Journal of Computers Vol. 30 No. 2, 2019, pp. 55-67
doi:10.3966/199115992019043002005

Event-based Feature Synthesis: Autonomous Data Science Engine

Thirat Limsurut[1*], Warasinee Chaisangmongkon[2]

[1] Institute of Field Robotic, King Mongkut's University of Technology Thonburi, Bangkok, Thailand
terus.limsurut@mail.kmutt.ac.th

[2] Institute of Field Robotic, King Mongkut's University of Technology Thonburi, Bangkok, Thailand
warasinee.cha@mail.kmutt.ac.th

Received 15 December 2018; Revised 15 February 2019; Accepted 15 February 2019

Abstract. In this paper, we develop the autonomous data science tool to allow artificial intelligence to solve data science problems. Data exploration and feature extraction are the most time-consuming steps in a data science process. We develop an event-based feature synthesis algorithm, which can automatically recognize relationships between different entities and events presented in the data, extract important features using statistical and mathematical functions, and filter out only features of high importance. Our algorithm can generate features for data science problems with single and multiple data tables and use them to fit random forest classifier. To test the robustness of our autonomous data science engine (ADE) framework against well-established Deep Feature Synthesis (DFS) framework, we put our data science bot to test in public data science challenges and assess the usefulness of our feature sets. ADE can achieve high accuracy scores in several competitions, for example, it can predict targets at the accuracy as high as 89.5%, beating 74% of human participants in Employee Access Challenge (Kaggle, 2013). In MOOC dropout prediction (KDD 2015), features from ADE can augment features from DFS framework and improve accuracy from 85.3% to 86.3%.

Keywords: automated machine learning, classification, data science, feature engineering

1 Introduction

Data science is a process of discovering insights and predicts future events from historical data. The construction of predictive models in data science process can be roughly broken down into 4 stages: (1) data exploration, where data are visualized and data scientists perform statistical analysis to understand data. (2) predictive problem formulation, where data scientists ask what can be predicted to provide insights that drive business decisions. (3) feature engineering, where raw datasets are transformed into features to be used in predictive modeling. (4) model fitting and selection, where several predictive models are fitted and evaluated; the best model will be selected for application.

Let us look at a well-known data science problem as an example. A retail grocery store wants to understand customers' purchase behaviors and offers the right discount coupon to the right customers. The store manager might look into customers' purchase and coupon redemption history in to discover insights that drive their offer decisions. Data scientist may explore the data in visualization software and develop a model to predict whether a given customer will use a discount coupon for a given product. To this end, data scientists have to engineer some features that might influence offer acceptance; for example, whether the customer has purchased the product before, or how well the product sold compared to products of the same category. Then these features will be entered into a predictive model, such as decision tree, to foretell customer's behavior upon receiving coupon offering (to use or not to use coupon).

* Corresponding Author

This paper tackle feature engineering problems, which is one of the most time-consuming steps in data science pipeline. We propose a new way to create features automatically; improving on the algorithm of the previous research work of automated feature generation and evaluates our feature constructions benchmarked by previous work.

We observed multiple data science problems and how data scientists approach feature engineering. There are a few common thought processes, which are based on how software applications are often designed. First, each data record represents an event that occurs in the real world. Each table contains records of events of the same type. For example, *customer history table* contains records of customer acquisition events, while purchase transaction records contain information about single purchase events. Second, each event involves one or more actors or entities and their properties. For example, a customer acquisition event may involve a single entity (customer) along with properties of the entity, such as customer age, customer gender, etc. A purchase transaction event involves the interaction between several entities (customers, products, coupons) and may contain properties of an entity (such as product categories and product brands). Third, each event may contain a dynamic property that changes from interaction to interaction. For example, at a given transaction, customer A may buy product B at a given price, while in another transaction where customer A interacts with product B, the price might change. Fourth, some tables will contain main entity, where each record is uniquely identifiable with entity ID, while some tables might not have main entity. For example, one would expect that in customer information database, each record corresponds to a single customer, so customer is the main entity of this table. While in transaction database, each transaction involves a customer, a product, and a coupon, so there is no main entity.

With this framework, we propose that every column in a database can be interpreted as one of the four types:

- Main entity ID – e.g. customer ID in *customer history table*.
- Entity ID – e.g. customer ID and product ID in *transaction table*.
- Entity properties – e.g. customer age and gender in *customer history table*.
- Event properties – e.g. product price in *transaction table*.

We further observed that data science problems often prescribe that data scientists predict an outcome or a property of an event. For example, predicting coupon usage would amount to predicting the outcome of the interaction between customer entity, product entity, and coupon entity. In order for data scientists to predict the outcome successfully, they need to consider characteristics of all participating entities. In the coupon usage example, data scientists may want to know whether customers spend a lot at the store or not, whether the coupon is a good offer, and whether the product is popular.

Based on this framework, we develop a method for computer to mimic human thought process during feature generation. Our paper is organized as followed. In section 2, we review related work, providing an overview of Deep Feature Synthesis [1] and other automated feature engineering [2-3], including feature selection algorithms [4] that inspired our research and auto-tune hyper parameter and model selection. Section 3 details the process of our framework and algorithm. In section 4, we explain the experiments used to test our algorithm in data science challenge problems and quantify the quality of our features compared to previous work and human data scientists. In section 5 and 6, we conclude our research and suggest future work.

