Understanding the Interplay of Simultaneous Model Selection and Representation Optimization for Classification Tasks

Fabian Bürger and Josef Pauli
Intelligent Systems Group, University of Duisburg-Essen, Bismarckstraße 90, 47057 Duisburg, Germany

Keywords: Model Selection, Representation Learning, Classification, Evolutionary Optimization.

Abstract: The development of classification systems that meet the desired accuracy levels for real world-tasks applications requires a lot of expertise. Numerous challenges, like noisy feature data, suboptimal algorithms and hyperparameters, degrade the generalization performance. On the other hand, almost countless solutions have been developed, e.g. feature selection, feature preprocessing, automatic algorithm and hyperparameter selection. Furthermore, representation learning is emerging to automatically learn better features. The challenge of finding a suitable and tuned algorithm combination for each learning task can be solved by automatic optimization frameworks. However, the more components are optimized simultaneously, the more complex their interplay becomes with respect to the generalization performance and optimization run time. This paper analyzes the interplay of the components in a holistic framework which optimizes the feature subset, feature preprocessing, representation learning, classifiers and all hyperparameters. The evaluation on a real-world dataset that suffers from the curse of dimensionality shows the potential benefits and risks of such holistic optimization frameworks.

1 INTRODUCTION

Classifier systems learn the connection between feature data and class labels and are potentially useful in many applications such as image-based object recognition, automated quality inspection and medical diagnosis systems. The development of classifier systems for real-world tasks is still challenging, even though many sophisticated classifiers, such as Support Vector Machines (SVM) or random forests, exist. The reasons for this are manifold: At first, the input feature data can be noisy, especially when raw sensor data is used, like pixel data from camera sensors. The development of task-specific features which are invariant to certain variations is time-consuming and requires a lot of expertise. Secondly, there is no best performing general-purpose machine learning algorithm and so a suitable one has to be chosen. Furthermore, most algorithms have hyperparameters that need to be adapted to each learning task.

Many well known, established solutions to almost all of these challenges exist, like feature selection algorithms, feature preprocessing methods or automatic algorithm and hyperparameter selection methods.

The field of representation learning aims at improving the feature data and it has shown a great performance boost in e.g. deep learning (Bengio et al., 2013). The goal is to automatically generate better suitable features out of low-level data. One way of learning a better representation is manifold learning that results in easier and lower dimensional features (Ma and Fu, 2011).

The large number of potentially useful approaches makes it difficult to select the optimal algorithm combination for each learning task. This challenge has motivated the development of automatic optimization frameworks that can handle a great amount of solutions. Currently, there are two promising optimization approaches to tackle the highly combinatorial so-called algorithm configuration problem. The first approach is Evolutionary Optimization (Bäck, 1996) which simulates the natural evolution process. It has been successfully used to select features and classifier hyperparameters (Huang and Chang, 2007; Ansótegui et al., 2009), also in combination with manifold learning (Bürger and Pauli, 2015). The second approach is Bayesian optimization (Snoek et al., 2012; Hutter et al., 2011) that collects all information about the optimization trajectory in a probabilistic model and evaluates the next most promising areas of the search space. The Auto-WEKA framework (Thornton et al., 2013) uses this approach to optimize features, algo-

Bürger, F. and Pauli, J.
Understanding the Interplay of Simultaneous Model Selection and Representation Optimization for Classification Tasks.
DOI: 10.5220/0005705302830290
In *Proceedings of the 5th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2016)*, pages 283-290
ISBN: 978-989-758-173-1

rithms and hyperparameters.

These holistic frameworks are successful, but the large degree of adaptability of the optimized classifier systems can cause overfitting that degrades the generalization performance. Therefore, a deeper understanding of the interplay of the involved optimization components is necessary. This paper quantifies the impact of feature selection, feature preprocessing, manifold and representation learning, classifier selection and hyperparameter tuning on generalization performance and optimization time using an Evolutionary Algorithm. Furthermore, three aspects regarding the optimization algorithm itself are analyzed as well.

