SymReg-MT: Iterative Multi-task Feature Learning Through Weighted Symbolic Regression

Michael Zwick, Holger Schöner, and Ehsan Rezaie

Software Competence Center Hagenberg, Softwarepark 21, 4232 Hagenberg, Austria,
{michael.zwick, holger.schoener, ehsan.rezaie}@scch.at,
WWW home page: http://www.scch.at

Abstract. We present a novel approach for unifying regression models learned in parallel on different but related datasets using *multi-task feature learning* based on *symbolic regression*. The *FFX framework* (Fast Function Extraction) is used for symbolic regression. It relies on regularized linear regression instead of genetic programming, thus providing a scalable and deterministic framework for implementation. FFX provides a basis for an iterative multi-task feature learning approach. Models learned on separate tasks are coupled by iteratively promoting common terms and penalizing seldom occurring terms, leading to improved consistency and interpretability of models across tasks and improved stability on new data. We conducted experiments on both real world datasets and synthetic datasets. The results show that the use of this basic approach already leads to models which *share more features*, show *less complexity* and still *retain* the same *model performance* when compared to a single symbolic regression run. Finally, we provide ideas and plans for future improvements based on our first implementation.

Keywords: multi-task feature learning, symbolic regression, FFX, fast function extraction, machine learning, ECML

1 Introduction

In the field of *multi-task learning*, different but related tasks in the same feature space — for each of which labeled data is available — are learned simultaneously, with the goal of improving the performance and robustness of each individual task by incorporating knowledge from all other tasks [3]. This can be done by using some kind of shared representation between tasks. [12] categorizes multi-task learning as a variant of *inductive transfer learning* where labeled data for all tasks are available and tasks are learned simultaneously. Using this categorization, in multi-task learning tasks are learned in parallel, whereas in other transfer learning variants a target task is learned using information from previously learned source tasks. Goals of transfer learning in general can be to improve generalization performance of the target tasks or to deal with small amounts of training data available for each task.

When dealing with different but related tasks in the same feature space, some of the input features can be *common* or *shared features*, i.e., they are relevant to other tasks in the multi-task setting. This also means they represent the commonality between tasks. Other features can be (task-)*specific*, they are only needed for a few tasks (or even just one task) to yield acceptable model performance. These features represent the differences between tasks. Trying to learn which features are common across all tasks is called *multi-task feature learning* [1].

The approach introduced in this contribution extends this idea of common and task-specific features to common and task-specific terms in a symbolic regression setting.

In *symbolic regression* [9, 15], target functions are free-form formulas, consisting of candidate terms to be selected and combined by the symbolic regression algorithm. However, a set of basis input features and operators has to be provided on which the regression algorithm can perform selection. Symbolic regression is used in problems where flexibility is of high importance. In addition, by providing basic input features and operators prior knowledge can be used to direct the learning algorithm in the right direction.

In this paper we present a multi-task feature selection framework for regression tasks called SymReg-MT. It operates on multiple related datasets sharing the same feature space by using an iterative approach to unify task models based on frequency of occurrence of features. We aim to improve the *trade-off between complexity and quality*, our focus is not primarily on reducing the model error when compared to learning the tasks separately. Our goal is to represent different but related tasks by similar models — with respect to the terms used in the models — and thereby to obtain simpler and more robust models. Furthermore our goal is to increase the interpretability of the task models as well as to reduce the amount of data needed for each individual task.

The approach described in this paper has its origins in an industrial application of one of our industry partners, where we successfully applied symbolic regression methods. In this use case we had related regression tasks and looked for the simplest models that could achieve a predefined error threshold (average as well as maximum error). Model selection based on a target error is further described in Section 4. In addition to testing our approach on data from one of our industrial projects, we used the method from the multi-task feature learning paper [1] for generating synthetic datasets for additional evaluation. More details about the exact process for generating the datasets can be found in section 6.1.

Section 2 describes related work also dealing with the subject of multi-task feature learning, especially identifying shared features. Section 3 defines the symbolic regression problem and describes the FFX framework our approach uses to learn the task models in each iteration. Section 4 presents our iterative approach to multi-task feature learning. Section 5 contains additional remarks concerning our implementation. Section 6 presents the results of our evaluations both on synthetic datasets and real world datasets. Finally, Section 7 concludes the pa-

per with a discussion of the presented framework and lists some of our ideas for further investigation and enhancement.

2 Related Work

In recent years, several papers addressing the problem of multi-task feature learning were published [18, 7, 10, 5, 8, 6], though our approach is the first one to use a symbolic regression framework as a basis for such an approach. Most of the papers in the related work are concerned with the problem of negative transfer, which occurs in the presence of an outlier task. In such a case, the assumption that all tasks in the multi-task setting share a common set of features does not hold. As a consequence, selected features are "forced upon" the outlier task which reduces its model performance. This problem is often addressed by learning "dirty models" [6], which allow for features that are solely assigned to a single task in addition to common features shared by all tasks.

