A Holistic Classification Optimization Framework with Feature Selection, Preprocessing, Manifold Learning and Classifiers

Fabian Bürger(✉) and Josef Pauli

Lehrstuhl für Intelligente Systeme, Universität Duisburg-Essen, Bismarckstraße 90, 47057 Duisburg, Germany
{fabian.buerger,josef.pauli}@uni-due.de
http://www.is.uni-due.de/

Abstract. All real-world classification problems require a carefully designed system to achieve the desired generalization performance. Developers need to select a useful feature subset and a classifier with suitable hyperparameters. Furthermore, a feature preprocessing method (e.g. scaling or pre-whitening) and a dimension reduction method (e.g. Principal Component Analysis (PCA), Autoencoders or other manifold learning algorithms) may improve the performance. The interplay of all these components is complex and a manual selection is time-consuming. This paper presents an automatic optimization framework that incorporates feature selection, several feature preprocessing methods, multiple feature transforms learned by manifold learning and multiple classifiers including all hyperparameters. The highly combinatorial optimization problem is solved with an evolutionary algorithm. Additionally, a multi-classifier based on the optimization trajectory is presented which improves the generalization. The evaluation on several datasets shows the effectiveness of the proposed framework.

Keywords: Feature selection · Model selection · Evolutionary optimization · Representation learning

1 Introduction

A classifier system that learns the connections from feature data to discrete class labels is useful for many applications such as medical diagnose systems or image-based object recognition. Several powerful methods have been established, like Support Vector Machines (SVM), that perform well on a large amount of tasks. However, the no-free-lunch theorem [1] states that there will never be a single best machine learning concept for all tasks. In practice, a lot of expertise is required for the development of a classification system to meet the generalization requirements. Numerous challenges occur in real-world applications, like high-dimensional and noisy feature data, too few training samples or suboptimal hyperparameters[1].

[1] Hyperparameters control the learning algorithm itself – e.g. the number of hidden layers in a neural network.

A. Fred et al. (Eds.): ICPRAM 2015, LNCS 9493, pp. 52–68, 2015.
DOI: 10.1007/978-3-319-27677-9_4

Furthermore, the feature data itself has a huge impact on the classification performance. There are three aspects that need to be considered: First, the selection of a reasonable subset of features is needed. A large amount of irrelevant, redundant or too noisy features tend to disturb classifiers due to the curse of dimensionality [2]. Secondly, the preprocessing of features usually improves the performance [3], especially when the distribution and value ranges differ greatly. Popular methods are e.g. feature scaling or pre-whitening. And third, representation learning with the goal of automatic feature construction out of low-level data is helpful [4]. This approach has gained more importance in the field of deep learning. Manifold learning is one variant of learning a simpler, low-dimensional representation from high-dimensional data. A great variety of such algorithms has been introduced, but their individual performance is highly dependent on the learning task.

Automatic optimization frameworks are designed to help the developer of machine learning systems to find an optimized combination of features, classifiers and hyperparameters. This paper presents an extended version of the classification pipeline framework presented in [5] and contains the fully automatic selection of features, feature preprocessing methods, manifold learning methods and classifiers. Furthermore, all hyperparameters – of the classifiers and the manifold learning methods – are optimized as well. As the interplay of all these components is complex an evolutionary optimization algorithm with an adapted variant of cross-validation is used to find good pipeline configurations.

Additionally, the optimization trajectory is exploited for a multi-pipeline classifier as well as graphical statistics to get deep insights into the classification problem itself. We compare our framework performance with the state-of-the-art optimization framework Auto-WEKA [6] with respect to classification accuracy and optimization speed.

2 Automatic Optimization Frameworks

When the supervised classification task is considered, a training dataset $T = \{(\mathbf{x}_i, y_i)\}$ with $1 \leq i \leq m$ training samples is provided. It contains feature vectors $\mathbf{x} \in \mathbb{R}^{d_{in}}$ and class labels $y_i \in \{\omega_1, \omega_2, \ldots, \omega_c\}$. The goal is to find a classifier model that predicts the correct class labels of previously unseen instances.

Automatic machine learning optimization frameworks try to find a suitable classifier model or a complete classification processing pipeline that fits to a given dataset T. A standard approach is the tuning of hyperparameters of a single classifier, e.g. the SVM. This problem is well discussed in many papers,e.g. in [7–9], to name a few. Usually, search-based approaches are used that evaluate different system configurations and hyperparameters with the goal to optimize the classification accuracy. Methods like cross-validation are used to estimate the generalization of a chosen algorithm [2].

Feature selection is one approach to dimension reduction with the strategy to remove irrelevant dimensions to overcome disturbing effects due to the peaking phenomenon [2]. Some frameworks, like [10–12], involve feature selection and hyperparameter optimization using evolutionary algorithms (see Sect. 4.1).

When more machine learning components should be optimized, the combinatorial complexity basically explodes. Furthermore, especially categorical variables and multiple algorithm portfolios introduce variable dependencies. The combinatorial optimization community developed sophisticated general purpose heuristics to handle problems with numerous and heterogeneous parameters which is also known as the *algorithm configuration problem*. With this respect, three approaches have to be mentioned. The ParamILS framework [13] is a local-search-based algorithm that limits the time spent for evaluating single configurations. Sequential model-based optimization (SMBO) [14] is a form of Bayesian optimization that keeps track of all knowledge of the objective function and its uncertainties to evaluate the next most promising configurations. SMBO is used in the Auto-WEKA framework [6] that optimizes features, classifiers and hyperparameters. A gender-based evolutionary approach is presented in [15] that handles variable dependencies in a tree structure.

However, there is no framework that targets holistic machine learning optimization covering feature selection, preprocessing, manifold learning and dimension reduction, classifier selection and hyperparameter tuning.

