Automatic Frankensteining: Creating Complex Ensembles Autonomously

Martin Wistuba *   Nicolas Schilling *   Lars Schmidt-Thieme *

Abstract

Automating machine learning by providing techniques that autonomously find the best algorithm, hyperparameter configuration and preprocessing is helpful for both researchers and practitioners. Therefore, it is not surprising that automated machine learning has become a very interesting field of research. While current research is mainly focusing on finding good pairs of algorithms and hyperparameter configurations, we will present an approach that automates the process of creating a top performing ensemble of several layers, different algorithms and hyperparameter configurations. These kinds of ensembles are called jokingly Frankenstein ensembles and proved their benefit on versatile data sets in many machine learning challenges. We compare our approach Automatic Frankensteining with the current state of the art for automated machine learning on 80 different data sets and can show that it outperforms them on the majority using the same training time. Furthermore, we compare Automatic Frankensteining on a large scale data set to more than 3,500 machine learning expert teams and are able to outperform more than 3,000 of them within 12 CPU hours.

1 Introduction

Algorithm selection and hyperparameter optimization is an omnipresent problem for data science practitioners and researchers. Nevertheless, they are usually not concerned about which algorithm or which hyperparameter configuration is selected but they want to have an accurate prediction model. Hence, it is not surprising that many main machine learning players such as Amazon, IBM, Google and SAS offer commercial tools that at least partially automate this process.

Automating machine learning also attracted a lot of attention of machine learning researchers in the past years. Various studies were able to show that automatic hyperparameter optimization is able to outperform human experts [2, 15]. Furthermore, the concept was extended to select preprocessing, algorithm and hyperparameter configurations altogether. The two most famous publicly available tools that offer these features are Auto-WEKA [18] and auto-sklearn [5]. Often, the tools for automating machine learning are extended by methods from meta-learning to transfer knowledge from observed data sets to new ones to initialize the search [19].

The current research focuses strongly on the question which algorithm/hyperparameter configuration combination is best. But in fact the true task that needs to be tackled is to find the strongest prediction model which of course does not need to be estimated from a single algorithm but might be an ensemble of many of them.

While detailed machine learning solutions for real problems created by companies are often confidential, the many competitions hosted by platforms such as Kaggle, DrivenData and CodaLab give great insight how high performance systems for company data can be created. The top performing solutions are often based on ensembles with many layers and various types of preprocessing, hyperparameter configurations and algorithms. These solutions are often very complex and hence this kind of ensembling is jokingly called Frankensteining named after the novel by Mary Shelly [16] in which the scientist Frankenstein creates an ugly artificial life form of many different components. Inspired by these solutions, we try to improve the current state of the art for automated machine learning by proposing a way that will find solutions like these Frankenstein ensembles that make intensive use of diverse algorithms and multi-layer ensembling. Hence, we propose *Automatic Frankensteining*, an automatic approach to generate deep stacked ensembles. While some research already identified that an average ensemble improves the performance [11, 5, 12], no one has looked into the direction of deep ensembles that consist of several layers. Thus, we are the first to propose a machine learning approach that autonomously identifies a good set of hybrid base learners for a stacked ensemble as well as the right combiner for all base learners. We compare our approach on 80 different classification data sets from the UCI repository to the current state of the art approaches Auto-WEKA and auto-sklearn and show that Automatic Frankensteining is outperforming its competitors on the large majority of data sets using the same CPU time. Finally, we compare our approach to the performance of more than 3,500 human machine learning expert teams on a large scale business data set.

*Information Systems and Machine Learning Lab, University of Hildesheim, Germany

Therefore, to the best of our knowledge, we are the first to compare automated machine learning to this very large amount of human experts. Additionally, the solution found by Automatic Frankensteining is outperforming more than 3,000 of the human experts on this task within 12 hours CPU time while each human expert team had 62 days.

2 Related Work

Automating machine learning has become a hot topic in the past years. Snoek et al. [17] were among the first to show that Bayesian optimization can be used to automate hyperparameter optimization. Soon afterwards, this idea was extended to the problem of selecting algorithms and preprocessing techniques [18]. Various approaches to make use of meta-knowledge either through initialization [6] or transfer surrogate models [1] have been proposed to accelerate the search for good models. Lately, first tries to make use of ensembling techniques have been proposed. Feurer et al. [5] identified that the many prediction models estimated during Bayesian optimization can be used afterwards to create a weighted ensemble. Lacoste et al. [11] proposes a Bayesian approach. They combine those models that perform best on a randomly selected subset of the validation data. Finally, Levesque [12] proposes to use Bayesian optimization directly to estimate which prediction model is the best candidate to be added to the ensemble. Thornton et al. [18] do not focus on the problem of creating strong ensembles but they make use of more advanced ensembling techniques by considering the structure of the ensemble as further hyperparameters.

