Optimizing Ensemble Selection with Evolutionary Algorithms

Seminar Paper submitted

to

Prof. Dr. Stefan Lessmann

Humboldt-Universität zu Berlin
School of Business and Economics
Chair of Information Systems

by

Lara Vomfell (573365)
Maria Kozlova (561736)

Applied Predictive Analytics

Berlin, July 15, 2016

Contents

1 Introduction

When confronted with a problem, you go around and ask experts on their opinion. In an overly simplified sense, that is how Ensemble Selection (ES) works.

Given a specific data forecasting problem, it is not trivial to decide which predictive model to use, on what metric it should be evaluated or what the optimal parameter settings are, especially when different models make different errors and have varying performances on different metrics. With Ensemble Selection, one trains a number of models and combines their predictions to retrieve a final prediction, much like people intuitively do when asking around for opinions. Ensemble Selection was first introduced by Dasarathy and Sheela (1979) and has been researched extensively since the 1990s under various names, e.g. committee of learners, classifier ensemble, consensus group or mixture of experts. The combination of different models, called classifiers in this paper, that make different errors into one ensemble typically improves the accuracy. This, however, immediately raises two questions, 1. which models should be included in the ensemble and 2. how? Not all classifier combinations yield better results than a single well-tuned classifier. As such, the combination needs to be optimized to maximize the overall accuracy. One could theoretically use an exhaustive search on all possible combinations, driving the computational cost through the roof. Other search algorithms are feasible with satisfying solutions. In this paper, different Evolutionary Algorithms (EAs) are employed to find the optimal ensemble.

Evolutionary Algorithms are a problem-solving technique where populations of solutions to a problem are developed similarly to behavior observed in nature. Their heuristic nature makes them a convenient and suitable application in many problems, Ensemble Selection among them.

Specifically, this paper investigates the synthesis of Evolutionary Algorithms with a novel bagged Ensemble Selection approach proposed by Caruana et al. (2004). To this end, four evolutionary algorithms are applied to data on fashion retail returns made available during the DATA MINING CUP competition 2016. Additionally, they are combined with bagging and compared to several baseline evaluations. The paper is organized as follow: Section 2 introduces the technique of Ensemble Selection, Section 3 explains the general mechanism behind Evolutionary Algorithms and how they can be used for Ensemble Selection, Section 4 expands on the algorithms used during the empirical study which is explain in Section 5. The results and some final conclusions are presented in Sections 6 and 7.

2 Ensemble Selection

Any supervised predictive model makes an assumption on the distribution underlying the data x and uses this approximation to the true distribution to infer from training data to predict the labels y of new data. During the training of the model, a trade-off between fit and generalization has to be considered. A classifier might be able to fit training data perfectly but fail to generalize to new data and vice versa.

One way of mitigating this issue is the construction of an ensemble which is "a set of classifiers whose individual decisions are combined in some way (typically by weighted or unweighted voting) to classify new examples." (Dietterich, 2000, p. 1) There are three main explanations as to why model ensembles tend to perform better than single classifiers: the first (*statistical*) reason the weighting of classifiers averages different assumptions on the data and therefore approximates the true data generating function better than a single assumption could. The second (*computational*) reason is that single classifiers tend to find local optima and combining their predictions has a higher potential of approximating the global optimum. The third (*representational*) reason asserts that not all types of classifiers are able to represent the true data generating process or at least cannot consider all representations in a finite sample (Dietterich, 2000, p. 2–3).

Following the notation by Tsoumakas et al. (2008), the problem of Ensemble Selection (ES) can be represented as follows: let x be realizations of X and let $h_i, i = 1, \ldots, k$ be a set of base classifiers that return probabilities $m_i(x, c_j)$ for $j = 1, \ldots, d$ for each label c_j. The ensemble then outputs:

$$y(x) = \arg\max \sum_{i=1}^{k} w_i m_i(x, c_j), \tag{1}$$

where w_i is the weight for classifier m_i. One seeks to optimize w such that the predictions are as accurate as possible (Tsoumakas et al., 2008, p. 2).

ES relies on two key assumptions: the single classifiers have to be 1. accurate and 2. diverse. Accuracy means that the classifier has be at least better than a random guess. Diversity means that the single classifiers have different decision boundaries, i.e. produce different predictions. Diversity among the classifiers helps achieve the averaging of different types of errors (Brown et al., 2005). An idealized visualization of this assumption can be seen in Figure 1. So far, there is no canonical definition of diversity and one can choose between a number of different measures. The question of whether more diverse ensembles always produce better predictions cannot be answered with certainty as the empirical results

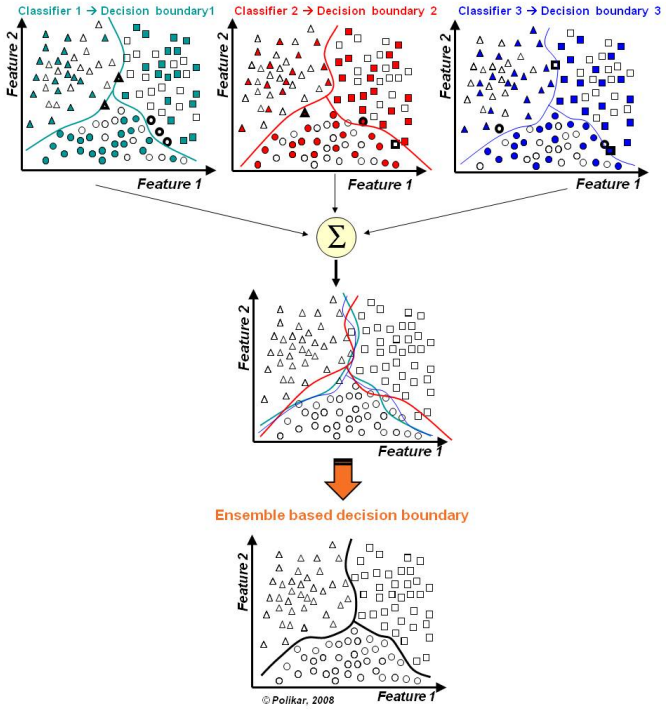

Figure 1: The combination of classifiers with different decision boundaries improves predictive power (Polikar, 2008).

are not straightforward (Kuncheva and Whitaker, 2003, p. 203). For an extensive overview see Kuncheva and Whitaker (2003) and Kuncheva (2005).