3 Classification Pipeline

In order to include all the aforementioned machine learning components into a holistic framework, a classification pipeline structure with 4 elements is proposed which is depicted in Fig. 1. Generally, the processing works like the pipes and filters pattern [16] while the pipeline has two modes: the *training mode* in which the training dataset T is needed and the *classification mode* in which new samples can be classified. A key design principle is a consecutive dimensionality reduction of the feature vectors while they pass through the pipeline. All pipeline elements contain important processing steps and multiple degrees of freedom that are summarized in the pipeline's configuration θ. It describes a set of important hyperparameters which have to be optimized for each learning task (see Sect. 4). The pipeline elements and their functionality are described in the following.

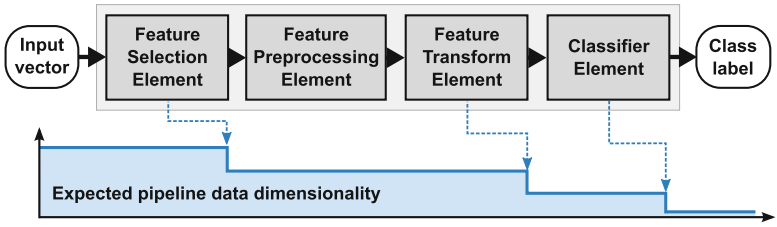


Fig. 1. Classification pipeline structure with expected data dimensionality.

3.1 Feature Selection Element

The first element is the feature selection element which removes irrelevant and noisy feature dimensions that could disturb any following algorithm. In training

and classification mode, it selects a subset $S_{FeatSet} \in \mathcal{P}(\{1, 2, ..., d_{in}\}) \setminus \emptyset$ of features. Feature selection is a difficult problem as $\mathcal{O}(2^{d_{in}})$ possible combinations exist and it has a great impact on the classification performance. The feature subset $S_{FeatSet}$ is included in θ.

3.2 Feature Preprocessing Element

The second element of the pipeline handles the preprocessing of the feature vectors. Almost all machine learning algorithms perform better when the numeric properties and value ranges of the features are stable. The preprocessing element uses a portfolio set S_{PrePoc} of commonly used feature preprocessing methods:

- *Rescaling* scales each feature dimension to a specific range, such that all values of all dimensions lie in the range of $[0, 1]$.
- *Standardization* is similar to rescaling, but the data is scaled to a normal distribution with zero mean and a standard deviation of 1.
- *L2-Normalization* scales each feature vector independently to unit length which is equivalent to an L_2-norm of 1.
- *Pre-Whitening* is a more complex preprocessing step that performs a decorrelation transformation resulting to a feature matrix with zero mean and having a covariance matrix equal to the identity matrix [17].
- The *identity* function does not change any feature data in this pipeline element. This leads to the best results for some datasets.

The preprocessing method $f_{PrePoc} \in S_{PrePoc}$ is a part of the pipeline configuration θ. In training mode, the selected method f_{PrePoc} is used to extract model variables from the training dataset T, such as minimum and maximum values in case of the rescaling method. In classification mode, incoming feature vectors are processed with f_{PrePoc}.

3.3 Feature Transform Element

The third element is the feature transform element which uses manifold learning algorithms to obtain a transform for a better suitable feature representation. Manifold learning describes a family of linear and nonlinear dimensionality reduction algorithms that analyze the topological properties of the feature data distribution to build a transformation function which embeds feature data into a low-dimensional space. In order to use manifold learning for real-world applications the following definition from [18] is used that requires just a set of D-dimensional data vectors. The assumption is that the datapoints lie on a lower-dimensional manifold with an intrinsic dimensionality $d < D$ which is embedded in the D-dimensional space. In practice, the target dimensionality d is not known and must be estimated. The manifold maybe non-Riemannian or disconnected which is likely for noisy real-world data. The goal is to find a feature transform function that embeds sample vectors into the lower dimensional vector space using $\tilde{\mathbf{x}}_i = f_{trans}(\mathbf{x}_i) \in \mathbb{R}^d$, while $\mathbf{x}_i \in \mathbb{R}^D$, without losing important information about the geometrical structure and distribution.

The pipeline element contains a portfolio set of possible transformations $S_{FeatTrans}$. Currently we use a set of 30 – mostly unsupervised – transforms provided by [19] which are listed in the appendix. Examples of linear transforms are e.g. Principal Component Analysis (PCA) or Linear Discriminant Analysis (LDA). Nonlinear techniques are e.g. Isomap, Kernel-PCA, Local Linear Embedding (LLE) or Autoencoders. References to these methods can be found e.g. in [18] or [20].

Furthermore, many of these manifold learning algorithms have – like classifiers – hyperparameters (see Sect. 3.5 for definition) that influence the performance. An example is the number of neighbors for neighborhood-based graph algorithms like Isomap [21]. The set of hyperparameters of a specific algorithm $f_{FeatTrans} \in S_{FeatTrans}$ is denoted as $S_{\mathbb{H}}(f_{FeatTrans})$. The choice of a method $f_{FeatTrans}$, the corresponding target dimensionality d and the hyperparameters $S_{\mathbb{H}}(f_{FeatTrans})$ are included into the pipeline configuration θ.

In training mode, the incoming feature vectors (and labels for the supervised methods like LDA) are used to learn the parameters of the manifold learning algorithm. In classification mode, previously unseen vectors need to be transformed to the new feature space. Unfortunately, not all methods directly support the so-called out-of-sample embedding. A direct extension is available only for parametric methods [18], e.g. PCA and Autoencoders. For spectral methods, like LLE, Isomap or Laplacian Eigenmaps, the Nyström theorem [22] can be used for an extension. For all other methods a rather naive non-parametric out-of-sample extension can be used (see [18,19]). It requires the storage of the complete set of base vectors and their corresponding transformed vectors. For each new feature vector, the nearest vector in the training dataset T and its corresponding transformed vector are determined which are used for an estimated linear projection.

