2016 IEEE 16th International Conference on Data Mining Workshops

Cognito: Automated Feature Engineering for Supervised Learning

Udayan Khurana Deepak Turaga Horst Samulowitz Srinivasan Parthasrathy
{ukhurana,turaga,samulowitz,spartha}@us.ibm.com
IBM TJ Watson Research Center

***Abstract*—Feature engineering involves constructing novel features from given data with the goal of improving predictive learning performance. Feature engineering is predominantly a human-intensive and time consuming step that is central to the data science workflow. In this paper, we present a novel system called *Cognito*, that performs automatic feature engineering on a given dataset for supervised learning. The system explores various feature construction choices in a hierarchical and non-exhaustive manner, while progressively maximizing the accuracy of the model through a greedy exploration strategy. Additionally, the system allows users to specify domain or data specific choices to prioritize the exploration. Cognito is capable of handling large datasets through sampling and built-in parallelism, and integrates well with a state-of-the-art model selection strategy. We present the design and operation of Cognito, along with experimental results on eight real datasets to demonstrate its efficacy.**

I. Introduction

A data scientist goes through multiple steps in the analysis and exploration of a new dataset. The process starts with data ingestion, selection and preparation, followed by feature generation and transformation, leading up to modeling, and operationalizing. Several aspects of the data exploration process make it a very manual and often artful process – requiring high degrees of skill, and with little repeatability. This has limited the adoption of data analysis based decision-making in the real-world, since insights derived are either insufficient, sub-optimal or costly to obtain. This problem is exacerbated in Big Data settings, where the data cannot be easily visualized, or analyzed on a single computer, requiring sophisticated programming and optimization skills in addition to the data science skills for effective data exploration.

Feature generation and transformation, called *feature engineering*, is largely manual and often the most time consuming step in a data science workflow. It is a complex exercise, performed in an iterative manner with trial and error, and driven by domain knowledge developed over time. Finding suitable features is a crucial ingredient in learning a good predictive model. For instance, in order to learn a model for predicting precipitation quantity from geo-temporal observations of atmospheric quantities such as humidity, wind, temperature, etc., a spatiotemporal aggregation *transform* plays a key role in adding a historical context to the data points, making it possible to fit a good regression model. Wind [8] provides an interesting account of many Kaggle competition winning works, which highlights the importance of feature engineering in practical problems. In order to scale the data exploration effort, and bring it to mainstream decision-making, we need to assist data scientists with automation in feature engineering. In this paper, we present a generalized and extensible system for effective automatic feature engineering for supervised learning problems. The basic building blocks of feature engineering are *transform* functions that can be applied to one or more attributes in the dataset to create new features, and a corresponding transformed dataset. We introduce the notion of a hierarchical *Transformation Tree*, where the root represents a given dataset, each node a transformed dataset, and edges represent transforms. Each node (dataset) is associated with an accuracy score that is measured as the performance of a predictive model built using that dataset. We thereby reduce the problem of feature construction to finding the node in the tree with maximum accuracy and propose three different greedy heuristic search algorithms for the solution. Our framework is extensible and allows easy addition of new transforms. It works in a domain independent manner, while providing a simple way for a domain expert to influence the search through specification of constraints relevant to the properties of the dataset. The framework also supports the addition of novel search strategies or modifying the existing ones with trivial effort.

The efficacy of engineered features also depends on the type of model being employed. For instance, an SVM using a polynomial kernel does not benefit from a *product transform* as it performs that mapping implicitly; however, a linear SVM may find it beneficial. A joint optimization between feature engineering and modeling is essential. Our system extends the feature engineering with one such state-of-the-art method for automated model selection, employing incremental sampling and using upper bound estimation of the model performance [6]. Our strategy results in performance comparable to best hand-tuned efforts.