This paper is organized as follows. Section 2 lists a selection of approaches to improve machine learning. Section 3 presents a holistic classification framework that incorporates these aspects. Section 4 discusses the Evolutionary Optimization strategy to adapt the algorithm configuration to each learning task. Section 5 presents the results of the evaluation of the framework. Finally, section 6 contains the conclusions.

2 MACHINE LEARNING SOLUTIONS

This section discusses approaches to improve machine learning subdivided into established methods and representation learning.

2.1 Established Approaches

There are some very popular "standard" approaches to improve the performance of machine learning systems, which are described in e.g. (Jain et al., 2000) or (Bishop and Nasrabadi, 2006).

Feature selection algorithms have the goal to choose a useful subset of features to remove irrelevant and noisy dimensions. Feature selection is a remedy to the curse of dimensionality. *Feature preprocessing* methods are relatively simple algorithms that e.g. normalize the value ranges of the features to a defined range or remove correlations between features which is known as pre-whitening (Juszczak et al., 2002).

Model selection algorithms have the goal to select an optimal classification model for a specific task. This can comprise the selection of a specific algorithm out of an portfolio of alternatives and the tuning of hyperparameters. The best performing model is usually determined by generalization estimation based on model validation techniques such as cross-validation.

2.2 Representation Learning

The development of task-specific features can be one of the most time-consuming parts of the development process. The field of *automatic feature construction* methods is one part of representation learning and aims at automatically learn better feature representations out of the provided data. The family of manifold learning methods provides such functionality – examples are Principal Component Analysis (PCA), Isomap, Local Linear Embedding (LLE) or Autoencoders. References to these methods can be found e.g. in (Van der Maaten et al., 2009) or (Ma and Fu, 2011). Most of these methods are unsupervised and thus do not require expensive, labeled ground truth data. However, the success of these methods depends on the datasets. Furthermore, not all methods provide a so-called direct out-of-sample extension that embeds unseen instances into the learned feature space, but only an approximation which can degrade the generalization performance.

3 HOLISTIC CLASSIFICATION FRAMEWORK

The classification pipeline is similar to our previous work (Bürger and Pauli, 2015) and incorporates all approaches discussed in section 2 into a holistic framework. The pipeline consists of four pipeline elements and is depicted in figure 1. The originally proposed "feature scaling" element is replaced by a generalized preprocessing element. The pipeline has two modes, namely the training and classification mode. The training mode is used to adapt and train the pipeline configuration θ using the training dataset T. This training dataset $T = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$ is the input of the framework and contains m labeled feature vectors $\mathbf{x}_i \in \mathbb{R}^{D_{in}}$ and corresponding class labels $y_i \in \{\omega_1, \omega_2, \dots, \omega_c\}$. The classification mode uses a specific configuration to set up a pipeline which processes the incoming feature vectors and returns class labels. The functionality of each pipeline element is described in the following.

3.1 Feature Selection Element

The first element contains the functionality of feature selection to remove irrelevant features as soon as possible. A feature subset $S_{FeatSet} \in \mathcal{P}(\{1, 2, ..., D_{in}\}) \setminus \emptyset$ is selected during the training mode. In the classification mode, the determined subset of dimensions is selected from the incoming features vectors and the

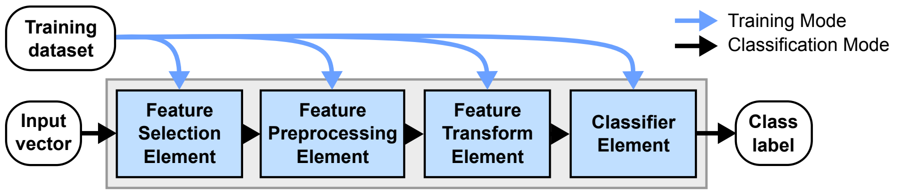

Figure 1: Classification pipeline structure and data processing in the training and classification mode.

others are removed. The feature dimensionality is decreased according to the number of selected features $|S_{FeatSet}|$.

