As this involves testing multiple hypotheses per measure, we use the well-known Holm correction [20] for multiple hypotheses. This procedure involves ranking the p-values from the smallest to largest (i.e., from most to least significant), and adjusting the significance level α according to the p-values' ranking. We set $\alpha = 0.05$ as usual, so the first p-value (p_1) is statistically significant if $p_1 < 0.017$. If this condition is not satisfied, the procedure stops and p_1, p_2 and p_3 are deemed non-significant. If p_1 is deemed significant, we then evaluate p_2, which is deemed significant if $p_2 < 0.025$. If p_2 is deemed significant, we then evaluate p_3, which is deemed significant if $p_3 < 0.05$.

Table 1 shows the characteristics of each dataset. These datasets are binary datasets that are engineered to PU datasets by hiding δ% of the positive class instances in the negative set, thus creating an unlabelled set – a common practice in PU learning [11]. Note that the Positive-class % column shows the percentage of positive class instances before some are hidden in the unlabelled set. δ takes the values 20%, 40%, and 60%, as it is important to test on a wide range of distributions [11].

Table 1. Dataset characteristics.

Dataset	No. instances	No. features	Positive-class %
Alzheimer's [21]	354	9	10.73%
Autism [22]	288	15	48.26%
Breast cancer Coimbra [22]	116	9	55.17%
Breast cancer Wisconsin [22]	569	30	37.26%
Breast cancer mutations [23]	1416	53	32.42%
Cervical cancer [22]	668	30	2.54%
Cirrhosis [24]	277	17	25.72%
Dermatology [22]	359	34	13.41%
Pima Indians Diabetes [22]	769	8	34.90%
Early Stage Diabetes [25]	521	17	61.54%
Heart Disease [22]	304	13	54.46%
Heart Failure [26]	300	12	32.11%
Hepatitis C [22]	590	13	9.51%
Kidney Disease [22]	159	24	27.22%
Liver Disease [22]	580	11	71.50%
Maternal Risk [22]	1014	6	26.82%
Parkinsons [22]	196	22	75.38%
Parkinsons Biomarkers [27]	131	29	23.08%
Spine [22]	311	6	48.39%
Stroke [28]	3427	15	5.25%

5 Computational Results

Table 2 shows the F-measure values achieved by all methods across the datasets for δ = 20%. GA-2 achieved the highest number of wins with 10.5, followed by GA-1 with 7.5. GA-2 outperformed DF-PU and S-EM with statistical significance (p=0.00002, Holm's adjusted α=0.017 for DF-PU; p=0.0008, Holm's adjusted α=0.025 for S-EM), but there was no significant difference between GA-2 and GA-1 (p=0.7841, α=0.05).

This trend continued for δ = 40%, as shown by Table 3. GA-2 achieved the highest number of wins with 8, followed by GA-1 with 5. GA-2 significantly outperformed DF-PU (p=0.0003, adjusted α=0.017) and S-EM (p=0.0073, adjusted α=0.025), but there was no significant difference between GA-2 and GA-1 (p=0.7562, α=0.05).

For δ = 60%, shown in Table 4, GA-1 and S-EM performed best with 6 wins each, followed by GA-2 with 5 and DF-PU with 3. GA-2 significantly outperformed DF-PU (p=0.0023, adjusted α=0.017), but there was no significant difference between GA-2 and GA-1 or S-EM (p=0.4980, adjusted α=0.025 GA-1; p=0.5706, α=0.05 S-EM). In total, across all Tables, GA-2 performed best with 23.5 wins, followed by GA-1 with 18.5, S-EM with 10.5, and DF-PU with 7.5.

Table 5 summarises the Precision and Recall values achieved by each method, showing the number of wins (out of 20 datasets) of each method and whether GA-2 performed statistically significantly better or worse than another method, for each measure and for each δ = 20%, 40%, 60% (the full results per dataset are not shown due to lack

of space). In terms of recall, DF-PU performed best overall, with GA-2 performing worst. GA-2 performed significantly worse than DF-PU for all 3 δ values. However, DF-PU generally predicted almost all instances as positive, thus achieving near 100% recall, but near 0% precision. Such classification is unhelpful, representing a bad trade-off between precision and recall, which led to the inferior results for DF-PU regarding F-measure, as shown earlier. GA-2 performed best in terms of Precision, significantly outperforming both DF-PU and S-EM for the 3 δ values.

