Towards Automatic Complex Feature Engineering: 19th International Conference, Dubai, United Arab Emirates, November 12-15, 2018, Proceedings, Part II

Chapter · November 2018

DOI: 10.1007/978-3-030-02925-8_22

CITATION
1

READS
293

3 authors, including:

Zhang Jianyu
TianJin Unversity
4 PUBLICATIONS **2** CITATIONS
SEE PROFILE

Francoise Soulie Fogelman
Hub France IA
153 PUBLICATIONS **2,083** CITATIONS
SEE PROFILE

Towards Automatic Complex Feature Engineering

Jianyu Zhang[1] [0000-0002-0693-6809], Françoise Fogelman-Soulié[1] [0000-0002-3172-6977], Christine Largeron[2] [1111-2222-3333-4444]

[1] Tianjin University, Tianjin, 300350, China
(edzhang,soulie)@tju.edu.cn
[2] Université de Lyon, F-42023, Saint-Etienne, France,
CNRS, UMR 5516, Laboratoire Hubert Curien, F-42000, Saint-Etienne, France
Université de Saint-Etienne, Jean-Monnet, F-42000, Saint-Etienne, France
christine.largeron@univ-st-etienne.fr

Abstract. Feature engineering is one of the most difficult and time-consuming tasks in data mining projects, and requires strong expert knowledge. Existing feature engineering techniques tend to use limited numbers of simple feature transformation methods and validate on simple datasets (small volume, simple structure), obviously limiting the benefits of feature engineering. In this paper, we propose a general *Automatic Feature Engineering Machine* framework (*AFEM* for short), which defines families of complex features and introduces them one family at a time (block bottom-up). We show that this framework covers most of the existing features used in the literature and allows us to efficiently generate complex feature families: in particular, local time, social network and representation-based families for relational and graph datasets, as well as composition of features. We validate our approach on two large realistic competitions datasets and a recommendation system task with social network. In the first two tasks, *AFEM* automatically reached ranks 15 and 12 compared to human teams; in the last task, it achieved 1.5% regression error reduction, compared to best results in the literature. Furthermore, in the context of big data and web applications, by balancing computation time and number of features / performance, in one case, we could reduce 2/3 computation time with only 0.2% AUC performance loss. Our code is publicly available on GitHub[1].

Keywords: Feature Engineering, Machine Learning for the Web, Social Network Computing, Big Data, Web Application.

1 Introduction

In recent years, it has been recognized in machine learning that *feature engineering* is the most critical factor for performance [1]: features are more important than the machine learning algorithm used. A *derived feature* is produced from the existing features set (originally the raw data) by some transformation: feature engineering (*FE* in the rest of this paper) aims at producing "good" derived features, i.e. features improv-

[1] https://github.com/TjuJianyu/AFEM

ing performances, through a model simpler and easier to train (i.e. requiring less examples) than with the original features set. The standard approach to *FE* is to carefully hand-craft meaningful features, a very time-consuming process, more art than science, requiring domain knowledge and trials and errors. It is not rare to spend 90% of a machine learning project on *FE*, so automating it would save significant time and effort. As a consequence, research groups and companies, such as IBM for example [4-6, 8] have started to produce methods for (semi-) automatic *FE*.

Since the number of possible derived features is potentially infinite, it is necessary to be able to control the generation of features. There exist basically two ways: the *top-down* approach (expansion-reduction [5]) consists in generating all the features for a full set of transformations, and then do feature selection and performance evaluation of a model trained on the enriched feature set; this is of course very costly in terms of computation, if the sets of transformations and original features are of large sizes, which is why most publications limit these sizes. In the *bottom-up* approach (evolution-centric [5]), features are progressively entered and the most interesting ones are selected, either using meta-features to characterize their quality [3] or through learning [4-5, 8]; this approach is more scalable than the previous one, but it still is compute-intensive, since it evaluates performances of derived features. Our contributions are as follows:

- We develop a modular, general and flexible framework, *AFEM*, for generating new features in relational datasets. The framework can handle many common types of features, both simple and more complex, structured in *families*, including e.g. timestamp, social network and representation related features.
- We introduce a *block-bottom-up* approach, intermediate between top-down and bottom-up, for progressively inserting and evaluating derived features, by introducing blocks of features and evaluating the most promising ones in each family.
- We evaluate *AFEM* on three realistic and relatively large datasets: in KDD cup 2014[2], *AFEM* got rank 15/472 on private leaderboard; in WSDM cup 2018[3] *AFEM* got 6th rank out of 1081 teams [10]; in Last.fm[4] rating prediction task, *AFEM* got 1.5% regression error reduction compared to literature.

