Deep Feature Synthesis: Towards Automating Data Science Endeavors

James Max Kanter
CSAIL, MIT
Cambridge, MA - 02139
kanter@mit.edu

Kalyan Veeramachaneni
CSAIL, MIT
Cambridge, MA- 02139
kalyan@csail.mit.edu

***Abstract*—In this paper, we develop the Data Science Machine, which is able to derive predictive models from raw data automatically. To achieve this automation, we first propose and develop the *Deep Feature Synthesis* algorithm for automatically generating features for relational datasets. The algorithm follows relationships in the data to a base field, and then sequentially applies mathematical functions along that path to create the final feature. Second, we implement a generalizable machine learning pipeline and tune it using a novel Gaussian Copula process based approach. We entered the Data Science Machine in 3 data science competitions that featured 906 other data science teams. Our approach beats 615 teams in these data science competitions. In 2 of the 3 competitions we beat a majority of competitors, and in the third, we achieved 94% of the best competitor's score. In the best case, with an ongoing competition, we beat 85.6% of the teams and achieved 95.7% of the top submissions score.**

I. Introduction

Data science consists of deriving insights, knowledge, and predictive models from data. This endeavor includes cleaning and curating at one end and dissemination of results at the other, and data collection and assimilation may also be involved. After the successful development and proliferation of systems and software that are able to efficiently *store*, *retrieve*, and *process* data, attention has now shifted to analytics, both *predictive* and *correlative*. Our goal is to make these endeavors more efficient, enjoyable, and successful.

To begin with, we observed that many data science problems, such as the ones released by KAGGLE, and competitions at conferences (KDD cup, IJCAI, ECML) have a few common properties. First, the data is structured and relational, usually presented as a set of tables with relational links. Second, the data captures some aspect of human interactions with a complex system. Third, the presented problem attempts to predict some aspect of human behavior, decisions, or activities (e.g., to predict whether a customer will buy again after a sale [IJCAI], whether a project will get funded by donors [KDD Cup 2014], or even where a taxi rider will choose to go [ECML]).

Given a prediction problem, the data scientist must first form variables, otherwise known as features. The data scientist may start by using some static fields (e.g. gender, age, etc.) from the tables as features, then form some specialized features by intuiting what might predict the outcome. Next, the scientist may develop new features that transform the raw fields into different measures (e.g. "*percentile of a certain feature*"). This is illustrated in the first three blocks of Figure 1.

Transforming raw data into features is often the part of the process that most heavily involves humans, because it is driven by intuition. While recent developments in deep learning and automated processing of images, text, and signals have enabled significant automation in feature engineering for those data types, feature engineering for relational and human behavioral data remains iterative, human-intuition driven, and challenging, and hence, time consuming. At the same time, because the efficacy of a machine learning algorithm relies heavily on the input features [1], any replacement for a human must be able to engineer them acceptably well .

To this end, we developed a feature synthesis algorithm called *Deep Feature Synthesis*. Although automatic in nature, the algorithm captures features that are usually supported by human intuition.

Once features are synthesized, one may select from several classification methodologies (svm, neural networks, etc.) and fine-tune the parameters [2], or cluster the data and build cluster-wise models (as shown in the fourth block of Figure 1). To explore these varied options, we also designed and implemented a generalized machine learning pathway and autotuning approach.

With these components in place, we present the Data Science Machine — an automated system for generating predictive models from raw data. It starts with a relational database and automatically generates features to be used for predictive modeling. Most parameters of the system are optimized automatically, in pursuit of good general purpose performance.

Our contributions through this paper are as follows:

- Designed the *Deep Feature Synthesis* algorithm, which is capable of generating features that express a rich feature space
- Developed an end-to-end Data Science Machine which can:
  - (a) automatically generate features via *Deep Feature Synthesis*
  - (b) autotune a machine learning pathway to extract the most value out of the synthesized features
  - (c) produce submissions for online data science competitions
- Matched human level performance when competing in data science competitions using the Data Science Machine

Paper Summary: In Section II, we explain the algorithm for *Deep Feature Synthesis*, and we detail the implementation in

