Neural Feature Learning From Relational Database

Hoang Thanh Lam*
Dublin Research Laboratory
IBM Research
Dublin, Ireland
t.l.hoang@ie.ibm.com

Tran Ngoc Minh
Dublin Research Laboratory
IBM Research
Dublin, Ireland
m.n.tran@ibm.com

Mathieu Sinn
Dublin Research Laboratory
IBM Research
Dublin, Ireland
mathsinn@ie.ibm.com

Beat Buesser
Dublin Research Laboratory
IBM Research
Dublin, Ireland
beat.buesser@ie.ibm.com

Martin Wistuba
Dublin Research Laboratory
IBM Research
Dublin, Ireland
martin.wistuba@ie.ibm.com

Abstract

Feature engineering is one of the most important but most tedious tasks in data science. This work studies automation of feature learning from relational database. We first prove theoretically that finding the optimal features from relational data for predictive tasks is NP-hard. We propose an efficient rule-based approach based on heuristics and a deep neural network to automatically learn appropriate features from relational data. We benchmark our approaches in ensembles in past Kaggle competitions. Our new approach wins late medals and beats the state-of-the-art solutions with significant margins. To the best of our knowledge, this is the first time an automated data science system could win medals in Kaggle competitions with complex relational database.

1 Introduction

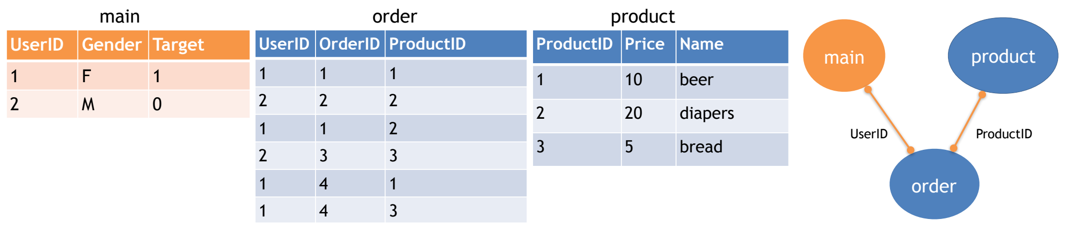
Often data science problems require machine learning models to be trained on table with one label and multiple feature columns. Data scientists must hand-craft these features from raw data. This process is called *feature engineering* and is one of the most tedious tasks in data science. Data scientists report that up to 95% of the total project time must be allocated to carefully hand-crafting features to achieve competitive models, for example see [2].

Here, we study methods to automate this step of feature engineering specifically for relational database because of four reasons. First, as mentioned data science projects involve many tedious trial-and-error steps and automation of these steps can significantly improve the productivity of data scientists. Second, before investing in data science projects with uncertain outcome quick estimates of the results can be achieved using automation. Third, automation democratizes data science by facilitating access for people with limited data science skills. Fourth, relational data is reported as the most popular type of data in industry reported by a recent survey of 14000 data scientists by Kaggle (2017) with at least 65% working daily with relational data.

The full automation of feature engineering for general purposes is very challenging, especially in applications where specific domain knowledge is an advantage. However, recent work by Kanter

*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

[2] http://blog.kaggle.com/2016/09/27/grupo-bimbo-inventory-demand-winners-interviewclustifier-alex-andrey/



main

UserID	Gender	Target
1	F	1
2	M	0

order

UserID	OrderID	ProductID
1	1	1
2	2	2
1	1	2
2	3	3
1	4	1
1	4	3

product

ProductID	Price	Name
1	10	beer
2	20	diapers
3	5	bread

Figure 1: A toy database and its relational graph.

and Veeramachaneni (2015) indicates that for relational data impressive performances like top 24-36% of all participants on Kaggle competitions can be achieved fully automated. A disadvantage of the cited work is its limitation to numerical data and neglection of temporal information. Moreover, the set of features contains redundant information because it is extracted using a set of predefined rules irrespective of the domain and targeted problems. In this work, we extend this prior art with a rule-based approach to deal with non-numerical and temporally ordered data and further improve this approach via feature learning using a deep neural network to learn relevant transformations.

We present experiments with different completed Kaggle competitions where our methods outperform the state-of-the-art baselines and achieve the top 3-8% of all participants. These results are achieved with minimal effort on data preparation and manual feature engineering within one week, while the competitions lasted for at least two months.

2 Backgrounds

Let $D = \{T_0, T_1, \cdots, T_n\}$ be a database of tables. Consider T_0 as the main table which has a target column, several foreign key columns and optional attribute columns. Each entry in the main table corresponds to a training example.

Example 1. *Figure 1 shows an example database with 3 tables. User table (main) contains a prediction target column indicating whether a user is a loyal customer. User shopping transactions are kept in the order table and the product table includes product price and names.*

A *relational graph* is a graph where nodes are tables and edges are links between tables via foreign-key relationships. Figure 1 shows the relational graph of the database in the same figure.

Definition 1 (Joining path)**.** *A joining path is a sequence* $p = T_0 \xrightarrow{c_1} T_1 \xrightarrow{c_2} T_2 \cdots \xrightarrow{c_k} T_k \mapsto c$*, where* T_0 *is the main table, each* T_i *is a table in the database,* c_i *is a foreign-key column connecting tables* T_{i-1} *and* T_i*, and* c *is a column (or a list of columns) in the last table* T_k *on the path.*

Example 2. *Joining the tables following the path* $p = main \xrightarrow{UserID} order \xrightarrow{ProductID} product \mapsto Price$*, we can obtain the price of all products that have been purchased by a user. The joined result can be represented as a relational tree defined in Definition 2 below.*

Definition 2 (Relational tree)**.** *Given a training example with identifier* e *and a joining path* $p = T_0 \xrightarrow{c_1} T_1 \xrightarrow{c_2} T_2 \cdots \xrightarrow{c_k} T_k \mapsto c$*, a relational tree, denoted as* t_e^p*, is a tree representation of the joined result for the entity* e *following the joining path* p*. The tree* t_e^p *has maximum depth* $d = k$*. The root of the tree corresponds to the training example* e*. Intermediate nodes at depth* $0 < j < k$ *represent the rows in the table* T_j*. A node at depth* $j - 1$ *connects to a node at depth* j *if the corresponding rows in tables* T_{j-1} *and table* T_j *share the same value of the foreign-key column* c_j*. Each leaf node of the tree represents the value of the data column* c *in the last table* T_k*.*