In contrast to the state of the art, we propose two innovations. First, we learn strong prediction models per algorithm instead of an overall strong model to obtain strong base learners for the ensemble. Second, we create deep ensembles instead of flat ensembles automatically. Therefore, we use Bayesian optimization to directly estimate strong base learners for ensembling techniques such as stacking instead of finding models that are good in general and create an ensemble as a by-product.

3 Background

In the next section we will formalize the problem and describe the basics of automating machine learning and ensembles.

3.1 Problem Definition In supervised machine learning one tries to learn a strong prediction model based on a labeled data set D with n instances and m predictors $X \in \mathbb{R}^{n\times m}$ with corresponding labels $y \in \mathbb{R}^n$ that minimizes a given loss function $\mathcal{L}$ for future data instances. This raises the questions which is the best machine learning algorithm of a set of possible algorithms $A_1, \ldots, A_k$ for a problem where a machine learning algorithm A uses the labeled training data to estimate a prediction model $\hat{y}$. Most machine learning algorithms also have parameters that are not learned during the training phase but need to be set upfront. A typical example is the trade-off parameter C of a linear support vector machine. These parameters are usually called *hyperparameters*. Since the resulting model is usually sensitive to the chosen hyperparameter configuration, we actually want to know which algorithm combined with what hyperparameter configuration leads to the model that achieves the smallest loss.

Let us formalize the different parts of machine learning to ease talking about them. A machine learning algorithm A is a function $A : \mathbb{R}^{n\times m} \times \mathcal{Y}^n \times \Lambda \to \mathcal{M}$ where Λ is the hyperparameter space, $\mathcal{Y}$ the target space and $\mathcal{M}$ the space of all prediction models. The result of this function is a prediction model $\hat{y} : \mathbb{R}^m \to \mathcal{Y}$ that allows to make predictions for new instances. The target space $\mathcal{Y}$ depends on the problem. For regression problems $\mathcal{Y} = \mathbb{R}^n$ and for classification problems $\mathcal{Y} = \mathbb{R}^{n\times c}$ for a set of classes $\mathcal{C} = \{1, \ldots, c\}$. For the groundtruth $\mathrm{Y} \in \mathcal{Y}$ of a classification problem usually yields that $y_{i,j} = 1$ if instance i belongs to class j and $y_{i,j} = 0$ otherwise.

We will focus on classification problems as a specific problem instance in this paper. Typical classification losses are the classification error

$$\mathcal{L}\left(\mathrm{Y}, \hat{\mathrm{Y}}\right) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\left(\arg\max_{j\in\mathcal{C}} y_{i,j} \neq \arg\max_{j\in\mathcal{C}} \hat{y}_{i,j}\right) \tag{3.1}$$

and the logistic loss

$$\mathcal{L}\left(\mathrm{Y}, \hat{\mathrm{Y}}\right) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c} y_{i,j}\log\left(\hat{y}_{i,j}\right) \quad , \tag{3.2}$$

where Y is the groundtruth and $\hat{\mathrm{Y}}$ the prediction.

The hyperparameter space Λ depends on the algorithm A. In the aforementioned case of a linear support vector machine with the only hyperparameter being the trade-off C, the hyperparameter space would equal $\mathbb{R}^+$, however, hyperparameter spaces are often multi-dimensional.

3.2 Automatic Hyperparameter Optimization Most state of the art methods for automatic hyperparameter optimization are based on Bayesian optimization [13]. The hyperparameter optimization problem for an algorithm A is defined as

$$\arg\min_{\boldsymbol{\lambda}\in\Lambda} f(\boldsymbol{\lambda}) = \arg\min_{\boldsymbol{\lambda}\in\Lambda} \mathcal{L}\left(\mathrm{Y}_{\text{valid}}, M_{\boldsymbol{\lambda}}\left(D_{\text{valid}}\right)\right) \tag{3.3}$$

with $M_{\boldsymbol{\lambda}} = A\left(D_{\text{train}}, \boldsymbol{\lambda}\right)$ for a data set $D \in \mathbb{R}^{n\times m} \times \mathcal{Y}$ which is split into train and validation fold. Adding algorithms to the representation of $\boldsymbol{\lambda}$, this representation can also be used to describe the problem of combined algorithm and hyperparameter configuration selection.