Table 2. F-measure values achieved by the four methods for $\delta = 20\%$.

Dataset	GA-1	GA-2	DF-PU	S-EM
Alzheimer's	0.529	**0.548**	0.195	0.321
Autism	0.960	**0.982**	0.648	0.820
Breast cancer Coi.	0.705	**0.711**	0.697	**0.711**
Breast cancer Wis.	0.954	**0.956**	0.543	0.898
Breast cancer mut.	0.893	**0.896**	0.489	0.892
Cervical cancer	0.828	**0.867**	0.061	0.054
Cirrhosis	**0.573**	0.446	0.405	0.436
Dermatology	0.860	**0.901**	0.228	0.718
Pima I. Diabetes	**0.677**	0.642	0.516	0.534
Early Diabetes	0.958	**0.978**	0.761	0.792
Heart Disease	**0.843**	0.836	0.705	0.811
Heart Failure	**0.770**	0.751	0.487	0.529
Hepatitis C	**0.953**	0.944	0.176	0.695
Kidney disease	0.976	0.925	0.428	**1.000**
Liver disease	**0.834**	0.831	**0.834**	0.816
Maternal health	0.476	**0.862**	0.403	0.454
Parkinson's	0.860	**0.935**	0.856	0.815
Parkinson's Biom.	**0.476**	0.282	0.354	0.333
Spine	0.652	**0.923**	0.652	0.820
Stroke	**0.474**	0.241	0.086	0.102
Total wins	7.5	**10.5**	0.5	1.5

Table 3. F-measure values achieved by the four methods for $\delta = 40\%$.

Dataset	GA-1	GA-2	DF-PU	S-EM
Alzheimer's	0.551	**0.576**	0.194	0.370
Autism	0.927	**0.940**	0.648	0.841
Breast cancer Coi.	0.687	0.671	**0.711**	0.704
Breast cancer Wis.	0.932	**0.936**	0.543	0.903
Breast cancer mut.	0.868	0.739	0.489	**0.893**
Cervical cancer	**0.903**	0.839	0.042	0.053
Cirrhosis	**0.464**	0.397	0.401	0.442
Dermatology	0.780	**0.896**	0.229	0.718
Pima I. Diabetes	**0.649**	0.646	0.516	0.526
Early Diabetes	**0.895**	0.887	0.756	0.859
Heart Disease	0.801	0.780	0.705	**0.828**
Heart Failure	0.652	**0.670**	0.486	0.508
Hepatitis C	0.771	**0.863**	0.171	0.708
Kidney disease	0.988	0.951	0.428	**1.000**
Liver disease	0.803	0.817	**0.832**	0.587
Maternal health	0.812	**0.813**	0.395	0.434
Parkinson's	0.836	0.843	**0.860**	0.748
Parkinson's Biom.	0.265	0.259	**0.354**	0.261
Spine	0.907	**0.917**	0.652	0.839
Stroke	**0.255**	0.239	0.094	0.102
Total wins	5	**8**	4	3

The performance of GA-2 did come at a computational cost, compared with DF-PU and S-EM. Whilst DF-PU and S-EM took about 4.9 minutes and 1.5 minutes on average per dataset respectively, GA-2 took about 3.7 hours. As such, GA-2 was 45x slower than DF-PU, and 150x slower than S-EM. All experiments were run on a 48 core GPU with 256GB of memory.

Table 4. F-measure values achieved by the four methods for $\delta = 60\%$.