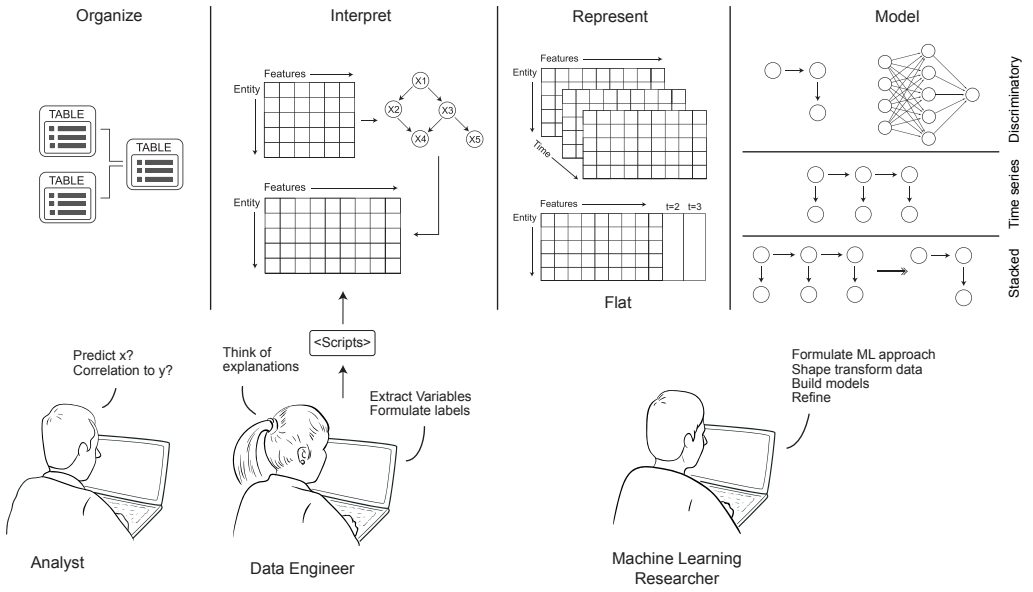

Fig. 1. A typical data science endeavor. It started with an analyst positing a question: Could we predict if x or y is correlated to z? These questions are usually defined based on some need of the business or researcher holding the data. Second, given a prediction problem, a data engineer posits explanatory variables and writes scripts to extract those variables. Given these variables/features and representations, the machine learning researcher builds models for given predictive goals, and iterates over different modeling techniques. Choices are made within the space of modeling approaches to identify the best generalizable approach for the prediction problem at hand. A data scientist can undergo the entire process; that is, positing the question and forming variables for building, iterating, and validating the models.

Section III. In Sections IV and V, we describe the generalized machine learning pathway and tuning process. We present our results in Section VI, and discuss them in VII. We summarize how the Data Science Machine fits in with related works in VIII. Finally, we conclude and propose future work in Section IX.

II. Deep Feature Synthesis

Deep Feature Synthesis is an algorithm that automatically generates features for relational datasets. In essence, the algorithm follows relationships in the data to a base field, and then sequentially applies mathematical functions along that path to create the final feature. By stacking calculations sequentially, we observe that we can define each new feature as having a certain depth, d. Hence, we call the algorithm *Deep Feature Synthesis*. In this section, we explain the motivation for *Deep Feature Synthesis*, define the feature synthesis abstractions, and present the algorithm.

A. Prototypical Problems and Motivation

Let us consider a hypothetical dataset for an e-commerce website. Our goal is calculate features that describe customers' behaviors based on all available data. The schema for this dataset is shown in Figure 2.

To think like a data scientist, we might start by asking questions that could be translated into features that describe a customer — for instance, "how often does this customer make a purchase?" or "how long has it been since this customer's last purchase?". We might also look at entities related to a customer, and ask questions about them — for instance, "how much does the total order price vary for the customer?" or "does this customer typically buy luxurious or economical products?". These questions can be turned into features by following relationships, aggregating values, and calculating new features. Our goal is to design an algorithm capable of generating the calculations that result in these types of features, or can act as proxy quantities.

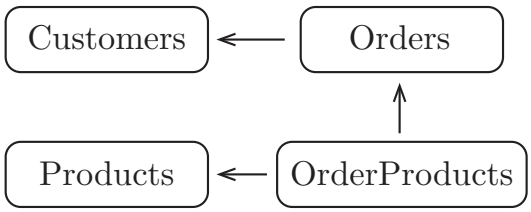

Fig. 2. A simplified schema for an e-commerce website. There are 4 entities. An arrow from one entity to another signifies that the first entity references the second in the database.