To minimize the function f, Bayesian optimization uses two different components: the surrogate model Ψ and the acquisition function a. The surrogate model Ψ is a regression model that provides some uncertainty about its prediction. This model approximates f by using all known observations of f stored in the history $\mathcal{H}$. The acquisition function uses the predicted mean and uncertainty of Ψ to assess to what degree a given $\boldsymbol{\lambda}$ finds the balance between exploitation and exploration. The chosen $\boldsymbol{\lambda}$ is then evaluated on f, the result stored in $\mathcal{H}$ and Ψ is updated. Algorithm 1 explains Bayesian optimization in detail. In our experiments we rely on the most common choices for both components. We choose a Gaussian process as a surrogate model and expected improvement [10] as the acquisition function.

```
Algorithm 1 Bayesian Optimization
Input: Hyperparameter space $\Lambda$, observation history $\mathcal{H}$, time limit $t$, acquisition function $a$, surrogate model $\Psi$.
Output: Best hyperparameter configuration found.
1: while time limit $t$ is not reached do
2:     Fit $\Psi$ to $\mathcal{H}$
3:     $\boldsymbol{\lambda} \leftarrow \arg\max_{\boldsymbol{\lambda}\in\Lambda} a\left(\boldsymbol{\lambda}, \Psi, f^{\min}\right)$
4:     Evaluate $f(\boldsymbol{\lambda})$
5:     $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\boldsymbol{\lambda}, f(\boldsymbol{\lambda}))\}$
6:     if $f(\boldsymbol{\lambda}) < f^{\min}$ then
7:         $\boldsymbol{\lambda}^{\min}, f^{\min} \leftarrow \boldsymbol{\lambda}, f(\boldsymbol{\lambda})$
8: return $\boldsymbol{\lambda}^{\min}$
```

3.3 Ensembling Techniques

An ensemble of classifiers is a combination of classifiers $h_1, \ldots, h_e$. The simplest way of combining them is by *voting* i.e. that each classifier has an equal vote and the majority decides which class is predicted. If the probability score is predicted one can combine the prediction by averaging the score over all classifiers.

$$\hat{y}(x) = \frac{1}{e}\sum_{i=1}^{e} h_i(x) \tag{3.4}$$

Instead of using an unweighted voting/averaging, one can also assign weights to the classifiers. This can be done manually or also learned on a hold-out data set [3]. Important for an ensemble is that its members are accurate by itself (better than random) and diverse which means they are stronger than the other members on specific parts of the data [8]. A more advanced technique to combine classifiers is *stacking.* In the case of stacking, the data set needs to be split into disjoint sets D_1 and D_2. The ensemble members will be trained on D_1 and predictions are estimated for all $(x, y) \in D_2$ and vice versa. After obtaining the meta-instances

$$\left(\begin{pmatrix} h_1(x) & \cdots & h_e(x) \end{pmatrix}^{\mathrm{T}}, y\right) , \tag{3.5}$$

a combiner algorithm is learned on these meta-instances which can be any classifier or even another ensemble.

4 Automatic Frankensteining

In the Automatic Frankensteining framework we distinguish between two different components. The *model selection component* is responsible for training models and finding near-optimal hyperparameter configurations. The *ensembling component* combines the models trained in the preceding model selection component to further boost the prediction performance and reduce the input space for the subsequent model selection component.

4.1 Model Selection Component

The task of the model selection component is to create useful meta-features or to provide the final predictions based on some given input. On an abstract level, the model selection component can be considered as a function $g : \mathcal{X} \to \mathcal{Y}^*$ that converts a data set representation X_2 to an intermediate representation $g(X_2)$. Therefore, it needs to be trained previously on a data set D_1 with predictors X_1 and corresponding labels y_1. This conversion is done by learning machine learning algorithms $A_1, \ldots, A_k$ on D_1 that create models $\hat{y} : \mathcal{X} \to \mathcal{Y}$. Using these models to create predictions $\left(\hat{y}_1(X_2), \ldots, \hat{y}_l(X_2)\right)^T$ we obtain a new representation.

Example 4.1. *Assume that only one model is used to generate a new representation for some predictors X_2. Then one can e.g. use a decision tree A_1 with a chosen hyperparameter configuration $\boldsymbol{\lambda}_1$ and train it on the data set D_1. A prediction model $\hat{y}$ is obtained which can be used to make predictions for X_2, i.e. $\hat{y}(X_2)$. These predictions are used as the new representation for X_2.*

As previously explained in Section 3.3, good meta-features are created by a diverse set of machine learning algorithms. Here, diversity rather than prediction performance is important. Nevertheless, very weak or useless prediction models will also lead to bad meta-