Bagging, **B**ootstrap **agg**regat**ing**, originally refers to the training of classifiers of one type on random resamples of the training data (Breiman, 1996). For the training of k classifiers, k training sets of size N are generated by drawing from the original data set of size N with replacement. While the individual classifiers' errors on the test set increase with that technique due to overfitting to the samples that occur more often in the training set, the combination of these overfitted classifiers makes the ensemble so strong Krogh (1996).

An alternative to Bagging is Boosting where in a sequence of classifiers, incorrectly classified samples are resampled more often to improve the accuracy of the single classifier (Opitz and Maclin, 1999, p. 173). Since 1996 both Bagging and Boosting have been expanded and applied at different steps of the classifier learning and final ES. Opitz and Maclin show that combining different classifiers that are trained on the same data, too, produces very good results (Opitz and Maclin, 1999, p. 170).

One potential issue in ES is overfitting, i.e. the inclusion of many classifiers in the ensemble may produce good results on the training set but result in poor performance on the test set

(Sun and Pfahringer, 2011, p. 253), (Cawley and Talbot, 2010, p. 2092). To prevent this, Caruana et al. (2004) present a novel approach. They extend the simple iterative inclusion or exclusion of classifiers to an ensemble in three ways:

Selection with Replacement Classifiers can be included in the ensemble several times meaning that good classifiers can be picked again instead of poorer ones when the initial top t models have already been selected.

Sorted Ensemble Initialization Instead of filling an empty ensemble, the initial ensemble is already comprised of the t best classifiers and filled from there on.

Bagged ES In this case, instead of bagging the training data, subsets of classifiers are created. One then optimizes the classifier combination within the subset and results across all subsets are averaged to obtain the final ensemble.

As a demonstration of bagged ES, imagine that one draws 15 samples from a classifier library of 1000 with a probability $p = 0.5$ such that each sample or bag contains 500 classifiers whose combination is then optimized with, for example, forward-selection. The 15 obtained sub-ensembles are then combined (Caruana et al., 2004, p. 3). The empirical results of Caruana et al. point strongly towards setting the ratio of models per bag to 0.5 as it shows the strongest advantage over the baseline best model. They also show that ensemble selection works extremely well with any type of performance metric, i.e. accuracy, ROC, root mean-squared error, etc. and accounted for one third of the total 8.7% outperformance of ES over the best model alone. Caruana et al. optimized the ES with forward stepwise selection where one iteratively adds classifiers until the performance converges. However, any selection or optimization procedure is possible. Linear methods include taking the simple average, the weighted average, or the median. Voting covers majority voting, plurality voting for multi-class problems, or weighted voting. For a more detailed view on ensemble combination see Chapter 4 in Zhou (2012). Usually, the classifiers are combined with regards to some objective function, the ensemble accuracy for example.

The problem of optimizing the classifier combination with regards to some objective function is a non-linear problem. A possible search heuristic for finding the optimal classifier combination are Evolutionary Algorithms, a class of algorithms that emulate different behavior found in nature: the evolution of genes or individuals over time or the movement of swarms or flocks. EAs solve the initial questions of which models to include and how to combine them heuristically and have shown competitive results in the past (Oliveira et al.,

2005).

3 Using Evolutionary Algorithms for Ensemble Selection

Ensemble Selection is essentially an optimization problem in a constrained search space. In order to find an effective solution, optimization strategies must exploit all available information. One can use derivative-based information, i.e. on the gradient of the search space. In the case of complex spaces with many local optima, these methods tend to get stuck and miss global solutions. Evolutionary Algorithms capitalize on a different strategy that often makes a suitable trade-off between exploration and exploitation of the search space. Hence, they are a search heuristic worthwhile exploring for ensemble selection.

3.1 Introduction to Evolutionary Algorithms

Evolutionary Algorithms are nature-inspired optimization algorithms. The idea of emulating natural selection and evolution emerged very early into the development of computational optimization; already in the 1950s, first evolution systems were employed to optimize real-valued functions (Mitchell, 1999, p. 2). The first genetic algorithm that mimics the way genes on a chromosome are encoded, was developed by John Holland in the 1960s. In his book *Adaptation in natural and artificial systems*, Holland presented an advanced, more abstract version of the GA that is able to solve many different problem along with a theoretical framework of how GAs optimize (Holland, 1975). Genetic Algorithms still follow the structure presented by Holland. As many terms are derived from biology and might deserve explanation, some concepts and vocabulary are briefly here. EAs are *population*-based, where the population is a group of candidate solutions to the problem at hand. The population can also be called swarm and its size is denoted by N. The candidate solution is usually called *chromosome*, but also particle, individual or string. It can be a binary-encoded or a real-valued numeric vector. The candidate solution consists of positions of *genes*, *loci* and contains as many loci as the problem has dimensions. The value at each locus is called *allele*. As an example, in the case of ensemble selection, the problem dimension is equal to the number of models D in the model library. So each chromosome x consists of D genes and the value at x_i corresponds to the weight assigned to model $i \in 1, \ldots, D$.

Typically, EAs consist of five stages which are explained here with the example of the canonical Genetic Algorithm: 1. Initialization, where the initial population of dimensions $N \times D$ is randomly created. 2. Evaluation: its fitness is evaluated on some cost function to

be optimized. 3. Selection: a fixed number of best performing chromosomes is selected for the next stages. 4. Crossover: two chromosomes are randomly selected and combined to produce offspring. 5. Mutation: the offspring is randomly mutated and included in the population. The steps 2–5 are repeated until a stopping criterion is reached, e.g. the optimum, a fixed number of generations or a convergence criterion.