Example 3. *Figure 2.a shows a relational tree for* $UserID = 1$ *following the joining path* $p = main \xrightarrow{UserID} order \xrightarrow{ProductID} product \mapsto Price$*. As can be seen, the that user made two orders represented by two intermediate nodes at depth* $d = 1$*. Besides, order 1 includes two products with* $ProductID = 1$ *and* $ProductID = 2$*, while order 4 consists of products with* $ProductID = 1$ *and* $ProductID = 3$*. The leaves of the tree carry the price of the purchased products.*

Definition 3 (Tree transformation)**.** *A transformation function* f *is a map from a relational tree* t_e^p *to a fixed size vector* $x \in R^l$*, i.e.* $f(t_e^p) = x$*. Vector* x *is called a feature vector.*

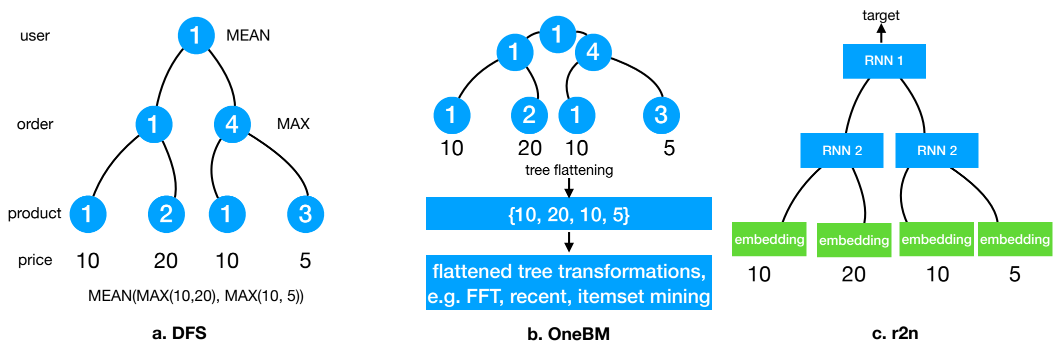


Figure 2: Tree transformations: DFS uses augmented aggregation functions at each level of the tree. OneBM flattens out the trees and transforms the flattened results using transformations that support unstructured and with temporally ordered data. R2N uses supervised learning to transform and embed any type of data using relational recurrent neural nets.

In general, feature engineering looks for relevant tree transformations to convert a tree into input feature vectors for machine learning models. For example, if we sum up the prices of all products carried at the leaves of the tree in Figure 2.a we obtain the purchased product price sum which can be a good predictor for the loyalty customer target.

3 Problem definition and complexity analysis

Given a relational database DB with m training examples $E = \{(e_1, y_1), (e_2, y_2), \cdots, (e_m, y_m)\}$, where $Y = \{y_1, \cdots, y_m\}$ is a set of labels. Let denote $P = \{p_1, p_2, \cdots, p_q\}$ as a set of joining paths extracted from the relational graph of DB. Recall that for each entity e_j following a joining path p_i we can obtain a relational tree $t_{e_j}^{p_i}$. Let $f_{p_i} \in F$ (the set of candidate transformations) be a tree transformation function associated with the path p_i, denote $f_{p_i}(t_{e_j}^{p_i}) = x_j^i$ as the feature vector extracted for e_j by following the path p_i.

Let $g(x_j^1 \oplus x_j^2 \oplus \cdots \oplus x_j^q) = \hat{y}_j$, be a machine learning model that estimates y_j from a concatenation of the feature vectors obtained from q joining paths. Denote $L_{P,F,g}(Y, \hat{Y})$ as the loss function defined over the set of ground-truth labels and the set of estimated labels $\hat{Y} = \{\hat{y}_1, \cdots, \hat{y}_m\}$

Problem 1 (Feature learning from relational data)**.** *Given a relational database, find the set of joining paths, transformations and models such that* $P^*, F^*, g^* = argminL_{P,F,g}(Y, \hat{Y})$.

The following theorem shows that Problem 1 as an optimization problem is NP-hard even when F and g are given (see the proof in supplementary material).

Theorem 1. *Given a relational graph, the candidate set of transformations F and model g , searching for the optimal path for predicting the correct label is an NP-Hard problem.*

Since the problem is hard, in sub-section 5.3, we explain efficient heuristic approaches for joining path generation. On the other hand, finding the best model g when the features are given is a model selection problem. That problem has been intensively studied in the machine learning literature, therefore, we limit the scope of this work to finding the relevant tree transformations.

4 A rule based approach for tree transformation

Given relational trees, there are different ways to transform the trees into features. In this section, we discuss rule-based approaches predefining tree transformations based on heuristics. The method discussed in this section is an extension of the Data Science Machine (DFS) by Kanter and Veeramachaneni (2015) so we first briefly recall the DFS algorithm.

In DFS, transformation function f is a composition of basic aggregation functions such as AVG, SUM, MIN and MAX augmented at each depth level of a tree. For instance, for the relational tree

Table 1: Transformation rules in OneBM

Data type	**transformation functions**
numerical	as is
categorical	order by frequency and use order as the transformation
timestamp	calendar features, gap to cut-off time if exists
timestamp series	series of gaps to cut-off time if exists
number multi-set	avg, variance, max, min, sum, count
multi-set of items	count, distinct count, high correlated items
timeseries	avg, max, min, sum, count, variance, recent(k), normalized count and sum to the max gap to cut-off
sequence, texts, set of texts	count, distinct count, high correlated symbols

t_1^p in Figure 2.a, a feature can be collected for $UserID = 1$ by applying MEAN and MAX at the root and first depth level respectively. The aggregation function at each node takes input from the children and outputs a value which is in turn served as an input to its parent. The example in Figure 2.a produces a feature $MEAN(MAX(10, 20), MAX(10, 5)) = 15$ corresponding to the average of the maximum price of purchased products by a user, which could be a good predictor for the user loyalty target.