Algorithm 2 Training the Model Selection Component

```
Input: Hyperparameter spaces $\Lambda_{1,\dots,k}$, observation histories $\mathcal{H}_{1,\dots,k}$, surrogate models $\varPsi_{1,\dots,k}$, acquisition function $a$, time limit $t$, train and validation data sets $D_{\text{train}} = (X_{\text{train}}, y_{\text{train}}), D_{\text{valid}} = (X_{\text{valid}}, y_{\text{valid}})$.
Output: Meta-features $\hat{\mathrm{Y}}$ for $D_{\text{valid}}$, loss $\mathrm{L}$ for columnwise predictions on $D_{\text{valid}}$.
 1: $\mathrm{L} \leftarrow ()\,,\ \hat{\mathrm{Y}} \leftarrow ()$
 2: while time limit $t$ is not reached do
 3:     Choose $j \in \{1, \dots k\}$ proportional to $|\Lambda_j|$.
 4:     Fit $\varPsi_j$ to $\mathcal{H}_j$
 5:     $\boldsymbol{\lambda} \leftarrow \arg\max_{\boldsymbol{\lambda} \in \Lambda_j} a\left(\boldsymbol{\lambda}, \varPsi_j, f_j^{\min}\right)$
 6:     if predicted run-time for $A_j\left(X_{\text{train}}, y_{\text{train}}, \boldsymbol{\lambda}\right)$ does not exceed the remaining run-time then
 7:         $\hat{y} \leftarrow A_j\left(X_{\text{train}}, y_{\text{train}}, \boldsymbol{\lambda}\right)$
 8:         $f_j\left(\boldsymbol{\lambda}\right) \leftarrow \mathcal{L}\left(y_{\text{valid}}, \hat{y}\left(X_{\text{valid}}\right)\right)$
 9:         $\mathcal{H}_j \leftarrow \mathcal{H}_j \cup \left\{\left(\boldsymbol{\lambda}, f_j\left(\boldsymbol{\lambda}\right)\right)\right\}$
10:         $\hat{\mathrm{Y}} \leftarrow \begin{pmatrix} \hat{\mathrm{Y}} & \hat{y}\left(X_{\text{valid}}\right) \end{pmatrix}$
11:         $\mathrm{L} \leftarrow \begin{pmatrix} \mathrm{L} & f_j\left(\boldsymbol{\lambda}\right) \end{pmatrix}$
12:         if $f_j\left(\boldsymbol{\lambda}\right) < f_j^{\min}$ then
13:             $f_j^{\min} \leftarrow f_j\left(\boldsymbol{\lambda}\right)$
14: return $\hat{\mathrm{Y}}, \mathrm{L}$
```

features. Hence, it is vital to select appropriate hyperparameter configurations for the machine learning algorithms. The model selection component will solve both issues. Having access to k machine learning algorithms $A_1, \dots, A_k$, it will automatically search for good hyperparameter configurations for each of them. In contrast to the state of the art for automatic model selection and hyperparameter tuning, which is interested in finding *the* best model with *the* best hyperparameter configuration, we are interested in finding predictive meta-features. Thus, we need to find for each algorithm a hyperparameter configuration that leads to a good prediction model itself. Hence, the hyperparameter search differs from the current state of the art approaches. While the state of the art is applying a global hyperparameter optimization over the whole algorithm and hyperparameter space, we apply a hyperparameter optimization per algorithm. Because the hyperparameter dimensionality between algorithms might differ a lot, as a logistic regression has only one to two hyperparameters while a neural network might have a dozen, each of the k individual hyperparameter searches is continued with probability proportional to the number of hyperparameters. For the automatic hyperparameter search, we make use of the current state of the art approach Bayesian optimization explained in Section 3.2.

Algorithm 2 summarizes the training procedure of the model selection component. Given a set of k learning algorithms $A_1, \dots, A_k$, a training and validation data set D_{train} and D_{valid}, k-many hyperparameter searches are executed in parallel for a given time limit t. Whenever resources are free to evaluate a new hyperparameter configuration $\boldsymbol{\lambda}$, an algorithm A_j is selected proportional to the dimensionality of the hyperparameter search. Following the typical Bayesian optimization method, the surrogate model $\varPsi_j$ is updated and the hyperparameter configuration for the chosen algorithm A_j is selected that maximizes the acquisition function. If the predicted run-time of evaluating A_j with the selected hyperparameter configuration $\boldsymbol{\lambda}$ exceeds the remaining run-time we continue with another algorithm. For predicting the run-time we use the approach proposed by Hutter et al. [9]. We run our k algorithms on different data sets of different sizes. Then, we use the number of instances, the number of predictors and the hyperparameter configuration to predict the run-time using a random forest. This will ensure that training the model selection component will finish within the specified time frame. In the case that no further algorithm can finish in the remaining time, we stop training the model selection component early.

After training the model selection component, it will provide predictions for D_{valid} for each model learned and its corresponding loss. This output will be used for training the following ensembling component. Furthermore, the model selection component now represents a function $g : \mathbb{R}^{n \times p} \to \mathbb{R}^{n \times q}$ that can map some predictors $X \in \mathbb{R}^{n \times p}$ to the meta-feature space. Therefore, the q prediction models trained during the training phase of the model selection component are used to make predictions

$$\hat{\mathrm{Y}} = \begin{pmatrix} \hat{y}_1\left(X\right) & \cdots & \hat{y}_q\left(X\right) \end{pmatrix} \tag{4.6}$$

which act as our meta-features.