The approach presented in this paper does not address the problem of negative transfer explicitly. External (expert) knowledge of the datasets is necessary to avoid trying to learn similar features for tasks whose models do not share the same features as the other tasks. The motivation for SymReg-MT are problems with highly related tasks. This is the case for our original problem and many other industrial applications where tool and workpiece combinations generate an immense number of different settings which are highly related because of the underlying industrial process, for which we require a similar structure over all settings in order to improve robustness and interpretability of the models.

[18] proposes an iterative hierarchical regularization approach for multi-task classification problems. The approach balances l_1/l_2- and l_1-norm regularization to share features between all tasks through the l_1/l_2-norm but at the same time allow sparse models without sharing by using l_1-norm. This is achieved by starting with pure l_1/l_2 regularization and subsequently putting more weight on l_1 regularization in later iterations allowing for more task specific features. The weight matrix $\mathbf{W}^l$ of the current level l is learned with respect to the weight matrix $\mathbf{W}^{l-1}$ of the previous level for $l = 1..L$ with L being the predefined number of levels in the algorithm.

[7] is directly based on [2] in which the authors use $l_{2,1}$-norm to ensure the selection of sparse features shared across all tasks. In [7], a shared feature representation is learned jointly while determining for each task with which other tasks to share. Tasks are automatically clustered into a predefined number of task groups with task relatedness being more prominent within each group. Regularization should then occur for tasks in the same group but not be imposed on tasks across groups.

[10] tries to learn a small number of latent basis tasks based on the input features resulting in a parameter matrix $\mathbf{L}$. The observed tasks to be learned eventually are linear combinations of these latent tasks and represented in a parameter matrix $\mathbf{S}$ which is assumed to be sparse with regards to the latent tasks. This is ensured by using l_1 regularization on $\mathbf{S}$. In contrast, Frobenius

norm is used on **L** for the latent tasks to avoid overfitting. The resulting cost function is not jointly convex for **L** and **S**, thus an alternating optimization strategy is applied until convergence.

[5] describes a domain adaption approach that is extendable to a multi-domain setting in which labeled data is available for all domains, which makes this domain adaption setting related to multi-task learning. The basic idea is to take the original feature vector and generate/copy one for each domain (source and target domain) plus one additional general feature vector. The datasets then get extended in a way that the source dataset only contains the general and the source-specific features and the target dataset only contains general and target-specific features. Using this extended input space, the learning algorithm has the ability to put increased weight on features which are only relevant to certain domains by using the domain version of the feature in the coefficient vector or use the general version if the feature is relevant to both domains.

[8] introduces the tree-guided group lasso approach for sparse multi-task feature selection regression. The task output structure is represented as a tree with leaf nodes representing task outputs and internal nodes representing clusters of outputs. Outputs can be grouped at multiple granularity indicating task relatedness. The structure of the tree is either available as prior knowledge or can be learned from data.

In general, all of the above mentioned approaches select input features directly. The approach presented in this paper, because it is based on a symbolic regression algorithm, selects basis functions automatically generated from the input features by the symbolic regression framework (see Section 3).

Preventing negative transfer was not an explicit goal in our approach. However, extending our framework with a task grouping capability comparable to the approaches described in this section, i.e., sharing features between groups while still allowing for outlier tasks, would be an interesting extension to our work (see also Section 7).

3 Preliminaries

This section gives a brief introduction to SR, followed by a description of "Fast Function Extraction", the SR framework on which the work presented in this paper is based on.

3.1 Symbolic Regression

Symbolic Regression (SR) [9] is a kind of regression analysis that searches a space of mathematical expressions for the purpose of deriving a model that best fits a given dataset. The mathematical expressions can consist of the datasets input features ($x_0, \dots x_n$), constants (π, e, ...), basic arithmetic operations (+, -, $\times$, $\div$) and mathematical functions (log, exp, ...). The actual set of inputs and operators which the symbolic regression algorithm is allowed to use for building the model has to be determined before the start of the algorithm. In contrast to

other regression methods, e.g. linear regression, no predefined model is given to the regression algorithm at the beginning. A symbolic regression algorithm has to determine both the structure and the parameters of the model from the data.

As a consequence, the search space is much larger when compared to other regression techniques. Therefore, symbolic regression relies on efficient heuristics to quickly identify subexpressions which can be combined to form an accurate model of the data, and eliminate subexpressions that do not contribute to the overall models performance. In symbolic regression, this is mainly done with *Genetic Programming* (GP) [14]. Candidate models are combined and altered using crossover and mutate operations in order to quickly converge to a (local) optimum while also covering a large part of the search space.

Symbolic regression research uses GP to such an extent that it is ofter considered a subfield of GP.