DFS works well for numerical data, however, it does not support non-numerical data. For instance, if the product name instead of price is considered, the given set of basic transformations become irrelevant. Moreover, when the nodes of the trees have temporal order the basic aggregations ignore temporal patterns in the data. Therefore, we extend DFS to One Button Machine (OneBM) to deal with such situation (see the supplementary material).

In OneBM, the tree is first flattened out by a GroupBy operation at the root node to produce a set or a sequence of values as can be seen in Figure 2.b . Depending on the type of the values carried at the leaves, different transformations are applied. For instance, the tree t_1^p in Figure 2 can be flattened out into a multi-set: $s = \{10, 20, 10, 5\}$. When there is an order associated with these values the multi-sets turn into timeseries. On the other hand, if the data column is the product name instead of price, we obtain an itemset or a sequence of purchased products.

For each type of such data, popular transformations such as correlated itemsets, correlated subsequences, autocorrelation coefficients can be extracted. A full list of the most popular transformations for each special type of data is described in Table 1. This simple approach allows us to incorporate additional user-defined transformations and being able to handle complex transformations beyond simple aggregations. As can be seen in experiments, this approach produces very good results in most Kaggle competitions. Once features are generated by the given set of rules, feature selection is needed to remove irrelevant features. Feature selection is a well-studied topic, please refer to the supplementary material for the discussion of our choice in feature selection.

The rule based approaches like OneBM and DFS specify the transformation functions based on heuristics regardless of the domain. In practice, predefined transformations can't be universally relevant for any use-case. In the next section we introduce an approach to go around this issue.

5 Neural feature learning

In this section, we discuss an approach that learns transformation from labelled data rather than being specified *a-priori* by the user.

5.1 Relational recurrent neural network

To simplify the discussion, we make some assumptions as follows (an extension to the general case is discussed in the next section):

- the last column c in the joining path p is a fixed-size numerical vector.
- all nodes at the same depth of the relational tree are ordered according to a predefined order.

With the given simplification, transformation function f and prediction function g can be learned from data by training a deep neural network structure that includes a set of recurrent neural networks (RNNs). We call the given network structure *relational recurrent neural network* (r2n) as it transforms relational data using recurrent neural networks.

There are many variants of RNN, in this work we assume that an RNN takes as input a sequence of vectors and outputs a vector. An RNN is denoted as $rnn(s, \theta)$, where s is a variable size sequence of vectors and θ is the network parameter. Although the discussion focuses on RNN cells, our framework also works for Long Short Term Memory (LSTM) or Gated Recent Unit (GRU) cells.

Definition 4 (Relational Recurrent Neural Network)**.** *For a given relational tree t_e^p, a relational recurrent neural network is a function denoted as $r2n(t_e^p, \theta)$ that maps the relational tree to a target value y_e. An $r2n$ is a tree of $RNNs$, in which at every intermediate node, there is an RNN that takes as input a sequence of output vectors of the $RNNs$ resident at its children nodes. In an $r2n$, all RNNs, resident at the same depth d, share the same parameter set θ_d.*

Example 4. *Figure 2.c shows an r2n of the tree depicted in Figure 2.a . As it is observed, an $r2n$ summarizes the data under every node at depth d in the relation tree via a function parametrized by an RNN with parameters θ_d (shared for all RNNs at the same depth). Compared to the DFS method in Figure 2.a, the transformations are learned from the data rather than be specified a-priori by the users.*

5.2 A universal r2n

In this section, we discuss a neural network structure that works for the general case even without the two assumptions made in Section 5.1.

5.2.1 Dealing with unstructured data

When input data is unstructured, we add at each leaf node an embedding layer that embeds the input into a vector of numerical values. The embedded layers can be learned jointly with the $r2n$ network as shown in Figure 2.c. For example, if the input is a categorical value, a direct look-up table is used, that maps each categorical value to a fixed size vector. If the input is a sequence, an RNN is used to embed a sequence to a vector. In general, the given list can be extended to handle more complicated data types such as graphs, images and GPS trajectories.

5.2.2 Dealing with unordered data

When data is not associated with an order, the input is a set instead of a sequence. In that case, the transformation function $f(s)$ takes input as a set, we call such function as set transformation. It is important to notice $f(s)$ is invariant in any random permutation of s. An interesting question to ask is if recurrent neural network can be used to approximate any set function $f(s)$. Unfortunately, the following theorem shows that, there is no recurrent neural network that can approximate any set function except the constant or the sum function.

Theorem 2 (Expressiveness)**.** *If a recurrent neural network $rnn(s, W, H, U)$ with linear activation is a set function, it is either a constant function or can be represented as:*

$$rnn(s, W, H, U) \quad = \quad c + h_0U + |s| * bU + UW * sum(s) \tag{1}$$

From equation 1 we can imply that an RNN cannot approximate the Max set and Min set functions unless we define an order on the input data. Therefore, we sort the input vectors according to vector mean value to ensure an order for input data.,

5.3 Joining path generation

So far, we have discussed feature learning from relational trees extracted from a database when a joining path is given. In this section, we discuss different strategies to generate relevant joining paths. Because finding the optimal paths is hard, we limit the maximum depth of the joining paths and propose simple heuristic traversing strategies: simple: only allows simple paths (no repeated nodes); forward only: nodes are assigned depth numbers based on a breadth-first traversal starting from the main table. Path generation only considers the paths such that latter nodes must be deeper than the former ones; all: all paths are considered

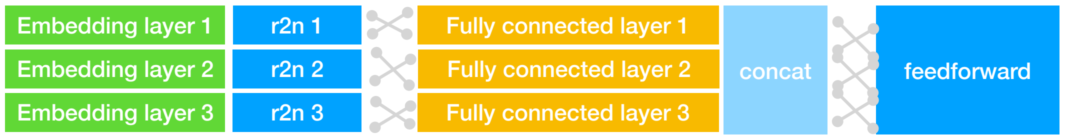


Figure 3: Network for multiple paths: data from each joining path is transformed using an embedding layer, an r2n before being combined by a fully connected layers and a feedforward layer.