4.2 Ensembling Component

Before using the output of the model selection component as final predictions or meta-features, the dimensionality needs to be reduced. One option is to use the validation performance to select only the predictions of the best model or the best model of each algorithm. This has the advantage of using strong prediction models from diverse algorithms but the disadvantage that many estimated models are not considered at all. Another option is to average the predictions or average them by algorithm. This usually leads to a lift in the prediction but will not work in our case. Here, we face the problem that we learned some models with bad hyperparameter configurations that led to very bad or even constant models that will deteriorate the overall prediction. There-

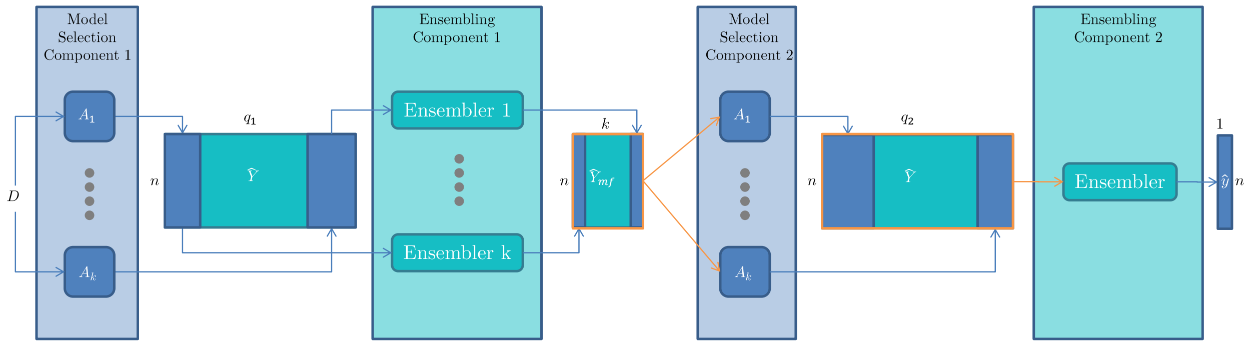

Figure 1: The final framework consists of two main layers. The first layer learns models on the original data. The estimated models are then ensembled by algorithm family. The resulting predictions lead to our meta-features that are used in the second main layer. Again, models are trained, this time on the meta-features. Finally, all models are ensembled to a single prediction vector.

fore, we propose to use a weighted ensemble to combine different models, i.e. the columns of $\hat{Y}$. We employ the bagged ensembling technique with replacement by Caruana et al. [3] to create b bags. Each bag chooses at random with probability r whether to consider a model or not. Then, s models are greedily selected and combined. The final prediction is the average prediction of all bags. This bagged approach will avoid overfitting on the meta-level. This way of ensembling might look complicated but in fact it is nothing but a weighted average of all models. Algorithm 3 summarizes the component. During the training phase the weights are estimated on some validation set. For prediction, these weights are used to combine the predictions of the model selection component. Since most weights are zero, many models can be discarded after the training procedure of the ensembling component has finished. This saves memory and reduces the prediction time.

Algorithm 3 Bagged Ensemble

Input: Number of bags b, fraction of models in a bag r, number of combined models in a bag s, groundtruth y_{valid}, predictions $\hat{Y} \in \mathbb{R}^{n \times q}$.

Output: Prediction of the ensemble.

```
1:  for $i \leftarrow 1$ to $b$ do
2:      $Q \subset \{1, \ldots, q\}$ s.t. $|Q| = rq$
3:      $\hat{y}_i \leftarrow \mathbf{0} \in \mathbb{R}^n$
4:      for $j \leftarrow 1$ to $s$ do
5:          $m_{\text{best}} \leftarrow 0,\ l_{\text{best}} \leftarrow \infty$
6:          for $m \in Q$ do
7:              $l \leftarrow \mathcal{L}\left(y_{\text{valid}}, \frac{1}{j}\left(\hat{y}_i + \hat{Y}_{\cdot,m}\right)\right)$
8:              if $l < l_{\text{best}}$ then
9:                  $m_{\text{best}} \leftarrow j,\ l_{\text{best}} \leftarrow l$
10:         $\hat{y}_i \leftarrow \hat{y}_i + \hat{Y}_{\cdot,m_{\text{best}}}$
11: return $\frac{1}{b \cdot s} \sum_{i=1}^{b} \hat{y}_i$
```

4.3 Final Framework In this section we will describe how the components described in the previous sections are combined to the final framework. Furthermore, we will describe on what part of the data the specific components have been trained and evaluated.

Figure 1 sketches our framework. While it looks complicated, it is the most simple version of our proposed method of Automatic Frankensteining. In fact, it is very similar to a stacked ensemble.