This basic setup can be varied and extended in numerous ways. Instead of random selection, one can implement fitness-proportionate selection or roulette-wheel selection, where better performing solutions are assigned a higher probability of selection. An alternative is tournament selection where randomly chosen chromosomes "perform again" each other and the best individuals are selected (Miller and Goldberg, 1995). There are many options on how to combine the parents to produce offspring, the simplest one being one-point crossover where both parents are split at a random point $i \in 1, \ldots, D$ and the first child inherits the characteristics of the first parent up until i from where on it posses the second parent's properties. The inverse is applied to the second child. Another common modification is elitism where a fixed number of best evaluated chromosomes in each generation are preserved and replace the worst performing chromosomes in step 5 and are directly transferred to the next generation.

EAs are often used in the following scenarios: the problem is very complex or high-dimensional. Also, because they work heuristically and not analytically, discontinuous or non-differentiable objective functions are not an issue to their performance as opposed to derivative-based algorithms. Also, as EAs can deal with the exploration/exploitation trade-off quite well so that search spaces with many local optima do not necessarily impede their convergence. Nevertheless, one needs to bear in mind that EAs do not find the analytical function optimum. With finite populations, they may never reach the global optimum or lose it over time due to mutation. EAs are best applied in situations where a reasonably satisfying solution in a reasonable time frame is sufficient. As De Jong argues: the genetic algorithm "is attempting to maximize cumulative payoff from arbitrary landscapes, deceptive or otherwise. In general, this is achieved by not investing too much effort in finding cleverly hidden peaks (the risk/reward ratio is too high)." (De Jong, 1993, p. 15). This notion can be transferred to all EAs (Mitchell, 1999, p. 92).

3.2 Optimizing Ensemble Selection

When using EAs, two fundamental questions need to be decided upon at the start: the function to be optimized and the encoding. EAs can be binary-encoded, meaning that each gene on the chromosome is either 0 or 1. In the case of Ensemble Selection, a 0 at x_i means that the model $i \in 1, \dots D$ is not included and vice versa. For the evaluation the 1's are divided by their sum, i.e. an exemplary vector $x = 01101$ of dimensions (1×5) is divided by the number of 1's, 3 in this case, such that the weights sum up to 1. As an alternative, the solution vector can be real-valued or numerical. Instead of 0/1, the value at x_i can be any value within the boundary, which in the case of ES is confined to $[0, 1]$. As the weights need to sum up to 1, they are divided by their cumulative sum. As an example, a preliminary solution vector $x = \{0.1, 0.5, 0.8, 0.2, 0.05\}$ is divided by its sum 1.65 so that the final solution vector is $x = \{0.06, 0.3, 0.48, 0.12, 0.03\}$. To prevent the EAs from including a large number of arbitrarily small weights, sparsity rules can be enforced where weights smaller than 1% can be excluded. Depending on the problem and the encoding, one potential issue with EAs is the production of invalid solutions, as especially the mutation step can change solutions such that they are outside of the defined constraints. In the case of ensemble weights, these constraints are $0 \leq w_i \leq 1$ and $\sum_{i=1}^{D} w_i = 1$.

EAs in general have a tendency to overfit and there is some experimental evidence that they overfit ensemble selection problems (Robilliard and Fonlupt (2002), Radtke et al. (2006), Dos Santos et al. (2009)). Complex validation strategies can be implemented to prevent this; their performance however is not overly effective and the effect size of overfitting is rather small (Dos Santos et al., 2008, p. 1424). In this paper, bagging as described by Caruana et al. (2004) is implemented to test whether a) overfitting is an issue and b) if it can be mitigated by bagging and c) whether bagging improves the solutions.

4 Algorithms implemented

While various EA are suitable for optimizing ES, the final selection for this study consists of the Genetic Algorithm (GA), the Backtracking Search Algorithm (BSA), Population-based Incremental Learning (PBIL) and Particle Swarm Optimization (PSO). As a benchmark of performance Hil-Climbing and Stochastic Hill-Climbing are implemented.

4.1 Genetic Algorithm

As the Genetic Algorithm (GA) has already been explained in its canonical form in Section 3.1 as a prototype of EAs, it will not be dealt with in much detail. For the ES case, a GA with elitism, one-point crossover and roulette-wheel selection is implemented.

4.2 Backtracking Search Optimization Algorithm

The Backtracking Search Optimization Algorithm (BSA) is comparatively new algorithm that was introduced in 2013 to solve numerical problems (Civicioglu, 2013). Out of all the algorithms taken into account, it can be considered the"sleekest" with only one control parameter. Additionally to the GA phases, a second Selection stage is implemented at the very beginning where an old population is maintained such that the BSA population preserves a memory. After initialization follows the new selection stage where the historical population is randomly updated according to *if* $a < b \,|\, a, b \sim U(0,1)$. It is then permuted and Mutation and Crossover follow. The population is mutated based on Equation (2):

$$Mutant = Population + F \cdot (historical\ Population - Population) \tag{2}$$

where $F = 3 \cdot \varepsilon$, $\varepsilon \sim N(0,1)$. The original BSA is developed for numerical, real-valued problems but it can easily be adapted to binary encoding.

4.3 Population-Based Incremental Learning

The Population-Based Incremental Learning Algorithm (PBIL) by Baluja (1994) can be considered an abstraction of the GA as it drops the population. Instead, it uses a probability vector from which population samples are drawn. This vector denotes the probability of a certain value appearing at a certain position of the solution vector. In a binary setting of the PBIL, the probability vector of a population denotes the probability of a 1 appearing at position i of the candidate solution x. Instead of recombination to obtain solutions with

high fitness, the probability vector from which the population is drawn is gradually shifted towards solutions with high fitness (Baluja, 1994, p. 11). The vector is initialized with 0.5 and is updated by the following rule:

$$p_i = p_i \cdot (1 - \gamma) + (\gamma \cdot x_i), \tag{3}$$

where p_i is the probability of generating a 1 bit at position i, x_i is the solution vector at position i towards which the probability vector at i is moved and γ is the learning rate. The learning rate affects how quickly the probability vector moves towards a feasible solution, i.e. how informative the previous search is. As such, it controls the trade-off between exploration and exploitation and premature convergence. It is usually suggested to set the learning rate to values around 0.1, as higher values lead to an early search focus whereas lower values enable greater search exploration, which is preferable when there is no certainty on the prevalence of local optima. (Baluja, 1994, p. 16–17). It should be noted that the PBIL and the probability vector so far are as described in Baluja (1994), meaning that only binary encoding is considered. The PBIL, however, can be adapted to real-valued encoding, thus making the implication of a probability vector more complicated.