2 Related Work

2.1 Autonomous Feature Extraction

Previous works have investigated automated feature engineering problems. One important work that influences our paper is Deep Feature Synthesis (or DFS for short). DFS views feature construction process as a set of functions we can apply to different data portion. There are 3 types of features described by DFS:

- Entity features (EFEAT), which are computed by so-called ‘transformative function’ for example transforming Date/Time feature into day-of-month, weekday, month-of-year features etc.
- Direct features, (DFEAT) which are features that are transferred from one table through direct relation. For example, if in *customer history table*, customers ID always belong to a segment ID, you can

transfer segment property to customer property.

- Relational features (RFEAT) which are features constructed by aggregating data from other tables that has many-to-one relationship follow the relation database. For example, when a customer ID appears on many records in *transaction table*, we can aggregate (sum, mean, max, count, etc. called aggregations function) with transactions and use the aggregated values as customer features.

Other systems offered different views and strategies on automated feature generation problem. For example, one button machine (or OneBM) [2] extended DFS by using depth-first search to find complex relational graph from relation between data tables, incrementally join the table and apply a set of transformative functions to different types of data. OneBM can realize complex data types such as time series and sequence of categorical values, and include pre-defined functions for each type, yielding useful features for prediction. Another example is the so-called relational recurrent neural network (R2N for short) [3], which apply a set of recurrent neural network to learn transformation functions and embedding features. R2N can deal with unstructured by embedding any data into a space of numeric vectors.

This paper explores a simple concept of event-based feature generation which exploits human-like intuition to build feature sets. Our approach is based on DFS, but less complex and more interpretable than OneBM.

2.2 Feature Selection

In most autonomous feature engineering systems, as massive number of features is generated, therefore feature selection is a crucial process to avoid overfitting and reduce memory usage. We utilized a software package called Boruta [4]. The algorithm implements wrapper method for feature selection, by creating 'shadow features' by shuffling values in each column and checking whether the real feature has higher importance than the shuffled feature, based on a random forest model [5]. Boruta is effective and simple as it requires no parameter tuning. Boruta will finally outputs ranks of all features, and we can simply eliminate non-important features using a rank threshold.

2.3 Automated Machine Learning

There are a wide range of algorithms that perform automated hyperparameter tuning such as Auto-Weka [6] and Auto-Sklearn [7]. These tools apply tree-based Bayesian optimization methods [8] to search for the space of learning algorithms and optimize their hyperparameter settings. Another interesting approach is tree-based Pipeline Optimization Tool (TPOT) [9], which is based on genetic programing (GP) [10]. First, TPOT splits data to training and validation data for cross-validation [11]. Next, TPOT applies GP algorithm to generate random tree-based pipelines. For each generation, the algorithm tries to select the top pipelines, maximizing classification accuracy and minimizing the number of operators. After that, TPOT copies the selected pipelines to create the next generation of pipelines. The process is repeated until TPOT reaches the limit number of generations.

3 Implementation

Our Autonomous Data Science Engine (ADE) follows a typical data science workflow as shown in Fig. 1. The process starts from exploring the data to decipher data structure (Data Explorer), following by transforming raw data into more generic data form (Data Preprocessor), and automatically generating massive amount of features (Feature Extraction). Then, we eliminate redundant features (Feature Selection) then use the remaining features to predict the target (Prediction).

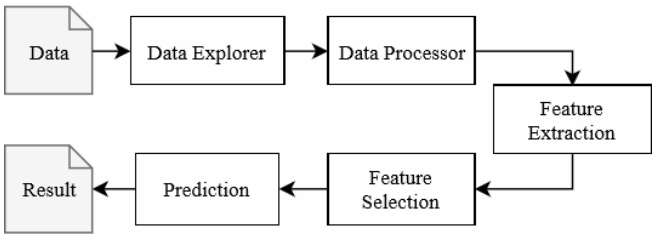

Fig. 1. The basic workflow of ADE

3.1 Data Explorer

Data exploration consists of 4 sub-processes: variable type confirmation, data type recognition, entity recognition, and table relationship mapping.

Variable Type Confirmation. Most data science software libraries can detect variable types automatically during data reading process. These automatically generate data types are often wrong or unfitted to the predictive analytic problem at hands, for example integer variable can be mis-encoded as floating if the number have floating point although it's zero. We implemented a rule-based algorithm to confirm data type for every single column. In our scheme, columns can be classified into 5 variable types: integer (Int), real numbers (Float), string (Object), Boolean and Date/Time. Fig. 2 displays the algorithm we used for variable type confirmation.