3.2 Feature Preprocessing Element

The second pipeline element contains the functionality of feature preprocessing to provide better usable features for the learning algorithms in the next two pipeline elements. This pipeline element has a portfolio of preprocessing algorithms $S_{PreProc}$ which contains four methods, namely feature rescaling of each dimension to a range of $[0,1]$, statistical standardization, L_2 normalization, pre-whitening and – additionally – the identity. In the training mode one method $f_{PreProc} \in S_{PreProc}$ is chosen. In the classification mode, the selected preprocessing is applied to the incoming feature vectors.

3.3 Feature Transform Element

The third element introduces the aspect of representation learning into the classification pipeline. This is realized in form of feature transforms that are learned using manifold learning techniques. The pipeline element contains a portfolio $S_{FeatTrans}$ with 32 manifold learning techniques provided by the Matlab Toolbox for Dimensionality Reduction (Van der Maaten, 2014). Additionally, similar to the feature preprocessing method, the identity is included, too. In the training mode a method $f_{FeatTrans} \in S_{FeatTrans}$ as well as values for the specific set of hyperparameters $S_{Hyp}(f_{FeatTrans})$ are chosen. Additionally, the target dimensionality D_{Target} of the resulting feature space needs to be chosen which is a common hyperparameter of all manifold learners. Then the manifold learner is trained with the feature vectors arriving from the feature preprocessing element. Most manifold learning algorithms are unsupervised, but the supervised ones additionally use the corresponding labels from the training dataset T. In the classification mode, the out-of-sample extension or approximation of the selected and trained manifold learner is applied to process the feature vectors.

3.4 Classifier Element

The fourth pipeline element is finally responsible to perform the actual classification. The pipeline element contains a portfolio $S_{Classifiers}$ of eight popular classifier concepts and variants, e.g. kernel Support Vector Machines (SVM), random forests and the multilayer perceptron. In the training mode, a classifier concept $f_{Classifier} \in S_{Classifiers}$ is selected as well as its specific set of hyperparameters $S_{Hyp}(f_{Classifier})$. The classifier is trained using the processed feature vectors and the corresponding labels in the training dataset T. In the classification mode, the trained classifier is used to classify instances.

3.5 Pipeline Configuration

The pipeline configuration contains all necessary information about the selected features, algorithms and hyperparameters and is defined as

$$\theta = (S_{FeatSet}, f_{PreProc}, f_{FeatTrans}, S_{Hyp}(f_{FeatTrans}), D_{Target}, f_{Classifier}, S_{Hyp}(f_{Classifier})). \quad (1)$$

The number of possible configurations is huge and also depends on the training dataset, because there are $2^{D_{in}} - 1$ possible feature subsets to choose from.

4 OPTIMIZATION FRAMEWORK

The pipeline configuration θ needs to be adapted to every learning task defined by the training dataset T. An optimization algorithm is required that can handle the large search space which will likely also contain a large number of local optima. Evolutionary Algorithms have the potential to cope with these challenges and it is known that they can be easily run in parallel – which is not the case for a Bayesian optimization approach.

4.1 Extended Evolution Strategies

Evolution Strategies (ES) (Beyer and Schwefel, 2002) are one variant of Evolutionary Algorithms which are suitable to solve high-dimensional optimization problems. A solution is coded into the genetic information of an individual and a population contains several individuals. Starting from an initial population which is usually generated randomly, the best individuals are selected according to their fitness. The fitness is usually determined using the objective function of the optimization problem (see section 4.3). The selected individuals are recombined and randomly mutated to generate a new generation of hopefully improved individuals. This procedure is repeated until the termination criteria are fulfilled, e.g. too little fitness improvements or a time limit.

The standard ES is only defined for numeric variables, but all necessary evolutionary operators can be extended to more data types which is described by the authors of (Bürger and Pauli, 2015). The configuration adaption problem requires a larger variety of variable types, because of the feature selection and hyperparameter tuning problem. Therefore, the following variable types are used:

- The real-valued variable type $V_{\mathbb{R}}$,
- the integer variable type $V_{\mathbb{Z}}$,
- the categorical variable type $V_{\mathbb{S}}$ and
- the Boolean variable type $V_{\mathbb{B}}$.