B. Feature Synthesis Abstractions

The input to *Deep Feature Synthesis* is a set of interconnected entities and the tables associated with them. There is a unique identifier for each instance of an entity in the table. Optionally, an entity can refer to an instance of a related entity by using the related entity's unique identifier. An instance of the entity has features which fall into one of the following data types: *numeric*, *categorical*, *timestamps* and *freetext*.

Notationally, for a given dataset, we have entities given by $E^{1\ldots K}$, where each entity table has $1 \ldots J$ features. We denote

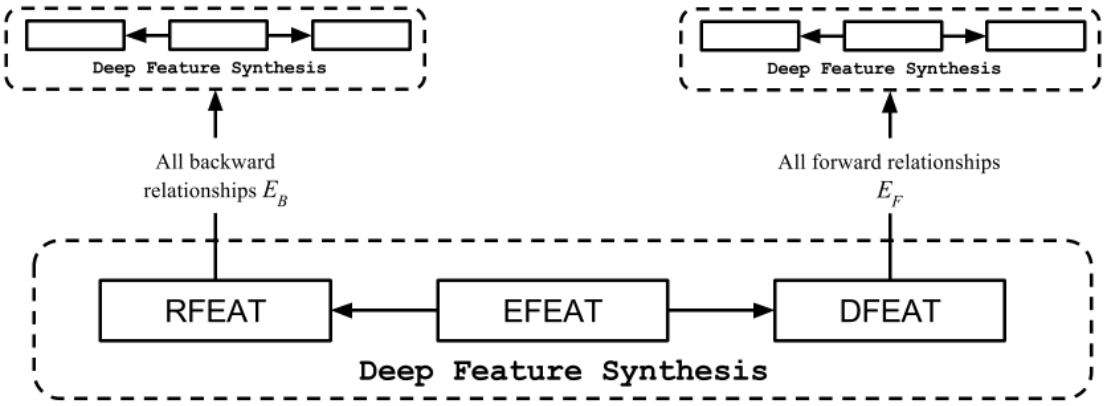

Fig. 3. Illustration of computational constraints when synthesizing each feature type. Both `rfeat` and `dfeat` features can be synthesized independently, while `efeat` features depend on both `rfeat` and `dfeat` features. One instantiation of an approach for *Deep Feature Synthesis* is given in Algorithm 1.

a specific entry as $x^k_{i,j}$, which is the value for feature j for the i^{th} instance of the k^{th}entity.

Next, given entities, their data tables, and relationships we define a number of mathematical functions that are applied at two different levels: at the *entity* level and at the *relational* level. We present these functions below. Consider an entity E^k for which we are assembling the features. For notational convenience we drop k that is used to represent an specific entity.

The first set of features is calculated by considering the features and their values in the table corresponding to the entity k alone. These are called *entity* features, and we describe them below.

Entity features (`efeat`): Entity features derive features by computing a value for each entry $x_{i,j}$. These features can be based on the computation function applied element-wise to the array $x_{:,j}$. Examples include functions that translate an existing feature in an entity table into another type of value, like conversion of a categorical string data type to a pre-decided unique numeric value or rounding of a numerical value. Other examples include translation of a timestamp into 4 distinct features — weekday (1-7), day of the month (1-30/31), month of the year (1-12) or hour of the day (1-24).

These features also include applying a function to the entire set of values for the j^{th} feature,$x_{:,j}$, and $x_{i,j}$, given by:

$$x_{i,j'} = efeat(x_{:,j}, i). \tag{1}$$

An example of such a computation is a cumulative distribution function (cdf)-based feature. To generate this feature, we form a density function over $x_{:,j}$, and then, evaluate the cumulative density value for $x_{i,j}$ (*or percentile*), thus forming a new feature.

The second set of features is derived by jointly analyzing two related entities, E^l and E^k.These two entities relate to each other in one of two ways: *forward* or *backward*.

Forward: A *forward* relationship is between an instance m of entity E^l, and a single instance of another entity i in E^k. This is considered the *forward* relationship because i has an explicit dependence on m. In the above e-commerce example, the *Orders* entity has a forward relationship with the *Customers*; that is, each order in the *Orders* table is related to only one customer.

Backward: The *backward* relation is the relationship from an instance i in E^k to all the instances $m = \{1 \dots M\}$ in E^l that have forward relationship to k. In the same example as above, the *Customers* entity has a backward relationship with *Orders*; that is, many orders can point to the same customer.

Direct Features (`dfeat`): Direct features are applied over the *forward* relationships. In these, features in a related entity $i \in E^k$ are directly transferred as features for the $m \in E^l$.