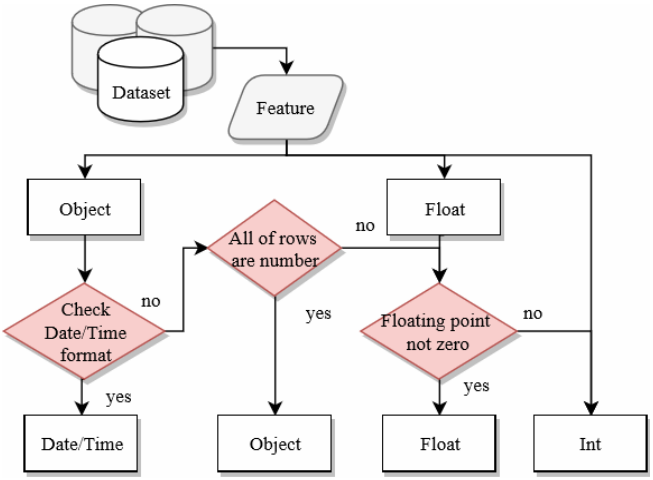

Fig. 2. Variable type confirmation algorithm

Data Type Recognition. After variable types are controlled, we then need to classify each column based on their characteristics, e.g. how they should be considered in a data science problem. We classify each column into 6 types: Numeric, Date/Time, Boolean, Category, and Text. Fig. 3 shows data type recognition algorithm

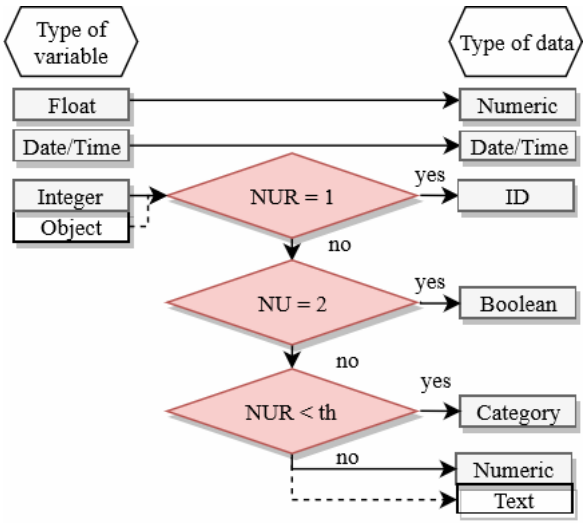

Fig. 3. Data type recognition algorithm. *NU* is number of unique values within a column. *NUR* is *NU* divided by the number of rows in the column. *th* is a threshold variable used to split between category and numeric/text data types (in this paper *th* = 0.25). Dotted line shows that Integer and Object variable types go through the same logical process which output Numeric and Text features respectively

Due to Float and Date/Time already get correct type in variable type confirmation, so we can determine Float and Date/Time variable features are Numeric and Date/Time data. In the same way with another data type, we can detect each type of data by condition of variable types.

Entity Recognition. Our event-based framework defines a role for each column in the table. The four possible roles are:

Main Entity (M). It is any column that satisfies 3 requirements: (a) data type must be ID; (b) each row in the column must be unique; (c) if there are more than two tables the set of IDs in the column must appear in another data table. For example, acquire valued shoppers dataset [18] in Fig. 4, customer-offer history contains customer ID and repeater label (the target labels that indicate which customer are highly probability to repeat purchase). ADE define customer ID is a main entity in this table because each record corresponds to a unique customer, but customer ID is not the main entity in transaction table, since the customer ID column there is not unique. Not every table will possess a main entity.

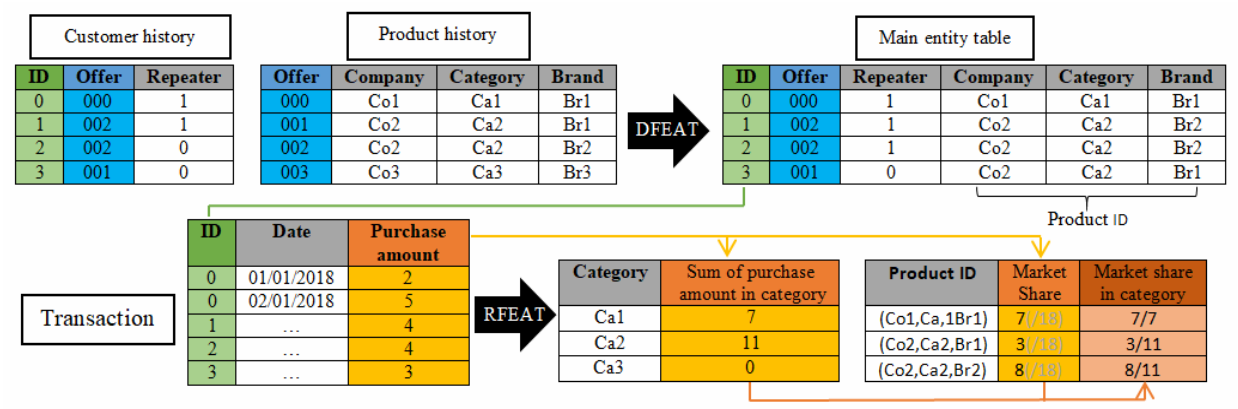

Fig. 4. The example of *Main Entity Table* which generated by direct features (DFEAT) and generate features by Relation between tables (RFEAT). For example, "Sum of purchase amount in category" was created by summation (aggregation function) of purchase amount in each group of Category from *Main Entity Table* like a "Market share" that grouped by Product ID and "Market share in category" which was generated from devided (mathematic function) "Market share" by "Sum of purchase amount in category"

Entity (E). It is similar to main entity in that the column must have ID data type, though ID must not be all unique. For example, customer ID in transaction table is an entity, not main entity.