The numeric variables also contain information about minimum and maximum values and the categorical variables the base set of all possible categories.

The parametrization of an ES-based optimization algorithm is summarized in the $(\mu,\kappa,\lambda,\rho)$ notation. In each generation $\lambda = 200$ individuals from $\rho = 3$ parents are generated while the best $\mu = 20$ individuals survive. The maximum lifespan of individuals is limited to $\kappa = 4$ generations. The initial population contains $\mu_{init} = 400$ random individuals. The number of individuals should be as high as possible to increase the chances to find better solutions. However, there must be a trade-off between computational complexity and the risk of getting stuck in local optima.

4.2 Evolutionary Configuration Adaptation

The pipeline configurations have to be coded into the genotype of the ES which is a sequence of N variables

$$G = [V_{*,1}, V_{*,2}, \dots, V_{*,N}],\ V_* \in \{V_{\mathbb{R}}, V_{\mathbb{Z}}, V_{\mathbb{S}}, V_{\mathbb{B}}\}. \quad (2)$$

The Evolutionary Configuration Adaption (*ECA*) schema is used to code a configuration into the genotype as depicted in figure 2. First, the feature selection problem is coded as D_{in} binary variables that indicate whether a feature is selected or not. The feature

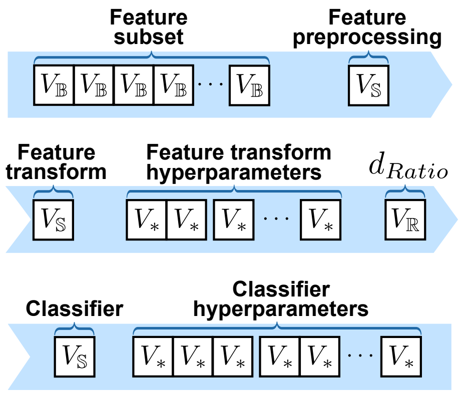
Feature subset
Feature preprocessing
$V_{\mathbb{B}}$ $V_{\mathbb{B}}$ $V_{\mathbb{B}}$ $V_{\mathbb{B}}$ ··· $V_{\mathbb{B}}$
$V_{\mathbb{S}}$
Feature transform
Feature transform hyperparameters
d_{Ratio}
$V_{\mathbb{S}}$
V_* V_* V_* ··· V_*
$V_{\mathbb{R}}$
Classifier
Classifier hyperparameters
$V_{\mathbb{S}}$
V_* V_* V_* V_* V_* ··· V_*

Figure 2: ES genotype coding schema of a pipeline configuration.

preprocessing method is coded as a single categorical variable. The feature transform is also selected with a categorical variable. All hyperparameters of all feature transforms are also appended to the genome with their corresponding variable type. When the individuals are transformed to a configuration, only those hyperparameters are used that belong to the selected feature transform. The target dimensionality depends on the number of selected features and therefore it is coded as a real-valued ratio $d_{Ratio} \in [0,1]$. The actual value for the target dimensionality is then calculated as $D_{Target} = \lfloor d_{Ratio} \cdot |S_{FeatSet}| \rfloor$. The classifier is coded similarly to the feature transform as categorical variable and the corresponding hyperparameters are also handled in the same way. Each individual I can be mapped to a valid pipeline configuration θ.

4.3 Fitness Function

The key role of ES is the selection of individuals I by their fitness $fit(I)$. In case of a classification pipeline, the fitness should be directly connected to the expected generalization performance of the whole pipeline. Therefore, a holistic k-fold cross-validation (HCV) is used that considers the generalization of all components. This is necessary, because especially the feature transform is expected to contribute to the generalization.