Dataset	GA-1	GA-2	DF-PU	S-EM
Alzheimer's	0.456	**0.529**	0.1711	0.3727
Autism	0.910	**0.927**	0.6447	0.8351
Breast cancer Coi.	0.510	0.553	0.6966	**0.6988**
Breast cancer Wis.	**0.906**	0.866	0.5392	0.9038
Breast cancer mut.	0.854	0.872	0.4853	**0.8922**
Cervical cancer	**0.714**	0.350	0.0444	0.0459
Cirrhosis	0.443	0.204	0.4046	**0.4589**
Dermatology	**0.828**	0.692	0.2194	0.7188
Pima I. Diabetes	0.606	**0.634**	0.5145	0.5442
Early Diabetes	**0.930**	0.894	0.7589	0.7925
Heart Disease	0.785	0.786	0.7024	**0.8290**
Heart Failure	**0.674**	0.671	0.4815	0.5571
Hepatitis C	0.588	**0.610**	0.1596	0.6087
Kidney disease	0.754	0.806	0.4279	**0.9512**
Liver disease	0.804	0.748	**0.8338**	0.7880
Maternal health	0.735	**0.738**	0.3895	0.4380
Parkinson's	0.818	0.792	**0.8596**	0.7619
Parkinson's Biom.	0.233	0.280	**0.3671**	0.3310
Spine	0.818	0.761	0.6522	**0.8300**
Stroke	**0.255**	0.243	0.0937	0.1022
Total wins	**6**	5	3	**6**

Table 5. Summary of Recall and Precision results across all datasets.

	Number of wins regarding Recall				Statistical significance results (> means better, < means worse)
δ	GA-1	GA-2	DF-PU	S-EM	
20%	3.83	0	10.33	5.83	GA-2 < DF-PU (p=0.00001)
40%	0	0	14.5	5.5	GA-2 < DF-PU (p=0.00002) GA-2 < S-EM (p=0.0136)
60%	0	0	14	6	GA-2 < DF-PU (p=0.000002) GA-2 < S-EM (p=0.00001)
δ	Number of wins regarding Precision				Statistical significance results
20%	7.83	10.83	0	1.33	GA-2 > DF-PU (p=0.00001) GA-2> S-EM (p=0.0003)
40%	7.33	11.33	0	1.33	GA-2 > DF-PU (p=0.001) GA-2 > S-EM (p=0.001)
60%	9.33	10.33	0	0.33	GA-2 > DF-PU (p=0.00001) GA-2 > S-EM (p=0.0001)

6 Conclusions

We recently proposed GA-Auto-PU, the first GA-based automated machine learning method for PU learning [7]. In this work we presented an improved version of the system which features an extended search space, incorporating spy-based heuristic methods into the genes of the individuals, which allows the creation of more sophisticated

PU learning algorithms. This new version of GA-Auto-PU was extensively compared against two established and well-performing PU learning methods, as well against the previous version of the system, across three distributions of engineered PU learning data in 20 datasets (representing in total 60 different PU learning problems). The new version of the system outperformed the previous version in general, and the new version outperformed the PU learning baselines with statistical significance in regard to F-measure, the most used performance measure in PU learning [11]. An analysis of the results for recall and precision (used to compute the F-measure) showed that the new system significantly outperforms the two baseline methods regarding precision, but it is significantly outperformed by the two baselines in most cases regarding recall.

Future work will look to explore other search and optimisation methods, such as Bayesian Optimisation, as well as expanding the GA's search space to include other types of PU learning methods.

References

1. Bekker, J. and Davis, J., 2020. Learning from positive and unlabeled data: A survey. *Machine Learning*, 109(4), 719-760.
2. Elkan, C. and Noto, K., 2008, August. Learning classifiers from only positive and unlabeled data. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 213-220.
3. Li, X. and Liu, B., 2003. Learning to classify texts using positive and unlabeled data. In *Proceedings of the 18th International Joint Conf. on Artificial Intelligence.* 3, 587-592.
4. Zheng, Y., Peng, H., Zhang, X., Zhao, Z., Gao, X. and Li, J., 2019. DDI-PULearn: a positive-unlabeled learning method for large-scale prediction of drug-drug interactions. *BMC Bioinformatics*, *20*(19), 1-12.
5. Yao, Q., Wang, M., Chen, Y., Dai, W., Li, Y.F., Tu, W.W., Yang, Q. and Yu, Y., 2018. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306*.
6. He, X., Zhao, K. and Chu, X., 2021. AutoML: A Survey of the State-of-the-Art. *Knowledge-Based Systems*, *212*, article 106622.
7. Saunders, J.D. and Freitas, A.A., 2022. GA-Auto-PU: A Genetic Algorithm-based Automated Machine Learning System for Positive-Unlabeled Learning. In *Proceedings of the GECCO'22 Companion (Genetic and Evolutionary Computation Conf.)*, 288-291. ACM.
8. Niu, G. du Plessis, M., Sakai, T., Ma, Y., and Sugiyama, M. 2016. Theoretical comparisons of positive-unlabeled learning against positive-negative learning. In *Proceedings of the 30th International Conf. on Neural Information Processing Systems (NIPS'16)*, 1207–1215.
9. Liu, B., Dai, Y., Li, X., Lee, W.S. and Yu, P.S., 2003. Building text classifiers using positive and unlabeled examples. In *Proceedings of the Third IEEE International Conference on Data Mining*, 179-186.
10. Zeng, X., Zhong, Y., Lin, W. and Zou, Q., 2020. Predicting disease-associated circular RNAs using deep forests combined with positive-unlabeled learning methods. *Briefings in Bioinformatics*, 21(4), 1425-1436.
11. Saunders, J.D. and Freitas, A.A, 2022. Evaluating the Predictive Performance of Positive-Unlabelled Classifiers: a brief critical review and practical recommendations for improvement. *arXiv preprint arXiv:2206.02423*.
12. Dempster, A., Laird, N.M., & Rubin, D., 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B, 39*, 1-38.