Relational features (`rfeat`): Relational features are applied over the *backward* relationships. They are derived for an instance i of entity E^k by applying a mathematical function to $x^l_{:,j|e^k=i}$, which is a collection of values for feature j in related entity E^l, assembled by extracting all the values for feature j in entity E^l where the identifier of E^k is $e^k = i$. This transformation is given by

$$x^k_{i,j'} = rfeat(x^l_{:,j|e^k=i}). \tag{2}$$

Some examples of `rfeat` functions are *min*, *max*, and *count*. Other `rfeat` functions include functions that could be applied to the probability density function over $x^l_{:,j|e^k=i}$.

C. Deep Feature Synthesis algorithm

To describe the *Deep Feature Synthesis* algorithm, we first consider a dataset of K entities, denoted as $E^{1\dots K}$. Our goal is to extract `rfeat`, `dfeat`, and `efeat` features for a target E^k. Additionally, we know all the entities with which E^k has FORWARD or BACKWARD relationships. These are denoted by sets E_F and E_B.

To start, we see that `efeat` features are created using features that already exist within the entity. We must first synthesize `rfeat` and `dfeat` features so we can apply `efeat` feature to the results. In order to generate `rfeat` features for E^k, we use features from entities in E_B. Thus, we must create all features types for each entity in E_B before

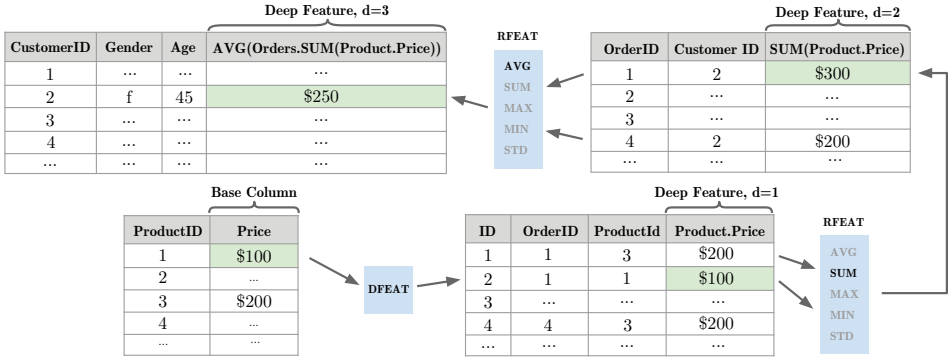

Fig. 4. An example of a feature that could be generated by *Deep Feature Synthesis*. The illustration shows how features are calculated at different depths, d, by traversing relationships between entities.

we can realize `rfeat` features for E^k. In a similar manner, we add `dfeat` features to E^k. These features are realized using the features from entities in E_F, so we must calculate all features for each entity in E_F first. Finally, with all `rfeat` and `dfeat` features added to E^k, we can generate `efeat` features. Figure 3 visualizes the sequence of computations in order to correctly generate each feature type.

Next, we consider a scenario where an entity related to the target entity has its own related entities. To handle this case we can recursively generate features using the same sequence described above. The recursion can terminate when we reach a certain depth or there are no related entities.

Algorithm 1 Generating features for target entity

```
1:  function MAKE_FEATURES(E^i, E^{1:M}, E_V)
2:      E_V = E_V ∪ E^i
3:      E_B = BACKWARD(E^i, E^{1...M})
4:      E_F = FORWARD(E^i, E^{1...M})
5:      for E^j ∈ E_B do
6:          MAKE_FEATURES(E^j, E^{1...M}, E_V)
7:          F^j = F^j ∪ RFEAT(E^i, E^j)
8:      for E^j ∈ E_F do
9:          if E^j ∈ E_V then
10:             CONTINUE
11:         MAKE_FEATURES(E^j, E^{1...M}, E_V)
12:         F^i = F^i ∪ DFEAT(E^i, E^j)
13:     F^i = F^i ∪ EFEAT(E^i)
```

The algorithm pseudocode for MAKE_FEATURES is presented above to make features, F^i, for the ith entity. The organization of recursive calls and calculation of each feature type is in accordance with the constraints explained above. The RFEAT, DFEAT, and EFEAT functions in the pseudocode are responsible for synthesizing their respective feature types based on the provided input. The algorithm stores and returns information to assist with later uses of the synthesized feature. This information includes not only feature values, but also *metadata* about base features and functions that were applied.