3.2 Fast Function Extraction

Fast Function Extraction (FFX) [11] is a SR framework that does not use GP for optimization. Unlike a SR algorithm based on GP, FFX does not work on arbitrary mathematical expression but restricts itself to *generalized linear models* (GLM) over a number of basis functions (see equation 1 taken from [11]).

$$\hat{y} = m(\mathbf{x}) = a_0 + \sum_{i=1}^{N_B} a_i * B_i(\mathbf{x}) \tag{1}$$

A model m maps an input vector $\mathbf{x}$ to an output value $\hat{y}$ by using a linear combination of N_B basis functions B_i with coefficients a_i. Because FFX tries to solve a SR problem, it has to come up with both the coefficients a_i and the basis functions B_i.

In a first step, FFX generates a hugh amount of linear as well as non-linear basis functions from the input features before a run. Examples are x_1, $x_2 x_3$, $\log x_5$, x_2/x_3, $\frac{\exp x_5}{\exp x_6}$. The challenge is then to efficiently reduce this large number of basis functions to the ones that are actually useful for building models which can explain the data. To achieve this, FFX uses *Elastic Net* (EN) regularization [17], a combination of lasso regularization [16] and ridge regression, for performing feature selection. The cost function for linear regression with EN regularization is given in equation 2, with $\mathbf{X}$ representing the matrix of the datasets input features, $\mathbf{y}$ the vector of target values and $\mathbf{a}$ the vector of regression coefficients. λ_1 and λ_2 are the regularization parameters for ridge regression (λ_1) and lasso regularization (λ_2).

$$\|\mathbf{y} - \mathbf{X} * \mathbf{a}\|^2 + \lambda_2 \|\mathbf{a}\|^2 + \lambda_1 \|\mathbf{a}\|_1 \tag{2}$$

For the purpose of pathwise learning (see next paragraph), EN regularization is reformulated using one regularization parameter λ and an additional mixin-parameter $\rho \in [0, 1]$ for balancing ridge regression ($\lambda_2 = \lambda(1 - \rho)$) and lasso regularization ($\lambda_1 = \lambda\rho$) (see equation 3).

$$\|\mathbf{y} - \mathbf{X} * \mathbf{a}\|^2 + \lambda(1-\rho)\|\mathbf{a}\|^2 + \lambda\rho\|\mathbf{a}\|_1 \quad (3)$$

In a second step, *Pathwise Learning* (PL) [4] varies the regularization parameter λ from high to low values in multiple iterations. The iterations yield models of differing complexity depending on λ. With a high λ, EN will only select a few basis functions and learn a model which is very simple but also has a high training error. Once λ gets lower, the learned models will get more complex and have a lower training error. At the end of the regularization path, the set of learned models represents a trade-off between accuracy and complexity. Simple models have higher training error but tend to be more robust and more suitable for interpretation, whereas complex models have lower training error but are not as easily interpreted and are likely to have higher out-of-sample error.

In a final step, FFX selects the non-dominated pareto optimal models out of all models generated by the regularization path with respect to model complexity and accuracy. A model is only retained in the final result set if there does not exist a different model with equal accuracy but fewer basis functions. This pareto front is then presented to the user to choose whichever model best fits the intended application.

Even though our approach described in Section 4 builds on the FFX framework as described in this section, it is important to mention that our approach is not limited to FFX, but could be combined with any SR framework, e.g., one that uses a GP approach[1].

4 Multi-task Symbolic Regression

In multi-task learning, information from one task is used to improve the performance of other related tasks. To achieve this transfer of information, the tasks are learned in parallel using some kind of shared representation [3]. In our system SymReg-MT, the tasks are learned in parallel while each task retains its own model. The shared representation is achieved by recording joint statistics about basis function weights of all tasks which are subsequently used to refine the models in further iterations.

The main idea of our approach is to calculate the frequency of occurrence of all basis functions (e.g. x_3/x_4) in the candidate models for all tasks. This is done after each learning run, i.e., after each iteration in which we learn all tasks in parallel and independent of each other using the FFX symbolic regression framework. Using the frequency values, each basis function is assigned a weight that is included in the elastic net regularization term for the next learning run. Thus, basis functions which appear in the candidate models of many tasks are preferred over basis functions appearing in only a few tasks. The motivation is that basis functions appearing in many candidate models have a higher probability of representing common features. By promoting these basis functions, the learning algorithm is encouraged to put increased importance on these common features.

[1] https://code.google.com/p/deap/

Furthermore, tasks which have not yet included such a feature in their candidate models are more likely to select it in the next learning run. Task-specific features have a high weight and are only selected when the performance gain outweighs the regularization penalty. This is how transfer of common features is achieved between the multiple tasks.

An initial approach for generating the weights after each iteration involves the following three steps, which are described in detail in the next paragraphs:

1. Generating *weight factors* for each basis function,
2. Updating the *weight* for each basis function using the weight factor,
3. *Normalizing* the product of the weights to 1.0 before starting the next iteration.

Note that in the approach presented, *high weight* does not mean high importance but *high penalty*, the algorithm tends to select features with low weight/penalty, features with high weight increase the regularization term and are less likely to be selected.