Table 2: Datasets and running time of OneBM (60 GPUs), DFS (1 GPU) and r2n (4 CPUs + 1 GPU)

Data	**♯ tables**	**♯ columns**	**size**	**OneBM**	**r2n**	**DFS**
KDD Cup 2014	4	51	0.9 GB	12.9 h	1 week	NA
Coupon purchase	7	50	2.2 GB	2.8 h	1 week	84.04 hours
Grupo Bimbo	5	19	7.2 GB	56 min.	1 week	2 weeks (not finished)

In our experiments, forward only is the most efficient which is our first choice. The other strategies are supported for the completeness. For any strategy, the joined tables can be very large, especially when the maximum depth is set high. Therefore, we apply sampling strategies and caching intermediate tables to save memory and speed up the join operations (see the supplementary material).

5.4 Networks for multiple joining paths

Recall that for each joining path p_i, we create an $r2n_i$ network that learns features from the data generated by the joining path. In order to jointly learn features from multiple joining paths $p_1, p_2, \cdots, p_m$, we use a fully connected layer that transform the output of the $r2n_i$ to a fixed size output vector before concatenating these vectors and use a feed-forward network to transform them into a desired final output size. The entire network structure is illustrated in Figure 3. For classification problems, additional softmax function is applied on the final output vector to obtain the class prediction distribution for classification problem.

6 Experiments

In this section we discuss the experimental results. DFS is considered as a baseline to compare to in addition to manual feature engineering approaches provided by Kaggle participants. Due to space limit, more details about data preparation, hyper-parameter automatic tuning, experimental settings can be found in Appendices (see supplementary material).

6.1 Datasets

Three Kaggle competitions with complex relational graphs and different problem types (classification, regression and ranking) were chosen for validation (see Figure 4). The data characteristics are described in Table 2 and the graphs are illustrated in Figure 4 where Coupon data has the most complex graph.

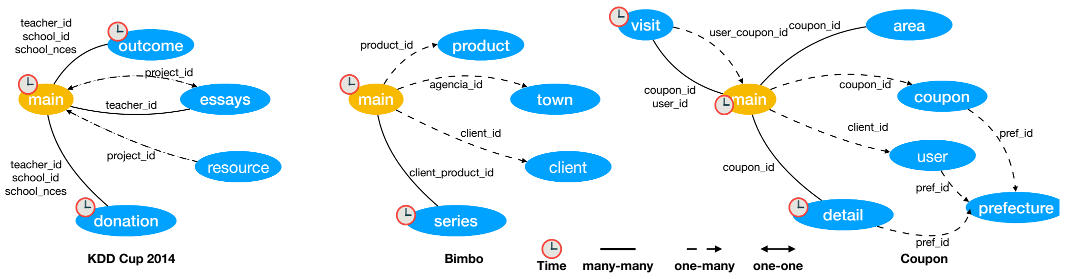


Figure 4: Relational graphs of the Kaggle datasets

Table 3: Data science competition results

Competition	Task	Metric	DFS	OneBM	R2N	Ensemble
KDD Cup 2014	Classification	AUC	0.586	0.622	0.619	**0.626**
Groupo Bimbo	Regression	LRMSE	NA	0.475	0.485	**0.472**
Coupon Purchase	Ranking	MAP@10	0.005635	**0.007416**	0.004228	0.006065

Table 4: Ranking of the best among r2n, OneBM and their ensemble

Competition	Rank	Top(%)	Medal
KDD Cup 2014	31/472	6.5	Silver
Groupo Bimbo	152/1969	7.7	Bronze
Coupon Purchase	30/1076	2.7	Silver

6.2 Experimental settings and running time

A Spark cluster with 60 cores, 300 GB of memory and 2 TB of disk space was used to run the rule-based OneBM algorithm. Features from OneBM were fit into an XGBOOST model known as the most popular model in the Kaggle community. Hyper-parameters of XGBOOST were auto-tuned via the Bayesian optimization for 50 iterations (see the settings in supplementary material).

To train the r2n networks, we used a machine with one GPU with 12 GB of memory and 4 CPU cores with 100 GB of memory. Training one model until convergence needed 7 days. Auto-tuning the r2n hyper-parameters was not considered because of limited time budget. The network structure hyperparameters are fixed based on computational feasibility in our available computing resource (see supplementary material). All results are reported based on the Kaggle private leaderboard ranking information. The running time of the algorithms is reported in Table 2.

6.3 Kaggle competition results and discussion

Table 3 reports the results of DFS, OneBM, r2n and a linear ensemble (with equal weights) of r2n and OneBM on three Kaggle competitions where results are available (the result of DFS on Bimbo was not available as its run didn't finish within 2-weeks time budget. In the KDD Cup 2014 competition, both OneBM and r2n outperformed DFS with a significant margin. OneBM always achieves better results than r2n because of the robustness of the XGBOOST model which was auto-tuned by Bayesian optimization. However, r2n provides additional benefits to the rule-based approach as the linear ensemble of these methods shows better results than each individual model in both KDD Cup 2014 and Grupo Bimbo competitions.

The result of r2n in the coupon purchase dataset is worse than the other methods. As shown in Figure 4 coupon purchase's graph is more complex which might need careful network structure search and hyper-parameter tuning to achieve better performance.

Figure 5 shows a comparison between the best of OneBM, r2n and their linear ensemble (marked with a dash-line) and all Kaggle participants. As it is observed, in terms of prediction accuracy, our

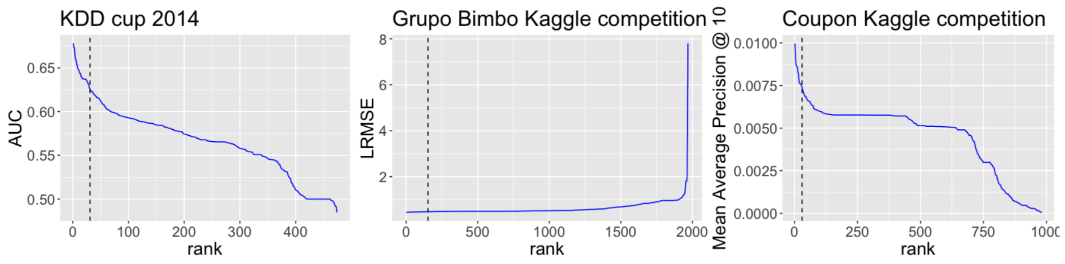


Figure 5: Final results on the private leaderboard ordered by rank. Our hypothetical rank is depicted by a dotted line.

method outperformed most participants and achieved results that were very close to the best teams. The ranking and achieved medals are reported in Table 4. To the best of our knowledge, before our solution, no other automatic system could win medals in Kaggle competitions.