Entity Property (PM). It is any column that conveys property of a main entity. For example, birthdate is a property of customer ID. Each unique entity ID is associated to only one value of entity property, e.g. a customer ID cannot have two different birthdates.

Event Property (PE). It is other columns that are not main entity, entity or entity properties. The values in these columns often change event by event. For example, even when customer ID and product ID are the same in two records, purchase price can change.

Our algorithm finds main entities and entities first before classifying the remaining columns as entity properties or event properties.

Table Relationship Mapping. For each pair of data tables D_i and D_j, this process will determine a value *R(i, j)* which classify relationship between two tables into forward (FW) versus backward (BW) relationships. Two data tables have a forward relationship if their linked keys have one-to-one or one-to-many relationship. For example, in Fig. 4, *product history table* has forward relationship with *customer history table* since the two tables are linked on Offer ID with one-to-many relationship. On the other hand, *transaction table* has backward relationship with *Main Entity Table*, as they are linked by customer ID with many-to-one relationship. Table 1 shows the process of determining R(i, j). Table relationships determined in this step will dictate the table merging process in the feature extraction step.

Table 1. Pseudo code for generate relation

Pseudo code: Generating relation table
1. **Function** Make_Relation_Table (D_i, D_j)
2. $K = F_i \cap F_j$
3. if $K \neq \varnothing$ then
4. **if** ($M_i \subset K$) **then**
5. *R(i, j)* = FW
6. **else if** ($M_j \subset K$) **then**
7. *R(i, j)* = BW

Where D_i is the data table in the dataset, F_i is the set of all feature names in D_i, M_i is the set of main entities in D_i and *R(i, j)* is relation between data table i and j.

3.2 Data Preprocessing

We applied standard data preprocessing procedures such as missing value imputation and data standardization. We eliminate columns with more than 10% missing values. Then we standardize all columns of Date/Time (using yy/MM/dd for dates and hh/mm/ss for times), Boolean (using true and false), and Categorical (encoding category labels as integers) data types.

3.3 Feature Extraction

The feature extraction process can be described as 3 separate sub-processes. (1) *Main Entity Table* generation. (2) Feature generation using transformative, aggregation, and mathematical functions. (3) On-the-fly feature selection. The feature extraction workflow is shown in Fig. 5.

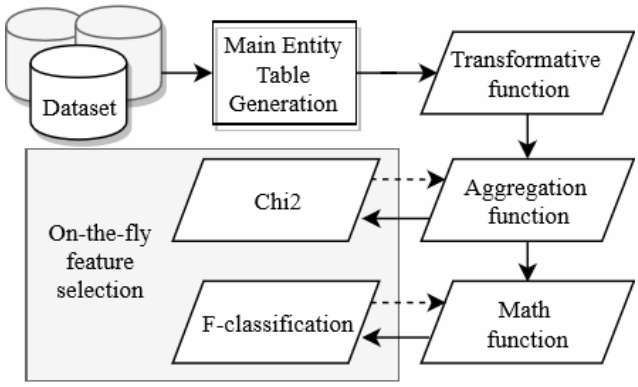

Fig. 5. The method in feature extraction process

Main Entity Table Generation. To mimic how a data scientist would approach a given problem, ADE must first orient itself to the data science problem at hands. ADE will find a *Main Table* which is the table where target entity and target labels are defined. Going back to acquire valued shoppers prediction example as depicted in Fig. 4, ADE would identify *customer history table* as the *Main Table* because this table consist of the entity (customer ID) would constitute target entities and repeater label column will be flagged as prediction target.

Once the *Main Table* is located, we merged tables with forward relationship with *Main Table* to form *Main Entity Table.* This merging process is similar to the approach proposed in Deep Feature Synthesis, ensuring that static properties of all entities are gathered in a single table for the feature generation process. For example, in Fig. 4, the *Main Entity Table* is constructed by merging *customer history table* with *product history table*. Other tables will be flagged as *Event Table*.

Feature Generation. After the *Main Entity Table* has been constructed, we are ready to generate some features. In the feature generation process, we apply 3 types of functions to the dataset.

Transformative Functions. These functions transform a feature in the Main Entity Table and create new feature(s). For example, from a Date/Time we can transform to Day, Month, Year, Weekday, Hour, and Minute. The set of transformative functions we used are included in Table 2.

Table 2. Aggregation function in each type of data

Type of data	Transformative function
Date/Time	day, month, year, hour, minute, weekend, weekday
Text	length of sentence, number of words, number of importance words

We apply pipeline of text mining [12] to extract feature in text data type. Where number of importance words is the new function which we create to deal with Text data type in general form such as description. This function will count the most frequency N words in features to transform features in Text data type.
Aggregation Functions. The power of event-based approach lies in the choice of aggregation procedures. In our coupon offer prediction example shown in Fig. 4, a human data scientist might aggregate amount of purchase in each product category as this feature would convey the power of each coupon and help predict whether the coupon would get used. For ADE to capture those types of features, we applied aggregation process as depicted in Fig. 6. The algorithm merges other tables with the Main Table. Then, the algorithm will take the merged table, group records by entities and categorical variables in the Main Entity Table and aggregate the remaining features with a set of aggregation functions depending on the data types of those features. The set of aggregation functions we used are included in Table 3.