The HCV method uses the training dataset T to generate $k = 5$ disjoint training and validation tuples $\{(T_{train,l}, T_{valid,l})\}$ with $1 \le l \le k$. In each cross-validation round, the classification pipeline is first trained using $T_{train,l}$. Subsequently, the instances of the validation dataset $T_{valid,l}$ are classified with this pipeline and the overall accuracy value $q_{acc,l}$ is calculated. This makes sure that the validation data is never used for training. During the cross-validation process, the average overall accuracy so far

$$\overline{q}_{acc,l} = \frac{1}{l}\sum_{j=1}^{l} q_{acc,j} \tag{3}$$

is calculated. This value is used for an early discarding system to save computation time. The cross-validation process is stopped prematurely for bad performing configurations that do not improve the previously observed best accuracy value q^*_{acc}. The early discarding system is relatively aggressive and the process is stopped if $\overline{q}_{acc,l} < q^*_{acc}$. The last average is used as fitness value for the individuals $fit(I) = \overline{q}_{acc,l}$.

This fitness metric is also used as termination criterion. The optimization is stopped when the best fitness of the current generation increases less than $\varepsilon = 10^{-3}$ (equal to 0.1 percent of accuracy) over the last three generations.

4.4 Initial Population Improvement

The feature selection problem is expected to have the largest impact on the problem complexity and therefore, it is promising to speedup and improve the ES optimization with an improved initial population. Random forests (Breiman, 2001) provide an integrated feature importance metric proposed by (Genuer et al., 2010). Before the actual ES optimization begins, this metric β_i is calculated for each feature with $1 \leq i \leq D_{in}$. The initial probability for each feature is determined by the importance metric using

$$p_i = p_{min} + (1 - p_{min})\frac{\beta_i - \beta_{min}}{\beta_{max} - \beta_{min}} \in [p_{min}, 1] \tag{4}$$

in which β_{min} and β_{max} are the minimum and maximum values of β_i across all features. The $p_{min} = 0.25$ value defines a minimum probability value for the least important variable. This makes sure that the selection of this variable does not become impossible.

4.5 Optimization Algorithm Variants

One of the main goals of this paper is to quantify and understand the impact of the components of the classification pipeline. This is achieved by the introduction of restricted variants of the *ECA* optimization algorithm that work with the leave-one-out principle for each component. The following variants are considered:

- The *ECA-full* variant uses all components,
- the *ECA-noFeatSel* variant does not consider feature selection and always uses the full feature subset,
- the *ECA-noPreProc* variant does not use any feature preprocessing method,
- the *ECA-noTrans* variant does not use manifold learning or feature transform methods,
- the *ECA-simpleClassifier* variant only contains a simple classifier, namely the naive Bayes classifier and
- the *ECA-defaultHyper* variant does not consider hyperparameter tuning and leaves all hyperparameters at their standard values.

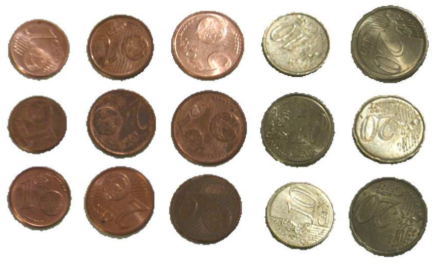
Figure 3: Three example images from each class of the coins dataset.

5 EVALUATION

An image-based object recognition task is used to evaluate the components of the framework. The task is to classify color images of five classes of Euro coins which are depicted in figure 3. There are 64 color images from 1-, 2-, 5-, 10- and 20-cent coins leading to 320 images in total. The following feature set is extracted from each image:

- The object area in pixels,
- statistical features of the gray value and color channel histogram,
- Hu moments of the gray value texture (Hu, 1962),
- Local Binary Patterns (Ojala et al., 2002) of the gray value texture,
- low-level pixel features in form of down-scaled gray value images: 5×5, 10×10, and 20×20 pixels.

The total dimensionality of the feature space is $D_{in} = 642$. In combination with the small number of samples negative effects due to the curse of dimensionality can be expected. However, this makes this dataset particularly interesting to analyze. The dataset is separated randomly into 50 % training and 50 % test dataset.