3.4 Classifier Element

The last element is the classifier element which uses a classifier $f_{Classifier} \in S_{Classifiers}$. The framework currently contains 6 "popular" classifier concepts: the naive Bayes classifier, k-nearest neighbors (kNN), Support Vector Machine (SVM) with different kernels (linear, polynomial and Gaussian), random forest, extreme learning machine (ELM) and multilayer perceptron (MLP). References to these concepts can be found e.g. in [17,23]. Each classifier can have an arbitrary number of hyperparameters (see Sect. 3.5 for definition) which are tuned during the optimization phase. The selection of the classifier $f_{Classifier}$ and its hyperparameters $S_{\mathbb{H}}(f_{Classifier})$ are included into θ.

In training mode, the chosen classifier is trained using the data processed by all previous pipeline elements while the labels stay the same as in the training set T. In classification mode, the classifier classifies the incoming vectors.

3.5 Hyperparameters

Most machine learning algorithms have hyperparameters that need to be carefully adapted to the current learning task. This is also true for manifold learning algorithms. Almost all hyperparameters can be categorized into two basic types:

- A *numerical hyperparameter* h_{num} can be either real or integer values $h_{num} \in \{\mathbb{R}, \mathbb{Z}\}$ that are usually bounded within a reasonable minimum and maximum value $h_{min} \leq h_{num} \leq h_{max}$.
- For *categorical hyperparameters* an item $h_{cat} \in H_{cat}$ has to be selected out of a set H_{cat}.

In order to facilitate a simpler notation, $S_{\mathbb{H}}(f) = \{h_1, h_2, \ldots, h_N\}$ with $h_j \in \{h_{num}, h_{cat}\}$ denotes the set of hyperparameters of an algorithm f. Note that the sets of hyperparameters of different algorithms are independent.

4 Optimization of the Pipeline Configuration

The pipeline configuration finally contains all important hyperparameters

$$\theta = (S_{FeatSet}, f_{PreProc}, f_{FeatTrans}, S_{\mathbb{H}}(f_{FeatTrans}), d_{FeatTrans}, f_{Classifier}, S_{\mathbb{H}}(f_{Classifier})) \quad (1)$$

which have to be optimized for each learning task. First, a suitable evaluation metric has to be involved to estimate the predictive performance of a pipeline configuration. Secondly, the highly combinatorial search problem to find the best configuration has to be solved within a reasonable time. Furthermore, the objective function is expected to be non-smooth with numerous local optima.

4.1 Extended Evolution Strategies

We choose evolutionary optimization because it is well-suited to solve complex optimization problems and can easily be parallelized. These algorithms are inspired by Darwin's Theory of Evolution [24] in which the fitness in terms of adaptation to the environment has a great impact on the survival and reproduction of individuals in a species. A key part of these algorithms is randomization which is helpful for objective functions with many local optima. Especially evolution strategies (ES) with extensions are suitable for the optimization of heterogeneous hyperparameters [25,26]. ES uses sets of solution candidates which are called population of individuals. Evolutionary operators for random initial population of individuals, selection, recombination and mutation perform the actual optimization. The variables of an optimization problem can conveniently be coded directly as real or integer number search space $V_{\mathbb{R}}$ and $V_{\mathbb{Z}}$. Extensions allow the use of a binary search space $V_{\mathbb{B}}$ to form bitstrings as well as a discrete set search space $V_{\mathbb{S}}$ to model categorical parameters [27]. Furthermore, numeric

hyperparameters with an exponential value range occur (e.g. Gaussian kernel parameters). In this case, the exponent $log_{10}(h_{num})$ is used for optimization in combination with $V_{\mathbb{R}}$.

The parameters for the ES strategies can be coded in the $(\mu, \kappa, \lambda, \rho)$ notation. The number of individuals that survive in each generation is denoted as μ. In each generation λ children from ρ parents are derived. The selection operator selects individuals for mating depending on their fitness which is directly connected to the objective function (see Sect. 4.2). Finally, individuals have a limited lifetime of κ generations.

Pipeline Configuration Coding. The classification pipeline configuration θ is transformed into a suitable ES variable representation in the following way (see Fig. 2). The feature subset is coded as binary mask consisting of d_{in} binary variables of type $V_{\mathbb{B}}$ which is the same idea than in [10]. All algorithm selection problems, namely feature preprocessing, feature transform and classifier, are handled as categorical variables $V_{\mathbb{S}}$. For the target dimensionality a factor $\alpha \in [0, 1]$ is coded as $V_{\mathbb{R}}$ genotype. It determines the fraction of the number of dimensions delivered by the feature selection that should be used as target dimensionality $d = \lfloor \alpha \cdot |S_{FeatSet}| \rfloor$ and $d \geq 1$.

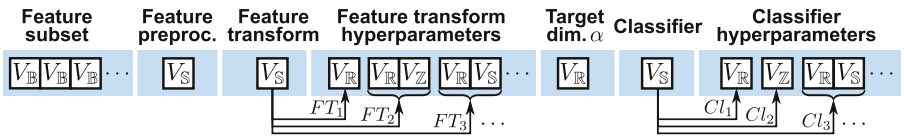


Fig. 2. Coding schema of a pipeline configuration θ for ES with four variable types $V_{\mathbb{R}}$, $V_{\mathbb{Z}}$, $V_{\mathbb{B}}$ and $V_{\mathbb{S}}$. Note that FT_i refers to the ith feature transform and Cl_j to the jth classifier.

