FTE: a Scalable Feature Transformation Engine for Machine Learning

Biruk B. Gebremariam
SAS Institute Inc.
Biruk.Gebremariam@sas.com

ABSTRACT

Data preprocessing and feature engineering are the most critical and time-consuming steps in a practical data science workflow. Due to the lack of generic solutions that apply across the board (to most data sets), data scientists apply these steps manually and iteratively. This manual, iterative process creates a bottleneck: a bottleneck in terms of both the data scientist's time and in the inefficient use of computing resources, especially in a cloud setting. Automated feature engineering strives to address these challenges by providing tools that essentially relieve the data scientist from these time-consuming steps. However, the most effective data preprocessing and feature engineering steps are problem specific, hence require domain expertise. As a result, automated feature engineering tools can either fail to provide good performance or tend to be computationally very demanding as they search for effective transformations in an exponentially large feature transformation and generation space. In this paper, we complement such tools by taking an orthogonal approach: we present a feature transformation engine, henceforth called FTE, and its compact feature transformation language called FTL. The FTL is a highly composable and expressive feature transformation language that can be used to construct first and higher order interpretable features. The FTE's execution engine is a scalable feature transformation engine that enables the data scientist to implement efficient data preprocessing and feature engineering workflow. We highlight this by recommending a specific workflow that best utilizes the FTE, and contrast this with a typical data preprocessing and feature transformation workflow. Finally, we demonstrate the FTE's efficiency and scalability with its sublinear-scaling with the number of generated features, and discuss how it accelerates data scientist productivity.

CCS CONCEPTS

• **Computing methodologies → Machine learning approaches**; **Machine learning algorithms**.

KEYWORDS

Feature Transformation; Feature Engineering; Automated Feature Engineering; Interpretable Models; Distributed Computing

ACM Reference Format:
Biruk B. Gebremariam. 2029. FTE: a Scalable Feature Transformation Engine for Machine Learning. In *KDD '20: The 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 21–26, 2020, San Diego, CA, United States.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

As the first, and perhaps most critical, step in machine learning and predictive modeling, data scientists spend a considerable amount of time preparing data in a form that is readily consumable by their predictive algorithms of choice (Ruiz [23]). These data preparation steps include data wrangling to setup the initial analytical base table, and a host of data preprocessing and feature transformation and generation steps. Some of these steps are imputation, outlier detection and treatment, nonlinear function transformation, mapping categorical variables to a continuous scale, discretization, categorical grouping, and miscellaneous interaction feature generation techniques [1].

Excepting toy problems, the composition of the most effective feature transformation steps depends on the particular modeling task at hand, and in particular, on the chosen predictive algorithm. This makes the selection of the most effective feature transformations and predictive algorithms a combinatorial problem that requires the exploration of a large feature transformation space. This problem is more pronounced in modern datasets due to their complexity and the significant data quality issues that commonly occur in them. The complexity and data quality issues in modern datasets are mostly due to the occurrence of a significant number of variables with one or more of these statistical profiles: irrelevance to the target concept, high missing rate, high cardinality, bad statistical distribution characterized by very low entropy and highly skewed class distribution for categorical variables, and high skewness and/or fat fails for continuous variables. In addition, the high dimensionality of these datasets, along with the occurrence of a significant number of outliers and a large number of interacting variables, adds to the challenges of building highly predictive models for these datasets (Donohu [4]; Guyon et al. [9]).

The impact of data quality issues on classification and regression algorithms depends on the nature of the algorithms. For instance, tree based algorithms (Brieman [3]) are better at picking interactions and are insensitive to outliers. This is in contrast to generalized linear models (GLMs) (Nelder et al. [20]) which require interactions to be explicitly built into the feature set. On the other hand, tree based algorithms may struggle to pick linear trends which can easily be captured by generalized linear models. Several studies show

[1]Since there is no clear distinction between data preprocessing and feature transformation, where appropriate, we use feature transformation as an umbrella term to include both

that most machine learning algorithms benefit from data preprocessing and that the benefit depends both on the preprocessing and the algorithm (Kotsiantis et al. [15]). For instance, the GA2M model (Lou et al. [17]) adds feature interactions explicitly into a baseline GAM model and achieves state of the art performance. Faced with this wide array of problems and solutions, the data scientist usually spends a considerable amount of time performing a mix and match between feature transformations and predictive algorithms.

Automated feature engineering strives to address these challenges by providing tools that essentially relieve the data scientist from these time-consuming steps. Recently, there has been significant progress in this space. The Data Science Machine (Kanter et al. [10]) performs feature generation from relational datasets. ExploreKit (Katz et al. [11]) is a framework that implements iterative feature generation and evaluation steps, along with selection using a machine learning approach to predict the potential usefulness of generated features. In contrast, Cognito (Udayan et al. [13]) implements a hierarchical feature construction approach, and uses a greedy exploration strategy to explore the large transformation space. More recently, various authors are applying reinforcement learning techniques for automated feature engineering. Udayan et. al [12] cast the manual and iterative feature engineering process into a reinforcement learning problem based on a performance driven exploration of a transformation graph. While most of these approaches achieve promising results, they suffer from two main shortcomings. First, most of them focus narrowly on continuous variable transformations and have limited scope when it comes to data preprocessing and categorical variable transformations. As a result, the data scientist is expected to do some preprocessing of the initial noisy data prior to using these tools. More importantly, most of these approaches are black-box approaches that target increased predictive performance as their sole objective.