The reported *cross-validation accuracy* is obtained on the training dataset and is equal to the fitness of the best individual. The *generalization performance* is measured by using the configuration with the highest fitness value to set up a classification pipeline and process and evaluate the test dataset with it. The *optimization time* is measured as well.

The performance of the *ECA* strategy is compared to two baseline methods, namely an SVM classifier with a Gaussian kernel (denoted as *Baseline-SVM*) and a random forest (denoted as *Baseline-RF*). The baselines also use a grid-based hyperparameter tuning with cross-validation. Furthermore, the performance of the Auto-WEKA framework (Thornton et al., 2013) with a time budget of 24 hours is compared as well. Each experiment with the framework is repeated ten times to obtain statistically relevant results. All performance metrics are compared relatively to the *ECA-full* strategy using a Welch test[1] (Howell, 2006) with a level of significance of $\alpha = 0.05$ to show if the deviations are statistically significant. If this is the case, the corresponding results are marked with an exclamation mark.

The framework is implemented in Matlab 2014b using the parallel computing toolbox and is running on a workstation computer with 6×2.5Ghz and 32GB of RAM.

5.1 Results using All Components

The *ECA-full* variant of the framework can use the full potential of all pipeline elements and is therefore the most promising and interesting approach. Table 1 shows the results regarding the accuracy values and optimization times. The two baseline methods, especially the SVM, perform badly on this dataset. An explanation is that this dataset suffers from the curse of dimensionality. The *ECA-full* variant shows much higher cross-validation and generalization accuracy values which also outperform the Auto-WEKA framework significantly.

The typical processing time of the *ECA-full* strategy stays under two hours. The two baseline methods are much faster, because these are simple classifiers which just undergo a grid search for the best hyperparameters. The Auto-WEKA framework uses its time budget of 24 hours to optimize and does not prematurely stop.

5.2 Impact of the Pipeline Components

The *ECA-full* variant provides the greatest amount of adaptability and performs well, but it is important to understand the contribution of the classification pipeline components. Figure 4 shows the impact of the pipeline components on the accuracy and the optimization times by comparing the restricted *ECA* variants (see section 4.5) to the *ECA-full* variant.

All restricted variants of the *ECA* strategy show worse average accuracy values than the *ECA-full* variant (see the left side of figure 4). This observation affects the cross-validation and the generalization accuracy values. All differences, except of the cross-validation accuracy of the *ECA-defaultHyper* variant, are also statistically significant. The classifier portfolio has the largest impact here, because the *ECA-simpleClassifier* variant which uses only the Bayes classifier achieves around 15 % less generalization accuracy on average. The other components are obviously not able to compensate a too simple classifier. The impact of the other components is smaller, but still significant. It is interesting that the feature preprocessing has a slightly larger impact than the feature transforms, because the *ECA-noPreProc* variant performs worse than the *ECA-noTrans* variant. This indicates that relatively simple preprocessing methods can lead to more improvement than 32 highly nonlinear manifold learning methods.

The hyperparameter tuning component has a relatively small impact on the accuracy values. It is likely the case that – by chance – the standard hyperparameters already work reasonably well for the dataset. Also the aspect of feature selection is less important here, because the accuracy values drop only about 1 - 2 % if feature selection is switched off.

The different pipeline components also affect the average processing times (see the right side of figure 4) and most of the differences are statistically significant. The use of feature selection speeds up the optimization tremendously by more then 50 minutes on average, because a lower dimensional feature space requires less computation time for most of the involved algorithms. The use of feature transforms slows down the optimization process by almost 50 minutes on average. This can be explained with the computationally complex models of the manifold learning algorithms, but also their potentially experimental implementation within Matlab. The use of the feature preprocessing methods also slows down the optimization process by more than 20 minutes on average. But the reason is not the computational complexity of the preprocessing methods – which are simple actually –, but that more complex feature transforms and classifiers perform better on preprocessed features and therefore are chosen more often in the ES optimization. Another reason is the larger search space. A similar explanation can be found to justify the same effect when hyperparameter tuning is active.