The paper is organized as follows: in section 2, we present definitions and notations, related work in section 3 and our approach in section 4. Experimental results are presented and analyzed in section 5. Section 6 concludes with a discussion of the potential of our approach and perspectives for further research.

2 Definitions and notations

In this paper, we assume that we have (at least) one *entity E* and a predictive task associated to it; we want to derive new features for *E*. For example, in Last.fm[4]

[2] https://www.kaggle.com/c/kdd-cup-2014-predicting-excitement-at-donors-choose
[3] https://wsdm-cup-2018.kkbox.events/
[4] http://files.grouplens.org/datasets/hetrec2011/hetrec2011-delicious-2k.zip

(Fig.1-c), User and Artist are two entities containing features of users (e.g. age, city) and artists and other tables allow to derive features for these two entities, e.g. how many times a user listened to an artist`s music.

Let be given an entity E, an individual i with entity index Id_E and a features set F, we will call *neighborhood* N of i, the result of a mapping *Neigh* from Id_E and p attributes of i in features set F: a neighborhood is a set of instances that are close with each other by some distance metrics. It could be a set of users, a time interval, a geographic region, etc. So we call S the space in which neighborhoods are and, for simplicity, do not further specify S. For example, in Fig.1-c, the neighborhood of a user could be the set of his friends with the same age living in the same city:

$$Neigh: \left(Id_E, a_1, a_2, \dots, a_p\right) \rightarrow N \tag{1}$$

A *function* of arity n is a mapping for deriving a feature from n features (this is a *global function*) or from n features and a neighborhood (a *local function*). For example, we can define the count of songs by a specific artist which users in N listened to. A derived feature is of *order k* if it depends upon k attributes. k is thus the number of distinct elements in the set of attributes $\{b_1, b_2, \dots, b_n; a_1, a_2, \dots, a_p\}$ the feature depends upon (through f and N in equation (2)). Obviously, derived features of large order will require a lot more computing time than features of order 1:

$$f: (b_1, b_2, \dots, b_n) \rightarrow b \qquad f: (b_1, b_2, \dots, b_n; N) \rightarrow b \tag{2}$$

A *transformation* is defined as $T = (T_{Func})$ (*global transformation*) or $T = \left(T_{Func},\ T_{Neigh}\right)$ (*local transformation*), where T_{Func} is a set of functions (global or local) and T_{Neigh} is a set of neighborhoods.

The *feature engineering process* consists in starting from an initial collection of features F_0, collected from the raw data, and progressively deriving enriched feature sets, $F_1, F_2, \dots, F_t, \dots$ through transformations $T_0, T_1, \dots, T_t, \dots$

$$F_t \rightarrow F_{t+1} = F_t \cup DF_t \tag{3}$$

where DF_t is the *set of derived features* φ obtained as:

$$\varphi \in DF_t: \varphi = f(b_1, b_2, \dots, b_n) \quad \text{if } T_t = (T^t_{Func}) \text{ is global,} \quad f \in T_t$$

$$\varphi \in DF_t: \varphi = f(b_1, b_2, \dots, b_n; N) \quad \text{if } T_t = \left(T^t_{Func}, T^t_{Neigh}\right) \text{ is local,} \quad (f, N) \in T_t \tag{4}$$

where $f \in T^t_{Func}$, $b_1, b_2, \dots, b_n; a_1, a_2, \dots, a_p \in F_t$, $N = Neigh\left(Id_E, a_1, a_2, \dots, a_p\right) \in T^t_{Neigh}$. By changing, at each time step t, the set of transformations T_t we can define various groups of features, we will describe some in section 4.