However, practical predictive modeling is based not only on performance metrics, but also on the business needs and regulations that apply to the business. Furthermore, the recent emphasis on model interpretability (Samek [24]) has made the construction of globally interpretable models of utmost importance (Rudin [22]). One of the effective ways to construct globally interpretable models is through feature transformations that bring the variables to representations that are more compatible with the chosen predictive algorithm. All these point to the need for the data scientist to have a closer control on the modeling workflow in general, and the data preprocessing and feature transformation steps in particular.

In the following sections, we describe the feature transformation engine (FTE) and its compact feature transformation language (FTL). The FTE increases data scientist productivity by taking an approach that is orthogonal to automated feature engineering. It provides the data scientist with a highly composable and expressive feature transformation language that can be used to construct first and higher order interpretable features. The FTE's scalable execution engine enables the data scientist to achieve faster feedback loops that result in more model iterations per unit time. In subsequent sections, we describe the FTE, its architecture, and its most effective workflow. Finally, we do performance comparison with a scikit-learn [21] based tool, discuss its sublinear-scaling, and conclude with future outlooks.

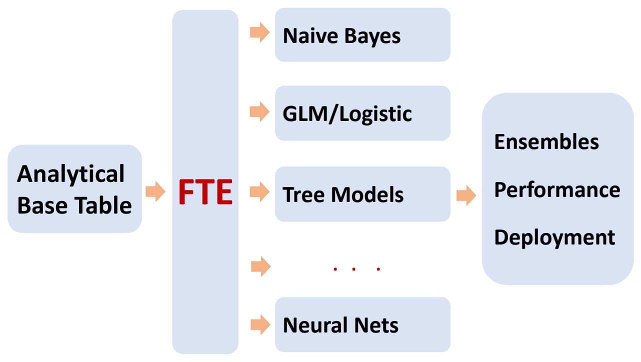

Figure 1: The FTE produces a single features table for downstream analytics.

2 FEATURE TRANSFORMATION ENGINE (FTE)

The FTE is a comprehensive, expressive and scalable feature transformation engine that executes user-specified feature transformations and produces a single table for downstream analytics. The FTE is available as the transform action in SAS Visual Analytics [25] and SAS Visual Data Mining and Machine Learning [26]. Figure 1 shows a typical usage of the FTE in a data science workflow.

2.1 FTE Architecture

Figure 2 shows a simplified architecture of the FTE. The basic building blocks of the FTE are transformation pipelines. A transformation pipeline is composed of input variables, a sequence of transformation operators, and a set of output features that are generated by the last transformation operator. The user defines transformation pipeline using the feature transformation language (FTL). The FTE compiles the FTL, deactivates invalid pipelines, and executes the remaining valid pipelines in an optimized manner to generate the final features.

2.2 Feature Transformation Language (FTL)

The FTL is an expressive and comprehensive feature transformation language that fits most of the data preprocessing and feature transformation needs of data scientists. There are two types of transformation pipelines:

(1) non-interaction pipelines: these are pipelines in which the number of input and output features is the same, and that each output feature is derived from a single feature through the application of a sequence of transformation operators
(2) interaction pipelines: these are pipelines in which each output feature is determined by the interaction of multiple features.

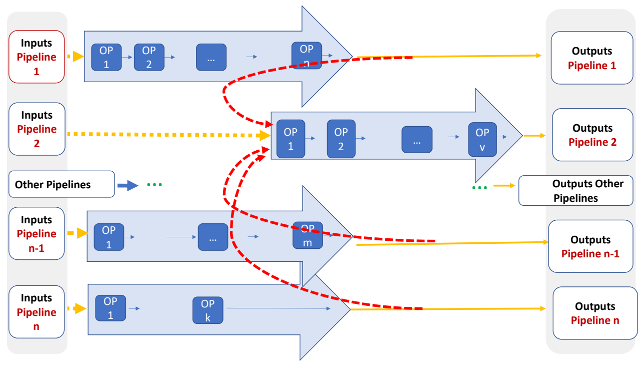

Figure 2: The architecture of the FTE: a multi-flow, expressive, and scalable feature transformation engine.

Table 1: Feature Transformation Single Input, Single Output (SISO) Operators

Operator	Input type	Output type
Impute	continuous, categorical	the input type
Outlier	continuous	the input type
Function	continuous	the input type
Map-interval	categorical	continuous
Hash	categorical	categorical
Datetime	datetime	indeterminate
Discretize	continuous	categorical
Cattrans	categorical	categorical

2.2.1 *Non-interaction Transformation Pipeline.*

A non-interaction transformation pipeline can be expressed as:

$$(\tilde{v}_1, \tilde{v}_2, ...\tilde{v}_n) = (\mathcal{T}^1(v_1), \mathcal{T}^1(v_2), .., \mathcal{T}^1(v_n)) \tag{1}$$

where v_is are the input variables, $\tilde{v}_i$ are the output features, and the $\mathcal{T}^1$ notation is adopted for uniform treatment with respect to p^{th} order interaction pipelines. The preceding equation should be understood as a vector equation where $\tilde{v}_i = \mathcal{T}^1(v_i)$. The transformation pipeline, $\mathcal{T}^1$, is defined by:

$$\mathcal{T}^1(v_i) = \text{Op}_1(\text{Op}_2(...(\text{Op}_m(v_i)))). \tag{2}$$

where Op_i denotes the sequence of transformation operators that compose the transformation pipeline. Figure 2 depicts a number of (n) user defined transformation pipelines. The pipelines labeled as "Pipeline 1", "Pipeline n-1", "Pipeline n" are non-interaction pipelines, while the one labeled as "Pipeline 2" is an interaction pipeline. The FTL defines two classes of transformation operators: SISO (single-input, single-output) operators, and MISO (multiple-input, single-output) operators. Only SISO operators can occur in non-interaction pipelines. Table 1 lists the available SISO operators along with their input-output measurement scales.