First, the given training data is split into two disjoint parts D_{train} and D_{blend} where D_{blend} contains 10% of the data. D_{train} is used in the first model selection component. The component is evaluating algorithm/hyperparameter configurations in a three-fold cross-validation. Thus, we are learning for each algorithm/hyperparameter configuration tuple three models, one for each fold. The following ensembling component also evaluates the weighted ensembles with a three-fold cross-validation with $b = 1$, $s = 30$ and $r = 1$. At this point, we can generate meta-features for D_{train} by using out-of-fold-predictions such that label information is not leaked. Furthermore, we can generate meta-features for arbitrary data by averaging the predictions of each of the three prediction models learned on a fold. We avoid retraining a model on the complete data to save time. Now, the training for the first two components is completed and they will not be updated any more.

Then, the last two components are trained. Using the first two components, we generate the meta-features for D_{blend} that is used as the training data for the last two components. D_{train} will be used at this point for

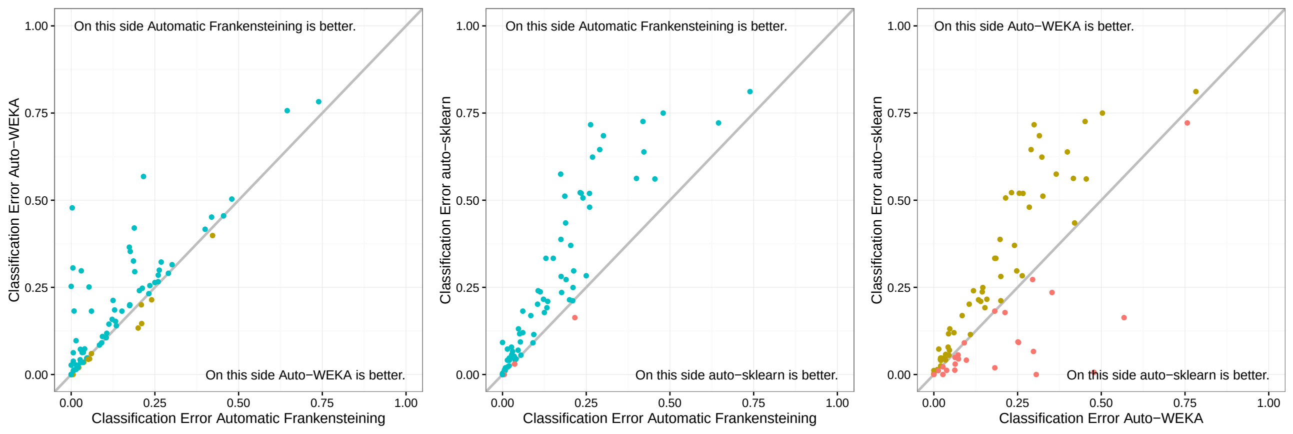

Figure 2: Our approach Automatic Frankensteining is only beaten on 11 data sets by Auto-WEKA and only 3 by auto-sklearn and hence provides the better solution for the majority of data sets.

evaluating the performance of the models. Actually, we trained the second model selection component twice. Once, only on the meta-features and once on the meta-features plus the original features. This allows to use possible interactions between features and meta-features. All models are finally combined with a big weighted ensemble with $b = 10$, $s = 30$ and $r = 0.5$.

Now, the automatically created ensemble can be used to predict for a new test data set D_{test}. The first model selection component computes the meta-features that are then processed by the ensemble component. These meta-features now are used for the second model selection component as features and their output is combined again by another ensembling component to the final prediction vector. For this step only predictions are done or combined.

We restrict us to following classifiers: naive Bayes (Gaussian and Bernoulli), logistic regression, support vector machine with radial basis kernel, k-nearest neighbors, extra randomized trees, gradient boosting classifier [14] and gradient boosting machine [4].

The first half of the available training time is used to train the first two components, the second half for the last two components.

5 Experimental Section

Our experimental section is divided into two parts. In the first part we focus on comparing our approach to the current state of the art for automatic machine learning. In the second part we focus on comparing against human machine learning experts.

5.1 Comparison to Other Approaches

We will compare our approach against Auto-WEKA [18] and auto-sklearn [5] using the authors' implementation and recommended settings.

Auto-WEKA is an addition to WEKA [7] which makes use of the different preprocessing techniques and algorithms offered by WEKA. Since stacking and other ensembles are also algorithms in WEKA, Auto-WEKA can make use of ensembles as well. In contrast to our approach, it is not steered to create an ensemble and thus it can happen that the final solution found by Auto-WEKA is not an ensemble. At its core, Auto-WEKA's search is controlled by Bayesian optimization.

auto-sklearn uses Bayesian optimization to find good algorithms, hyperparameter configurations and preprocessing techniques provided by scikit-learn [14]. After finishing the search, auto-sklearn uses a weighted ensemble [3] to combine estimated prediction models. It has also the feature to use meta-knowledge to initialize [6] the search i.e. that it first evaluates those algorithm/hyperparameter configuration pairs that have been good on many other data sets. We disabled this feature to avoid a distortion of the results for two reasons. First, Auto-WEKA does not contain this feature and second, we perform our evaluation on UCI data sets which are likely used to estimate the initialization sequence and thus auto-sklearn would make use of test information.