As a solution, Servet et al. (1997) suggest an interval-based probability vector. It is assumed that the solution is continuous and bounded, such that for each position i, an bounded interval $[low_i, up_i]$ can be defined. The value at position i of the probability vector then denotes the probability that x_i is greater than a threshold. The initial threshold is defined as $\frac{low_i + up_i}{2}$ and is updated throughout the iterations. If $p_i \geq 0.9$, i.e. the probability that x_i is greater than the threshold is ≥ 0.9, the threshold is updated to $\frac{\frac{low_i + up_i}{2} + up_i}{2}$, i.e. low_i is updated to the previous threshold and p_i is reinitialized at 0.5. Both p_i and the up_i are updated equivalently whenever $p_i \leq 0.1$. In the case of ES in this pape, the interval is slightly redefined from $\frac{low_i + up_i}{2}$ to $\frac{low_i + up_i}{\sum i}$ such that the initial probability assumes equal weights.

4.4 Particle Swarm Optimization

The Particle Swarm Optimization Algorithm (PSO) draws on swarm intelligence to find the optimum. The candidate solutions, called particles, move through the search space by random mutations, called velocities. The velocity of each particle consists of three components: an inertia weight, i.e. the "unwillingness" of a particle to leave its current position which also prevents it from moving too erratically, a cognitive component, i.e. the particle's ability to

remember a previous good solution, and a social component, i.e. the particle's ability to move towards optimal solutions found by its neighbors in the swarm. The PSO was introduced in 1995 by Kennedy and Eberhart and simulates the social behavior of bird flocking or fish swarming. The particles evolve by the following equations:

$$v_{i,t+1} = \gamma v_{i,t} + \varphi_1 \delta(0,1)(P_i - x_{i,t}) + \varphi_2 \delta(P_g - x_{i,t}) \tag{4}$$

$$x_{i,t+1} = x_{i,t} + v_{i,t+1} \tag{5}$$

where $\delta \sim U(0,1)$, $x_{i,t}$ is the value at position i and time step t of the candidate solution x, P_i refers to the previously best personal solution and P_g denotes the best solution in the neighborhood.

The inertia weight γ controls how fast each particle moves and was not in the proposition for the original PSO (Kennedy and Eberhart, 1995). It was added three years later as it adds significantly to the algorithm's ability to search and exploit the search space (Shi and Eberhart, 1998, p. 70). For $\gamma > 1.2$ the swarm keeps on exploiting new search areas, for $\gamma < 0.8$ it exploits local areas more and finds the balance between the two for values $0.8 < \gamma < 1.2$ However, experimental studies have shown that non-fixed γ values also perform extremely well and that the so called random inertia weight $\gamma = 0.5 + \delta/2$ performs well on a number of different problems (Bansal et al., 2011, p. 643). φ_1 and φ_2, the acceleration coefficients control the influence of information on the movement of the particles. Usually, one sets $\varphi_1 = \varphi_2$ to values around 2 (Kennedy and Eberhart, 1995, p. 1946).

Additionally, the information neighborhood of each particle, also called topology, is not a trivial setup as it determines how much information is available to each particle. The two most common ones are the *lbest* and *gbest* topologies but there are many other variants. In the *lbest*, each particle i is informed on its own position and its two adjacent neighbors, resulting in a ring lattice (see Figure 2). In a *gbest* setting, which is also the setting in the initial algorithm, the particles are informed by the global neighborhood. The information flow in this social network influences the convergence behavior strongly: In a study by Kennedy and Mendes (2002), different topologies were evaluated and *gbest* performed worst in both accuracy and speed, while *lbest* performed slightly better when removing particle i (the self) from the information (Kennedy and Mendes, 2002, p. 1674). For this paper, both the *gbest* and *lbest* without self are tested; more complex topologies von Neumann neighborhood were not considered due to the popularity of the former and the complexity of the latter.

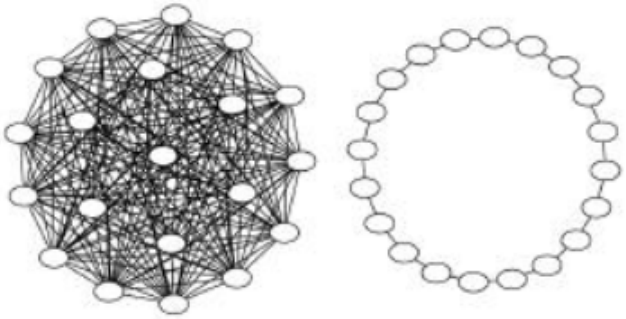

Figure 2: A visualization of the *gbest* and *lbest* topologies (Kennedy and Mendes, 2002, p. 1671).

4.5 (Stochastic) Hill-climbing

Simple Hill-climbing is an iterative search algorithm that does not belong to the class of evolutionary algorithms. It is, however, a powerful and fast algorithm that XXX. HC systematically modifies an initial candidate solution in directions that yield higher fitness values. If the first modification did not improve the current fitness value, the solution moves into another direction. Ideally, in a convex space with only one optimum, HC returns the hill-top, the best solution after a fixed number of iterations (Skiena, 2008, p. 252). As solutions with lower fitness values are not further investigated, HC is prone to get stuck in local optima as it cannot return to previous points and must go upwards (Russell and Norvig, 1995, p. 113). One resolution to mitigate this problem is Stochastic Hill-climbing where a random neighbor around the current solution is selected and evaluated. If it yields higher fitness, it replaces the current solution. instead of systematic mutation, randomly mutated candidate are evaluated and accepted if they yield a higher fitness. With this probabilistic element, the HC is forced to consider solutions further away from the current solution.