A full specification of the SISO operators requires the specification of arguments that unambiguously define their operations: $\text{Op}_i(\beta_1, \beta_2, ...\beta_n)$ where the β_j denote the arguments and n is the number of arguments for the operator. The value of n is operator dependent. In addition, arguments are defined in their order of precedence such that in cases where incompatibility arises, an argument can be overridden by arguments that precede it. These arguments bring two benefits: comprehensiveness and the ability to fine-tune pipelines. For instance, for the discretize operator, in addition to the well known arguments that specify the discretization method and the (maximum) number of bins, there are arguments that control finer aspects such as the fate of missing values. In the wider context of the end-to-end data science workflow, these arguments essentially act as additional hyperparameters. The following lists the most important arguments for the SISO operators:

1. Impute: β_1: imputation methods such as *mean, median, mode, random, ...*
2. Outlier: β_1: outlier detection methods such as *z-score, robust z-score, IQR, percentile,...* β_2: outlier treatment methods such as *trim, winsor, ...*
3. Function: β_1: function methods such as *standardize, center, range, power, ...* β_2: location estimates such as *mean, median, winsorized mean, Tukey's biweight,...* β_3: scale estimate such as *std, iqr, MAD, Tukey's biweight,...*
4. Map-interval: β_1: map-interval target encoding and unsupervised methods such as *(mean, max, min, WOE, probability)* encoding, and *count, freq...*
5. Hash: β_1: hashing methods such as *murmur3, ...* β_2: number of buckets
6. Datetime: β_1: datetime methods such as *year, month, day, leapyear, weekend, ...*
7. Discretize: β_1: discretize methods such as *bucket, quantile, dtree, rtree, MDLP, chimerge,...* β_2: number of buckets
8. Cattrans: β_1: grouping methods such as *label, rare, dtree, rtree, chimerge,...* β_2: number of buckets

Due to the different input-output characterstics of the SISO operators, the FTE can deactivate a transformation pipeline that is composed of incompatible operators. For instance, a pipeline that has Map-interval followed by Cattrans is deactivated.

2.2.2 *Interaction Transformation Pipeline.*

A p^{th} order interaction transformation pipeline is one where each output feature is determined by the interaction of p input variables or output features from other non-interaction pipelines. The FTL allows inputs to interaction transformation pipelines to originate either from the input dataset or from the outputs of non-interaction transformation pipelines. This is one of the key aspects of the FTL that give its strong expressiveness. Interaction transformation pipelines can be expressed as:

$$(\tilde{v}_1, \tilde{v}_2, ...\tilde{v}_m) = (\mathcal{T}^p(\tilde{v}_1^1, ..., \tilde{v}_p^1), .., \mathcal{T}^p(\tilde{v}_1^m, ..., \tilde{v}_p^m)) \tag{3}$$

where the components of the preceding vector equation are given by:

$$\tilde{v}_j = \mathcal{T}^p(\tilde{v}_1^j, ..., \tilde{v}_p^j) = \mathcal{T}^1(S(\tilde{v}_1^j, ..., \tilde{v}_p^j)) \tag{4}$$

where S is a MISO (multiple input, single output) operator that synthesizes multiple input variables (or features) into a single output feature. Note that the MISO operator (synthesizer) is the very first operator in interaction pipelines. The output from S is piped through the rest of the transformation pipeline which can be any T^1, essentially a non-interaction pipeline. In the $\tilde{v}_j^i$ notation, i stands for the i^{th} output feature from the interaction pipeline, and j is the j^{th} input feature to the MISO operator S. An example of interaction transformation pipeline is depicted in figure 2 where the pipeline denoted as "Pipeline 2" is a 4th order interaction pipeline taking a set of inputs from the original input dataset and the outputs from non-interaction pipelines "Pipeline 1", "Pipeline n-1", and "Pipeline n". Note that the FTL constitutes an identity, non-interaction transformation from the set of input variables in the input dataset that are piped directly into the interaction transformation pipeline.

The FTL groups the available MISO operators into three groups:

(1) continuous linear or nonlinear transformation functions: for a p^{th} order interaction pipeline, these operators (functions) map p continuous variables into a single continuous output.
(2) categorical combination function: for a p^{th} order interaction pipeline, this operator maps p categorical variables into a single categorical output using distinct-levels operation.
(3) continuous-categorical interaction operators: these operators take a group of continuous and interval variables, p variables in total, and map them into a single continuous output. The operations are primarily relational, group-by like operation where aggregate statistics are computed for the interval inputs, per distinct levels of the categorical inputs.

Here also, the FTE deactivates invalid interaction transformation pipelines. These include interaction pipelines for which any one of the input transformation pipelines is not active.

2.3 Implementation Novelties

The FTL, and its execution engine, the FTE, come with some engineering novelties.

2.3.1 Highly Composable and Expressive Language for First or Higher Order Interpretable Features. The FTL makes transformation operators the lowest granular level, and defines a highly composable syntax in which operators can be included or excluded in particular pipelines as needed. This abstraction of transformation pipelines in terms of a handful of transformation operators, that come with sensible defaults, puts only a small cognitive load on the user. Except for a few compatibility restrictions between consecutive transformation operators, transformation pipelines are fully user configurable. In addition, the fact that the inputs to interaction pipelines can be both the original input variables and the outputs from other non-interaction pipelines opens up the combinatorial feature transformation space for efficient exploration. In (Bachrach et al. [1]), the authors highlight the fact that features derived from the outputs of decision trees provide a significant performance lift for a logistic regression model. The FTL's interaction pipelines can be used to efficiently explore such higher order interaction features.