7 Related Work

The data science work-flow includes five basic steps: problem formulation, data acquisition, data curation, feature engineering, model selection and hyper-parameter tuning. Most related works focus on automating the last two steps which will be reviewed in the following subsections.

7.1 Automatic model selection and tuning

Auto-Weka by Kotthoff et al. (2016); Thornton, Hutter, Hoos, and Leyton-Brown (Thornton et al.) and Auto-SkLearn by Feurer et al. (2015) are two popular tools trying to find the best combination of data pre-processing, hyper-parameter tuning and model selection. Both works are based on Bayesian optimization Brochu et al. (2010) to avoid exhaustive grid-search. Cognitive Automation of Data Science (CADS) Biem et al. (2015) is another system built on top of Weka, SPSS and R to automate model selection and hyper-parameter tuning processes. Besides these works, TPOT by Olson et al. (2016) is another system that uses genetic programming to find the best model configuration and pre-processing work-flow. In summary, automation of hyper-parameter tuning and model selection is a very attractive research topic with very rich literature. The key difference between our work and these works is that, while the state-of-the-art focuses on optimization of models given a ready set of features stored in a single table, our work focuses on preparing features as an input to these systems from relational databases with multiple tables. Therefore, these works are orthogonal to each other.

7.2 Automatic feature engineering

Different from automation of model selection and tuning where the literature is very rich, only a few works have been proposed to completely automate feature engineering for general problems. The main reason is that feature engineering is both domain and data specific. Therefore, we discuss related work for relational database.

DFS by Kanter and Veeramachaneni (2015) is the first system that automates feature engineering from relational data with multiple tables. DFS has been shown to achieve good results on public data science competitions. OneBM is closely related to work in inductive logic programming, e.g. see Muggleton and Raedt (1994) where relational data is unfolded via propositionalizing, e.g. see Kramer et al. (2001) or Wordification by Perovšek et al. (2015) discretises the data into words from which the joined results can be considered as a bag of words. Each word in the bag is a feature for further predictive modelling. Wordification is a rule-based approach, which does not support unstructured and temporally ordered data. In Knobbe et al. (1999), the authors proposed an approach to learn multi-relational decision tree induction for relational data. This work does not support temporally ordered data and is limited to decision tree models. Our work extended DFS to deal with non-numerical and temporally ordered data. Moreover, we resolved the redundancy issues of rule-based approaches via learning features rather than relying on predefined rules.

Besides, works in statistical relational learning (StarAI) presented in Getoor and Taskar (2007) are also related to our work. Recently, a deep relational learning approach was proposed by Kazemi and Poole (2017) to learn to predict object's properties using object's neighbourhood information. However, the given prior art does not support temporally ordered data and unstructured properties of objects. Besides, an important additional contribution of our work is the study of the theoretical complexity of the feature learning problem for relational data as well as the universal expressiveness of the network structures used for feature learning.

Cognito by Khurana, Turaga, Samulowitz, and Parthasarathy (Khurana et al.) automates feature engineering for one table. It applies recursively a set of predefined mathematical transformations on the table's columns to obtain new features from the original data. Since it does not support relational databases with multiple tables and is orthogonal to our approach.

8 Conclusion and future work

We have shown that feature engineering for relational data can be automated using predefined sets of heuristic transformations or by the r2n network structure. This opens many interesting research directions for the future. For example, the r2n network structure in this work is not auto-tuned due to the efficiency issue. Future work could focus on efficient methods for network structure search to boost the current results even more. Second, the rule based approach seems to be the best choice so far because of its effectiveness and efficiency, yet there are chances to improve the results further if a smarter graph traversal policy is considered. Although we have proved that finding the best joining path is NP-hard, the theoretical analysis assumes that there is no domain knowledge about the data. We believe that exploitation of semantic relation between tables and columns can lead to better search algorithm and better features.

References

Alain Biem, Maria Butrico, Mark Feblowitz, Tim Klinger, Yuri Malitsky, Kenney Ng, Adam Perer, Chandra Reddy, Anton Riabov, Horst Samulowitz, Daby M. Sow, Gerald Tesauro, and Deepak S. Turaga. 2015. Towards Cognitive Automation of Data Science. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. 4268–4269.

Eric Brochu, Vlad M Cora, and Nando De Freitas. 2010. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599* (2010).

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*. 2962–2970.

Lise Getoor and Ben Taskar. 2007. *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Kaggle. 2017. The state of machine learning and data science. A survey of 14000 data scientist.. `https://www.kaggle.com/surveys/2017`.

James Max Kanter and Kalyan Veeramachaneni. 2015. Deep feature synthesis: Towards automating data science endeavors. In *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*. IEEE, 1–10.

Richard M Karp. 1972. Reducibility among combinatorial problems. In *Complexity of computer computations*. Springer, 85–103.

Seyed Mehran Kazemi and David Poole. 2017. RelNN: A Deep Neural Model for Relational Learning. *CoRR* abs/1712.02831 (2017). arXiv:1712.02831

Udayan Khurana, Deepak Turaga, Horst Samulowitz, and Srinivasan Parthasarathy (Eds.). *Cognito: Automated Feature Engineering for Supervised Learning , ICDM 2016*.

Arno J. Knobbe, Arno Siebes, and Daniël van der Wallen. 1999. Multi-relational Decision Tree Induction. In *Principles of Data Mining and Knowledge Discovery, Third European Conference, PKDD '99, Prague, Czech Republic, September 15-18, 1999, Proceedings*. 378–383.

Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. 2016. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research* 17 (2016), 1–5.

Stefan Kramer, Nada Lavrac, and Peter Flach. 2001. Propositionalization Approaches to Relational Data Mining. In *Relational Data Mining*, Saso Dzeroski and Nada Lavrac (Eds.). Springer New York Inc., New York, NY, USA, 262–286.