Algorithm 1 shows our approach in pseudocode form. Due to the fact that FFX generates the basis functions it considers for learning on-the-fly, we cannot initialize the weights of all possible basis functions from the beginning. We therefore start out with an empty set of weights, and after each FFX run, we initialize the weights of new basis functions, i.e., basis functions we have not seen in a prior learning run. The initial value at the first iteration is 1.0, afterwards we initialize new basis functions with the median weight of all known basis functions, in order to prevent new and unproven basis functions from overpowering established ones. How the weight is adjusted between each iteration using a weight factor is described in detail in the next paragraph.

In the following, N_B is the number of different basis functions and B_j a basis function with $j \in \{1, \dots N_B\}$. Each basis function is assigned a real-valued weight $w_j \in [1, W_{max}]$. At the first iteration, $i = 1$, each basis function gets a weight $w_j := 1$, which means the basis function is not penalized at this point. After each symbolic regression learning run, the weights are multiplied by a real-valued weight factor $f_j \in [1, 2]$ which is dependent on the number of occurrence o_j of the respective basis function B_j over all tasks (see equation (4)). Basis functions with $o_j = o_{max}$ get a factor $f_j = 1$, which means they retain their current weight and are not penalized further. o_{max} is the highest number of occurrence of all basis functions in the current iteration. Basis functions which were considered during learning but eventually not selected ($o_j = 0$) get the maximum factor $f_j = 2$, i.e., their weight is doubled. Figure 1 shows the weight factor function for a dataset with 20 tasks and an $o_{max} = 18$. The *scale factor* s_f in equation 4 determines the steepness of the curve.

$$f_j = \frac{o_{max} - o_j}{o_{max}(s_f \cdot o_j + 1.0)} + 1.0 \tag{4}$$

The weight of each basis function is then multiplied with the weight factor to get the weight for the next iteration i. However, a weight can never be higher than

Algorithm 1 High-level structure of SymReg-MT

```
1: procedure SymReg-MT
2:     i ← 0                                                    ▷ iteration
3:     w^i ← ∅                          ▷ FFX assigns default weights to new BFs
4:     repeat
5:         i ← i + 1
6:         for all tasks do                                     ▷ in parallel
7:             run symbolic regression (FFX)
8:             perform model selection      ▷ with additional degradation prevention
9:         end for
10:        assign weights to new base functions ▷ median weight of old base functions
11:        compute frequency of occurence o
12:        generate weight factors f(o)          ▷ with o_max as reference value
13:        w^i ← w^{i−1} · f                      ▷ update weights from factors
14:    until w^i = w^{i−1}
15: end procedure
```

W_{max}, which is the maximum penalty for any basis function (see equation (5)). For our experimental results reported in Section 6, the value W_{max} was set to $W_{max} := 10$.

$$w_j^i = \begin{cases} w_j^{i-1} f_j, & \text{if } w_j^{i-1} f_j < W_{max} \\ W_{max}, & \text{otherwise} \end{cases} \tag{5}$$

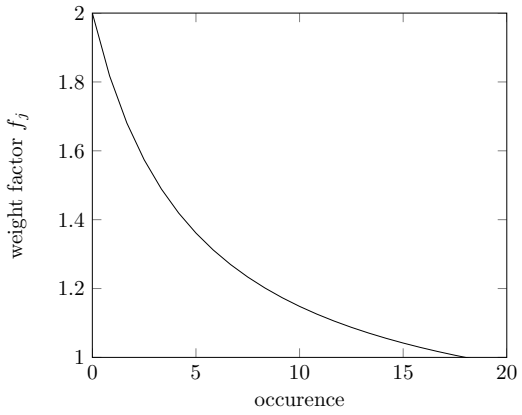


Fig. 1. Example of weight factor f_j for $o_{max} = 18$ and $T = 20$.

A single run of the FFX algorithm on the dataset for a given task yields the whole pareto front of models from the poor-performing but simple to the most accurate but complex ones (see also section 3.2). For results reported later, we

select exactly one model h_t^i for each task t and at each iteration i from the whole pareto front, according to our initial industrial problem setting. The selection criterion we use is a maximal training error level RMSE_{target}. For each task, we then select the simplest model (i.e. with the least number of basis functions), which at the same time achieves the target quality. We used this approach because our industrial use case, which provided the idea for our implementation, had this maximum error requirement. In the future, we also plan to apply our approach in a different way, i.e., aiming for better model quality while keeping model complexity under a given maximum.

In addition to our target RMSE, there are additional selection criteria, which make sure that our task models do not degenerate. At a given iteration i, the model selection retains the model of iteration $i-1$ for a certain task t, if one of the following holds:

- No new model of the current iteration i can reach the target RMSE_{target}.
- h_t^i is more complex that h_t^{i-1}
- The selected model h_t^i has the same complexity as h_t^{i-1} but cannot achieve higher quality, i.e., $\text{RMSE}(h_t^i) \geq \text{RMSE}(h_t^{i-1})$.

Complexity currently is simply defined as the number of basis functions in a model, regardless of the complexity of the basis functions appearing in the model. We intend to refine the definition of model complexity in our framework in the future (see Section 7).