2.3.2 Multi-Flow. The FTE executes each transformation pipeline as an independent task. Variables can be inputs to multiple interaction and non-interaction transformations. In addition, both interaction and non-interaction pipelines can process multiple variables. This is the FTE's multi-flow capability. For instance, a group of variables can be operated with median-imputation followed by *log* transformation in one pipeline, and discretized with the MDLP discretization (Fayyad et al. [28]) in another. This multi-flow capability is an important contribution as different classification and regression algorithms have different feature transformation needs (Lui et al. [16]). This implies that a single FTE run can be used to produce a single table that contains the features that are to be consumed by a number of predictive algorithms. In addition to reducing the number of data reads, and more importantly data writes, this leads to a gain in data scientist productivity by freeing the data scientist from a cumbersome, error prone process.

2.3.3 Intelligent Pipeline Compilation and Execution: Pipeline Similarity and Variable Flow Analysis. The application of an operator in a transformation pipeline can be naively executed by computing the required fit parameters of the operator from the data that is output by the preceding operator. For instance, a pipeline that contains standardization followed by discretization can execute the standardization, generate an output dataset and pipe that to the discretization operator. Unfortunately, this naive execution of pipelines is very inefficient. The inefficiency becomes even more apparent when there is a need to execute multiple transformation pipelines.

The FTE performs intelligent pipeline compilation, with its pipeline similarity and variable flow analysis, in order to avoid inefficient executions. A pipeline's order is the number of operators that occur in a pipeline. Algorithm 1 is used to determine the maximum pipeline order, $p_{\max}$. In general, the number of data passes that is required by the FTE to execute the transformation pipelines is proportional to $p_{\max}$. This is followed by the construction of a pipeline similarity matrix, as outlined in Algorithm 2. This defines the maximum order up to which the operator sequence in two pipelines is similar. Armed with this, the FTE performs an iterative data pass in which transformation parameters are computed, in lockstep, for all pipelines whose order is greater than or equal to the currently executing order. These are the active pipelines for the current order. Since variables can be transformed in multiple pipelines, the FTE performs variable flow analysis (Algorithm 3) at the beginning of each data pass. The output from this analysis is the non-redundant set of computations that need to carried for the successful completion of the step across all active pipelines. As a result of this intelligent pipeline compilation and execution, the FTE avoid execution inefficiencies at two levels, namely, at the level of the individual pipelines, and across the pipelines. The impact of this on scalability is discussed in Section 4.

Algorithm 1 Maximum Pipeline Order

```
Require: The set of transformation pipelines $\mathcal{S}$
1: Index pipelines $\mathcal{T}_i \in \mathcal{S}$ according to some order.
2: Set $p_{\max} = 0$.
3: Set $i = 0$.
4: for $i < (|\mathcal{S}|)$ do
5:     if $|\mathcal{T}_i| > p_{\max}$ then
6:         Set $p_{\max} = |\mathcal{T}_i|$            ▷ Track the maximum order
7:     end if
8: end for
```

Algorithm 2 Pipeline Similarity Analysis

```
Require: The set of transformation pipelines $\mathcal{S}$
1:  Index pipelines $\mathcal{T}_i \in \mathcal{S}$ according to some order.
2:  Fill symmetric pipeline similarity matrix $\mathcal{M}$ with zero.
3:  Set $i = 0$.
4:  for $i < (|\mathcal{S}|)$ do
5:      Set $j = i + 1$
6:      for $j < (|\mathcal{S}|)$ do
7:          Set $k = 0$
8:          Set $m = \min(|\mathcal{T}_i|, |\mathcal{T}_j|)$.
9:          Set $p = 0$
10:         for $k < m$ do
11:             if $\mathcal{T}_i[k] == \mathcal{T}_j[k]$ then
12:                 Set $p = p + 1$            ▷ So far the same operators
13:             else
14:                 break
15:             end if
16:         end for
17:         Set $\mathcal{M}[i, j] = p$             ▷ Store into the matrix.
18:     end for
19: end for
```

Algorithm 3 Variable Flow Analysis

```
Require: The set of transformation pipelines $\mathcal{S}$
Require: The pipeline similarity matrix $\mathcal{M}$
Require: The analysis variable $v$
Require: The analysis order $p$
Require: $\mathcal{V}(\mathcal{T}_i)$: returns input variable set of $\mathcal{T}_i$
1:  $\mathcal{S}_p^v$: set of pipelines $\mathcal{T}_i \in \mathcal{S}$ and $v \in \mathcal{V}(\mathcal{T}_i)$ and $|\mathcal{T}_i| \leq p$
2:  Index pipelines $\mathcal{T}_i \in \mathcal{S}_p^v$ according to some order.
3:  Fill variable flow similarity vector $\mathcal{F}_p^v$ with zero.
4:  Set $i = 1$.
5:  for $i < (|\mathcal{S}_p^v|)$ do
6:      Set $j = 0$
7:      Set $k = 0$
8:      for $j < i$ do
9:          if $\mathcal{T}_i[p] == \mathcal{T}_j[p]$ then
10:             Set $k = j$                    ▷ So far the same operators
11:             break
12:         end if
13:     end for
14:     Set $\mathcal{F}_p^v[i] = k$            ▷ Store reference pipeline index.
15: end for
```

2.3.4 Single Write Execution Resulting in no Temporary Dataset Generation. In typical data science libraries such as scikit-learn (Pedregosa et al. [21]), pipelining of feature transformations is fairly easy. In fact scikit-learn comes with the pipeline module to easily do that. One can also simulate multi-flow operation through iterative, horizontal stacking of the outputs from the transformation pipelines. However, compared to the FTE, these come with a corresponding increase in the number of data reads, and more importantly, data writes that result in the generation of temporary datasets. The FTE implements a three-step process: compilation, fit (statistical parameter computation), and write. It is only during the single-write stage that all output features are written to the output data set.This strategy avoids the generation of temporary datasets, resulting in the FTE's significant performance gain compared with scikit-learn based implementations. This is discussed in more detail in 4.