13. Brazdil, P., Carrier, C.G., Soares, C. and Vilalta, R., 2008. Metalearning: Applications to data mining. Springer Science & Business Media.
14. Olson, R.S., Bartley, N., Urbanowicz, R.J. and Moore, J.H., 2016, July. Evaluation of a tree-based pipeline optimization tool for automating data science. *In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2016)*, 485-492.
15. Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T.A.M.T., 2002. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation, 6(2)*, 182-197.
16. Deb, K. Multi-objective Optimization Using Evolutionary Algorithms. Chichester: John Wiley & Sons, 2001.
17. Freitas, A.A., 2004. A critical review of multi-objective optimization in data mining: a position paper. *ACM SIGKDD Explorations Newsletter, 6(2)*, 77-86.
18. de Sá, A.G., Pinto, W.J.G., Oliveira, L.O.V. and Pappa, G.L., 2017. RECIPE: a grammar-based framework for automatically evolving classification pipelines. *In Proceedings of the European Conference on Genetic Programming*, 246-261.
19. Wilcoxon, F., Katti, S.K. and Wilcox, R.A., 1963. Critical values and probability levels for the Wilcoxon rank sum test and the Wilcoxon signed rank test. Selected Tables in Mathematical Statistics, 1, 171-259.
20. Demšar, J., 2006. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research, 7*, 1-30.
21. Marcus, D.S., Fotenos, A.F., Csernansky, J.G., Morris, J.C. and Buckner, R.L., 2010. Open access series of imaging studies: longitudinal MRI data in nondemented and demented older adults. *Journal of Cognitive Neuroscience, 22(12)*, 2677-2684.
22. Asuncion, A. and Newman, D., 2007. UCI machine learning repository.
23. Pereira, B., Chin, S.F., Rueda, O.M., Vollan, H.K.M., Provenzano, E., Bardwell, H.A., Pugh, M., Jones, L., Russell, R., Sammut, S.J. and Tsui, D.W., 2016. The somatic mutation profiles of 2,433 breast cancers refine their genomic and transcriptomic landscapes. *Nature Communications, 7(1)*, 1-16.
24. Fleming, T.R. and Harrington, D.P., 1991. Counting Processes and Survival Analysis. New York: John Wiley and Sons Inc.
25. Islam, M.F., Ferdousi, R., Rahman, S. and Bushra, H.Y., 2020. Likelihood prediction of diabetes at early stage using data mining techniques. *In Computer Vision and Machine Intelligence in Medical Image Analysis*, 113-125.
26. Chicco, D. and Jurman, G., 2020. Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone. *BMC Medical Informatics and Decision Making, 20(1)*, 1-16.
27. Hlavnička, J., Čmejla, R., Tykalová, T., Šonka, K., Růžička, E. and Rusz, J., 2017. Automated analysis of connected speech reveals early biomarkers of Parkinson's disease in patients with rapid eye movement sleep behaviour disorder. *Scientific Reports, 7(1)*, 1-13.
28. Emon, M.U., Keya, M.S., Meghla, T.I., Rahman, M.M., Al Mamun, M.S. and Kaiser, M.S., 2020. Performance Analysis of Machine Learning Approaches in Stroke Prediction. *In 2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, 1464-1469.