The handling of hyperparameters is more difficult as they depend on the selection of the corresponding algorithm. In [5], the pipeline configuration only contained *one* set of hyperparameters, namely the classifier's. Two variants have been proposed: a combination of ES with grid search and a "complete" evolutionary optimization. Grid search is likely not feasible to optimize *both* hyperparameter sets from classifier and feature transform. Therefore, a similar approach as the proposed complete evolutionary strategy (denoted as *CES*) is used that concatenates all hyperparameters of all algorithms in a linear way with their corresponding type (typically $V_{\mathbb{R}}, V_{\mathbb{Z}}$ or $V_{\mathbb{S}}$). Another benefit is that numeric values can be adapted much finer compared to the grid search approach. All hyperparameters are evolved in parallel, but the selection of a specific algorithm acts as a switch that activates only the corresponding values when the configuration is used for a classification pipeline.

Evolutionary Parameters. First, an initial population of 200 random individuals is generated to get a reasonably large sample of the huge search space.

In each generation $\lambda = 100$ individuals from $\rho = 3$ parents are generated. The selection operator selects $\mu = 25$ individuals out of the newly generated offspring and the parent generation. In order to increase the diversity of solutions to overcome local optima, all individuals have a limited lifespan of $\kappa = 5$ generations.

The algorithm terminates when the improvement of the best fitness is less than $\epsilon = 10^{-3}$ (equal to 0.1 % of accuracy) after at least three consecutive generations. However, to prevent a premature end of the optimization, at least 10 generations are evaluated.

Mutation. Standard ES algorithms just define a mutation operator for real vectors. However, the proposed system needs to be extended to mutate the heterogeneous variable types. For numerical variables $V_{\mathbb{R}}$ and $V_{\mathbb{Z}}$, an additive, normally distributed random variable is used whose standard deviation is initialized depending on the expected value boundaries of the corresponding hyperparameter $\sigma = 0.2 \cdot (h_{max} - h_{min})$. Each of the categorical and binary variables $V_{\mathbb{S}}, V_{\mathbb{B}}$ have a probability variable p_{mut} which is initialized with $p_{mut} = 0.1$. This probability defines a mutation to select either a random item or a bit flip, respectively. All mutation parameters are adapted during the optimization process as well. However, the originally proposed correlation between mutation variables [25] is not considered due to high number of additional variables that would be needed.

Initial Population Improvement of the Feature Subset. Due to the large search space – especially due to the feature selection – an improvement of the initial subsets compared to pure random subsets is expected to improve both optimization runtime and accuracy. Before every main optimization with the full set of framework components, a fast pre-optimization is performed that usually is done within less than a minute. This pre-optimization just contains feature selection and preprocessing methods in combination with the hyperparameter-free naive Bayes classifier. The initial population size contains 200 individuals, while 5 generations are performed using $\lambda_{pre} = 50$, $\rho_{pre} = 3$ and $\mu_{pre} = 20$. The feature subsets of the last surviving individuals of the pre-optimization are used as initial pool of feature subsets for the main optimization. One of the "good" subsets from the pool is chosen randomly for each individual of the initial generation of the main optimization.

4.2 Optimization Target Function

The evaluation metric of a configuration θ plays a central role to evaluate the generalization of the whole pipeline and is, of course, needed for the fitness of individuals. A common way to minimize the risk of overfitting of classifiers is k-fold cross-validation [2]. In the proposed pipeline multiple processing steps influence the generalization. Especially, the feature transform element with its out-of-sample function has a special role as the "intelligence" is potentially moved from the classifier to the feature transform: A highly nonlinear feature transform might work best with a simple, e.g. linear classifier. Furthermore, the preprocessing element also extracts model parameters from the training data

and processes unseen feature vectors with these parameters. Problems may occur even with the simple rescaling method to $[0, 1]$: single outlier values with a very high value compared to the others make this method unstable.

Adapted Cross-Validation. In order estimate the generalization of a pipeline configuration θ, all components of the pipeline have to be involved into an adapted cross-validation process which is depicted in Fig. 3. The training set T is separated into $K = 5$ cross-validation tuples with disjoint training and validation datasets $\{(T_{train,l}, T_{valid,l})\}$. In each round, all models and parameters are estimated using only the training data $T_{train,l}$. The validation dataset is processed separately and the predictions of the classifier are used to calculate the accuracy acc_l. Finally, the average cross validation accuracy $acc_{avg} = \frac{1}{K} \sum_{l=1}^{K} acc_l$ and the standard deviation $acc_{sd} = \sqrt{\frac{1}{K} \sum_{l=1}^{K} (acc_l - acc_{avg})^2}$ are computed.

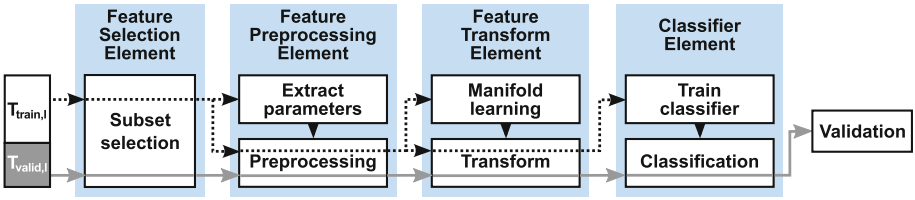


Fig. 3. Processing of the lth cross-validation round with training and validation data $T_{train,l}$ and $T_{valid,l}$ that incorporates feature selection, preprocessing, manifold learning and classifier into the generalization estimation. Note that the validation set is never used to estimate parameters or train any algorithm.