2.3.5 Comprehensive Ranking Statistics. The fact that the FTE implements a highly composable language and has a multi-flow capability means that the data scientist can produce a practically unlimited number of highly correlated features in a single operation. As a result, feature selection (Guyon et al. [9]) becomes an important problem. Though feature selection is outside its scope, the FTE provides a host of feature ranking statistics that include mutual information, weight of evidence, Pearson correlation, Chi-square, G2, and many more. These are augmented with several feature rankings such as global ranking among all generated features and local ranking among features originating from the same transformation pipeline. The data scientist can make judicious use of these ranking statistics in order to select effective feature transformations.

2.3.6 Modern Data Sketch Algorithms. The MISO categorical transformation function defined in Section 2.2.2 can result in a proliferation of a large number of high cardinality categorical, intermediate variables. As a result, exact treatment of the distinct levels that result from this MISO operator can be memory intensive. The FTE uses the the hyperloglog++ algorithm (Heule et al. [8]) to estimate the resulting cardinalities, and dynamically switch to an approximate Misra-Gries (Misra et al. [19]) based operation if a user configurable distinct count limit is exceeded. During this process, only the most frequent levels are resolved, while the rest are grouped into a single, rare category. In practice this approximation does not introduce performance degradation as most interactions, among categorical variables, result in intermediate variables that are highly skewed in frequency distribution. This makes them amenable for approximate analysis of their frequency distribution.

2.3.7 Parallel and Distributed. The FTE, as a component in a SAS Viya platform [27], executes in a highly distributed platform. As a result, all of its key kernels including the estimation of all intermediate statistical parameters, the data read, and the single-write operations all execute in a parallel and distributed fashion.

3 HOW TO USE THE FTE?

The architecture of the FTE, and the syntax of the FTL, encourage the adoption of a particular approach for feature transformation. We first describe a typical workflow. We follow this with the description of our recommended approach, especially for effective utilization of the FTE on modern, high dimensional datasets.

3.1 A typical Data Preprocessing and Feature Transformation Workflow

Modern data sets can have hunderds to thousands of variables. We want to illustrate this with the kdd98 dataset [7]. The kdd98 dataset originates from a direct marketing campaign to maximize donation profits. The dataset has about 480 input variables and two target variables: a binary target for a classification problem, and a continuous target for a regression problem. The binary target indicates whether a donation was made or not, while the continuous target indicates the amount of the donation. Data exploration shows that the data suffers from significant data quality issues. See the next section for more details. After data exploration, the data scientist usually performs a sequence of preprocessing operations that typically include:

- identify noisy, relevant and irrelvant variables
- discard noisy and irrelevant variables
- impute missing values and/or create missing indicator variables
- identify highly skewed and fat-tailed continuous variables and either discretize them or perform transformations such as Box-Cox (Box et al. [2]) and Yeo-Johnson (Yeo et al. [29])
- group rare levels for categorical variables
- transform high cardinality variables and so on.

The data scientist generates the table that contains the final features only after stringing together a sequence of these transformations, a highly cumbersome endeavor. In addition, the fact that variables usually suffer from several data quality issues at the same time implies that the data scientist may need to try multiple transformations per variable to find effective transformations. This, when combined with model selection, leads to the aforementioned combinatorial explosion which introduces a significant bottleneck in the data science workflow.

3.2 Recommended Workflow with the FTE

The FTE, and its FTL, encourage a more granular approach to data preprocessing and feature transformation: a higher granularity than the process outlined in the previous section, while a much smaller granularity compared with automated feature engineering. Specifically, the multi-flow capability of the FTE fits with the need to transform variables in multiple ways. We start with the assumption that effective transformations for variables that share similar statistical profiles constitute a small set, however big the variable group is. We call this the *group-transform* assumption. This implies that the FTE can be used most effectively and it can provide effective transformations by defining a small set of transformations for each group of variables. One can generate meaningful and reliable variable groups with a novel high dimensional data exploration technique that we call AVAPT.

3.2.1 AVAPT. AVAPT stands for automatic variable analysis and grouping for pipelined transformation. It is a data exploration technique that takes the variables of a dataset, characterized by multi-dimensional statistical profiles, as the basic units of exploration. This is in contrast to traditional data exploration which emphasize data points as the basic units of exploration. The main output from AVAPT is a hierarchical variable grouping in which the groups are characterized by multi-dimensional statistical profiles. The statistical profiles include cardinality, Shannon and Gini entropies (Breiman [3]), class frequency skewness, and missing rate for categorical variables, and moment and robust skewness (Kim et al. [14]), moment and robust kurtosis (Kim et al. [14]), percentage outliers, coefficient of variation, and missing rate for continuous variables. Note that one can add mutual information between the variables and the target, for instance, to these statistical profiles. Additionally, interaction detection schemes (Hawkins et al. [6]) can be used