5 Implementation

In section 4 we presented the basic idea of how the transfer of knowledge between the different tasks works from one iteration to the next. This explanation was implementation independent. In this section, we briefly mention some implementation details of our implementation based on the python framework *scikit-learn* [13], which is used by the FFX framework.

We define a maximum number of iterations to perform before the algorithm terminates to prevent against infinite iterations, although this was not necessary for any of our experiments. Before starting the symbolic regression learning run for a given iteration, the weight vector $\mathbf{w}$ is normalized using the geometric mean (GM) of the weights, resulting in $\mathbf{w}_{norm} := \frac{\mathbf{w}}{\text{GM}(\mathbf{w})}$ such that $\prod_{j=1}^{N_B} w_j = 1$.

To make the FFX algorithm use the base function weights during feature selection, we implemented an alternate version of the sklearn *elastic net coordinated descent* implementation by multiplying the weights $\mathbf{w}$ to the model coefficients $\mathbf{a}$ ($\odot$ denoting element-wise multiplication):

$$\|\mathbf{y} - \mathbf{X} * \mathbf{a}\|^2 + \lambda(1-\rho)\|\mathbf{a} \odot \mathbf{w}\|^2 + \lambda\rho\|\mathbf{a} \odot \mathbf{w}\|_1 \tag{6}$$

For an explanation of the cost function in Equation 6 we refer back to Section 3.2.

6 Results

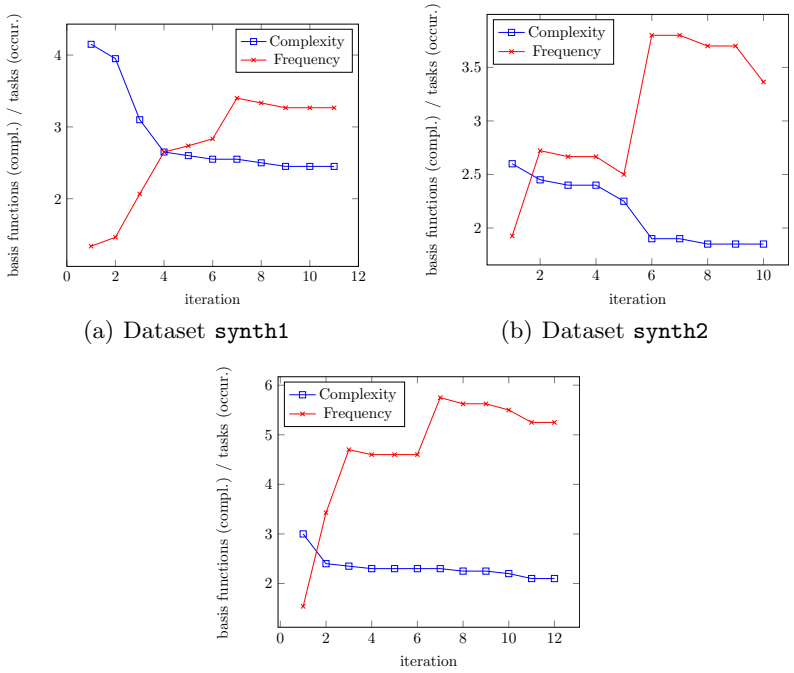
In order to check the viability of our approach, we tested the framework on both synthetic and real world datasets. The synthetic datasets were generated similar to the method described in [1]. Both real datasets stem from a project with one of our industrial partners, we anonymized the names of the input features in order to protect our partner company's intellectual property. For both the synthetic as well as the real world datasets we set $W_{max} := 10$ and $s_f := T/10$, with T denoting the number of tasks.

6.1 Synthetic Dataset

For generating the synthetic datasets, we assumed a multi-task learning scenario with $T = 20$ tasks. There are $C = 5$ common features $cm_0, \dots cm_{C-1}$ relevant for all tasks and $I = 20$ irrelevant features $ir_0, \dots ir_{I-1}$. For each of the T tasks, we created a C-dimensional vector $\mathbf{c}_t$ ($t \in \{1, \dots T\}$) from a 5-dimensional Gaussian distribution with covariance Diag(1, 0.25, 0.1, 0.05, 0.01) representing the importance of the common features for each task. An I-dimensional vector $\mathbf{i}_t = \mathbf{0}$ represents the importance of the irrelevant features. Input feature values $cm_{t,0}, \dots cm_{t,C-1}$ and $ir_{t,0}, \dots ir_{t,I-1}$ are then drawn uniformly from $[0, 1]$ and the output (target) for one sample is then calculated by $y_{ti} = \langle \mathbf{c}_t, \mathbf{cm}_t \rangle + \langle \mathbf{i}_t = \mathbf{0}, \mathbf{ir}_t \rangle$, with $\langle \cdot \rangle$ denoting the scalar product. For each task 10 training samples and 5 test samples were created. Using the method just described, we created 3 different datasets `synth1`–`3` containing $10 + 5$ samples for each of the T tasks for testing our algorithm. The reason for the small number of training samples is to test generalization quality where a task needs to incorporate knowledge from related tasks.