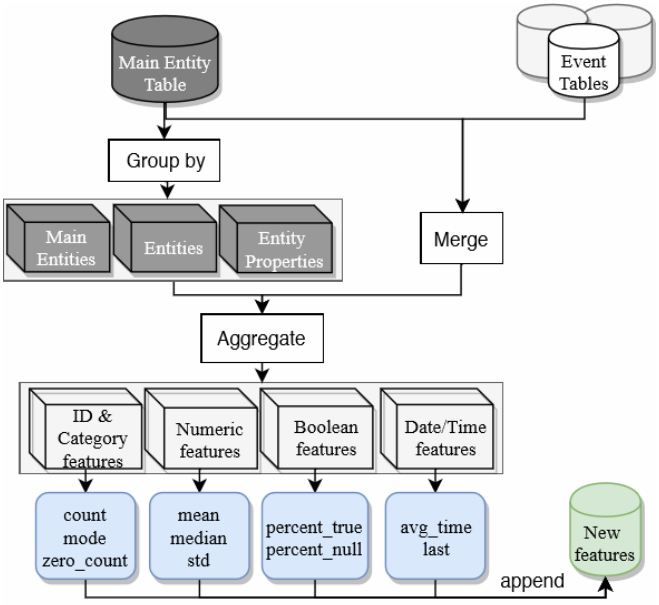

Fig. 6. The flow chart to generate features. Let's start with grouping main entities, entities and entity properties in *Main Entity Table* then aggregate with features in each type of data (gray box) by aggregation function (blue block)

Table 3. Aggregation function in each type of data

Type of data	Aggregation function
Numeric	mean, min, max, sum median, mode, standard deviation, zero-count
Categorical	count, mode, number of unique, zero count, last
Boolean	percent true
Date/ Time	average time, last

In this process, we will generate a large number of new features. The number of features generated can be determined by a few variables: x is the number of features in *Main Entity Table* which used for grouping operations; x_{type} is the number of features that passes through a given aggregation function in each type; fn_{type} is the number of aggregation functions for each type; g_{max} is maximum number of

element for grouping (for example, if algorithm can group by 2 categories, g_{max} would equal to 2). The number of features created from each type, x'_{type} is given by:

$$x'_{type} = \left(\frac{x}{g_{\max}} \right) \times x_{type} \times fn_{type}. \quad \textbf{(1)}$$

Mathematical Functions. In our coupon offer prediction example shown in Fig. 4, a human data scientist might think to create another feature which captures market share of each product in its own category. This is accomplished by dividing two aggregated features (sum of purchase amount grouped by product divided by sum of purchase amount grouped by product category). To capture these features, we create additional features by applying addition, subtraction, multiplication, and division to every pair of numeric features after the aggregation processes. After this operation, the number of features generated can by computed similarly to Equation 1 by setting $g_{max} = 2$ and setting fn_{type} to be the number of mathematical functions used.

On-the-fly Feature Selection. As a large number of features are generated, we need to eliminate some features right after its creation to save storage space and computational load. To eliminate useless features rapidly, we computed Chi-Squared value for discrete features and ANOVA f-value for numerical features and sort features based on scores. Then, the algorithm will compute Pearson correlations among features from highest scores to lowest scores. For any feature pairs with higher than 0.7 correlations, the feature with lower score will be eliminated. On-the-fly feature selection is performed after each loop of aggregation (once after features are generated by grouping 1 column and again after grouping by 2 columns) and after mathematical feature generations if the number of features generated are more than 150 features.

3.4 Modeling

Lastly, all generated features will be cleaned before the modeling process. For numeric features, we impute missing values by zeros. For categorical and Boolean features, we create a new category for records with missing values. For Date/Time features, we simply eliminate the original feature and rely solely on the transformed features. We assign these steps were done after aggregation process, since including it prior may propagate errors through the aggregation process. After all features are processed we repeated on-the-fly feature selection process and, to reduce the number of features further, we utilized a software package called Boruta to perform wrapper feature selection method with random forest ensemble for all experiments and select features with highest importance. The final numbers of features range from 30-100 depending on the size of dataset because we need to use number of features from ADE close to DFS in experiment. Then the features are fed into random forest using scikit-learn library with number of estimators = 1000.

4 Autonomous Data Science Engine: Experiment

We assess the usefulness of features generated from ADE by putting it to test in 5 public data science competitions which are all binary classification problems. For each competition, we pass the training dataset into ADE to perform variable type confirmation, data type recognition, entity recognition, table relationship mapping, data preprocessing, feature extraction, feature selection, and random forest model fitting. Then we applied fitted process and models to the test dataset.

For small prediction problems with only one table in the dataset, we selected **Employee Access Challenge (Kaggle, 2013)** and **Santander customer satisfaction (Kaggle, 2016)**. For larger problems with multiple tables in the datasets, we selected **Facebook Recruiting IV Human or Robot (Kaggle 2015)**, **Donorchoose Application Screening (Kaggle 2018)**, and **MOOC dropout prediction (KDD cup 2015)**.