In our algorithm, we choose to calculate `rfeat` features before `dfeat` features, even though there are no constraints between them. Additionally, E_V keeps track of the entities that we have "visited". On line 10, we make sure to not included `dfeat` features for entities we have visited. This ensures we avoid calculating `rfeat` features unnecessarily. For example, consider a case where we are making features for the *Customer* entity in our example e-commerce database. It would not make sense to create a `dfeat` for each order that pulled in the age of the customer who placed the order because when we later create `rfeat` for *Customers*, we are going to aggregate each order based on the customer who placed it, and the orders in each group will have same value for `dfeat` features.

Figure 4 shows an example of a feature that is recursively generated. In this example, we eventually calculate the average order size for every customer. However, in order to realize that value, we do intermediate calculations, starting with the *Product* entity. First, we calculate a `dfeat` feature to add the product's price to the *ProductOrders* entity. Next, we calculate an `rfeat` feature for *Orders* by applying the SUM function to all instances of *ProductOrders* related to a given instance of the *Orders* entity. Finally, we calculate another `rfeat` feature to calculate the average total order size for each customer.

D. Growth of number of features

The feature space that can be enumerated by *Deep Feature Synthesis* grows very quickly. To understand how it grows, we analyze the number of features, z, the algorithm will synthesize for a given entity. Due to the recursive nature of feature synthesis, the number of features created for an entity depends on the number created for related entities. Thus, we let z_i represent the number of features we create for an entity if we recurse i times. Additionally, we assume that all entities in our dataset start with $O(j)$ features, and have $O(n)$ forward relationships and $O(m)$ backward relationships.

First, we synthesize `rfeat` type features for $O(m)$ entities. If we assume there are $O(r)$ `rfeat` functions, then we synthesize $O(r \cdot z_{i-1})$ features for each of the m entities in a backward relationship for a total of $O(r \cdot z_{i-1} \cdot m)$ additional features. Next, we make one `dfeat` feature for every feature in entities in a forward relationship. This means we add $O(z_{i-1} \cdot n)$ features. Finally, we must make `efeat` features using the j original features, as well as the $O(z_{i-1} \cdot (r \cdot m + n))$ newly synthesized features. We assume there are $O(e)$ `efeat` functions. Therefore, we will synthesize an additional $O(e \cdot j + e(z_{i-1} \cdot (r \cdot m + n)))$ `efeat` features.

Combining all `rfeat`, `dfeat`, and `efeat` features, we see that $z_i = O(z_{i-1} \cdot (r \cdot m + n)(e + 1) + e \cdot j)$. At z_0 we can only calculate `efeat` features, so $z_0 = O(e \cdot j)$. Let $p = (r \cdot m + n)(e + 1)$ and $q = e \cdot j$. Substituting, we get

$$z_i = O\left(z_{i-1} \cdot p + q\right),$$

and replacing z_{i-1} by $z_{i-2} \cdot p + q$ we get

$$z_i = O\left(z_{i-2} \cdot p^2 + q \cdot p + q\right).$$

Continuing the expansion till z_0 we get

$$z_i = O\left(z_0 \cdot p^i + q \cdot p^{i-1} + q \cdot p^{i-2} \cdots + q\right).$$

Replacing $z_0 = O(e \cdot j) = q$ in the above equation we get

$$z_i = O\left(q \cdot p^i + q \cdot p^{i-1} + q \cdot p^{i-2} \cdots + q\right)$$

$$z_i = O\left(q \cdot \left(\sum_{u=0}^{i} p^u\right)\right).$$

Thus, the closed form for z_i is

$$z_i = O\left((e \cdot j) \sum_{u=0}^{i} (r \cdot m + n)^u \cdot (e + 1)^u\right)$$

III. Deep Feature Synthesis: Implementation

In implementing *Deep Feature Synthesis*, we aim to rapidly deploy the Data Science Machine and evaluate the synthesized features when it encounters a new dataset.

The Data Science Machine and accompanying *Deep Feature Synthesis* algorithm are built on top of the MySQL database using the InnoDB engine for tables. All raw datasets were manually converted to a MySQL schema for processing by the Data Science Machine. We implement the logic for calculating, managing, and manipulating the synthesized features in Python.

We chose to use a relational database due to the natural parallels between how they store data and the requirements of *Deep Feature Synthesis*: in each, entities are represented as tables and features are represented as columns.