Our approach can be summarized as: (a) We build an extensible framework for capturing, representing, and employing the best-practice patterns in feature engineering and modeling; (b) We formulate the feature engineering problem as a search on the transformation tree and develop an incremental search strategy to efficiently explore the set of available transforms and their compositions for feature engineering; (c) We couple this with an incremental model search strategy that leverages data sampling and upper-bounding of performance to rapidly provide estimates of the best choice for models and their parameters, even for Big Data; (d) We build a library of data

2375-9259/16 $31.00 © 2016 IEEE
DOI 10.1109/ICDMW.2016.179

transform and modeling algorithms by curating them from open source and commercial platforms (e.g. R, Weka, Sklearn, Spark MLLib). We also explicitly capture their available parameterizations; (e) We evaluate our system with ten real datasets from a variety of domains, and show significant gains in short periods of execution time without human intervention.

A. Related Work

FICUS by Markovitch et al. [5] uses a variant of beam search over the space of possible features, which are constructed through certain constructor functions. They use a proxy measure of accuracy – information gain (through the complexity of an associated decision tree) to guide their search. FCTree [2] divides the training data recursively based on the information gain provided original or new features (that are randomly generated) to assess their performance. Data science machine by Kanter et al. [3] performs aggregates and other transforms on a dataset when presented in a normalized relational schema. It applies all transforms at once to the base dataset, followed by one step of feature selection. Dor et al. [1] perform random feature construction followed by feature selection. A survey by Sodhi [7] points to more relevant work in the domain.

We briefly list certain limitations of the past work. The use of information gain as a proxy to model accuracy, while efficient to compute, has two prominent drawbacks. First, it does not capture correlations between features, and secondly, it does not optimize feature engineering for the model being used. Generation of features without a sequential application of the transforms is insufficient to capture complex features that are formed as a combination of the basic transforms. Finally, applying feature selection to all the new features at once becomes a scalability bottleneck due to the explosion in feature dimensions.

It is worth noting that deep learning techniques implicitly performs feature construction. However, it requires large amounts of data to prevent overfitting, and the features generated are not human interpretable. Deep learning has shown impressive performance in speech, image and video processing, but it has not been used much for general purpose data analysis due to the need for large amounts of training data. Other methods, such as kernel tricks also generate useful features, but their scope is limited to specific type of features and specific models.

II. System Description

The problem of performing feature engineering can be broadly stated as finding the minimal set of transforms that maximizes the model performance, i.e., accuracy. The solution space of this problem is unbounded. The set of potential transforms is an open one, i.e., the features can be transformed in an indefinite number of ways. Even in a simpler case where the number of transforms is fixed in number (k), the number of ways in which these functions can be composed to generate features is unlimited. Considering a limit on the complexity of such combinations (d, at most), for f features, the possible new features are: $O(f.d^{k+1})$, and the possible datasets with the addition of one or more of these features is $O(2^{f.d^{k+1}})$. Given the complexity of the *data space*, we adopt a greedy incremental search approach. We define the essential aspects of Cognito below, followed by exploration on the transformation tree. We then explain the role of feature selection in speeding up the tree construction, followed by the integration of model selection and sampling.

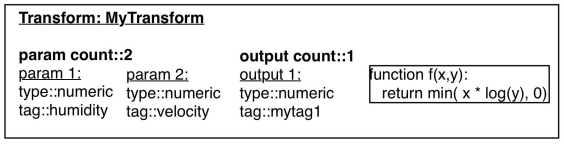

Fig. 1. Example of a transform specifying the input parameter criteria (type and tags), and output feature, and the transformation function logic.

A *feature*, $F(n)$, is defined by (a) vector of n-values and; (b) A data type, such as *Numeric*, *String*, *Date* or a categorical class; (c) An arbitrary number of string annotations, also known as *tags* that characterize the feature semantics and may be used to find a match for transform inputs. A *dataset*, $D(k, n)$, is defined by a set of $k-1$ n-valued features, an n-valued prediction target vector, and the problem task (classification or regression). The column and row cardinalities of D, written as $|D|_c(=k)$ and $|D|_r(=n)$, respectively. The data is expected in an ARFF file[1], where a feature tag is specified with the comment `%@attributetag <tag>`, immediately following the definition of the attribute.