Both the simple HC and the SHC are simple to implement and perform very well on a number of optimization problems (Jacobson and Yücesan, 2004). They are often used as benchmark comparisons because one can justify (or fail to justify) the use of more complex algorithms like EAs when simpler and faster methods like HC perform competitively (Mitchell and Holland (1993), Wattenberg and Juels (1996), Prügel-Bennett (2004)).

5 Experimental Design

All algorithms were hand-coded in R and tested on the publicly available data from the DATA MINING CUP 2016 on fashion retail returns was used (DATA MINING CUP, 2016). As in the contest submissions of Humboldt University of Berlin, a model library of 26 Random Forests and 26 XGBoost, 54 models in total, was used. The high server usage at the Research Data Center at Humboldt University where all computations for this paper were handled limited both the variety of different parameter settings, the number of iterations and the data sets on which the algorithms were run. To deal with these restrictions, the DMC data set was reduced to 30% of its original size. This sub-set was then split into a training and a validation set with a 70/30 ratio, so that each algorithm optimized on 372,000 predictions. The evaluation of the optimization function, which is usually called several times in each iteration of the evolutionary algorithms is quite computationally expensive, so that even with the reduced data set, the number of iterations was limited to 500.

Each configuration of the five evolutionary algorithms was optimized in four seperate experiments: 1. binary-encoded, 2. real-valued, 3. binary-encoded + Bagging and 4. real-valued + Bagging. Bagging used of ten bags each.

All experiments followed the same structure: the solution populations were optimized over 500 generations (or iterations) where their performance was evaluated on a validation set. As a performance measure, the Brier Score was used, which can be intuitively understood a version of the Mean Squared Error for binary outcomes. It is defined as:

$$BS = \frac{1}{N}\sum_{i=1}^{N}(p_i - o_i)^2 \tag{6}$$

where N denotes the sample size, p_i is the predicted probability and o_i is the actual binary outcome (i.e. 1 or 0). The Brier Score lies in the [0,1] interval where a low value indicates a good forecast and the higher the Brier Score, the poorer the forecast performance. In the Bagging experiments, each algorithm is optimized on ten different subsets of the full classifier set and the results are then averaged. Additionally, the ensemble weights yielded by the algorithms were tested on the classification set of the DMC. The results for this data were recently published and are available on the DMC website.

Algorithms	Abbr.	Psize	Parameters		
			Mixrate		
BSA	BSA1	30	1		
	BSA2	100	1		
			Crossover	Elitism	pMutation
GA	GA1	30	0.50	0.03	0.05
	GA2	100	0.50	0.03	0.05
	GA3	30	0.80	0.03	0.05
	GA4	100	0.80	0.03	0.05
			Learning Rate	Shift	Prob
PBIL	PBIL1	30	0.05	0.05	0.02
	PBIL2	100	0.05	0.05	0.02
	PBIL3	30	0.10	0.05	0.02
	PBIL4	100	0.10	0.05	0.02
			Phi	Inertia	Initial Speed
PSO-G	PSOG1	30	1.80	$0.5 + \delta/2$	0.01
	PSOG2	100	1.80	$0.5 + \delta/2$	0.01
	PSOG3	30	2.10	$0.5 + \delta/2$	0.01
	PSOG4	100	2.10	$0.5 + \delta/2$	0.01
			Phi	Inertia	Initial Speed
PSO-L	PSOL1	30	1.80	$0.5 + \delta/2$	0.01
	PSOL2	100	1.80	$0.5 + \delta/2$	0.01
	PSOL3	30	2.10	$0.5 + \delta/2$	0.01
	PSOL4	100	2.10	$0.5 + \delta/2$	0.01

Table 1: The parameter configurations for each variant of the five evolutionary algorithms run; $\delta \sim U(0, 1)$

6 Results

The runtime of the algorithms was an unpleasant surprise and is most likely due to the high server usage at the RDC as test runs with a small number of iterations finished quickly on standard laptops. In the final run, each configuration took approximately 23 hours to run, whereas the bagged implementations took 27 hours on average. The binary-encoded variants ran a bit faster as their code was slightly less complex than their real-valued counterparts but still took 21 hours.

Figures 3 and 4 show the average convergence of each algorithm across all parameter settings. The GA and BSA converge as expected, whereas both PSO implementations show only little improvement to their random initialization. The PBIL is seemingly unable to hold any good solutions over time and tried solutions in a random walk fashion. The parameter setting might be at fault here as the bagged experiments yielded very similar results.

The algorithms yield strongly varying results in terms of ensemble sizes. Some variants pick as many as 26 of the 54 classifiers, whereas others decide on only 2 - 3. A complete

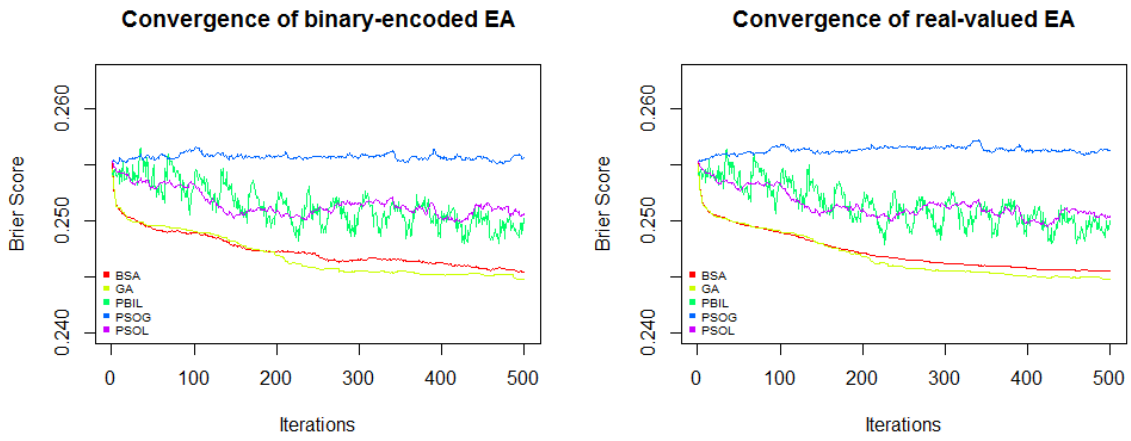

Figure 3: Convergence Results of Experiments 1 and 2.