We chose 80 different classification data sets from the UCI repository. If these data sets already provided a train/test split we merged them. We shuffled the instances and split them into 80% training data and 20% test data that is used for evaluation purposes only. Our evaluation measure is the classification error for which all methods are optimized directly. Each method got 30 minutes time to create the best predictions.

Figure 2 shows the classification error on the test

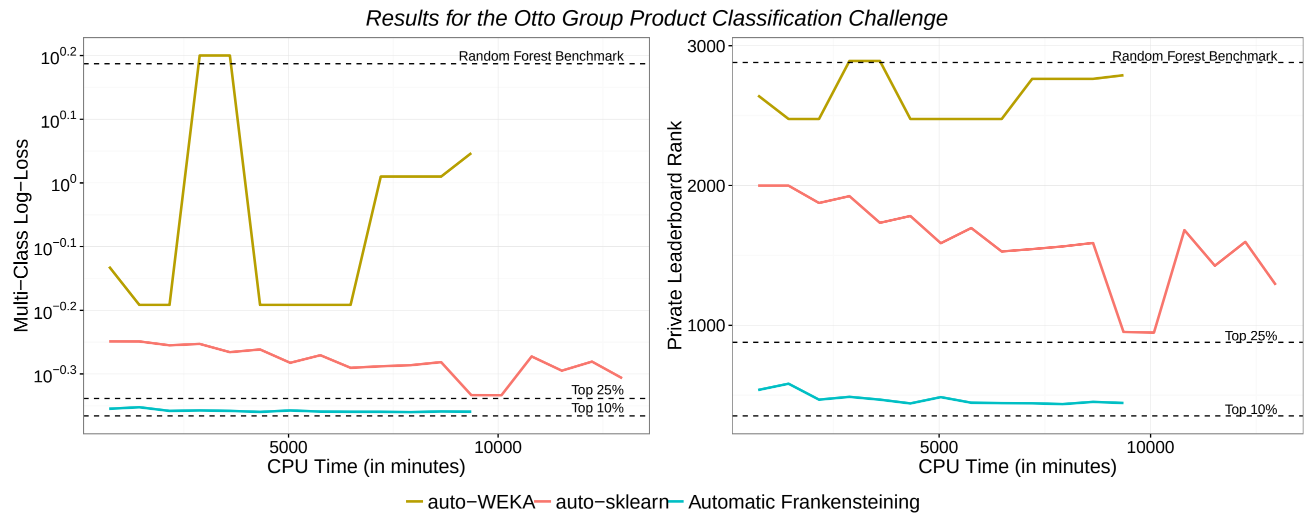

Figure 3: Automatic Frankensteining (bottom) achieves within hours already a very small loss on the private leaderboard, outperforming the automated machine learning baselines WEKA (top) and auto-sklearn (middle) as well as the majority of human participants. The dotted lines indicate the performane of the Random Forest Benchmark, the top 25% and top 10% of the participants.

split where each point represents one data set. The classification error for each approach can be read from the axis. Points on the diagonal indicate ties, points on the one or the other side indicate wins for the named method. Our approach Automatic Frankensteining finds in 55 out of 80 cases a better prediction model than Auto-WEKA and has 14 ties. In comparison to auto-sklearn the difference is even stronger. Automatic Frankensteining finds for 74 data sets a stronger model and only in 3 cases auto-sklearn provides the better solution. The average rank of Automatic Frankensteining is 1.28, for Auto-WEKA 2.06 and for auto-sklearn 2.66. Comparing Auto-WEKA and auto-sklearn head to head, Auto-WEKA finds on average the better prediction models. It finds in 56 cases the better solution, only in 21 cases auto-sklearn is the better option

5.2 Comparison to Human Experts To compare our approach against a wide range of machine learning experts, we applied Automatic Frankensteining on the Otto Group Product Classification Challenge[1], one of the most popular Kaggle challenges. In this challenge, the Otto Group, one of the world's biggest e-commerce companies, asked participants to decide to which of their main categories a product described by 93 features belongs. The challenge data contains business data from more than 20 countries of the Otto Group for more than 200,000 products. The performance was measured based on the multi-class logistic loss as defined in Equation 3.2. More than 3,500 teams participated in this challenge and tried for 62 days to be among the top three to claim their share of the $10,000 price money. The organizers provided a Random Forest Benchmark baseline which was able to achieve a score of 1.5387 on the private leaderboard, the best solution achieved a score of 0.38243.

The labels for the leaderboards were never made publicly available but it is still possible to submit your predictions and let Kaggle evaluate it for you. This means the following for our experiments. First, we are limited by the number of submissions and evalutions because they involve time-consuming interactions with the Kaggle interface. Hence, we were restricted to just few evaluations. Second, since the label is still hidden, it is impossible that our results are distorted by any means because the split and evaluation are maintained by a third party. This also guarantees a high degree of reproducability.