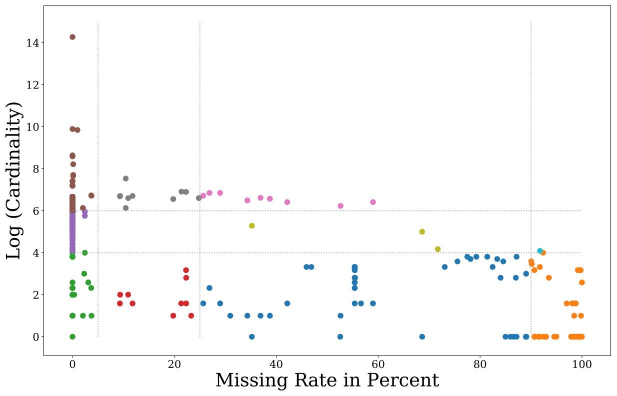

Figure 3: Projection of KDD98 categorical variables into (missing rate, log(cardinality)) subspace.

to define potentially interacting groups of variables for defining interaction transformation pipelines.

The first step in AVAPT is to discretize the values of these statistical profiles into *low*, *medium*, and *high* classes according to some predefined thresholds. This is followed by putting all variables having the same multi-dimensional, discretized profile into the same group.

Figure 3 shows the two dimensional projection, along the missing rate and cardinality (log), of the categorical variables in the kdd98 dataset. The horizontal and vertical lines are threshold lines that discretize the raw values into *low*, *medium*, and *high* classes. The missing rate has one more threshold line at about 90%. This is to denote variables that can safely be dropped as they have very high missing rate. It can be seen that there are a large number of categorical variables that have very significant amount of missing values, and/or very high cardinality.

Figure 4 shows the categorical variables projected along another pair of dimensions: the Shannon entropy and cardinality (log) subspace. Note that the plot limits the cardinality to the low and medium cardinality as we do not recommend computing statistical profiles for high cardinality variables. This is inline with standard practice in data science where high cardinality variables have a special set of transformations that are mostly based on target-encoding (D. Micci-Barreca [18]).

Figure 5 projects continuous variables across the missing rate and robust skewness (abs) dimensions. The procedure that we used for identifying variables as continuous or categorical denoted only 38 variables as continuous. Still, one can see an interesting distribution of these variables in the two extreme left and extreme right regions.

These various projections indicate the interesting structures that emerge from applying AVAPT to high dimensional datasets. These structures are the motivation for the *group-transform* assumption. Once the groups are determined, the next step is the enumeration of the set of transformation pipelines that are to be applied to each group. It is possible to recommend specific transformations. For instance, for a group containing continuous variables with a

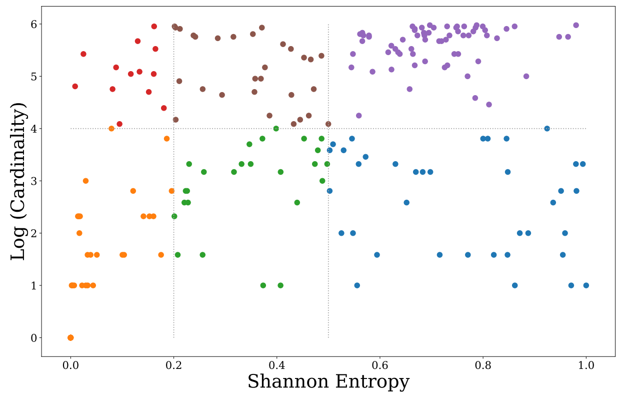

Figure 4: Projection of KDD98 categorical variables into (Shannon entropy, log(cardinality)) subspace.

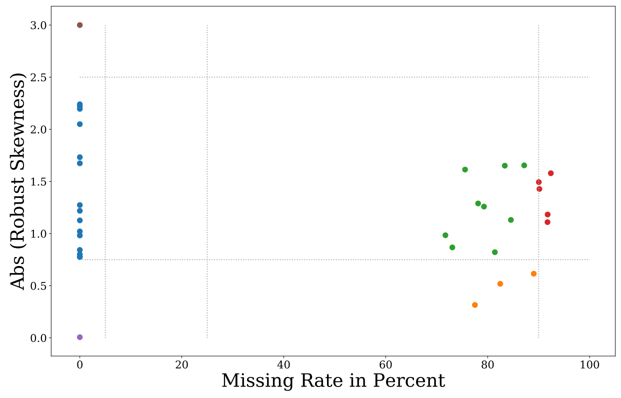

Figure 5: Projection of KDD98 continuous variables into (missing rate, robust skewness) subspace.

statistical profile: *(skewness=high, kurtosis=low, outlier=low, missing=high)*, two of the potentially effective transformation pipelines are a pipeline with median imputation and Box-Cox transformation and a pipeline that discretizes the variables with one of the available, preferably supervised, discretization methods (and a separate bin for the missing values). The enumeration of these potentially effective transformations is outside the scope of this work.