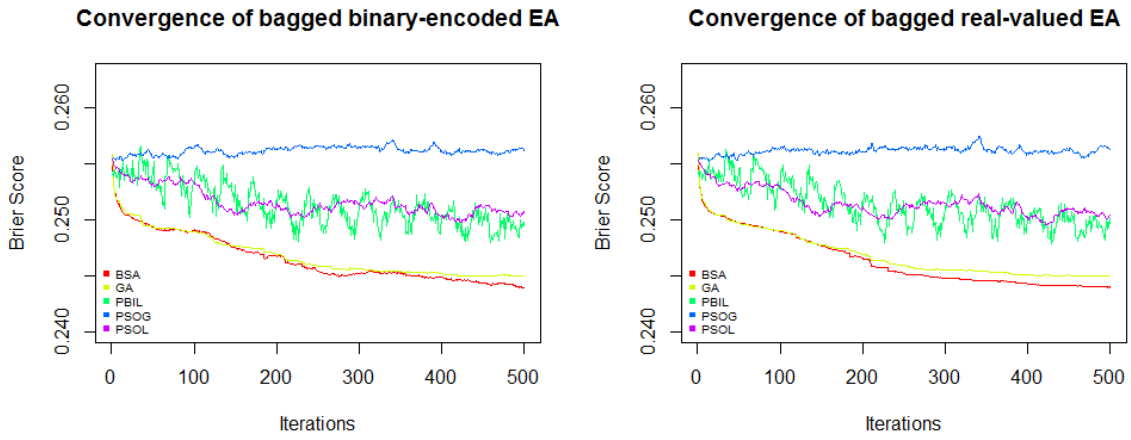

Figure 4: Convergence Results of Experiments 1 and 2.

Algorithms	ES (binary)	ES (real)	Algorithms	ES (binary)	ES (real)
BSA1	5	26	PSOG1	8	13
BSA2	3	13	PSOG2	6	14
BSA.BAG1	9	18	PSOG3	5	7
BSA.BAG2	3	6	PSOG4	2	4
GA1	4	4	PSOG.BAG1	7	10
GA2	5	5	PSOG.BAG2	3	5
GA3	4	5	PSOG.BAG3	4	4
GA4	3	3	PSOG.BAG4	2	4
GA.BAG1	2	3	PSOL1	4	5
GA.BAG2	3	4	PSOL2	6	8
GA.BAG3	2	3	PSOL3	3	12
GA.BAG4	3	4	PSOL4	7	8
PBIL1	9	17	PSOL.BAG1	4	6
PBIL2	3	26	PSOL.BAG2	6	6
PBIL3	7	9	PSOL.BAG3	2	7
PBIL4	7	9	PSOL.BAG4	5	11
PBIL.BAG1	8	12	SHC	3	
PBIL.BAG2	5	16	SHC.BAG	11	
PBIL.BAG3	6	7	HC	2	
PBIL.BAG4	7	8	HC.BAG	6	

Table 2: Ensemble Sizes Chosen by the EA

overview can be found in Table 2. Some results by Opitz and Maclin (1999) indicate that the marginal benefit of inclusion is highest for the first few models in the ensemble and decreases quickly (Opitz and Maclin, 1999, p. 190). Even though the classifiers in the library can be considered diverse due to their wide parameter range, it is intuitive that after a certain number of RF and XGBoost models in the ensemble, the performance does not improve anymore and the errors increase. The correlation of performance and ensemble size which is evident from Figure 5 is extremely interesting: For both the binary- and the real-coded versions, sparse solutions tend to yield better results. Still, the error rate increases quicker in the binary case. This should not come as a surprise because real-valued solutions may include many classifiers, many of them by a small weight only, so they do not decrease the performance. In the binary case, the weights are less likely to be arbitrarily small so any classifier in the ensemble has a larger relative weight.

The complete results of the experiments can be found in Table 3, where the columns refer to the performance on the validation and classification set. The best performing configurations for each of the two data sets are printed in boldface. When two scores were equal, both are emphasized. Overall, the real-valued solutions tend to perform slightly better than their binary counterparts.

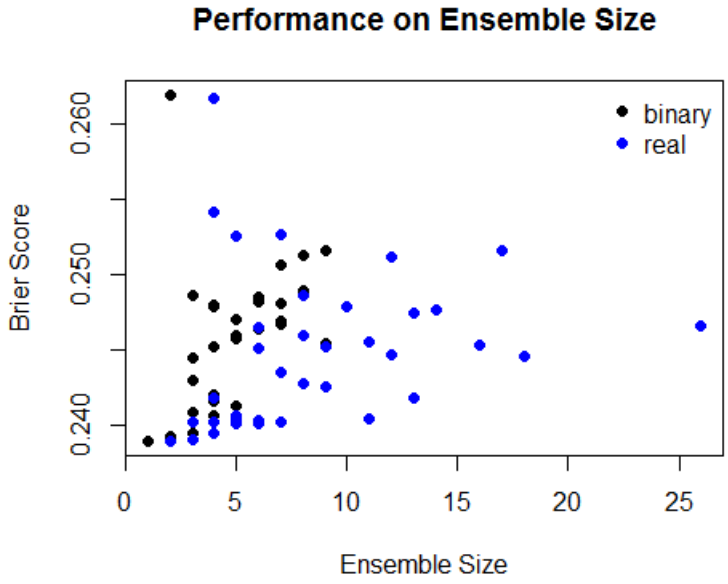

Figure 5: A scatterplot of the error rate on the ensemble size.