J. Močkus. 1975. *On bayesian methods for seeking the extremum*. Springer Berlin Heidelberg, Berlin, Heidelberg, 400–404.

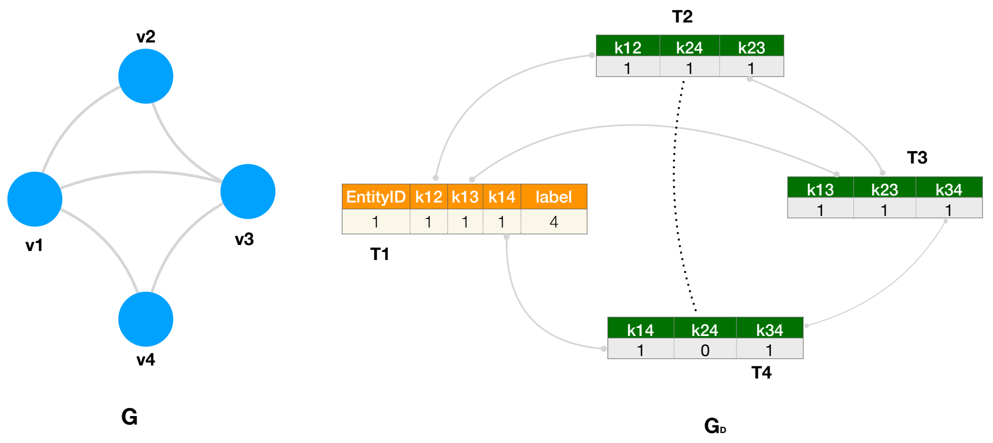


Figure 6: A reduction from Hamiltonian cycle problem to the problem finding the optimal joining path for engineering features from relational data for a given predictive analytics problem.

Stephen Muggleton and Luc De Raedt. 1994. Inductive Logic Programming: Theory and Methods. *JOURNAL OF LOGIC PROGRAMMING* 19, 20 (1994), 629–679.

Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. 2016. Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016 (GECCO '16)*. ACM, New York, NY, USA, 485–492.

Matic Perovšek, Anže Vavpetič, Janez Kranjc, Bojan Cestnik, and Nada Lavrač. 2015. Wordification: Propositionalization by unfolding relational data into bags of words. *Expert Systems with Applications* 42, 17 (2015), 6442–6456.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *NIPS 2012*. 2960–2968.

C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. In *Proc. of KDD-2013*.

Appendices

A NP-hardness proof

Proof. Problem 1 is an optimization problem, the decision version asks whether there exists a solution such that $L_{P,F,g}(Y,\hat{Y}) = 0$. We prove the NP-hardness by reducing the given decision problem to the *Hamiltonian cycle problem* which was well-known as an NP-Complete problem as discussed in Karp (1972). Given a graph $G(E,V)$, where E is a set of undirected edges and $V = \{v_1, v_2, \cdots, v_n\}$ is a set of n nodes, the Hamiltonian cycle problem asks if there exists a cycle on the graph such that all nodes are visited on the cycle and every node is visited exactly twice.

Given an instance of the Hamiltonian cycle problem, we create an instance of the feature learning problem with a relational graph as demonstrated in Figure 6. We assume that we have a database $D = \{T_1, T_2, \cdots, T_n\}$ with n tables, each table T_i corresponds to exactly one node $v_i \in V$. Assume that each table has only one row. For each pair of tables T_i and T_j (where $i < j$), there is a foreign key k_{ij} presenting in both tables such that the values of k_{ij} in T_i and T_j are the same if and only if there is an edge $(v_i, v_j) \in E$.

Assume that T_1 is the main table which has an additional label column with value equal to n. We also assume that all the keys k_{ij} have unique value in each table it presents which means that for each entry in T_i there is at most one entry in T_j with the same k_{ij} value and vice versa, we call such relations between tables one-one. Recall that the relational graph G_D constructed for the database D, where nodes are tables and edges are relational links, is a fully connected graph. Let p is a path defined on G_D and starts from the main table T_1. Because all the relations between tables are $one - one$, following the joining path p we can either obtain an empty or a set containing at most one element, denoted as J_p.

A cycle in a graph is simple if all nodes are visited exactly twice. Let assume F as the set of functions such that: if J_p is empty then $f(J_p) = 0$ and if J_p is not empty then $f(J_p) = k$ where k is the length of the longest simple cycle which is a sub-graph of p. Let g be the identity function. The decision problem asks whether there exists a path p such that $L_{p,F,g}(Y, \hat{Y}) = 0$ is equivalent to asking whether $g(f(J_p)) = n$ or $f(J_p) = n$ assuming g is an identity function.

Assume that $g(f(J_p)) = n$, we can imply that J_p is not empty and p is a Hamiltonian cycle in G_D. Since J_p is not empty, p is a sub-graph of G. Hence G also possess at least one Hamiltonian cycle. On the other hands, if p is a sub-graph of G and it is a Hamiltonian cycle, then since G is a sub-graph of the fully connected graph G_D we must have $g(f(J_p)) = n$ as well.

The given reduction from the Hamiltonian cycle problem is a polynomial time reduction because the time for construction of the database D is linear in the size of the graph G. Therefore, the NP-hardness follows.

□

B Proof of expressiveness theorem

First, we need to prove the following lemma:

Lemma 1. *A recurrent neural network $rnn(s, W, H, U)$ with linear activation is a set function if and only if $H = 1$ or $rnn(s, W, H, U)$ is a constant.*

Proof. Denote s as a set of numbers and $p(s)$ is any random permutation of s. A set function $f(s)$ is a map from any set s to a real-value. Function f is invariant with respect to set-permutation operation, i.e. $f(s) = f(p(s))$. For simplicity, we prove the lemma when the input is a set of scalar numbers. The general case for a set of vectors is proved in a similar way.