Early Rejection. The random character of ES leads to a relatively high fraction of suboptimal solutions that need to be evaluated. We propose an early rejection strategy that discards inferior configurations as soon as possible during cross-validation to save computation time for potentially better solutions. Two criteria lead to an early rejection: At first, if any configuration performs worse than guessing, thus $acc_l < 1/number\ classes$, the cross-validation is stopped. The second criterion uses a statistical method to test if a configuration will likely be equal or better than the currently best one. The cross-validation performance value $acc_{avg,best}$ and $acc_{sd,best}$ of the overall best configuration are stored. In each cross-validation round, a one-sided confidence interval for the accuracy mean is calculated that determines a minimum average accuracy of the lth round $acc_{min,l} = acc_{avg,best} - z \cdot acc_{sd,best}/\sqrt{l}$. The parameter z is the confidence level; we use $z = 1.96$ which is equal to an error probability of 2.5 %. The cross-validation is stopped if any $acc_l < acc_{min,l}$.

4.3 Multi-pipeline Classifier

The presented optimization method leads to a result list of N_{Res} configurations $R = \{(\theta_j, q_j)\}$, $1 \leq j \leq N_{Res}$, with a corresponding fitness q_j. The configurations

can be sorted by their fitness q_j and, at first glance, the configuration with the highest fitness is the most interesting result. However, this solution could be randomly picked and therefore quite "unusual" and also potentially overfitted to the training dataset, even though cross-validation is used.

The distribution of the top-n configurations can be used to generate a multi-pipeline classifier. Multi-classifier systems have the potential to improve the generalization capabilities compared to a single classifier when the diversity of the different models is large enough [28]. A multi-pipeline classifier is defined such that the top-n configurations are used to set up n pipelines with the corresponding configuration θ_j. In classification mode, all pipelines are classifying the input vector parallelly and finally, a majority voting is performed to select the most frequent label.

5 Experiments

For the evaluation of the presented framework 11 classification problems from the UCI database [29] have been used with different dimensionalities and classes (see Table 1). In order to test the generalization capabilities the instances of all datasets have been divided randomly into 50 % train and 50 % test sets. The proposed optimization algorithm, denoted as *CES (complete evolutionary strategy)*, is evaluated and compared to a *baseline* classifier. For the baseline we choose a popular standard approach with an SVM with a Gaussian kernel, in combination with the full feature set (no feature selection), no preprocessing, no feature transform and grid-based tuned hyperparameters.

The proposed evolutionary algorithm uses random components which may lead to non-reproducible results and local optima. In order to overcome this problem in the evaluations, all experiments have been repeated 3 times. Currently, the framework is implemented in Matlab 2014 using the Parallel Computing Toolbox and is run on an Intel Xeon workstation with 6 × 2.5 Ghz and 32 GB of RAM[2].

Table 1. Dataset information from the UCI database [29].

Index	Tataset	Dimensions	Classes	Index	Dataset	Dimensions	Classes
1	iris	4	3	7	australian	14	2
2	diabetes	8	2	8	vehicle	18	4
3	breast-cancer	9	2	9	ionosphere	34	2
4	contraceptive	9	3	10	sonar	60	2
5	glass	9	6	11	semeion-digits	256	10
6	statlogheart	13	2				

[2] The average memory consumption of the proposed system is below 8 GB.

5.1 Evaluation of the Optimization Process

First, the optimization process on the training datasets itself is evaluated. Table 2 shows the average cross-validation accuracies compared to the baseline SVM. The accuracy values for CES are significantly higher compared to the SVM for all datasets – the average accuracy improvement is $6.36 \pm 6.02\,\%$. These results show that the proposed classification pipeline is able to adapt very well to any learning task due do its large repertoire of algorithms and hyperparameters. The results are mostly stable, however the standard deviation is slightly higher for three datasets, namely *glass*, *vehicle* and *sonar*. This indicates that the optimization algorithm was stuck in local optima.

The optimization runtimes for each dataset can be found in Table 3. The average runtime for the 11 datasets is 131.4 ± 96.0 min. There is no general link between the runtime and the dimensionality of the dataset. The main runtime heavily depends on the complexity of the selected algorithms in the pipeline configuration. Especially some feature transforms need much more time than others. Note that the runtime of the algorithms is not considered in the optimization process yet.

Table 2. Average training cross-validation accuracies compared to the baseline SVM.

Dataset	Baseline	CES	Dataset	Baseline	CES
1	96.00	**98.67 ± 0.00**	7	68.76	**88.35 ± 0.43**
2	76.81	**81.17 ± 0.60**	8	75.94	**82.55 ± 2.05**
3	97.07	**98.15 ± 0.45**	9	92.65	**95.48 ± 0.56**
4	52.70	**55.68 ± 0.27**	10	85.71	**89.21 ± 1.98**
5	68.96	**80.43 ± 2.24**	11	92.12	**93.21 ± 0.69**
6	74.07	**87.90 ± 1.54**			

Table 3. Average optimization runtimes in minutes.

Dataset	CES	Dataset	CES	Dataset	CES
1	29.1 ± 5.0	5	59.8 ± 36.0	9	131.4 ± 18.6
2	132.0 ± 30.3	6	39.4 ± 22.1	10	47.6 ± 1.1
3	182.0 ± 73.8	7	124.8 ± 48.7	11	287.5 ± 118.5
4	313.3 ± 55.6	8	99.0 ± 26.9		

5.2 Analysis of the Optimization Trajectory

In the following the optimization processes of two datasets, namely *statlogheart*[3] and *glass*[4], are discussed in detail. Figures 4 (a) and 5 (a) show the fitness

[3] The *statlogheart* dataset is a medical application in which diagnoses are correlated with absence of presence of serious heart diseases.