3 Related work

It has long been known in Machine Learning that original data might not be best for building a simple model with good performance. Various researchers have indeed demonstrated the benefit of using additional features: for example, the final winner in the Netflix prize [9] designed 16 *Global Effects* and 24 *Global Time Effects* features. Deep learning automatically learns features from raw data, but training requires careful design of the network structure, as well as enough examples to learn the large number of the network parameters (weights). In many common tasks with relational datasets, pure deep learning may not work well, because of the complex structure and semantic meaning of the relations. Moreover features learnt by deep learning are opaque, while derived features have explicit meaning. The literature on *FE* is not very abundant. It contains two main lines of approaches. In the *bottom-up approach* (or evolution-centric), features are progressively added and evaluated, either using meta-features (e.g. information gain for LFE [8]) or using the performance of a true prediction model (Cognito [4], or reinforcement learning [5]). In the *top-down approach* (or expansion-reduction), all features are generated and used to build a true prediction model; then, based on that model, most important features are selected. This approach is not feasible in practice with large datasets and / or large number of features. To the best of our knowledge, the first automatic *FE* work on "flat" data is ExploreKit [3]; it uses a predefined structured set of operations to generate all possible candidate features, and learns to rank and select them through a machine learning model with a series of meta-features. It is very compute-intensive due to the generation of all possible features and the evaluation of each feature through the performance of a true prediction model. Data Science Machine *DSM* [2] and One Button Machine *OneBM* [11] can do automatic *FE* from relational data through predefined aggregation operators, but cannot handle complex data structure and features. Until now, most research on automatic *FE* proposes very simple features, but, to the best of our knowledge, nobody uses more complex features, such as social network-based or representation-based features. Some works like *DSM* and *OneBM* extract features from timestamp, but do not get complex time-based features, which can actually be very significant in many time-related tasks.

To summarize, there are three major issues of feature engineering to be considered:

- *FE* is rather domain-specific, and very time-consuming. It is thus critical to automate this process as much as possible (a *holy grail of machine learning* according to [1]), with a *generic, domain-independent* method.
- Adding more features increases the dimension of the attributes space. Evaluating a large number of derived features may thus become a problem: the top-down approach might not be feasible at all.
- Incremental *evaluation of derived features* (bottom-up) is certainly dependent upon the order of inclusion of derived features. *AFEM* proposes a heuristic approach to this problem, but a full solution for this is not in the scope of this paper.

4 Method

We want to solve a prediction task *P* for an entity *E* and measure the performance through some indicator *Perf*. In any prediction task we want to solve, we assume that there is a unique index which labels each observation of entity *E* in the dataset.

4.1 Families of features

A *family* of features is defined through a set of transformations of a given type (global or local transformations). Families should be generic (domain-independent) and computable for most datasets. We will use the following families:

- Ψ_{stat} is the family of *statistical features*. They are global, order-1 features such as: {max, min, sum, mean, var, std ...} on numerical features or {count, count-distinct, most frequent ...} on categorical features; or order 2 such as {ratio, mean-difference ...}. This family is the one most commonly used in the literature.
- Ψ_{time} is the family of *time-based features*, for attributes with time-stamps. They include global features such as, for order-1 for example: {day-of-week, day-of-month, ..., time-to-last, time-since-last, ...}; and local features, obtained from local Ψ_{stat} defined on time-based neighborhoods, such as for example: {max, min, sum, mean, var, std ...} or {count, count-distinct, most frequent ...} in time windows {last-hour, last-week, ...}. These features are very common in the literature.
- Ψ_{SN} is the family of *social graph-based features*, when observations of entity *E* are nodes in a social graph. They include global features such as for example: {degree, clustering coefficient, community index ...}; and local features, obtained from local Ψ_{stat}, defined on graph-based neighborhoods, such as for example: {max, min, sum, mean, var, std ...} in {first-circle, second-circle, community...}.
- Ψ_{Rep} is the family of *representation-based features*. These are mostly global features obtained through embedding of the original data, e.g. *SVD*, *PCA*, *AE* (deep learning auto-encoder) features.