A *transform*, τ^p is defined as an operation over a given dataset D_i, that produces another dataset D_o, i.e., $D_o = \tau^p(D_i)$, where p symbolizes a set of parameters used by a transform type, τ. The definition of a transformation involves (a) a function, f and, (b) a set of type matching constraints, C, for f's input. The application of a transform over a dataset involves row-wise application of instances f taking as input all valid combinations of feature columns in D that satisfy C, each producing one or more new columns. Note that the enumeration of candidate sets for input is taken care of by the platform, similar to that in [4]. The transform specification requires the criteria for each parameter and the core function itself, as specified symbolically in Figure 1. Note that a transformation only adds new features, therefore, $D_i \subseteq D_o$. Also, the target feature and problem type remain invariant across a transform. For transforms, τ_1 and τ_2, the expression $\tau_1(\tau_2(D))$ is also written as $\tau_1.\tau_2(D)$ for visual clarity.

We classify transforms based on whether the output value in a row is a function of only other values in the same row *(Type–A)* or may depend on values from other rows *(Type–B)*. Transforms are usually unary or binary, based on the number of inputs columns they take as input. Composition of multiple transforms effectively gives us higher order transforms as well. Examples of Type–A unary transform functions are: *logarithm; square; square root; exponential;*

[1]ARFF: http://www.cs.waikato.ac.nz/ml/weka/arff.html

rounding; absolute value; reciprocal; extract date, time, day of week, month and year from timestamp, z-normalization, min-max normalization. Examples of binary Type-A transform functions are: *difference, sum, product*. In contrast, examples of Type-B aggregate transforms are: *Aggregation by an entity using different statistical functions such as Count, Slope, Min, Max, Average; Geospatial Aggregation; Entity based Temporal Aggregation*, etc.

A. Transformation Tree and Search

The *Transformation tree* is a tree where the root corresponds to the input dataset, each other node corresponds to a transformed dataset and each (directed) edge represents a transforms, which when applied to a parent, yields the child dataset. Figure 2 shows a transformation tree, with the root at the left. Each node has an accuracy measure associated with it based on the cross-validation performance of the specified model learnt on that dataset. We translate the problem of feature construction into exploring the nodes of the corresponding transformation tree to find the node with the maximum accuracy. We adopt one of the following strategies to explore the tree. (a) *Depth-first traversal*: We start off with a random transform at the root and explore a branch until we cease to obtain an improvement in accuracy for γ consecutive nodes. If so, we mark that node as "done" and choose the highest accuracy node among the remaining active nodes explored thus far to continue the exploration in a similar manner; and so on. Ties are broken at random. (b) *Global traversal*: promotes global search for the most promising child of any nodes explored so far. The promise of a future node is evaluated through a relative measure of how successful the edge transform leading up to it, has performed so far in the tree. It is also a function of whether that transform has already been applied in the hierarchy (less priority) or is yet to be used. (c) *Balanced traversal*: is always based on a time or node budget where we chose from the following two strategies at each step until the budget is exhausted – explore (global options) or exploit (work on depth) using a biased random selection varying between exploration at the beginning and exploitation towards the end of the search ($P[explore] = \frac{TimeRemaining}{TimeBudget}$, $P[exploit] = 1 - P[explore]$).

B. Feature Selection

Application of a transform can severely increase the number of columns in the dataset. For instance, the application of a *log* transform on ten numeric features, produces another ten, doubling the count. The application of a product transform produces $\binom{10}{2}$ or 45 additional columns. When transforms are applied in a cascaded manner (deeper in the tree), the count multiplies even further. Learning algorithms do not scale well with increasing column count. In order to control the explosive growth of column counts, we provide the option of performing *feature selection* at each node on the newly generated features post the transform application. This pruning ensures a manageable size of the dataset throughout the tree, and speeds up the execution. However, its drawback is the premature discarding of features on which additional transformations might have yielded useful features.

A final round of feature selection is performed on the winning candidate (node with the highest accuracy measure) for *all* columns, original and constructed, before reporting the final resulting dataset.

Data	#Row	#Col	Before FE Accuracy	After FE Accuracy	%gain
Australian Credit	690	15	0.631	0.675	7%
Svmguide3	1243	22	0.617	0.796	29%
Svmguide1	3089	5	0.951	0.971	2%
Ionosphere	351	35	0.742	0.805	8%
Pima Diabetes	768	9	0.621	0.688	11%
German Credit	1k	25	0.690	0.752	9%
Weather	1m	424	0.660	0.810	23%
Energy	100	6	0.392	0.548	40%

TABLE I
IMPROVEMENT IN ACCURACY FOR VARIOUS DATASETS USING COGNITO.