All mathematical functions must provide a Feature Function interface. A Feature Function takes one or two entities as inputs depending on the type of feature. Two entities are required in the case of relational level features, and direct features where the source is another entity. The function definition is responsible for determining which columns in the relationship it can be applied to and how to calculate the output value.

For performance reasons, the Data Science Machine builds Feature Functions on top of the functions provided by MySQL. Currently, the Data Science Machine implements the following `rfeat` functions: `AVG()`, `MAX()`, `MIN()`, `SUM()`, `STD()`, and `COUNT()`. In addition, the system implements the following `efeat` functions: `length()` to calculate the number of characters in a text field, and `WEEKDAY()` and `MONTH()` to convert dates to the day of the week or month they occurred. Despite their simplicity, this small base of functions is enough to create a wide range of features for us to use in evaluating the Data Science Machine.

Filter Objects provide a flexible way to select subsets of data for `rfeat` functions. They enable construction of new Filter Objects by combining existing Filter Objects using logical operators such as “and” or “or”. Filter Objects provide an interface to return the columns being filtered, as well as a way to be serialized to a MySQL query.

Filter Objects implement two useful pieces of functionality for *Deep Feature Synthesis*. First, they provide a way to apply `rfeat` functions only to instances where a certain condition is true. We call this usage a *categorical filter*. For example, "total amount of money spent by this customer on products *where* company X manufactured the product" constitutes a categorical filter. Second, they allow construction of time interval features by specifying an upper and lower bound on a date field. For instance, "number of orders this customer placed *where* the order was placed after March 31st and before April 30th.

During use, the Data Science Machine constructs database queries only when necessary. The data required to calculate a feature may be stored in multiple tables, so we must join tables to make sure all columns in the `SELECT` clause or `WHERE` clause of a query are accessible. An example query is in Figure 5. Queries are further optimized to perform multiple calculations at once to reduce the number joins and table scans.

```
UPDATE Donors_1 target_table
LEFT JOIN (
        SELECT donor_acctid, SUM(amount) as val
        FROM Donations rt
        GROUP BY donor_acctid
        ) b
ON b.donor_acctid = target_table.donor_acctid
SET target_table.Donors_1__100 = b.val
WHERE b.donor_acctid = target_table.donor_acctid
```

Fig. 5. An example MySQL query auto-generated by the Data Science Machine to create a feature for the Donors entity in the KDD cup 2014 dataset. This query calculates the total amount given by each donor.

IV. Predictive Machine Learning Pathway

To use the features created by *Deep Feature Synthesis*, we implement a generalized machine learning pathway.

The first step is to formulate a prediction problem. We achieve this by selecting one of the features in the dataset to model. We call this feature we wish to predict the target value.

After a target value is selected, we assemble the features that are appropriate for use in prediction. We call these features *predictors*. If predictors are computed using common base data as the target value, or if they rely on data that does not exist at the time of the occurrence of the target value, they are filtered out as invalid. The Data Science Machine also maintains a database of *metadata* associated with each entity-feature. This *metadata* contains information about the base fields in the original database that were used to form the feature, as well as any time dependencies contained within it.

A. Reusable machine learning pathways

With a target feature and predictors selected, the Data Science Machine implements a parametrized pathway for data preprocessing, feature selection, dimensionality reduction, modeling, and evaluation. To tune the parameters, the Data

TABLE I. SUMMARY OF PARAMETERS IN THE MACHINE LEARNING PIPELINE AND THE OPTIMIZED VALUES FROM RUNNING GCP BY DATASET.

Param	Default	Range	Function	KDD cup 2014	IJCAI	KDD cup 2015
k	1	[1, 6]	The number of clusters to make	1	1	2
n_c	100	[10, 500]	The number of SVD dimensions	389	271	420
γ	100	[10, 100]	The percentage of top features selected	32	65	18
rr	1	[1, 10]	The ratio to re-weight underrepresented classes	4	8	1
n	200	[50, 500]	The number of decision trees in a random forest	387	453	480
m_d	None	[1, 20]	The maximum depth of the decision trees	19	10	10
β	50	[1, 100]	The maximum percentage of features used in decision trees	32	51	34

Science Machine provides a tool for performing intelligent parameter optimization. The following steps are followed for machine learning and building predictive models:

Data preprocessing: Prior to entering the machine learning pathway, we clean the data by removing the null values, converting the categorical variables using one-hot encoding, and normalizing the features.