For each dataset, we made small adjustments in ADE configurations due to computational limits. These configurations will be described as the results of each competition are presented.

4.1 Prediction Performance for Single-table Problems

Employee Access Challenge (Kaggle, 2013) [13]. Competitors predict whether employees should have access to resources using 9 features such as titles, roles, manager ID, etc.

All original features in this dataset are of categorical data type. Therefore, we perform one-hot encoding on original features and top 35 features generated from ADE using Boruta (rank threshold = 30) before random forest model fitting. After that we filtered features with Chi2 scores and keep only top 500 features and then pass it through Boruta to keep features with ranks from 1-200.

Santander Customer Satisfaction (Kaggle, 2016) [14]. Competitors predict whether customer is happy or unhappy with their banking experience from 370 customer features. Due to large number of features, we select 42 top features using Boruta (rank threshold = 40) before passing the dataset to ADE process.

Table 4 compares number of features and the accuracy scores between baseline and ADE. Fig. 7 shows performance comparison between ADE and human data science competitors.

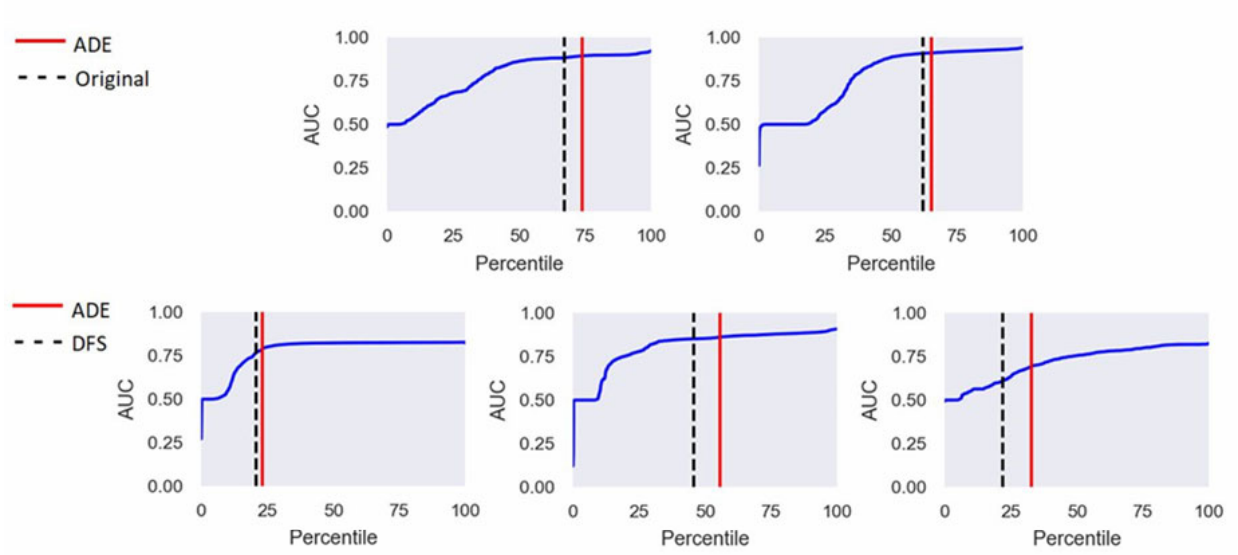

Fig. 7. The AUC scores and percentile of participant in single-table dataset (top): Employee Access Challenge and Santander Customer Satisfaction and multi-tables dataset (bottom): Facebook Recruiting IV, Donorchoose application screening and KDD cup 2015. Where Red line is the score by ADE, black line in single-table dataset is the score using original features and in multi-tables dataset using DFS

Table 4. Summary of the score of prediction in 2 datasets from single-tables problems

Datasets / **Result**	Employee access challenge		Santander-customer-satisfaction	
	N	*S*	*N*	*S*
No synthesis (baseline)	201	0.8826	42	0.7680
Synthesis (ADE)	204	**0.8950**	49	**0.7907**

Where *N* is the number of selected features that are entered into random forest and *S* is the accuracy score.

ADE process produces better result than baseline; 1% improvement for Employee Access Challenge and 3% improvement for Santander Customer Satisfaction. However, these small improvements can result in dramatic improvement in competition ranking as shown in Fig. 7, especially in Employee Access Challenge, ADE boosted competition ranking from 64th percentile to 74th percentile. These improvements were achieved with no model tuning, so better performance can be attained with more tuning.

4.2 Prediction Performance for Multi-table Problems

Facebook Recruiting IV Human or Robot (Kaggle, 2015) [15]. Competitors predict whether the auction participants are human or robot from bid data, device, time, countries, and a series of behavioral logs.

Donorchoose Application Screening (Kaggle, 2018) [16]: Competitors predict whether project proposal will be approved using metadata about project, teacher, and school and text project descriptions.
MOOC Dropout Prediction (KDD cup, 2015) [17]: Predict whether a student will drop out from an online learning platform using several databases from student info to course info and all online activities.