Figure 2 shows the progression of average model complexity and average basis function frequency over all 20 tasks. The average frequency is calculated from all basis functions occurring at least once in any model. While model complexity is monotonically decreasing in all 3 datasets, basis function frequency increases abruptly in the first couple of iterations, where many of the irrelevant features are eliminated.

To get an idea of how the models change from the first unweighted iteration to the final result, table 1 shows the models for 5 selected tasks based on dataset `synth3` at the first and the last iteration. Looking at the first iteration, which corresponds to a single symbolic regression run without multi-task feature learning, both common features and irrelevant features are used to fit the models to the data. By the end of the last iteration, only task number 3 could not be improved any further and remained to use irrelevant features ir_5 and ir_{10} in combination with non-linear basis functions. It is worth mentioning, that for the 3 datasets `synth1-3`, terms including irrelevant features were the exception, the final models of the majority of the tasks use only common features cm_0–cm_4, which also are used by the models in a strictly linear way, i.e. no non-linear basis functions like, e.g., $\exp cm_1$ appeared.



(a) Dataset `synth1` (b) Dataset `synth2`

(c) Dataset `synth3`

Fig. 2. Model complexity and similarity for synthetic datasets ($T = 20$), selecting the simplest model achieving the required accuracy in each iteration.

These results demonstrate, that by coupling models for similar but differing individual tasks, less data can be sufficient to select relevant features and obtain models with testing error similar to models learned in isolation (see section 6.3). As a consequence, model complexity is lowered, and interpretability of models can be improved by showing similarity between tasks.

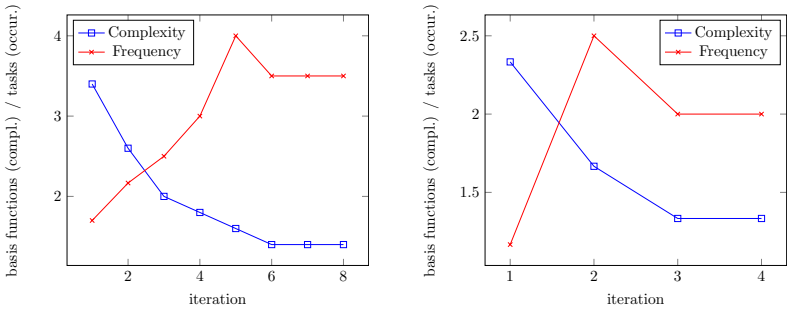
6.2 Real World Datasets

In figure 3, model complexity and basis function frequency is shown over the whole run for two real world datasets from a parts manufacturing setting, where the tasks differ by the tools with which a machine was equipped. The tools share a general shape but differ in their detailed geometry, thus introducing different but highly related tasks. The problem has 20 input features, dataset `real1` has 5, dataset `real2` has 3 tasks. The sample size for each of the tasks varies between 21 and 151 samples.

Table 2 shows the evolution of the task models over the first 6 iterations. The algorithm stopped after iteration 8, but in the last two iterations, only the coefficients did change slightly, the selected terms stayed the same for all tasks. The models in the first iteration are the results from single individual symbolic

Table 1. Trace of a SymReg-MT run on synthetic dataset `synth1`.

Iter.	Task	Model
1	1	$-0.144 - 0.102 \log cm_2 - 0.0880 \exp cm_3 + 0.0601 \exp ir_3 + 0.0495 \log cm_1$
	3	$0.225/(1.0 - 0.0818 \exp cm_2 \exp ir_5 - 0.0575 \exp cm_0 \exp ir_5 - 0.00928 \exp ir_{10})$
	13	$-0.257 - 0.160 cm_0 \exp cm_1 + 0.0474 \exp ir_{18} / \exp cm_1 - 0.0469 \exp cm_1 / \exp cm_3 + 0.00234 \exp ir_{18} / \exp cm_0 + 0.000202 \exp ir_{18} / cm_0$
	17	$(-0.0967 - 0.529 cm_0 - 0.0993 ir_2 + 0.000518 cm_4)/(1.0 - 0.127 ir_{10} - 0.109 cm_2 - 0.0564 ir_8 - 0.0508 ir_{17} - 0.0453 ir_2 - 0.0309 ir_9)$
	19	$0.719 + 0.148 (ir_4)^2 / ir_{15} - \frac{0.140}{\sqrt{cm_2}} - 0.0901 (ir_{15})^2 + 0.0858 \sqrt{cm_2} (ir_4)^2 - 0.0784 (ir_{15})^2 - \frac{0.0372}{ir_5} - 0.0245 ir_{15} / cm_0 - \frac{0.0102}{(ir_5)^2}$
12	1	$-0.0136 - 0.354 cm_2 - 0.263 cm_3 + 0.166 cm_1 + 0.152 cm_0$
	3	$0.225/(1.0 - 0.0818 \exp cm_2 \exp ir_5 - 0.0575 \exp cm_0 \exp ir_5 - 0.00928 \exp ir_{10})$
	13	$0.0226 - 0.489 cm_0 - 0.345 cm_1$
	17	$-0.0860 - 0.960 cm_0 + 0.108 cm_1$
	19	$0.00472 + 1.09 cm_0 - 0.622 cm_1 + 0.380 cm_2$