4 PERFORMANCE EXPERIMENTS: SUBLINEAR SCALING OF THE FTE

The scikit-learn pipeline module [21] is the closest that comes in design to the FTE. However, as it does not have a multi-flow capability, one needs to simulate the multi-flow capability of the FTE with a manual horizontal stacking of the features that are generated by the pipelines. In this performance test, we construct a simple transformation pipeline set that contains three pipelines: $\mathcal{S} = \{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3\}$, where $\mathcal{T}_1$ generates a missing indicator, $\mathcal{T}_2$ does mean imputation, and $\mathcal{T}_3$ does mode imputation for categorical variables. Note that we designed this simple test in such a way that we do not need to do horizontal stacking, thereby avoiding the

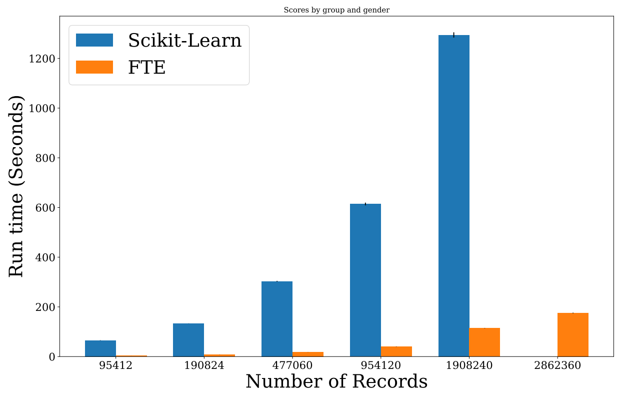

Figure 6: Performance comparison between FTE and scikit-learn.

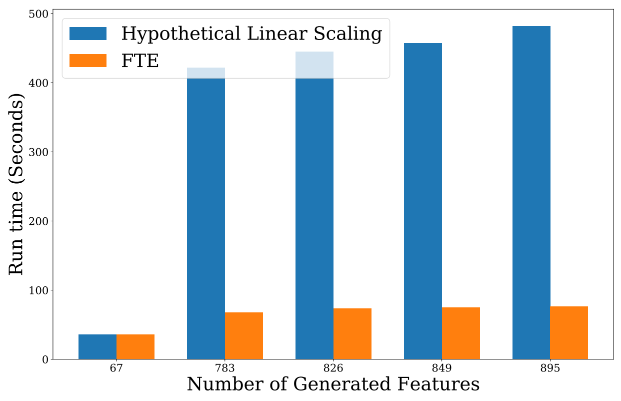

Figure 7: FTE's sublinear scaling with respect to the number of generated features.

additional overhead that can make the interpretation of the results more difficult. Also note that the actual code for the scikit-learn based run uses the open source library (feature engine).

The experiments were conducted on a PC with the following specs (intel i7-7700 CPU, and 32GB RAM). Figure 6 shows that the run-times on a PC with the following specs (in a single processor, 8 core setting, with 32GM RAM). See Section A for a notebook to reproduce these results (you will need access to SAS Visual Data Mining and Machine Learning [26]). For this experiment, we used the KDD98 dataset. As can be seen, the FTE, even for this very small pipeline set, is considerably faster than scikit-learn. In fact, for the last experiment where the number of records is about three million, scikit-learn runs out of memory (never completes), and hence the runtime is not reported.

To further illustrate the scalability of the FTE, we run it by increasing the number of pipelines that are executed in parallel. We do this by using the featureMachine action in SAS Visual Data Mining and Machine Learning [26]. FeatureMachine drives the FTE in a more granular and powerful fashion. Figure 7 shows the result of this experiment. This experiment also uses the KDD98 dataset, with about half a million sampled observations. The experiment

Table 2: Progressive Model Performance Improvement with the FTE

model	test AUC
model1	0.89 ± 0.03
model2	0.91 ± 0.05
model3	0.92 ± 0.03

compares the runtimes of the FTE with a hypothetical, linearly scaling runtime (with the number of generated features). As can be seen, the FTE achieves an impressive sublinear scaling. Delving deeper into the transformations requested by featureMachine, we note that many variables take part in several transformations. This is a natural problem for the FTE's optimized execution. Though there can be other contributing factors, this sublinear scaling behavior is a direct result of the FTE's intelligent compilation with pipeline similarity and variable flow analysis (Section 2.3.3).

Finally, Table 2 shows the progressive model performance (AUC) improvement of a simple FTE-based workflow. The experiments were conducted on SAS Visual Data Mining and Machine Learning [26]. We used the UCI [5] Adult dataset. We used 25% of the data for test, and the rest for training. We experimented with three models:

(1) model1: the first model is a baseline model that uses just the raw data
(2) model2: the second model defines two transformation pipeline sets. The first set applies to the continuous variables. We define $\mathcal{S}_1 = \{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3\}$ where $\mathcal{T}_1$ is a median imputation, $\mathcal{T}_2$ is a decision tree based discretization, and $\mathcal{T}_3$ is a discretization, followed by a map-interval WOE-encoding transformation. The second set applies to the categorical variables, $\mathcal{S}_2 = \{\mathcal{T}_4, \mathcal{T}_5\}$, where $\mathcal{T}_4$ is a label transformation and $\mathcal{T}_5$ is a map-interval WOE-encoding transformation.
(3) model3: the third model includes all the pipelines defined for the second model and added an interaction: $\mathcal{S}_3 = \{\mathcal{T}_6\}$. This pipeline is an all bivariate categorical-categorical interaction. For this, the continuous variables are initially transformed with a transformation pipeline that contains an equal-frequency discretization operator.