Consider the special case when $s = \{x_0, x_1\}$ and $p(s) = \{x_1, x_0\}$. According to definition of recurrent neural net we have:

$$\begin{aligned} h_t &= b + H * h_{t-1} + W * x_t && (2)\\ o_t &= c + U * h_t && (3)\end{aligned}$$

from which we have $rnn(s) = o_2$, where:

$$\begin{aligned} h_1 &= b + H * h_0 + W * x_0 && (4)\\ o_1 &= c + U * h_1 && (5)\\ h_2 &= b + H * h_1 + W * x_1 && (6)\\ o_2 &= c + U * h_2 && (7)\end{aligned}$$

In a similar way we can obtain the value of $rnn(p(s)) = o_2^*$, where:

$$\begin{aligned} h_1^* &= b + H * h_0^* + W * x_1 && (8)\\ o_1^* &= c + U * h_1^* && (9)\\ h_2^* &= b + H * h_1^* + W * x_0 && (10)\\ o_2^* &= c + U * h_2^* && (11)\end{aligned}$$

Since $rnn(p(s)) = rnn(s)$, we infer that:

$$U * (H - 1) * W * (x_0 - x_1) = 0 \qquad (12)$$

The last equation holds for all value of x_0, x_1, therefore, either $H = 1$, $W = 0$ or $U = 0$. The lemma is proved. □

Proof. According to Lemma 1, $rnn(s, W, H, U)$ is either a constant function or $H = 1$. Replace $H = 1$ to the formula of an RNN we can easily obtain equation 1. □

C Efficient implementation on GPU

Deep learning techniques takes the advantage of fast matrix computation capabilities of GPU to speed up its training time. The speed-up is highly dependent upon if the computation can be packed into a fixed size tensor before sending it to GPUs for massive parallel matrix computation. A problem with the network is that the structures of relational trees are different even for a given joining path. For instances, the relational trees in Figure 3 have different structures depending on input data. This issue makes it difficult to normalize the computation across different relational trees in the same mini-batch to take the advantage of GPU computation.

In this section, we discuss a computation normalization approach that allows speeding up the implementation 5x-10x using GPU computation under the assumption that the input to an r2n network are relational trees we set $\{D_{p_1}, D_{p_2}, \cdots, D_{p_q}\}$, where $D_{p_i} = \{t_1^{p_i}, t_2^{p_i}, \cdots, t_m^{p_i}\}$.

It is important to notice that $t_i^{p_k}$ and $t_i^{p_l}$ have different structure when p_l and p_k are different. Therefore, normalization across joining paths is not a reasonable approach. For a given joining path p_i, the trees $t_k^{p_i}$ and $t_l^{p_i}$ in the set D_{p_i} may have different structures as well. Fortunately, those trees share commons properties:

- they have the same maximum depth equal to the length of the path p_i
- transformation based on RNN at each depth of the trees are shared

Thanks to the common properties between the trees in D_{p_i} the computation across the trees can be normalized. The input data at each depth of all the trees in D_{p_i} (or a mini-batch) are transformed at once using the shared transformation network at the given depth. The output of the transformation is a list, for which we just need to identify which output corresponds to which tree for further transformation at the parent nodes of the trees.

D Model hyper-parameter tuning with Bayesian optimization

The quality of the prediction highly depends on various hyper-parameters present in the machine learning method. Since their optimal choice highly depends on the data, fixing them arbitrarily will provide suboptimal results. Therefore, we make use of Bayesian optimization Močkus (1975), the state-of-the-art approach for efficient and automated hyperparameter optimization Snoek et al. (2012).

For Bayesian optimization, the problem of hyperparameter optimization is formulated as a black-box function minimization problem. We define this black-box function f by

$$f \,:\, \Lambda \rightarrow \mathbb{R}\ , \tag{13}$$

where f maps a hyperparameter configuration $\boldsymbol{\lambda} \in \Lambda$ to its loss on the validation data set. The evaluation of f is a time-consuming step because it involves training our neural network on the training data set and evaluating it on the validation data set. However, minimizing f will provide us the optimal hyperparameter configuration.

$$\boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\lambda}\in\Lambda} f\left(\boldsymbol{\lambda}\right)\ . \tag{14}$$

Bayesian optimization efficiently minimizes this expensive black-box function sequentially. In each optimization iteration the function f is approximated by a Bayesian machine learning model, the surrogate model. We selected a Gaussian process with Matérn $5/2$ kernel as our surrogate model. This Bayesian model provides point and uncertainty estimates for the whole hyperparameter search space. According to a heuristic considering both mean and uncertainty prediction, the most promising hyperparameter configuration is evaluated next. We select expected improvement as our acquisition function Močkus (1975).

E Data preparation

The following steps are needed to turn the raw data into the format that our system requires:

1. For every dataset we need to create a main table with training instances. The training data must reflect exactly how the test data was created. This ensures the consistency between training and test settings.
2. Users need to explicitly declare the database schema.
3. Each interested entity is identified by a key column. We added additional key columns to represent those entities if the keys are missing in the original data.

It is important to notice that, the first step is an obligation for all Kaggle participants. The second step is trivial as it only requires declaring the table column's special types and primary/foreign key columns. Basic column types such as numerical, Boolean, timestamps, categorical etc., are automatically determined by our system. The last step requires knowledge about the data but time spent on creating additional key columns is negligible compared to creating hand-crafted features.

Grupo Bimbo participants were asked to predict weekly sales of fresh bakery products on the shelves of over 1 million stores across Mexico. The database contains 4 different tables:

- *sale series*: the sale log with weekly sale in units of fresh bakery products. Since the evaluation is based on Root Mean Squared Logarithmic Error (RMSLE), we take the logarithm of the demand.
- *town state*: geographical location of the stores
- *product*: additional information, e.g. product names
- *client*: information about the clients

The historical sale data spans from week 1-9 while the test data spans from weeks 10-11. We created the main table from the sale series table with data of the weeks 8-9. Data of prior weeks was not considered because there was a shortage of historical sales for the starting weeks. The main table has a target column which is the demand of the products and several foreign key columns and some static attributes of the products.