In training, the bagged GA yields the best results with BSA and the PSO-L being very comparatively in their error rate. Interestingly, the population size has opposing effects on the performance across algorithms. Overall, the GA performs better with a small population size whereas the increase of solution vectors in the population yields a performance increase for all others. As was somewhat expected after reading the paper by Kennedy and Mendes (2002), the PSO-G with a global information topology performs the worst out of all the configurations.

When compared to the baseline evaluations, the EAs do not measure up. All solutions, even the best solutions have a notable increase in the BS compared to the single best model. Even a simple average of the four best models outperforms 31 of the binary and 28 of the real-valued solutions. Interestingly, both HC and SHC suffer from bagging and yield best results on the full model library. HC maintains the error rate of the best performing model by only adding another classifier to the ensemble, with SHC also only considering three classifiers. Both the HC and SHC weigh the two classifiers more cleverly than the bagged GA1 and GA3 which also only include two classifiers.

This paper also considered the problem of overfitting, so that the results of applying the ensemble weights to the classification data are surprising. Five binary and four real-valued solutions yield BS that are significantly lower than the single best classifier. HC and SHC only perform as good as the best model and do not lower the BS further.

7 Conclusion

The results obtained by the EAs in the four experiments on the DMC data are very much promising and show that they are able to find good solutions with respect to Ensemble Selection. Throughout the experiments, there is no clear winner with the bagged genetic algorithm with a small population size, the bagged and unbagged BSA with a large population and the binary PBIL yielding strong solutions that even outperform the single best classifier on the classification data.

Given more time and more computational capacities, one could possible improve the performance of the tested algorithms even further as the comparative results show room for further investigation: Regarding the promising results of the PSO, the implementation of even more information topologies as suggested by Kennedy and Mendes (2002) could further improve them. Also, some of the slow convergence of the PSO could be attributed to the use of the random inertia weight as proposed by Bansal et al. (2011) so systematic testing of different weights might prove useful.

Regarding the three question raised at the end of Section 3.2, a) overfitting was not an issue in this setup so that b) cannot be answered. For c) the effect of bagging is not entirely straightforward in the application on the DMC data. For all algorithms tested, some configurations produce better results with bagging, some without. The effect of the number of bags on the performance could be further studied.

Still, the results of this study should not be overestimated. The surprisingly good performance of the sets on the classification data is not necessarily indicative of their ability to find the best ensemble. All EAs did not perform convincingly on the training data which they could fit to and were easily outperformed by other heuristic methods like Hill-Climbing and Stochastic Hill-Climbing. Using Cross Validation or more data subsamples could elucidate this issue.

Still, one has to carefully weigh the pros and cons of investing in optimizing and tuning an evolutionary algorithm instead of sticking with a simple-to-implement method that compares in performance. Based on our results, we recommend the latter.

References

Baluja, S. (1994): "Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning," Tech. rep., DTIC Document.

Bansal, J., P. Singh, M. Saraswat, A. Verma, S. S. Jadon, and A. Abraham (2011): "Inertia weight strategies in particle swarm optimization," in *Nature and Biologically Inspired Computing (NaBIC), 2011 Third World Congress on*, IEEE, 633–640.

Breiman, L. (1996): "Bagging Predictors," *Machine Learning*, 24, 123–140.

Brown, G., J. Wyatt, R. Harris, and X. Yao (2005): "Diversity creation methods: a survey and categorisation," *Information Fusion*, 6, 5–20.

Caruana, R., A. Niculescu-Mizil, G. Crew, and A. Ksikes (2004): "Ensemble selection from libraries of models," in *Proceedings of the twenty-first international conference on Machine learning*, ACM.

Cawley, G. C. and N. L. Talbot (2010): "On over-fitting in model selection and subsequent selection bias in performance evaluation," *Journal of Machine Learning Research*, 11, 2079–2107.

Civicioglu, P. (2013): "Backtracking search optimization algorithm for numerical optimization problems," *Applied Mathematics and Computation*, 219, 8121–8144.

Dasarathy, B. V. and B. V. Sheela (1979): "A composite classifier system design: concepts and methodology," *Proceedings of the IEEE*, 67, 708–713.

DATA MINING CUP (2016): "Predicting returns for the sales of discounted articles and voucher redemptions," Available under: data.mining-cup.de; last visited 07/14/2016.

De Jong, K. A. (1993): "Genetic algorithms are NOT function optimizers," *Foundations of genetic algorithms*, 2, 5–17.

Dietterich, T. G. (2000): "Ensemble methods in machine learning," in *International workshop on multiple classifier systems*, Springer, 1–15.

Dos Santos, E. M., L. S. Oliveira, R. Sabourin, and P. Maupin (2008): "Overfitting in the selection of classifier ensembles: a comparative study between pso and ga," in *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, ACM, 1423–1424.

Dos Santos, E. M., R. Sabourin, and P. Maupin (2009): "Overfitting cautious selection of classifier ensembles with genetic algorithms," *Information Fusion*, 10, 150–162.

Holland, J. H. (1975): *Adaptation in natural and artificial systems*, U Michigan Press.

Jacobson, S. H. and E. Yücesan (2004): "Analyzing the Performance of Generalized Hill Climbing Algorithms," *Journal of Heuristics*, 10, 387–405.

Kennedy, J. and R. Eberhart (1995): "Particle swarm optimization," in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol. 4, 1942–1948.

Kennedy, J. and R. Mendes (2002): "Population structure and particle swarm performance," in *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, vol. 2, 1671–1676.

Krogh, P. S. A. (1996): "Learning with ensembles: How over-fitting can be useful," in *Proceedings of the 1995 Conference*, vol. 8, 190.

Kuncheva, L. I. (2005): "Using diversity measures for generating error-correcting output codes in classifier ensembles," *Pattern Recognition Letters*, 26, 83–90.

Kuncheva, L. I. and C. J. Whitaker (2003): "Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy," *Machine Learning*, 51, 181–207.