For multi-table problems, we can use DFS as a benchmark. We passed the datasets into ADE and DFS to extract features.

For each competition, we compare the prediction performance among baseline, ADE, and DFS feature sets. We also examine the difference in performance when adjusting g_{max} variable in the ADE feature generation process and assess the performance when ADE feature sets are combined with DFS feature sets. Table 5 summarizes the results of the experiments.

Table 5. Summary of the score of prediction in 3 datasets from multi-tables problems

Method \ Datasets		Facebook recruiting IV		DonorChoose application screening		MOOC dropout prediction	
		N	*S*	*N*	*S*	*N*	*S*
Original		11		18		13	
DFS		20	0.9094	30	0.6130	48	0.8531
ADE	(g_{max} = 1)	10	0.8539	49	0.6802	46	0.8596
DFS–ADE	(g_{max} = 1)	22	0.8907	30	**0.6929**	94	**0.8632**
ADE	(g_{max} = 2)	11	0.8882	49	0.6759	28	0.7960
DFS–ADE	(g_{max} = 2)	24	**0.9114**	25	0.6801	66	0.8520

The results of experiments show that although we cannot definitively conclude that ADE yields better feature sets than DFS, ADE feature set can always help enhance the accuracy of the model.

For DonorChoose application screening and MOOC dropout prediction problems, the accuracies are better when g_{max} = 1 compared to the cases where g_{max} = 2, although the features in the first case is a subset of the features in the latter case. This shows that model performance depends a lot on the feature selection process, since features generated from g_{max} = 2 processes should be dropped out if it worsens model prediction, resulting in similar performance for any value of g_{max}.

To understand how ADE and DFS features complement each other, we explore the correlations between two feature sets in DonorChoose application screening competition. The correlation distribution is shown in Fig. 8. The result shows that feature sets from ADE and DFS are rarely correlated, with correlation values peaking at zeros. This is one main reason why ADE features enhance model prediction.

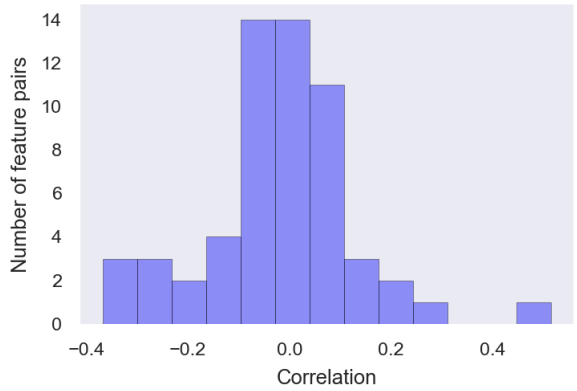

Fig. 8. Distribution of correlation values between DFS and ADE feature sets in DonorChoose application screening challenge

To understand how ADE and DFS features complement each other when they are merged, we examine the quality of features that DFS and ADE generate from DonorChoose application screening dataset (using g_{max} = 1 for ADE). For each feature, we assess feature importance values from random forest model, Chi2 scores (for categorical features) and f-values from ANOVA (for numerical features). Fig. 9 shows these scores ranked from the highest to the lowest, color-coded by feature origins (original feature sets, or DFS, or ADE). The results show that although features from DFS are ranked first in all

measurements, ADE contributes more features in the top rank, boosting the quality of the feature set when performing prediction.

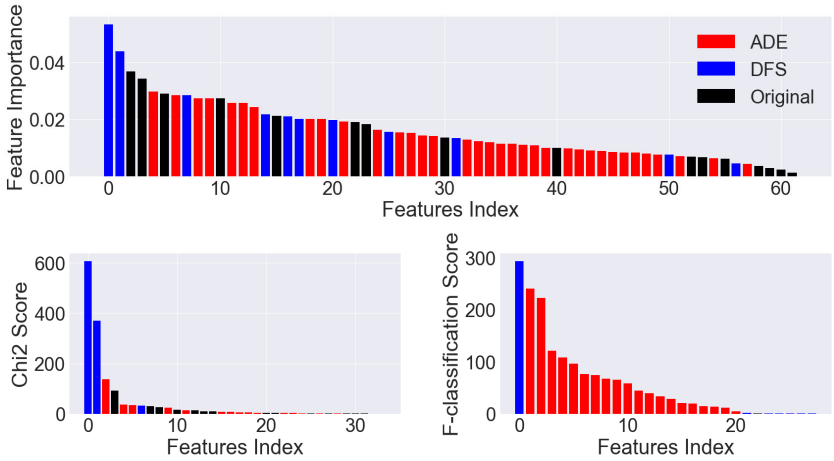

Fig. 9. Feature importance, chi2 and ANOVA f-values for each feature in Donorchoose application screening dataset

This result highlights one important difference between DFS and ADE. DFS generates features from grouping by entity ID only but does not consider group-by operations for entity properties like ADE. This could be the reason why DFS cannot achieve high accuracy in many of these experiments. For example, in DonorChoose application screening challenges, based on proposals that teachers submitted to receive support for needed resources for their class projects, the algorithm must predict the chance that the applications will be approved. In this prediction, the features that are ranked the highest from DFS are the quality of project description (whether your descriptions are common length or exceptionally detailed). Interestingly, the standard deviation of resource prices is also ranked high, indicating that projects that require variety of resources with some expensive items in the list often do well. These features are properties of the project, which is the main entity ID. In contrast, ADE can produce additional features that come from grouping other entity properties. For example, the top-ranked features from ADE reflect the quality of resource descriptions and teacher prefixes. These secondary features reflect additional information about the projects that help model make more accurate predictions.