The main focus in this experiment is the comparison of Automatic Frankensteining to human experts. Nevertheless, it is also interesting to see how the current state of the art for automated machine learning performs. Unfortunately, Auto-WEKA only allows to optimize for the classification error and not for arbitrary metrics. We still report the results of Auto-WEKA but this is the reason why we see rather bad and strange results by Auto-WEKA. Automatic Frankensteining and auto-sklearn directly optimize for the right loss. We de-

[1]https://www.kaggle.com/c/otto-group-product-classification-challenge

cided to give each approach 12, 24, 36, ..., 156 hours and submit the predictions after this time to the Kaggle platform. Since we saw a dramatic improvement by auto-sklearn after 156 hours, we also investigated whether auto-sklearn can improve even more, if more time is provided. Figure 3 presents the results with respect to the multi-class log-loss and the rank on the private leaderboard for the predictions estimated after the given CPU time. As explained before, Auto-WEKA shows some strange behaviour. But since the log-loss correlates with the classification error, Auto-WEKA is able to beat at least the baseline provided by the challenge organizers. Automatic Frankensteining achieves already after 12 hours a very good result and can improve, if more computational run-time is provided. After running for 2 days on the data set, Automatic Frankensteining has converged and does not show any further improvement. It is not able to reach the top 10% of the participants and reached a score just under rank 400. Yet, this is quite an impressive result considering that the better performing human experts likely had to spend much more time to achieve the same result. Furthermore, Automatic Frankensteining is able to demonstrate for a further data set that it delivers the better predictions than auto-sklearn.

We also investigated the performance of the individual components. Each single model created by Automatic Frankensteining provided worse predictions than the final ensemble. So we can confirm the results found by the human experts during this challenge, that only a Frankenstein ensemble can provide the most powerful predictions.

6 Conclusions

We proposed Automatic Frankensteining, an automatic way of learning ensembles with many different algorithms with several layers. In a comparison on 80 different data sets, Automatic Frankensteining was able to outperform the current state of the art for automatic machine learning for the large majority of the data sets. Furthermore, in an additional experiment we compared Automatic Frankensteining on a large scale business data set with more than 3,500 human machine learning expert teams and were able to achieve a better score than more than 3,000 teams within just 12 hours CPU time. In the future we will further improve our method to finally reach the level of very strong experts. Therefore, we have to focus strongly on components such as automatic feature engineering which currently give human experts a not negligible advantage over our approach. We already used an early version of Automatic Frankensteining in the ECML/PKDD 2016 Discovery Challenge on Bank Card Usage. While this early version was not cappable of reaching very high performances, it still gave us some insights that we used to improve our solutions which eventually led to winning the challenge.

Acknowledgements

The authors gratefully acknowledge the co-funding of their work by the German Research Foundation (DFG) under grant SCHM 2583/6-1.

References

[1] Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michèle Sebag. Collaborative hyperparameter tuning. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 199–207, 2013.

[2] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain.*, pages 2546–2554, 2011.

[3] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*, 2004.

[4] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, San Francisco, CA, USA - August 13 - 17, 2016*, 2016.

[5] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2962–2970, 2015.

[6] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Initializing bayesian hyperparameter optimization via meta-learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 1128–1135, 2015.

[7] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[8] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(10):993–1001, 1990.

[9] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artif. Intell.*, 206:79–111, 2014.

[10] Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *J. of Global Optimization*, 13(4):455–492, December 1998.

[11] Alexandre Lacoste, Hugo Larochelle, Mario Marchand, and François Laviolette. Sequential model-based ensemble optimization. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence, UAI 2014, Quebec City, Quebec, Canada, July 23-27, 2014*, pages 440–448, 2014.

[12] Julien-Charles Levesque, Christian Gagné, and Robert Sabourin. Bayesian hyperparameter optimization for ensemble learning. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence, UAI 2016, June 25-29, 2016, New York City, NY, USA*, 2016.

[13] Jonas Mockus, Vytautas Tiešis, and Antanas Žilinskas. The application of bayesian methods for seeking the extremum. *Towards Global Optimization*, 2:117–129, 1978.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[15] Nicolas Pinto, David Doukhan, James J DiCarlo, and David D Cox. A high-throughput screening approach to discovering good forms of biologically inspired visual representation. *PLoS Computational Biology*, 5(11):e1000579, 2009. PMID: 19956750.

[16] Mary Wollstonecraft Shelley. *Frankenstein; or, The Modern Prometheus.* Lackington, Hughes, Harding, Mavor & Jones, 1818.

[17] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 2960–2968, 2012.

[18] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 847–855, New York, NY, USA, 2013. ACM.

[19] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Learning hyperparameter optimization initializations. In *International Conference on Data Science and Advanced Analytics, DSAA 2015, Paris, France, October 19 - 21, 2015*, 2015.