Miller, B. L. and D. E. Goldberg (1995): "Genetic algorithms, tournament selection, and the effects of noise," *Complex systems*, 9, 193–212.

Mitchell, M. (1999): *An introduction to genetic algorithms*, Cambridge, Massachusetts London, England, fifth printing ed.

Mitchell, M. and J. H. Holland (1993): "When will a genetic algorithm outperform hill-climbing?" Tech. Rep. SGI Working Paper 1993-06-037, Santa Fe Institute.

Oliveira, L. S., M. Morita, R. Sabourin, and F. Bortolozzi (2005): "Multi-objective genetic algorithms to create ensemble of classifiers," in *International Conference on Evolutionary Multi-Criterion Optimization*, Springer, 592–606.

Opitz, D. and R. Maclin (1999): "Popular ensemble methods: An empirical study," *Journal of Artificial Intelligence Research*, 11, 169–198.

Polikar, R. (2008): "Combining classifiers with different decision boundaries reduce error," Available under http://www.scholarpedia.org/article/File:Combining_classifiers2.jpg, last visited 07/14/2016.

Prügel-Bennett, A. (2004): "When a genetic algorithm outperforms hill-climbing," *Theoretical Computer Science*, 320, 135 – 153.

Radtke, P., R. Sabourin, and T. Wong (2006): "Impact of solution over-fit on evolutionary multi-objective optimization for classification systems," *Proceedings of IJCNN, Vancouver, Canada*.

Robilliard, D. and C. Fonlupt (2002): *Backwarding: An Overfitting Control for Genetic Programming in a Remote Sensing Application*, Springer Berlin Heidelberg, 245–254.

Russell, S. and P. Norvig (1995): *Artificial Intelligence: A modern approach*, vol. 25, Citeseer.

Servet, I., L. Travé-Massuyès, and D. Stern (1997): "Telephone network traffic overloading diagnosis and evolutionary computation techniques," in *European Conference on Artificial Evolution*, Springer, 137–144.

Shi, Y. and R. Eberhart (1998): "A modified particle swarm optimizer," in *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, IEEE, 69–73.

Skiena, S. S. (2008): *The algorithm design manual*, Springer.

Sun, Q. and B. Pfahringer (2011): "Bagging ensemble selection," in *Australasian Joint Conference on Artificial Intelligence*, Springer, 251–260.

Tsoumakas, G., I. Partalas, and I. Vlahavas (2008): "A taxonomy and short review of ensemble selection," in *Workshop on Supervised and Unsupervised Ensemble Methods and Their Applications*.

Wattenberg, M. and A. Juels (1996): "Stochastic hillclimbing as a baseline method for evaluating genetic algorithms," in *Proceedings of the 1995 Conference*, vol. 8, 430.

Zhou, Z.-H. (2012): *Ensemble methods: foundations and algorithms*, CRC press.

Declaration of Authorship

We hereby confirm that we have authored this seminar paper independently and without use of others than the indicated sources. All passages which are literally or in general matter taken out of publications or other sources are marked as such.

Berlin, July 15

Maria Kozlova
Lara Vomfell

	BS on Validation		BS on Classification	
Algorithms	Binary	Real	Binary	Real
BSA1	0.2413	0.2466	0.2951	0.3007
BSA2	0.2396	0.2419	**0.2945**	0.2957
BSA.BAG1	0.2454	0.2446	0.2986	0.2980
BSA.BAG2	0.2396	0.2403	**0.2945**	0.2953
GA1	0.2416	0.2395	0.2954	0.2947
GA2	0.2413	0.2407	0.2951	0.2949
GA3	0.2406	0.2402	0.2948	0.2947
GA4	0.2430	0.2402	0.2966	0.2951
GA.BAG1	**0.2393**	**0.2391**	0.2946	0.2947
GA.BAG2	0.2445	0.2402	0.3008	0.2961
GA.BAG3	**0.2393**	**0.2391**	0.2946	**0.2946**
GA.BAG4	0.2445	0.2418	0.3008	0.2980
PBIL1	0.2516	0.2516	0.3051	0.3049
PBIL2	0.2396	0.2466	**0.2945**	0.3007
PBIL3	0.2469	0.2452	0.3017	0.3001
PBIL4	0.2467	0.2426	0.3000	0.2972
PBIL.BAG1	0.2513	0.2512	0.3054	0.3048
PBIL.BAG2	0.2458	0.2454	0.3003	0.2995
PBIL.BAG3	0.2485	0.2436	0.3029	0.2988
PBIL.BAG4	0.2467	0.2428	0.3000	0.2974
PSOG1	0.2489	0.2475	0.3051	0.3035
PSOG2	0.2482	0.2477	0.3047	0.3037
PSOG3	0.2460	0.2527	0.3022	0.3097
PSOG4	0.2619	0.2617	0.3195	0.3192
PSOG.BAG1	0.2506	0.2479	0.3071	0.3040
PSOG.BAG2	0.2486	0.2526	0.3028	0.3070
PSOG.BAG3	0.2479	0.2541	0.3047	0.3115
PSOG.BAG4	0.2619	0.2617	0.3195	0.3192
PSOL1	0.2452	0.2405	0.2978	0.2957
PSOL2	0.2464	0.2459	0.3002	0.3007
PSOL3	0.2409	0.2447	0.2951	0.2974
PSOL4	0.2481	0.2486	0.3038	0.3035
PSOL.BAG1	0.2479	0.2451	0.3047	0.3012
PSOL.BAG2	0.2482	0.2465	0.3012	0.3008
PSOL.BAG3	**0.2393**	0.2403	0.2946	0.2957
PSOL.BAG4	0.2470	0.2456	0.3028	0.2995
HC	0.2389		0.2948	
HC.BAG	0.2401		0.2949	
SHC	0.2390		0.2948	
SHC.BAG	0.2404		0.2949	
BEST	0.2389		0.2948	
AVRG	0.2553		0.3092	
BEST4	0.2406		0.2947	

Table 3: Performance Results of Evolutionary Algorithms on Validation and Classification Data