(a) Dataset `real1` (b) Dataset `real2`

Fig. 3. Model complexity and similarity for real world datasets, $T = 5$ (a), $T = 3$ (b).

regression runs with basis function weight equal to 1. As the iterations progress, our algorithm promotes common feature x_2, the basis function x_1 also appears in 2 out of 5 models. In contrast to the first iteration, there is no task specific basis function which is used only by one model.

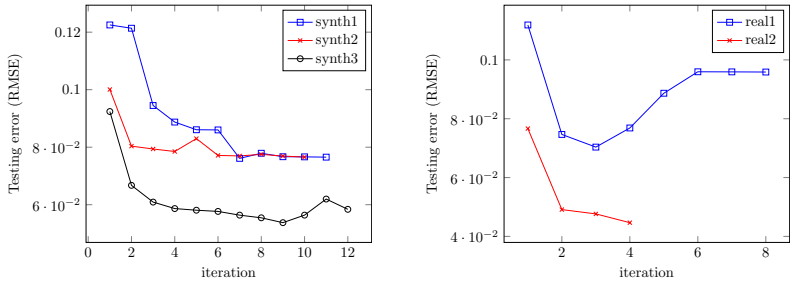
6.3 Model Quality

As described in section 1, we are not primarily interested in using multi-task learning for improving the model accuracy. However, in figure 4 we show the change in average model quality as RSME on the testing set between iterations. Because at each iteration we select the simplest model for each task which meets our target RMSE, the model error is not expected to change much. One observation with both synthetic (figure 4(a)) and real (figure 4(b)) datasets is

Table 2. Trace of a SymReg-MT run on real world dataset `real1`.

Iter.	Task	Model
1	1	$0.255 + 0.678x_5 + 0.381x_3 + 0.160x_4 + 0.145x_2$
	2	$1.50 + 0.487\sqrt{x_7}\sqrt{x_2} + 0.0463(x_3)^2$
	3	$0.592 + 0.743x_1 + 0.346x_2 + 0.190x_7$
	4	$0.582 + 1.30x_1 + 0.300x_2 + 0.123x_7$
	5	$1.27 + 1.46x_1 + 0.993x_5 + 0.560x_4 + 0.0513x_{11} + 0.0115x_9$
2	1	$0.254 + 1.10x_5 + 0.189x_2$
	2	$0.252 + 1.41x_4 + 0.207x_2$
	3	$0.335 + 1.05x_1 + 0.379x_2$
	4	$0.162 + 1.59x_1 + 0.363x_2$
	5	$1.27 + 1.46x_1 + 0.993x_5 + 0.560x_4 + 0.0513x_{11} + 0.0115x_9$
3	1	$0.157 + 1.15x_5 + 0.203x_2$
	2	$0.252 + 1.41x_4 + 0.207x_2$
	3	$0.335 + 1.05x_1 + 0.379x_2$
	4	$0.162 + 1.59x_1 + 0.363x_2$
	5	$0.349 + 3.35x_1 + 0.203x_2$
4	1	$0.692 + 0.383x_2$
	2	$0.252 + 1.41x_4 + 0.207x_2$
	3	$0.335 + 1.05x_1 + 0.379x_2$
	4	$0.162 + 1.59x_1 + 0.363x_2$
	5	$0.272 + 3.37x_1 + 0.218x_2$
5	1	$0.594 + 0.411x_2$
	2	$0.960 + 0.444x_2$
	3	$0.335 + 1.05x_1 + 0.379x_2$
	4	$0.162 + 1.59x_1 + 0.363x_2$
	5	$0.272 + 3.37x_1 + 0.218x_2$
6	1	$0.514 + 0.434x_2$
	2	$0.705 + 0.517x_2$
	3	$0.763 + 0.583x_2$
	4	$0.162 + 1.59x_1 + 0.363x_2$
	5	$0.272 + 3.37x_1 + 0.218x_2$

that model error tends to drop during the first couple of iterations and then in some cases increases towards the end of a run where the algorithm trades quality for simplicity and uniformity of the models. Most notably, this can be seen in figure 4(b) for dataset `real1`. Many datasets can decrease their error over the whole learning run without explicitly enforcing this via the cost function or model selection, but only indirectly by reducing complexity and increasing similarity of the task models. However, there is a fair amount of randomness involved in the variation of model performance through the iterations, because the model selection usually cannot select a model with a training error of exactly RMSE_{target}.



(a) Synthetic datasets (b) Real datasets

Fig. 4. Model error (RMSE) at all iterations.