Obviously, many more families could be defined: for example, when data on (longitude, latitude) are available, the Ψ_{Geo} family includes features in neighborhoods such as {street, district, city, region, country…}.(We will not use this family here).

4.2 Process for deriving features

Let us now describe our Automatic Feature Engineering Machine (*AFEM*) process to derive features. *AFEM* is an extension of *DSM* [2] and *OneBM* [6]. It can handle categorical and numeric features in different tables, but also history features with timestamp and social network features. It is a *block bottom-up approach* (like [2, 6]), but with blocks of features of a same family entered at successive time steps. We start from an initial features set F_0, collected from the raw data on entity *E* and progressively derive enriched feature sets, $F_1, F_2, \ldots, F_t, \ldots$ through transformations $T_0, T_1, \ldots, T_t, \ldots$, by equations (3-4). At each stage, transformations T_t are from one family of features only. To limit computation time, we will always generate global order-1 features first, then local order-1, global order-2, local order-2... We will choose the order in which we put the families in: usually, in this paper, we use the order Ψ_{Stat}, Ψ_{time}, Ψ_{SN}, Ψ_{Rep}. Of course any order is possible. Let us denote $\Psi^0, \Psi^1, \ldots, \Psi^t, \ldots$ the succession of family blocks successively applied. Once we have introduced, at time *t*, a block of features from one family (for example global order-1 Ψ_{Stat}), we build a model on $F_{t+1} = F_t \cup DF_t$, evaluate performance *Perf* and select the top-k_t most important features: we retain the k_t features with importance larger than some ratio α of total importance (for example, $\alpha = 1\%$). The process is described in Table 1. Obviously, through this process, features can be easily composed.

Table 1. *AFEM*, Automatic Feature Engineering Machine Process for deriving features

```
For entity E, prediction task P, importance Imp, ratio $\alpha$, performance evaluation Perf
Input: $\Psi^0, \Psi^1, \ldots, \Psi^T$, $\alpha$, initial feature set $F_0$        Output: Feature set $F_{T+1}$
For t=0 to T do
  • $\Psi^t = (T^t_{Func})$ or $\Psi^t = (T^t_{Func}, T^t_{Neigh})$
  • $F_t \to F_{t+1} = F_t \cup DF_t$ with
    − $DF_t = \{\varphi = f(b_1, b_2, \ldots, b_n) / f \in T^t_{Func}\}$,              if $\Psi^t$ is global
    − $DF_t = \{\varphi = f(b_1, b_2, \ldots, b_n; N) / f \in T^t_{Func}, N \in T^t_{Neigh}\}$   if $\Psi^t$ is local
  • Build model on $F_{t+1}$, evaluate its performance Perf_t, select Top-k_t features with
    importance Imp larger than $\alpha$
  • $F_{t+1} := \{\text{Top } k_t \text{ most important features}\}$; t :=t+1
End
```

5 Experimental results

5.1 Datasets

In this section, we evaluate our *AFEM* framework on three datasets, which we have chosen because they are large and complex enough to allow for significant evaluation, and comparison to challenges or the literature [2, 6]. The datasets are as follows and their characteristics and structures are shown in Table 2 and Fig. 1.

- KDD'14[2]: participants are asked to predict whether a crowd-funded project will be "exciting" based on project descriptions and donation information.
- WSDM'18[3]: participants are asked to predict whether a user will listen to a song again after the first observable listening event based on user, song and context.
- Last.fm[4]: in this task, we need to predict the rating that a user gives to an artist based on the user social network and user-artist tagging behaviors.

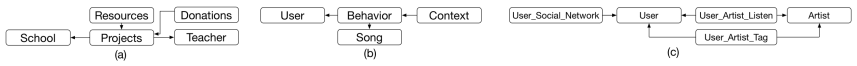

Fig. 1. Dataset structures: KDD'14 (a), WSDM'18 (b) and Last.fm (c). An arrow from one entity to another signifies that the first entity references the second in the dataset.