5 Conclusion

We have presented an automated feature engineering framework that executes data science workflow autonomously from data exploration to model fitting. Our system improves upon the previous work in various ways. First, ADE can generate new features even for single-table problems, improving model accuracy compared to vanilla model fitting. Second, ADE executes group-by and aggregate operations beyond main entity level and create an uncorrelated feature sets to augment DFS framework. Third, ADE added many transformative functions that allow us to generate numeric features from text features.

The main challenge in this work is computational resource limit, as we need to create lots of features to find good ones. We handled this challenge by applying on-the-fly feature selection algorithm. Future works can address the problem with more robust feature selection algorithm and implement parallel feature extraction processes. Furthermore, better model performance can be achieved with an additional automated model tuning algorithm. We decided to omit that to get a clear evaluation of feature generation algorithm. In general, the use of automated tuning algorithm might exacerbate another key challenge with automated machine learning, which is overfitting. As feature selection and model selection process are done recursively algorithm is likely to overfit the validation dataset as well as the training set. Therefore, the fitting process requires dataset size to be large enough.

In conclusion, our work demonstrates a new approach to feature engineering that significantly improves the previously presented framework and allows organizations to tackle machine learning problems at lower lead time and lower cost. This work has made a significant step towards the development of an autonomous data scientist, by which great business and data-analytic challenges can

be solved with relatively low effort required from highly valued workforce members, freeing their time for greater emphasis on the human-centric and creative aspects of their endeavors.

Acknowledgments

The authors would like to thank The Asahi Glass Foundation for the support. This work was partially supported by the Natural Science Foundation.

References

[1] J.M. Kanter, K. Veeramachaneni, Deep feature synthesis: towards automating data science endeavors, in: Proc. 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA), 2015.

[2] H.T. Lam, J.-M. Thiebaut, M. Sinn, B. Chen, T. Mai, O. Alkan, One button machine for automating feature. <http://arxiv.org/abs/1706.00327>, 2017.

[3] H.T. Lam, T.N. Minh, M. Sinn, B. Buesser, M. Wistuba, Neural feature learning from relational database, <http://arxiv.org/abs/1801.05372>, 2018.

[4] M.B. Kursa, W. R. Rudnicki, Feature selection with the boruta package, Journal of Statistical Software 36(11)(2010) 1-8.

[5] L. Breiman, Random forests, Machine Learning 45(1)(2001) 5-32.

[6] C. Thornton, F. Hutter, H.H. Hoos, K. Leyton-Brown, Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms, in: Proc. 2013 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2013.

[7] M. Feurer, A. Klein, K. Eggensperger, J.T. Springenberg, M. Blum, F. Hutter, Efficient and robust automated machine learning, in: Advances. 2015 Neural Information Processing Systems (NIPS), 2015.

[8] E. Brochu, V.M. Cora, N.d. Freitas, A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. <http://arxiv.org/abs/1012.2599>, 2010.

[9] R.S. Olson, J.H. Moore, TPOT: a tree-based pipeline optimization tool, in: Proc. Workshop. 2016 Automatic Machine Learning, 2016.

[10] W. Banzhaf, F.D. Francone, R.E. Keller, P. Nordin, Genetic Programming: An Introduction, Morgan Kaufmann, San Francisco, 1998.

[11] P. Refaeilzadeh, L. Tang, H. Liu, Cross-validation, in: L. Liu, M.T. Özsu (Eds.), Encyclopedia of Database Systems, 2009, pp. 532-538.

[12] S. Vijayarani, J. Ilamathi, Nithya, Preprocessing techniques for text minings- an overview, International Journal of Computer Science & Communication Networks 5(1)(2015) 7-16.

[13] Amazon, Amazon.com- employee access challenge. <https://www.kaggle.com/c/amazon-employee-access-challenge/>, 2013 (accessed 02.05.17).

[14] B. Santander, Santander Value Prediction Challenge. <https://www.kaggle.com/c/santander-value-prediction-challenge/>, 2018 (accessed 04.06.18).

[15] Facebook, Facebook recruiting IV: human or robot?. <https://www.kaggle.com/c/facebook-recruiting-iv-human-or-bot/>, 2015 (accessed 04.06.18).

[16] DonorsChoose.org, DonorsChoose.org application screening. <https://www.kaggle.com/c/donorschoose-application-screening/>, 2018 (accessed 21.06.18).

[17] XuetangX, KDD Cup 2015- predicting dropouts in MOOC. <http://kddcup2015.com/information.html/>, 2015 (accessed 21.06.18).

[18] Acquire valued shoppers challenge. <https://www.kaggle.com/c/acquire-valued-shoppers-challenge/>, 2014 (accessed 15.11.16).