C. Model Selection, Sampling and Accuracy Estimation

The best set of features may be different for different modeling approaches. While Cognito provides users with the choice of models to work with, it can also perform automated model selection. The selection is based on determining the best-performing models through incremental data allocation and on estimating the model performances based on upper bounds [6]. The approach analyzes the learning curve of each available model based on the performance on the currently allocated amount of training samples. It uses slope of the learning curve to estimate how a model would perform had one allocated it all the data. While this estimate is used to hone in on the best model as quickly as possible during the model selection process, it can also be used to guide search within Cognito when faced with larger data sets. Performing automated model selection at every node of the tree is significantly more expensive than using a single model. Instead, we adopt a progressive filtering of model choices where the deeper levels of the tree have fewer choices of models to select from, based on the performance higher up in the tree.

III. IMPLEMENTATION AND EXPERIMENTS

Cognito has been implemented in Java and makes use of analytic models and transforms from open source libraries such as Weka, LibSVM, different R packages. The web interface uses JSP for processing web requests and AJAX with D3 and JQuery for dynamic rendering of web pages. The system expects an ARFF file from the user. Due to space constraints, we omit a detailed description of the architecture and implementation details in this paper. Table II-B provides an account of performance of Cognito on several datasets from a wide variety of domains. The first 4 datasets (classification) are from the UCI ML repository[2], the next two (classification) from LIBSVM [3], the weather dataset (regression) is about

[2]UCI ML repository: http://archive.ics.uci.edu/ml/
[3]LIBSVM: https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/binary.html