7 Discussion and Future Work

Our approach to multi-task feature learning described in this paper uses a simple method of weighing base functions solely based on the frequency of occurrence when looking at the models across different tasks. Together with our safety conditions for preventing model degradation during the later iterations in a SymReg-MT run, the results on real and synthetic datasets show that our approach can significantly decrease model complexity of all tasks as well as promote similarity, i.e. common features, between tasks while keeping the prediction quality on testing data at comparable levels. In many cases, the simpler models also yield higher model performance. We intend to use our current implementation as a starting point for investigating several possible improvements.

Our current approach uses a method for model selection based on a predefined target performance. This can lead to a problematic selection of basis functions. Models which lie close together in the pareto front generated by the FFX framework can contain completely different kinds of basis functions. Only a minor change in the target performance can lead to the selection of completely different basis functions, which can heavily influence the weights of basis functions and thus alter the course of future iterations. In the next version of our algorithm, we intend to not only select one model for each task, but to consider the whole pareto front of models and all basis functions contained therein. Furthermore, we intend to include the quality and complexity of models in the weighing of basis functions, i.e., basis functions appearing in better and/or less complex models get penalized less. Our current algorithm also does not take into account the complexity of basis functions itself, meaning that simple linear basis functions like x_1 are considered to be as complex as more complicated ones like, e.g., the non-linear basis function $\exp \frac{x_1}{x_2} \exp x_3$.

Currently our internal shared representation consists of one weight vector for all tasks. A task-specific feature which is only useful for a single tasks has a high value in the weight vector and gets selected only when the gain in model performance outweighs the penalty in regularization. It would be interesting to investigate "dirty models" as described in the related work in section 2, which

allows for truly task-specific features only used in one model, e.g., by further adapting the regularization term the FFX framework uses for regression on the candidate terms.

As already mentioned, our algorithm has to take additional measures to prevent model degradation by suppressing specific terms in later iterations. We intend to address this issue and investigate the way the penalizing weights are determined, e.g., by deriving it directly from the cost function. The results for the presented basic approach are encouraging to continue working in this direction.

Besides improving the existing algorithm, another topic of further investigation concerns the application to completely new tasks, by interpolation of coefficients between directly neighboring tasks as determined, e.g., by tool geometry. As adjacent models are similar to each other and share common features after a learning run, this opens the possibility to interpolate the feature coefficients for an unknown task in between. Finally, as already mentioned in Section 4, we plan to apply our approach in a different way, i.e., aiming for better model quality while keeping model complexity under a given maximum.

Acknowledgments. This work has been supported by the State of Upper Austria and the Austrian Research Promotion Agency (FFG).

References

1. A. Argyriou, T. Evgeniou, and M. Pontil. Multi-task feature learning. In *Advances in Neural Information Processing Systems 19*. MIT Press, 2007.
2. A. Argyriou, T. Evgeniou, and M. Pontil. Convex multi-task feature learning. *Mach. Learn.*, 73(3):243–272, Dec. 2008.
3. R. Caruana. Multitask learning. *Mach. Learn.*, 28(1):41–75, July 1997.
4. J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.
5. H. D. III. Frustratingly easy domain adaptation. In J. A. Carroll, A. van den Bosch, and A. Zaenen, editors, *ACL*. The Association for Computational Linguistics, 2007.
6. A. Jalali, S. Sanghavi, P. Ravikumar, and C. Ruan. A dirty model for multi-task learning. In *In NIPS*, 2010.
7. Z. Kang, K. Grauman, and F. Sha. Learning with whom to share in multi-task feature learning. In L. Getoor and T. Scheffer, editors, *ICML*, pages 521–528. Omnipress, 2011.
8. S. Kim and E. P. Xing. Tree-guided group lasso for multi-task regression with structured sparsity. In *In Proceedings of the 27th International Conference on Machine Learning*, pages 543–550. Omnipress, 2010.
9. J. R. Koza. *Genetic programming - on the programming of computers by means of natural selection*. Complex adaptive systems. MIT Press, Cambridge, MA, USA, 1992.
10. A. Kumar and H. Daum III. Learning task grouping and overlap in multi-task learning. In *ICML*. icml.cc / Omnipress, 2012.

11. T. McConaghy. FFX: Fast, scalable, deterministic symbolic regression technology. In R. Riolo, E. Vladislavleva, and J. H. Moore, editors, *Genetic Programming Theory and Practice IX*, Genetic and Evolutionary Computation, pages 235–260. Springer New York, 2011.
12. S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Trans. on Knowl. and Data Eng.*, 22(10):1345–1359, Oct. 2010.
13. 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.
14. R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 2008.
15. M. Schmidt and H. Lipson. Distilling Free-Form Natural Laws from Experimental Data. *Science*, 324(5923):81–85, Apr. 2009.
16. R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994.
17. H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B*, 67:301–320, 2005.
18. A. Zweig and D. Weinshall. Hierarchical regularization cascade for joint learning. In *ICML (3)*, volume 28 of *JMLR Proceedings*, pages 37–45. JMLR.org, 2013.