Table 2. Datasets used for evaluation

Dataset	# Rows	# Features	With Timestamp	With Graph
KDD'14	664,098	51	Yes	No
WSDM'18	9,934,208	19	Yes	No
Last.fm	92,834	7	Yes	Yes

5.2 Evaluation methodology

In our experiments, we use LightGBM, Random Forest or XGBoost[5] to build our models and select most important features. For all the experiments, we split the dataset in three parts: training, validation and test . We train the model in successive epochs on the training set and validate the results on the validation set, until there is no more improvement. We then retrain the model on the full data (train + validation) for that number of epochs and apply it on test set. To measure feature importance, we use Gini Importance / Mean Decrease in Impurity in Random Forest and split-based feature importance measure for XGBoost and LightGBM. In all experiments, we set feature importance ratio $\alpha = 1\%$ by default. The performance evaluators we use are those common for the algorithms we chose: AUC, MAE or RMSE. All experiments are run on a Dell R920 in-memory server with 96 cores, 2 TB RAM, in Python[1].

5.3 Results

In this section, we present the results obtained by our *AFEM* approach (Table 1) and compare them to those obtained on competition sites or to other *FE* techniques in the literature [2, 6] and to the global top-down approach. The *top-down* approach is very time-consuming and our *AFEM* approach, could be faster, but hopefully not degrade performances. Features generated are shown in Table 3.

Table 3. Number of features generated by *AFEM* for each task

Feature family	KDD 2014	WSDM 2018	Last.fm
Original	33	5	2
Statistic Ψ_{stat}	14	19	12
Time-based Ψ_{time}	384	28	-
Graph-based Ψ_{SN}	-	-	7
Representation-based Ψ_{Rep}	-	60	-
Total	431	112	21

KDD Cup 2014. We have used our *AFEM* approach, generating features in families Ψ_{stat} and Ψ_{time}. In Table 4, we compare the performances of *AFEM* (LightGBM model) and the *top-down* approach. As shown, performances regularly increase with the addition of new derived features and *AFEM* outperforms the *top-down* approach. We also compare *AFEM* to *DSM* and *OneBM* [2, 6] in Table 5 (results from *DSM* and *OneBM* are from the original papers, number of features are not available). *AFEM* outperforms the other two automatic *FE* techniques: on *DSM* our *AFEM* + LighGBM improves AUC performance by 5,41% and 130 ranks; for *OneBM* + XGBoost, resp. Random Forest our *AFEM* + LighGBM, resp. XGBoost, Random Forest improves resp. AUC by 4.35% and 66 ranks, 4,32%, 66 ranks and 0.07%, 3 ranks. In web-based applications, computing time, at the time when we apply a model, needs to be very small ensuring short latency to avoid losing customers, with a compromise between improved performance and shorter computation time. Fig. 2 shows the AUC perfor-

[5] http://lightgbm.readthedocs.io, http://scikit-learn.org, http://xgboost.readthedocs.io

mance and time cost of feature generation and training the model with different feature importance ratios α, as well as model apply time. As we increase α, we select less and less features for training, and thus less features when we apply it. In the situation where we choose α =0.17 instead of 0.01 (the default used previously), we save 2/3 computation time of feature generation and model training with only 0.2% AUC reduction; with the number of features reduced from 431 to 125, the time for applying the model is reduced from 1,163s to 264s.

Table 4. Performances on private leaderboard of *AFEM* iterative and *top-down* approaches on KDD Cup 2014. FS(x) indicates features selected from feature set x; $FT_{previous}$ features from previous step (previous row). Best results are shown **in bold**.

Approach	Feature used	#features	AUC	Rank	Top%
Iterative approach	Original features	33	0.58210	169	36%
	FS($FT_{previous}$)	28	0.58271	164	35%
	$FT_{previous}$ +Ψ_{stat}	70 (+42)	0.58659	142	30%
	FS($FT_{previous}$)	55	0.59463	90	19%
	$FT_{previous}$ + Ψ_{time}	**411 (+356)**	**0.64290**	**15**	**3%**
	FS($FT_{previous}$)	286	0.64045	15	3%
Top-down approach	Original features + Ψ_{stat} + Ψ_{time}	431	0.63919	17	4%
	FS($FT_{previous}$)	296	0.63845	17	4%

Table 5. Performances on private leaderboard of *AFEM*, *DSM* and *OneBM* on KDD Cup 2014