Coupon Purchase participants were asked to predict the top ten coupons which were purchased by the users in the test weeks. The dataset includes over one year of historical logs about coupon purchases and user activities:

- *coupon list*: coupon's info: location, discount price and the shop
- *coupon detail*: more detailed information about the coupons
- *coupon area*: categorical information about the coupon types and its display category on the website
- *coupon visit*: historical log about user activities on the coupon websites. User and coupon keys are concatenated to create a user-coupon key that represents the user-coupon pair which is the target entity of our prediction problem.
- *user*: demographic information about the users
- *prefecture*: user and coupon geographical information

We cast the recommendation problem into a classification problem by creating a main table with 40 weeks of data before the testing week. To ensure that the training data is consistent with the test data, for each week, we find coupons with released date falling into the following week and create an entry in the main table for each user-coupon pair. We label the entry as positive if the coupon was purchased by that user in the following week and negative otherwise. The main table has three foreign keys to represent the coupons, the users and the user-coupon pairs.

KDD Cup 2014 participants were asked to predict which project proposals are successful based on their data about:

- *projects*: project descriptions, school and teacher profiles and locations. The project table is considered as the main table in our experiment as it contains the target column.
- *essays*: written by teachers who proposed the proposal as a project goal statement.

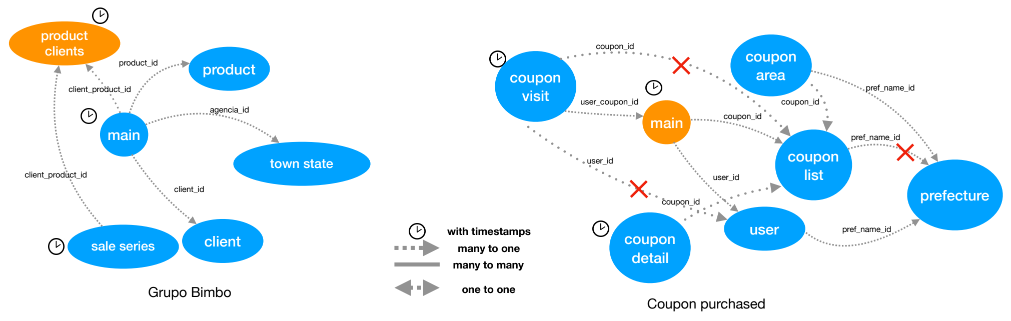


Figure 7: Tweaked relational graphs of the Kaggle data used in our experiments for DFS

- *resources*: information about the requested resources
- *donation*: ignored as no data for test set.
- *outcome*: historical outcome of the past projects. We add three missing key columns (school ID, teacher ID, school NCES ID) to the outcome table to connect it to the main table. This allows our system to explore the historical outcome for each school, teacher and school NCES ID.

The relational graphs of the datasets are shown in Figure 4. In all datasets, we experimented with the forward only graph traversal policy. In the given policy, the maximum search depth is always set to the maximum depth of the breadth-first search of the relational graph starting from the main table.

F Parameter settings

Table 5 reports the hyper-parameters used in our experiments. Since running full automatic hyper-parameter tuning and network structure search is computationally expensive we chose the size of the network based on our available computing resource. While the parameters related to optimization like the learning rate was chosen based on popular choice in the literature.

Table 5: Parameter settings for OneBM and the r2n networks

parameter	**value**
The number of recent values	10
Maximum joined table size	10^9
The number of highest correlated items	10
Min correlation	10^{-16}
Min info-gain	10^{-16}
Optimization algorithm for backprop	ADAM
Learning rate of ADAM	0.01
Initial weights for FC and feed-forwards	Xavier
Output size of FCs	10
The number of hidden layers in feedforward layers	1
The number of hidden layer size in feedforward layers	1024
RNN cell	LSTM
LSTM cell size	18
Max input sequence size	50
Early termination after no improvement on	25% training data
Validation ratio	10% training data

G Baseline method settings

DFS is currently considered as the state of the art for automation of feature engineering for relational data. Recently, DFS was open-sourced[3]. We compared OneBM and r2n to DFS (version 0.1.14). It is important to notice that the open-source version of DFS has been improved a lot since its first publication Kanter and Veeramachaneni (2015). For example, in the first version described in the publication there is no concept of temporal index which is very important to avoid mining leakages.

To use DFS properly, it requires knowledge about the data to create additional tables for interesting entities and to avoid creating diamond relational graphs because DFS doesn't support diamond loops in the graph and does not allow many-many relations. The results of Grupo Bimbo and Coupon purchase competitions were reported using the open-source DFS after consulting with the authors on how to use DFS properly on these datasets.

For the Bimbo and Coupon purchased data, the relational graphs shown in Figure 4 are not supported by DFS as they contain many-many relations and diamond subgraphs. Therefore, we tweaked these graphs to let it run under the DFS framework. Particularly, for Bimbo data the relation between main and series tables is many-many. To go around this problem, we created an additional table called product-client from the sale series table. Each entry in the new table encodes the product, client pairs. The product-client is the main table correspond to product-client pair. Since the competition asked for predicting sales of every pair of product-client at different time points, we created a cut-off time-stamp table, where each entry corresponds to exactly one cut-off timestamp. The new relational graph is presented in Figure 7. We run DFS with maximum depth set to 1 and 2 and 3.

For Coupon datasets, more efforts are needed to prepare a proper input for DFS because the original relational graph contains both diamond loops and many-many relations. The latter issue can be resolved by changing the connections as demonstrated in Figure 7. To avoid diamond loops, we need to delete some relations. We decided to delete the relations (marked with an X) in Figure 7. Alternatively, we also tried to delete the relation between the main and coupon-visit table but that led to much worse prediction than the given choice.

In general, the results are highly dependent on how we use the tool so for the KDD cup 2014, to have a fair comparison, we used the results in the original publication Kanter and Veeramachaneni (2015). For KDD Cup 2015 and IJCAI 2015, the competitions are closed for submissions so we couldn't report the results.

H Acknowledgements

We would like to thank Johann Thiebaut, Dr. Tiep Mai, Dr. Bei Chen, Dr. Oznur Alkan, Dr. Olivier Verscheure, Dr. Eric Bouillet, Dr. Horst C. Samulowitz, Dr. Udayan Khurana and Tejaswina Pedapat for useful discussion and support during the development of the project. We would like to thank Max Kanter for explaining us how to use DFS (featuretools) for the experiments.

[3]https://www.featuretools.com/