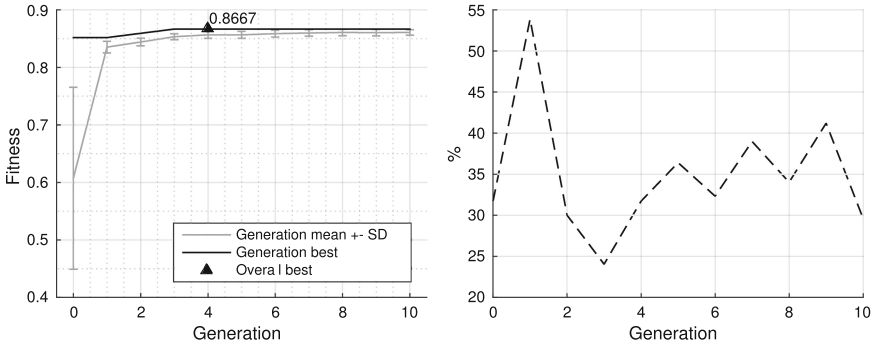
[4] The *glass* dataset is a forensic application in which glass is classified by oxide contents with the goal to identify the origins of the glass.

developments during the optimization. In both cases, a fast and steep gain of the mean fitness of the populations can be observed. The best fitness values per generation start at a relatively high level and increase much slower than the mean fitness. This indicates that the initial population already contains very well performing individuals which are fine-tuned in the main optimization.

Additionally, the effectiveness of the early rejection strategy during cross-validation can be seen in Figs. 4 (b) and 5 (b). These graphs depict the percentage of saved cross-validation rounds depending on the generation. The generation number zero is the initial population of the main optimization and during these evaluations, only the "worse-than-guessing" criterion is applied. In generation one, the fitness values of the best individuals from the initial generation are available to apply the confidence interval criterion to save more cross-validation rounds. As a large fraction of individuals perform inferiorly in the first generation, the ratio of saved evaluations is maximal there – for both datasets. Totally, 34.6 % of cross-validation rounds have been saved for the *statlogheart* dataset and and 41.5 % for the *glass* dataset, respectively.

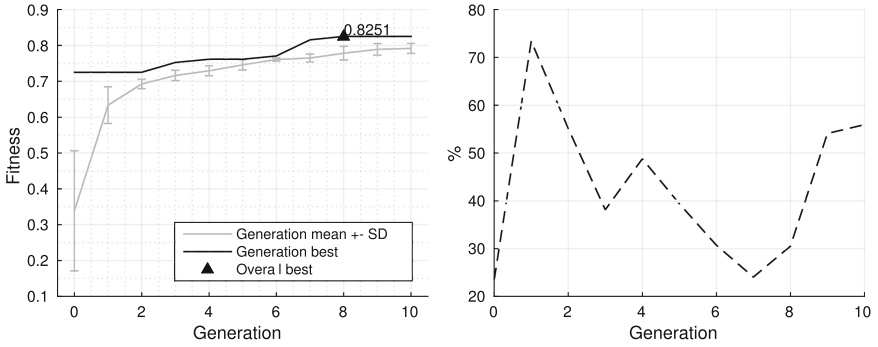
5.3 Top Configuration Graph

The trajectory of the optimization can be exploited to get insight into the classification problem and its solutions. After the optimization terminates, the sorted result list R (see Sect. 4.3) is available. However, it is hard to analyze the configurations in text or table form. One way of visually analyze the solutions in R is the top configuration graph which is depicted in Fig. 6. The graph shows the distribution of frequencies of features, feature preprocessing methods, feature transforms and classifiers with a different shading. Additionally, components are considered as connected if they appear in the same configuration. These connections are shown as edges which are also shaded according to their frequency. The idea behind this graph is that features and algorithms which have been selected more often play a more important role for the classification problem.



(a) Fitness development (b) Saved evaluations by early rejection

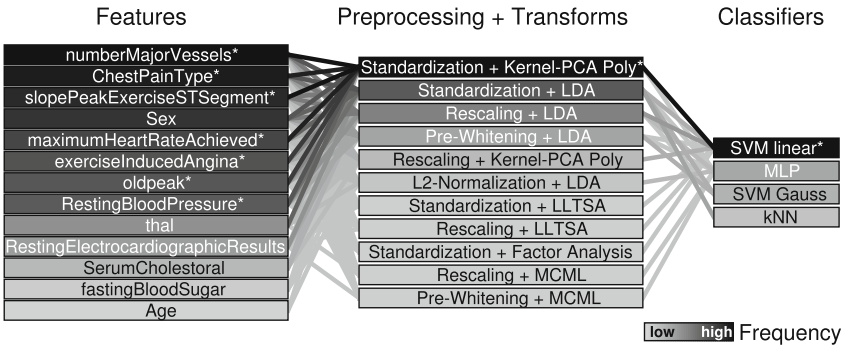
Fig. 4. Exemplary optimization statistics for the *statlogheart* dataset.



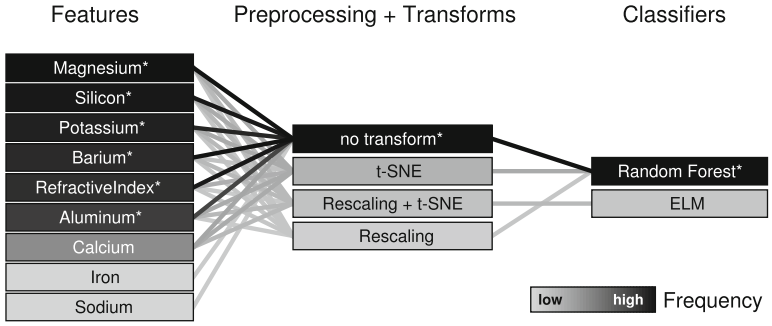
(a) Fitness development (b) Saved evaluations by early rejection

Fig. 5. Exemplary optimization statistics for the *glass* dataset.

The two examples found in Fig. 6 show the top-50 configuration graphs for the *statlogheart* dataset in (a) and the *glass* dataset in (b).



(a) Top-50 configuration graph for the *statlogheart* dataset.



(b) Top-50 configuration graph for *glass* dataset.

Fig. 6. Examples of graphical analyses of the distribution of the top-50 configurations. The asterisk (*) denotes elements that occur in the overall best configuration.