[1]The Welch test is a statistical method to show if the means of two samples are significantly different while no assumptions about the sample variances are made.

Table 1: Comparison of cross-validation and generalization accuracy results as well as optimization times (mean ±1σ).

	Cross-validation acc.	Generalization acc.	Optimization time [min]
ECA-full	**0.935** ± 0.015	**0.951** ± 0.013	73.74 ± 20.28
Baseline-SVM	0.119 ± 0.013 (!)	0.281 ± 0.020 (!)	**0.02** ± 0.01 (!)
Baseline-RF	0.623 ± 0.028 (!)	0.708 ± 0.045 (!)	0.48 ± 0.01 (!)
Auto-WEKA	(not comparable)	0.930 ± 0.014 (!)	1440 = 24 hours (!)

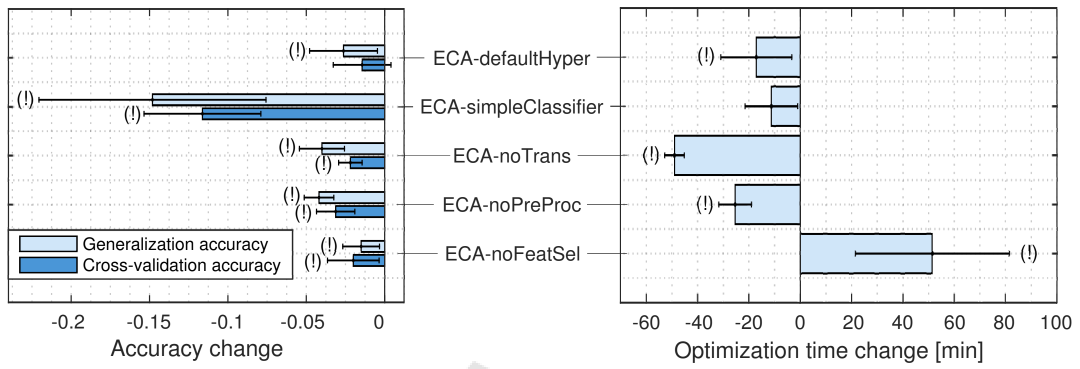

Figure 4: Impact of the pipeline components on accuracy values and optimization times compared to the *ECA-full* variant. The standard deviations are denoted as lines around the end of the bars.

5.3 Impact of the ECA Algorithm

In the following, the impact of three central aspects of the *ECA* optimization algorithm are investigated by switching off the corresponding component of the *ECA-full* algorithm. Table 2 lists the results regarding the accuracy values and the optimization times.

5.3.1 Impact of Holistic Cross-validation

Instead of HCV (see section 4.3), a classifier-only cross-validation approach is used for this experiment. The pipeline is trained using the full training dataset and only the classifier performs cross-validation. With this procedure, the generalization of the feature transform is never considered. This leads to a cross-validation accuracy value of 100 %, but the resulting generalization accuracy is unusably bad. However, the average optimization time drops slightly due to the reduced computational complexity of the classifier-only cross-validation.

5.3.2 Impact of Early Discarding

The early discarding system is an aggressive system to save computation time (see section 4.3). It can be seen that the average computation time rises tremendously by a factor of around 3.8 when the early discarding system is not used. The system does not have any significant positive or negative effect on the accuracy values.

5.3.3 Impact of Initial Population Improvement

The initial population improvement affects the feature selection (see section 4.4). If there is no initial improvement and a completely random initialization of the features is used, the cross-validation and generalization accuracy values drop significantly. The computation time actually drops slightly, but not significantly, if the initial population is not improved. It is likely the case that the termination criterion of the ES algorithm was fulfilled too early.