Feature selection and dimensionality reduction: *Deep Feature Synthesis* generates a large number of features per entity. To reduce the size of the feature space, we employ two techniques sequentially: first, we use Truncated SVD transformation and select n_c components of the SVD; then, we rank each SVD-feature by calculating its *f-value w.r.t* to the target value, and select the $\gamma\%$ highest ranking features.

Modeling: For modeling, we use a random forest by constructing n decision trees. Each decision tree has a depth of m_d and uses a fraction of the features denoted by β. For many datasets, it can be powerful to have a separate model for different clusters of data points. To incorporate this, we separate training points into k clusters using a kmeans clustering technique. We then train a distinct random forest for each cluster. To predict a label for a test sample, a trained cluster classifier assigns a cluster label to the data point and then applies the corresponding model.

In classification problems, sometimes one of the target value classes is underrepresented. In order to compensate for this, the modeling stage can re-weight an underrepresented class by a factor of rr.

In the modeling stage we have introduced four parameters: n, m_d, β, k, rr. Next, we present how to autotune this pathway.

V. BAYESIAN PARAMETER OPTIMIZATION USING GAUSSIAN COPULA PROCESSES

Many stages of the machine learning pipeline have parameters that could be tuned, and this tuning may have a noticeable impact on the model performance. A naive grid search would lead to search in the space of $6 * 490 * 90 * 10 * 450 * 20 * 100 = 2,381,400,000,000$ (two trillion, three hundred eighty-one billion, four hundred million) possibilities if we consider all possible combinations of parameter values. To aid in the exploration of this space, we use a Gaussian Copula Process (GCP). GCP is used to model the relationship f between parameter choices and the performance of the whole pathway (Model). Then we sample new parameters and predict what their performance would be (Sample). Finally, we apply selection strategies to choose what parameters to use next (Optimize). We describe each of these steps in detail below.

Model: Typically, a Gaussian process is used to model f such that for any finite set of N points $\bar{p}_{1\ldots n}$ in $\mathcal{X}$, $\{\ f(\bar{p}_i)\ \}_{i=1}^{N}$ has a multivariate Gaussian distribution on $\mathbb{R}^N$. The properties of such a process are determined by the mean function (commonly taken as zero for centered data) and the covariance function $K : P \times P \to \mathbb{R}$. For an extensive review of Gaussian Processes, see [3], while their use in Bayesian optimization is largely explained in [4] and their usage in tuning parameters of classifiers is explained in [2].

In this paper, we introduce a novel approach for parameter optimization based on Copula processes as defined by Wilson et al. in [5] through *warping* the output space. In GCP, instead of a Gaussian Process to model the multivariate distribution of $\{\ f(\bar{p}_i)\ \}_{i=1}^{N}$, it is done through a mapping $\Psi : \mathbb{R} \to \mathbb{R}$ that transforms $\{\ f(\bar{p}_i)\ \}_{i=1}^{N}$ into $\{\ \Psi \circ f(\bar{p}_i)\ \}_{i=1}^{N}$, which is then modeled as a Gaussian Process. By doing this, we change the assumed Gaussian marginal distribution of each $f(\bar{p})$ into a more complex one.

The mapping function: In [5], a parametrized mapping is learned so that the transformed output is best modelled by a Gaussian Process. However, such a mapping is unstable; for many experiments on the same data set, different mappings were learned. Moreover, the induced univariate distribution was most of the time almost Gaussian and the parametrized mapping could not offer a greater flexibility. To overcome this, we utilize a novel approach where a marginal distribution is learned from the observed data through kernel density estimation. More specifically, consider the parameters $\bar{p} = \{p_1 \ldots p_m\}$ and the performance function $f(\bar{p})$. Our first step in the transformation models the density of $\{\ f(\bar{p}_i)\ \}_{i=1}^{N}$ using a kernel density estimator, and then estimates the *cdf* of this density. We then generate the *cdf* values for each value of $\{\ f(\bar{p}_i)\ \}_{i=1}^{N}$ and are given by $g = cdf(f(\bar{p}))$. Assuming that g is a sample from a standard normal, we apply ψ^{-1} to values in g to generate the final values given by $h = \psi^{-1}(g)$. h represents the transformation of $f(.)$ which we wish to model using a regular Gaussian process. Hence the input to the Gaussian process modeling is $\bar{p}_{1\ldots n}$ and the corresponding $h_{1\ldots n}$ values.

Fitting the covariance function: We use a generalization of a squared exponential with a periodic component, inspired by [3], pp. 119-221. We learn the model parameters *via* likelihood maximization.