Methods	#features	AUC	Rank	Top %
DSM without tuning	N/A	0.55481	314	66.5%
DSM with tuning	N/A	0.58630	145	30.7%
OneBM + Random Forest	90	0.58983	118	25.0%
OneBM + XGBoost	90	0.59696	81	17.2%
AFEM + Random Forest	286	0.59052	115	24.4%
AFEM + XGBoost	286	0.64014	15	3.2%
AFEM + LightGBM	**286**	**0.64045**	**15**	3.2%

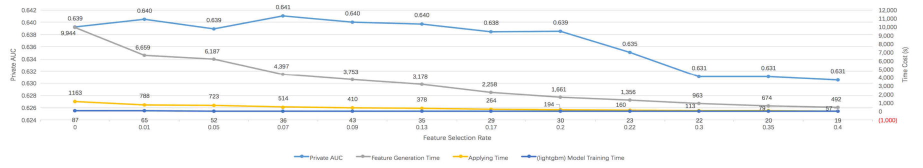

Fig. 2. AUC performance and time cost of feature generation, training model and total cost with different feature importance ratios α.

WSDM Cup 2018. In this task, there is no explicit timestamp in the dataset, but data are ordered chronologically. We use the index of examples as timestamp in *AFEM*. In Table 6, we show the performance of *AFEM* with LightGBM classifier. In our iterative approach, we can achieve 12th (top 1.1%) out of 1081 human teams. We can even get top 13.1% using statistic and time-based families only. However, in this case, *AFEM* is very slightly worse than the *top-down* method. We can also use *AFEM* with a *human-in-the-loop* approach. To illustrate this, we choose a model from an ensemble used in [10], which ranked 6th in WSDM'2018. In this single model, the authors use Ψ_{stat} features, Ψ_{Rep} features, similarity features and high-order features (a total of 186 features). We use all these features, then apply our *AFEM* approach on this initial feature set, except some duplicated features. We choose the same classification model as in [10], a LightGBM model with the same hyper-parameters. In Table 7, we show the improvement of *AFEM* for this Top model [10] in WSDM'18. *AFEM* can improve AUC by 0.43% for the top single model, which is very hard at this rank.

Table 6. Performances on private leaderboard of *AFEM* iterative approach and *top-down* approach on WSDM'18. Same notations as Table 5.

Approach	Feature used	#features	AUC	Rank	Top%
Iteratively approach	Original features	5	0.62233	882	81.6%
	$FT_{previous} + \Psi_{stat}$	24	0.67010	561	51.9%
	FS($FT_{previous}$)	22	0.67342	408	37.7%
	$FT_{previous} + \Psi_{time}$	50	0.68897	142	13.1%
	FS($FT_{previous}$)	40	0.68857	175	16.2%
	$FT_{previous} + \Psi_{Rep}$	100	0.71989	12	1.1%
	FS($FT_{previous}$)	93	0.71978	12	1.1%
Top-down approach	Original features+Ψ_{stat}+ Ψ_{time}+Ψ_{Rep}	112	0.71917	12	1.1%
	FS($FT_{previous}$)	**94**	**0.71992**	**12**	1.1%

Table 7. Performances on private leaderboard of *AFEM* for Top Model on WSDM'18.

	#features	AUC	Rank
Top model	186	0.72928	7
Top model + *AFEM*	**186+16**	**0.73359**	**6**

Last.fm. In this rating prediction task, we compare RMSE and MAE performance of our *AFEM* with BPMFSRIC [7] and other two approaches BPMF and Hao Ma`s method from BPMFSRC work. BPMF uses Bayesian probabilistic matrix factorization to predict the rating; BPMFSRIC adds social relations and item contents on top of BPMF; Hao Ma's method uses social networks to regularize the process of matrix factorization. To allow for comparisons, we map listening counts into ratings of 1 to 5 in the same way as BPMFSR. We use 80% ratings data to train and validate the model and apply the model on the remaining 20% data. The whole process is repeated 10 times to calculate confidence intervals. All these settings are the same as in the literature. Furthermore, the MAE and RMSE performance of BPMFSRIC, BPMF and Hao Ma's method on Last.fm dataset are collected from BPMFSRIC [7] directly. Our *AFEM* method here derives 7 graph-based features Table 8 shows the MAE and RMSE performance of *AFEM* and other three methods on Last.fm dataset. *AFEM*+LightGBM significantly outperforms all the other methods, on both performance indicators reducing regression errors RMSE by 1.5% and MAE by 1.1%.