For the *statlogheart* dataset, many different feature preprocessing and transforms perform well. Standardization, Kernel-PCA with a polynomial kernel and the LDA methods perform best. The feature preprocessing and transforms seem to make the problem linearly separable, as a linear SVM performs best in most cases. The shading of the features indicate their importance and, e.g. the feature *numberMajorVessels* and *ChestPainType* are very relevant, while the features *fastingBloodSugar* and *Age* seem to be the most irrelevant for predicting heart diseases.

For the *glass* dataset, no feature preprocessing method and transform have been successful. The distribution of features seems to be highly non-smooth as only the random forest performed well in most cases. The feature analysis reveals that the features *Magnesium*, *Silicon*, *Potassium*, *Barium*, *RefractiveIndex* and *Aluminium* are much more important than *Calcium*, *Iron* and *Sodium* to classify glass samples.

5.4 Evaluation of the Generalization

The huge accuracy improvements during cross-validation are promising, but the risk of overfitting is evident. Table 4 shows the accuracies of the proposed classification pipelines on the test datasets which have not been used during cross-validation. The generalization of a single classification pipeline using the best configuration in terms of fitness (denoted as top-1) is in many cases better than the generalization of the baseline classifier, but the average improvement is marginal (0.73 % with a high standard deviation). This would usually not justify the optimization time of several hours. The multi-pipeline classifier improves the accuracy greatly for many datasets; the average accuracy improvement

Table 4. Average generalization accuracies on test datasets compared to baseline and Auto-WEKA (24h of time budget). The number of pipelines for the best top-n multi-pipeline classifier is denoted in parentheses (n).

Dataset	Baseline	Top-1	Top-10	Best Top-n	AutoWEKA
1	97.33	97.33 ± 0.00	98.22 ± 1.54	**99.11** ± 1.54 (5)	92.27
2	73.96	74.91 ± 0.15	74.57 ± 0.60	74.91 ± 0.15 (1)	**75.83**
3	**96.77**	86.12 ± 17.96	96.29 ± 0.45	**96.77** ± 0.78 (15)	96.72
4	54.29	56.19 ± 0.59	57.23 ± 1.19	**57.55** ± 1.30 (11)	57.17
5	63.81	65.71 ± 7.19	68.57 ± 7.44	69.52 ± 8.30 (4)	**74.86**
6	72.59	81.73 ± 1.86	82.22 ± 2.22	82.96 ± 2.96 (23)	**83.70**
7	71.51	85.47 ± 0.77	85.95 ± 0.17	**86.24** ± 0.61 (6)	85.29
8	77.49	80.25 ± 4.12	81.99 ± 0.24	**83.10** ± 1.68 (30)	81.18
9	96.57	92.57 ± 1.98	95.24 ± 0.87	**96.76** ± 0.87 (50)	96.11
10	**87.38**	79.94 ± 1.48	82.52 ± 3.88	83.17 ± 2.24 (21)	85.63
11	95.21	94.75 ± 0.65	95.71 ± 0.44	**96.47** ± 0.13 (38)	94.13
	Delta Baseline:	0.73 ± 6.88	2.87 ± 5.38	3.60 ± 5.29	3.27 ± 6.11

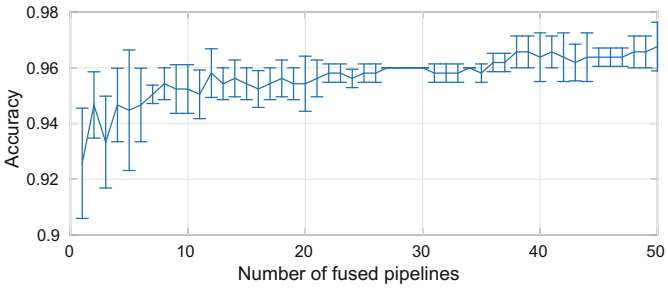


Fig. 7. Exemplary accuracy on test dataset of a multi-pipeline classifier for the *ionosphere* dataset depending on the number of pipelines n. The plot shows means and standard deviations of the repetitions.

compared to the baseline is 2.87 % if the 10 best pipelines are used. However, the optimal number of pipelines with the best generalization depends on the dataset. It can be observed for many datasets that a fusion of a higher number of pipelines leads to a better classification performance. Figure 7 shows the increasing performance depending on the number of fused pipelines for the *ionosphere* dataset.

The classifiers optimized by the Auto-WEKA framework with a time budget of 24 h also show a good performance on many datasets that is in the range of the best multi-pipeline classifiers. However, it performed rather poorly on the very simple *iris*[5] dataset. This shows that overfitting is also an issue for Auto-WEKA.

6 Conclusions

In this work, a holistic classification pipeline framework with feature selection, portfolios of feature preprocessing methods, feature transforms and classifiers is presented. An evolutionary algorithm is used that optimizes the configuration of the pipeline consisting of feature subset, algorithm selection and hyperparameters relatively efficiently. An adapted variant of cross-validation is proposed that incorporates the generalization performance of feature preprocessing, feature transforms and the classifier. A multi-pipeline classifier is used to improve the generalization of the classifier system. The framework does not require expert knowledge to reach state-of-the-art performance within a few hours. Additionally, graphical analyses of the best configurations help to reveal information about latent properties of the learning task.

The evaluation of the framework shows that overfitting is still a problem even though cross-validation is used. Sometimes, the standard SVM without special preprocessing generalizes best. However, the fusion of multiple pipelines shows a much better generalization performance, but introduces a higher computational cost. On the other hand, the datasets were rather low-dimensional and the performance improvement is expected to be larger in higher dimensional feature

[5] The popular *iris* dataset correlates variants of the iris plant with leaf dimensions.

spaces. In future work, the generalization estimation needs to be improved with advanced methods like e.g. bootstrapping. The runtime of the selected algorithms should also be considered during the optimization process to make it faster.