5 CONCLUSIONS

In this paper, we discussed the feature transformation engine (FTE), and its accompanying feature transformation language (FTL). The FTE complements automated feature engineering tools. Unlike automated feature engineering tools, the FTE keeps the data scientist at the center and addresses the key computational challenges that data scientists face in the data preprocessing and feature transformation and generation steps. The FTE achieves this through its highly composable and expressive feature transformation language that encourages the construction of first and higher order interpretable features, its multi-flow capabilities, and its single-write execution engine. In addition, the FTE's multi-flow capability encourages a different and, potentially, effective data exploration and feature transformation paradigm for modern, high dimensional datasets. This is based on the grouping of variables and defining a common set of transformation pipelines for variables in a group. We demonstrated how the grouping can be generated using a novel data exploration scheme, AVAPT, that is based on multi-dimensional statistical profiling. These all lead to fast feedback loops in data science workflows, thereby increasing data scientist productivity. Finally, we conducted a limited performance study that demonstrated the FTE's superior performance compared to scikit-learn's pipeline module. Further experiments demonstrated that the FTE scales sublinearly with the number of generated features. There are a number of developments that can be pursued in the future. These include the assessment of the impact of the FTE on the construction of globally interpretable models, the expansion of the operator space to include feature extraction operators (at the loss of interpretability), and large scale meta-learning studies that enumerate potentially effective transformations for the groups of variables generated by AVAPT.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous referees of KDD 2020 for their valuable comments and helpful suggestions.

REFERENCES

[1] Yoram Bachrach, Michal Kosinski, Thore Graepel, Pushmeet Kohli, and David Stillwell. 2012. Personality and patterns of Facebook usage. *Proceedings of the 4th Annual ACM Web Science Conference* (2012).

[2] George Box and David Cox. 1964. An Analysis of Transformations. *Journal of the Royal Statistical Society: Series B (Methodological)* (1964).

[3] Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone. 1984. Classification and regression trees. (1984).

[4] David L. Donoho. 2000. High-dimensional data analysis: The curses and blessings of dimensionality. (2000).

[5] "Dheeru Dua and Casey Graff". "2017". "UCI Machine Learning Repository". "http://archive.ics.uci.edu/ml"

[6] Douglas Hawkins and Gordon Kass. 1982. Automatic Interaction Detection. *Topics in Applied Multivariate Analysis* (1982).

[7] Seth Hettich and Stephen Bay. 1998. The UCI KDD Archive. (1998). https://kdd.ics.uci.edu/databases/kddcup98/kddcup98.html

[8] Stefan Heule, Marc Nunkesser, and Alexander Hall. 2013. HyperLogLog in Practice: Algorithmic Engineering of a State of The Art Cardinality Estimation Algorithm. *Proceedings of the 16th International Conference on Extending Database Technology* (2013).

[9] Guyon I and Elisseeff A. 2003. An introduction to variable and feature selection. *Journal of Machine Learning Research* (2003).

[10] J. M. Kanter and K. Veeramachaneni. 2015. Towards Automating Data Science Endeavors. *Proceedings of the International Conference on Data Science and Advance Analytics* (2015).

[11] G. Katz, E. C. R. Shin, and D. Song. 2016. ExploreKit: Automatic Feature Generation and Selection. *International Conference on Data Mining (ICDM)* (2016).

[12] Udayan Khurana, Horst Samulowitz, and Deepak S. Turaga. 2017. Feature Engineering for Predictive Modeling using Reinforcement Learning. *CoRR* abs/1709.07150 (2017). arXiv:1709.07150 http://arxiv.org/abs/1709.07150

[13] Udayan Khurana, Deepak Turaga, Horst Samulowitz, and Srinivasan Parthasrathy. 2016. Cognito: Automated Feature Engineering for Supervised Learning. *International Conference on Data Mining (ICDM)* (2016), 979–984.

[14] Tae-Hwan Kim and Halbert White. 2004. On more robust estimation of skewness and kurtosis. *Finance Research Letters* (2004).

[15] Sotiris Kotsiantis, Dimitris Kanellopoulos, and P. Pintelas. 2006. Data Preprocessing for Supervised Learning. *International Journal of Computer Science* 1 (01 2006), 111–117.

[16] Huan Liu, Farhad Hussain, Chew Lim Tan, and Manoranjan Dash. 2002. Discretization: An Enabling Technique. *Data Mining and Knowledge Discovery* 6, 4 (01 Oct 2002), 393–423. https://doi.org/10.1023/A:1016304305535

[17] Yink Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker. 2013. Accurate intelligible models with pairwise interactions. *KDD 2013* (2013).

[18] D. Micci-Barreca. 2001. A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *ACM SIGKDD Explorations Newsletter* (2001).

[19] J . Misra and David Gries. 1982. Finding repeated elements. *Science of Computer Programming* (1982).

[20] John Nelder and Robert Wedderburn. 1972. Generalized Linear Models. *Journal of the Royal Statistical Society. Series A (General)* (1972). https://doi.org/10.2307/

2344614

[21] Fabian Pedregosa, Gal Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and douard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* (2011).

[22] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence* (2019).

[23] Armand Ruiz. 2017. The 80/20 data science dilemma. *InfoWorld* (2017). https://www.infoworld.com/article/3228245/the-80-20-data-science-dilemma.html

[24] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Muller. 2017. Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models. *arXiv:1708.08296* (2017).

[25] SAS. 2019. SAS Visual Analytics. (2019). https://www.sas.com/en_us/software/visual-analytics.html

[26] SAS. 2019. SAS Visual Data Mining and Machine Learning. (2019). https://support.sas.com/en/software/visual-data-mining-and-machine-learning-support.html

[27] SAS. 2019. SAS Viya. (2019). https://www.sas.com/en_us/software/viya.html

[28] Fayyad U. and Irani. 1993. Multi-interval discretization of continuous-valued attributes for classification learning. *Proceedings of the 13th IJCAI* (1993).

[29] In-Kwon Yeo and Richard Johnson. 2000. A New Family of Power Transformations to Improve Normality or Symmetry. *Biometrika* (2000).

A ONLINE RESOURCES

Some resources are here: https://github.com/biruk23eth/fte. You need the SAS Visual Data Mining and Machine Learning product to reproduce the results.