6 CONCLUSIONS

This paper analyzed the impact of multiple aspects of a holistic model selection and representation optimization framework on accuracy values and optimization times. The first experiment analyzed the impact of the components of the classification pipeline. It was shown that all involved components, namely feature selection, multiple preprocessing methods, multiple feature transforms, multiple classifiers and hyperparameter tuning are potentially contributing to the generalization performance. However, multiple observations have been made that indicate that the interplay between the involved components is complex – especially regarding the optimization run times. It can be expected that some effects are highly dependent on the dataset.

The second experiment analyzed the impact of

Table 2: Impact of the central components of the *ECA* optimization algorithm on the accuracy values and optimization times (mean $\pm 1\sigma$).

	Cross-validation acc.	Generalization acc.	Optimization time [min]
ECA-full	0.935 ± 0.015	**0.951** ± 0.013	73.74 ± 20.28
No holistic cross-validation	**1.000** ± 0.0 (!)	0.287 ± 0.022 (!)	59.95 ± 9.16
No early discarding system	0.938 ± 0.024	0.933 ± 0.027	276.15 ± 59.64 (!)
No initial population improvement	0.912 ± 0.022 (!)	0.925 ± 0.025 (!)	**59.30** ± 14.95

three aspects of the optimization algorithm itself. At first, it was shown that the incorporation of the feature transform into the cross-validation process is absolutely necessary. Secondly, the early discarding system greatly improves the optimization speed while the resulting accuracy values are not affected in a negative way. And lastly, the incorporation of prior knowledge about the importance of features into the optimization algorithm improved the accuracy.

Ultimately, it can be concluded that not only the amount of optimized components is important, but also the suitability of the optimization algorithm itself. However, further experiments on other datasets need to be conducted to explore the full variety of effects regarding the complex interplay of the machine learning components.

REFERENCES

Ansótegui, C., Sellmann, M., and Tierney, K. (2009). A gender-based genetic algorithm for the automatic configuration of algorithms. In Gent, I., editor, *Principles and Practice of Constraint Programming - CP 2009*, volume 5732 of *Lecture Notes in Computer Science*, pages 142–157. Springer Berlin Heidelberg.

Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Oxford, UK.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1798–1828.

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

Bishop, C. M. and Nasrabadi, N. M. (2006). *Pattern recognition and machine learning*, volume 1. Springer New York.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Bürger, F. and Pauli, J. (2015). Representation optimization with feature selection and manifold learning in a holistic classification framework. In De Marsico, M. and Fred, A., editors, *ICPRAM 2015 - Proceedings of the International Conference on Pattern Recognition Applications and Methods*, volume 1, pages 35–44, Lisbon, Portugal. INSTICC, SCITEPRESS.

Genuer, R., Poggi, J.-M., and Tuleau-Malot, C. (2010). Variable selection using random forests. *Pattern Recognition Letters*, 31(14):2225 – 2236.

Howell, D. C. (2006). *Statistical Methods for Psychology*. Wadsworth Publishing.

Hu, M.-K. (1962). Visual pattern recognition by moment invariants. *Information Theory, IRE Transactions on*, 8(2):179–187.

Huang, H.-L. and Chang, F.-L. (2007). ESVM: Evolutionary support vector machine for automatic feature selection and classification of microarray data. *Biosystems*, 90(2):516 – 528.

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

Jain, A. K., Duin, R. P. W., and Mao, J. (2000). Statistical pattern recognition: a review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(1):4–37.

Juszczak, P., Tax, D., and Duin, R. (2002). Feature scaling in support vector data description. In *Proc. ASCI*, pages 95–102. Citeseer.

Ma, Y. and Fu, Y. (2011). *Manifold Learning Theory and Applications*. CRC Press.

Ojala, T., Pietikainen, M., and Maenpaa, T. (2002). Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):971–987.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959.

Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2013). Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proc. of KDD-2013*, pages 847–855.

Van der Maaten, L. (2014). *Matlab Toolbox for Dimensionality Reduction*. http://lvdmaaten.github.io/drtoolbox/.

Van der Maaten, L., Postma, E., and Van Den Herik, H. (2009). Dimensionality reduction: A comparative review. *Journal of Machine Learning Research*, 10:1–41.