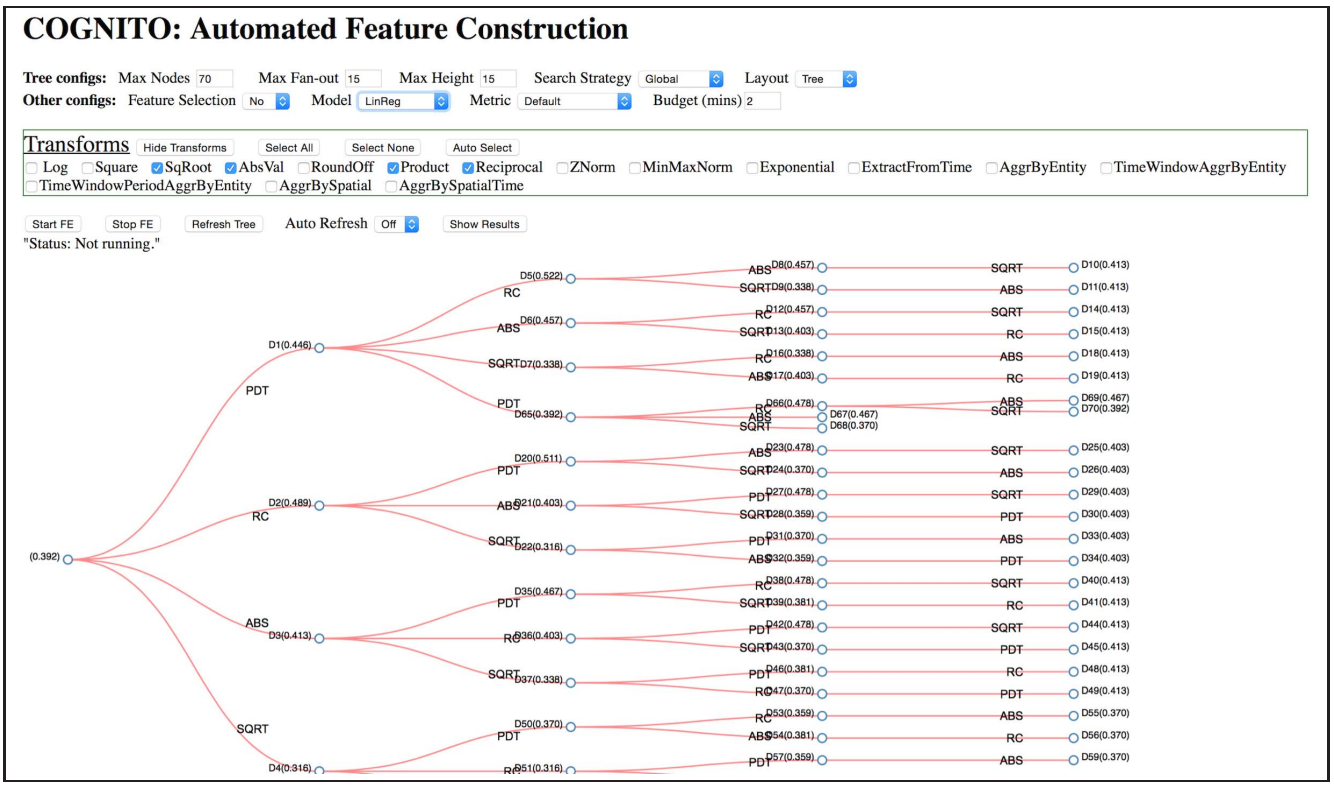

Fig. 2. In a transformation tree, a node corresponds to a *dataset* (root is the input dataset, rest are transformed) and an *edge* to a transform. In the given example, *feature engineering* is performed for a regression dataset using four transforms and results in an increases in accuracy from 0.392 to 0.522.

precipitation quantity prediction from NOAA[4]; the last one is a client proprietary dataset (regression) about predicting energy consumption. We observe accuracy gain for different datasets, even for the "Svmguide1" dataset which has a very high accuracy to begin with. All experiments were run on a single desktop machine within a budget of 1 minute, except the weather dataset, whose budget was 4 minutes.

IV. Demonstration Proposal

Cognito is simple and intuitive to operate. A sample run can be seen here[5]. The homepage expects the user to upload a valid ARFF file, upon which the system provides a preview of the dataset. The user can edit any feature tags at this point, go back and upload a different file, or proceed to feature engineering. At this stage, the user is presented with a default list of parameters for the run, such as the tree bounds, model(s) to test, transformations to choose from, whether to use feature selection or not, time budget for the run, amongst others. The user may modify the configuration, and upon launching the process, the growing tree is visible in real time on the screen. At any time, the user can click the `GetResult` button to see the composition of the data version with the highest accuracy so far. The user is presented with the lineage of each feature in the resulting dataset starting from the original dataset.

[4]National Oceanic and Atmospheric Administration: http://www.noaa.gov
[5]Using Cognito: https://www.youtube.com/watch?v=hJlG0mvynDo

Demonstration Plan: During the conference, we propose to demonstrate the efficacy of Cognito by letting the users work with a dataset of their choice (one that is publicly downloadable). Additionally, we will also provide the datasets listed in Table II-B. We would emphasize on the following aspects: (a) interpretation of the transformed features; (b) impact of search strategy in finding the best result; (c) impact of the choice of model (including automated model selection option) to the final result; (d) utility of user annotation. Finally, we plan to use this exercise to solicit feedback about the system to further improve the interactivity, the search strategy, the set of transforms such that we improve its usability.

References

[1] O. Dor and Y. Reich. Strengthening learning algorithms by feature discovery. *Information Sciences*, 2012.
[2] W. Fan et al. Generalized and heuristic-free feature construction for improved accuracy. In *SDM*, 2010.
[3] J. M. Kanter and K. Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *IEEE DSAA*, 2015.
[4] U. Khurana, S. Parthasarathy, and D. S. Turaga. READ: Rapid data exploration, analysis and discovery. In *EDBT*, 2014.
[5] S. Markovitch and D. Rosenstein. Feature generation using general constructor functions. *Machine Learning*, 2002.
[6] A. Sabharwal, H. Samulowitz, and G. Tesauro. Selecting near-optimal learners via incremental data allocation. In *AAAI*, 2016.
[7] P. Sondhi. Feature construction methods: a survey. 2009.
[8] D. K. Wind. Concepts in predictive machine learning. Master's thesis, Technical University of Denmark, 2014.