Table 8. MAE and RMSE performance of Hao Ma's method, BPMF, BPMFSRIC and *AFEM* + LightGBM on Last.fm dataset.

Methods	Hao Ma`s	BPMF	BPMFSRIC	AFEM+LightGBM
MAE	0.4492$\pm$0.0013	0.3241$\pm$0.0012	0.3244 $\pm$0.0014	**0.3131$\pm$0.0011**
RMSE	0.6418$\pm$0.0026	0.4467$\pm$0.0024	0.4451$\pm$0.0026	**0.4305$\pm$0.0023**

6 Conclusion and Future Work

We propose a general feature engineering framework *AFEM* and define different complex feature families for performing automatic feature engineering on relational data. We present a block bottom-up evaluation and feature selection heuristic for selecting the most significant features. We show how to monitor the generation / selection process to limit the number of derived features and allow computation time compatible with Web applications requirements. We evaluate the performance of *AFEM* on 3 datasets from different domains. *AFEM* outperforms most of the human solutions and literature. It gets from rank 882 with raw data to rank 12 with automatically generated features in one competition; it largely outperforms state-of-the-art *DSM* and *OneBM* in another competition; it outperforms state-of-the-art on a classical dataset using social network features. But still many questions remain. In our future work, we intend to add more features and more complex feature families. The order of generating and evaluating different families is still a problem: using some algorithms such as reinforcement learning to learn the whole process is part of our future work.

7 References

1. Domingos, P. M.: A few useful things to know about machine learning. Communications of the ACM, 55(10), pp. 78–87 (2012).
2. Kanter, J.M., Veeramachaneni, K.: Deep feature synthesis: Towards automating data science endeavors. In: 2nd International Conference on Data Science and Advanced Analytics DSAA, pp. 1–10, IEEE, Paris, France (2015).
3. Katz, G., Shin, E.C.R., Song, D.: ExploreKit: Automatic Feature Generation and Selection. In: 16th International Conference on Data Mining ICDM 2016, pp. 979–984, IEEE, Barcelona, Spain (2016).
4. Khurana, U., Turaga, D., Samulowitz, H., Parthasrathy, S.: Cognito: Automated Feature Engineering for Supervised Learning. In 16th International Conference on Data Mining Workshops ICDMW, pp. 1304–1307, IEEE, Barcelona, Spain (2016).
5. Khurana, U., Samulowitz, H., Turaga, D.: Feature Engineering for Predictive Modeling using Reinforcement Learning. In: 32nd AAAI Conference on Artificial Intelligence AAAI-18, New Orleans, USA (2018).
6. Lam, H.T., Thiebaut, J.-M., Sinn, M., Chen, B., Mai, T., Alkan, O.: One button machine for automating feature engineering in relational databases. arXiv preprint arXiv:1706.00327 (2017).
7. Liu, J., Wu, C., Liu, W.: Bayesian probabilistic matrix factorization with social relations and item contents for recommendation. Decision Support Systems, 55(3), 838-850. (2013).
8. Nargesian, F., Samulowitz, H., Khurana, U., Khalil, E.B., Turaga, D.S.: Learning Feature Engineering for Classification. In: 26th International Joint Conference on Artificial Intelligence IJCAI 2017, pp. 2529–2535, Melbourne, Australia (2017).
9. Töscher, A., Jahrer, M. The Big Chaos Solution to the Netflix Grand Prize. AT&T Labs - Research Tech report (September 5, 2009).
10. Zhang, J., Fogelman-Soulié, F. KKbox's Music Recommendation Challenge Solution with Feature engineering. In: 11th ACM International Conference on Web Search and Data Mining WSDM 2018, Cup Workshop, Los Angeles, California, USA (2018).