Acknowledgements. This work was funded by the European Commission within the Ziel2.NRW programme "NanoMikro+Werkstoffe.NRW".

Appendix

List of feature transforms and manifold learning methods that are used in the portfolio $S_{FeatTrans}$ of the feature transform element:
Autoencoder, CFA (Coordinated Factor Analysis), Diffusion Maps, Factor Analysis, FastICA (Independent Component Analysis), GPLVM (Gaussian Process Latent Variable Models), Hessian LLE, Identity (no transform), Isomap, Kernel-LDA (extension to LDA with e.g. Gaussian or polynomial kernels), Kernel-PCA (extension to PCA with e.g. Gaussian or polynomial kernels), Landmark Isomap, Laplacian Eigenmaps, LDA (Linear Discriminant Analysis / Fisher Discriminant Analysis / FDA), LLC (Locally Linear Coordination), LLE (Locally Linear Embedding), LLTSA (Linear LTSA), LMNN (Large-Margin Nearest Neighbor), LPP (Locality Preserving Projection), LTSA (Local Tangent Space Analysis), Manifold Charting, MCML (Maximally Collapsing Metric Learning), NCA (Neighborhood Components Analysis), NPE (Neighborhood Preserving Embedding), PCA (Principal Component Analysis), Sammon Mapping, SNE (Stochastic Neighbor Embedding), SPE (Structure Preserving Embedding), Symmetric SNE, t-SNE (parametric).

References

1. Wolpert, D.H.: The lack of a priori distinctions between learning algorithms. Neural Comput. **8**, 1341–1390 (1996)
2. Jain, A.K., Duin, R.P.W., Mao, J.: Statistical pattern recognition: a review. IEEE Trans. Pattern Anal. Mach. Intell. **22**, 4–37 (2000)
3. Juszczak, P., Tax, D., Duin, R.: Feature scaling in support vector data description. In: Proceedings ASCI, pp. 95–102. Citeseer (2002)
4. Bengio, Y., Courville, A., Vincent, P.: Representation learning: a review and new perspectives. IEEE Trans. Pattern Anal. Mach. Intelli. **35**, 1798–1828 (2013)
5. Bürger, F., Pauli, J.: Representation optimization with feature selection and manifold learning in a holistic classification framework. In: De Marsico, M., Fred, A., eds.: Proceedings of the International Conference on Pattern Recognition Applications and Methods ICPRAM 2015, vol. 1, pp. 35–44. INSTICC, SCITEPRESS, Lisbon (2015)
6. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In: Proceedings of KDD-2013, pp. 847–855 (2013)
7. Bengio, Y.: Gradient-based optimization of hyperparameters. Neural Comput. **12**, 1889–1900 (2000)

8. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B., et al.: Algorithms for hyper-parameter optimization. In: 25th Annual Conference on Neural Information Processing Systems (NIPS 2011) (2011)
9. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**, 281–305 (2012)
10. Huang, C.L., Wang, C.J.: A GA-based feature selection and parameters optimization for support vector machines. Expert Syst. Appl. **31**, 231–240 (2006)
11. Huang, H.L., Chang, F.L.: ESVM: evolutionary support vector machine for automatic feature selection and classification of microarray data. Biosyst. **90**, 516–528 (2007)
12. Åberg, M., Wessberg, J.: Evolutionary optimization of classifiers and features for single trial EEG discrimination. Biomed. Eng. Online **6**, 32 (2007)
13. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. J. Artif. Intell. Res. **36**, 267–306 (2009)
14. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C.A.C. (ed.) LION 2011. LNCS, vol. 6683, pp. 507–523. Springer, Heidelberg (2011)
15. Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 142–157. Springer, Heidelberg (2009)
16. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns, vol. 1, Wiley, New York (1996)
17. Bishop, C.M., Nasrabadi, N.M.: Pattern recognition and machine learning. vol. 1, Springer, New York (2006)
18. Van der Maaten, L., Postma, E., Van Den Herik, H.: Dimensionality reduction: a comparative review. J. Mach. Learn. Res. **10**, 1–41 (2009)
19. Van der Maaten, L.: Matlab Toolbox for Dimensionality Reduction (2014). http://homepage.tudelft.nl/19j49/Matlab_Toolbox_for_Dimensionality_Reduction.html
20. Ma, Y., Fu, Y.: Manifold Learning Theory and Applications. CRC Press, Boca Raton (2011)
21. Tenenbaum, J.B., De Silva, V., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. Science **290**, 2319–2323 (2000)
22. Bengio, Y., Paiement, J.f., Vincent, P., Delalleau, O., Roux, N.L., Ouimet, M.: Out-of-sample extensions for LLE, Isomap, MDS, eigenmaps, and spectral clustering. In: Advances in Neural Information Processing Systems (2003)
23. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: theory and applications. Neurocomputing **70**, 489–501 (2006)
24. Darwin, C.: On the Origins of Species by Means of Natural Selection. Murray, London (1859)
25. Bäck, T.: Evolutionary algorithms in theory and practice. Oxford UniversityPress (1996)
26. Beyer, H.G., Schwefel, H.P.: Evolution strategies - a comprehensive introduction. Nat. Comput. **1**, 3–52 (2002)
27. Müller, M.: Ein Entwurfsmuster für die multikriterielle Parameteradaption mit Evolutionsstrategien in der Bildverarbeitung. VDI-Verlag (2012)
28. Ranawana, R., Palade, V.: Multi-classifier systems: review and a roadmap for developers. Int. J. Hybrid Intell. Syst. **3**, 35–61 (2006)
29. Bache, K., Lichman, M.: UCI machine learning repository (2013). http://archive.ics.uci.edu/ml/