Sample: We sample iteratively some points in $\mathcal{P}$, and predict their corresponding outcome value $f(\bar{p})$ using the GCP model, and then decide which point to select next.

Optimize: This final step is usually made by maximizing the *acquisition function* a. The inputs to this function are derived from f to balance between exploration (testing points in unexplored areas of the input space $\mathcal{P}$) and exploitation (testing points for which we predict a high $f(\bar{p})$ value). In particular, this enables us to avoid concentrating on the search

near local optima. Given a set of observations $(p_{1...n}, f(p_n))$, we can thus randomly sample $\bar{p}'_i$ in $\mathcal{P}$, predict their output f'_i and choose the $\bar{p}'^*$ that maximizes a [2].

VI. Experimental Results

As the Data Science Machine is the first of its kind, we wish to address a number of questions that are at the intersection of "how well did the machine do?", "did it produce any valuable features?", and "did automation work?". We demonstrate the effectiveness of the Data Science Machine by applying it to datasets where many data scientists competed for the best results — KDD cup 2014, IJCAI, and KDD cup 2015. In each one of these competitions, hundreds of data science enthusiasts participated in solving a prediction problem defined by the organizers. The schemas in Figure 6 show the entities for each dataset, and we briefly describe each of the problems below.

KDD *cup 2014 - Project Excitement:* Using past projects' histories on DonorsChoose.org, predict if a crowd-funded project is "exciting".

IJCAI - *Repeat Buyer Prediction:* Using past merchant and customer shopping data, predict if a customer making a purchase during a promotional period will turn into a repeat buyer.

KDD *cup 2015 - Student Dropout:* Using student interaction with resources on an online course, predict if the student will dropout in the next 10 days.

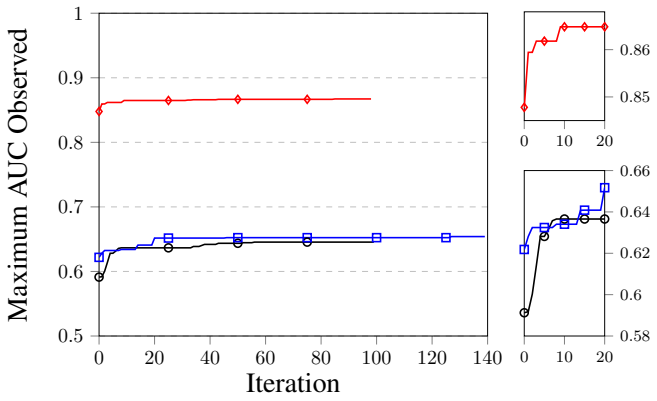

Fig. 7. The maximum cross validation AUC score found by iteration for all three datasets. From top to bottom: KDD cup 2014, IJCAI, KDD cup 2015

To compete, data scientists engage in *feature engineering*, *modeling*, and fine *tuning*. All three of these activities can be performed by the Data Science Machine with minimal human involvement beyond setting initial parameters of the Data Science Machine such that the computational limits are maximized. We first ran Data Science Machine with fully automated feature generation (*Deep Feature Synthesis*). For KDD cup 2014, we also constructed time interval features, and for IJCAI and KDD cup 2015, we created a few categorical filters. The results of running *Deep Feature Synthesis* are presented in Table III.

To determine the optimal parameters for the machine learning pathway, the Data Science Machine runs the Gaussian Copula Process based tuning approach. The parameters of the best run are shown in Table I. The results of running the Data Science Machine with and without tuning are presented in Table II.

We compare the results from our experiments with the public performance in these competitions to help us determine how Data Science Machine compares to human performance. Table IV shows how the Data Science Machine fared relative to other competitors on the leaderboard. The table presents some basic information about the teams that beat the Data Science Machine, and the teams that were beaten by it. To put these scores in perspective, Figure 8 shows how the Data Science Machine's score compares to other competitors at each percentile of the leaderboard. In the next section, we present our interpretation of these results.

VII. Discussion

In this section, we discuss how the results on all three datasets reflect the effectiveness of the Data Science Machine in emulating a data scientist.

Creating valuable synthesized features: In two out of three competitions, the Data Science Machine achieved > 90% of the best score achieved by any competitor. In its best performance, in KDD15, it beat approximately 86% of other competitors. If the synthesized features had no value, we would not expect the Data Science Machine to perform so well in the three competitions. These results demonstrate that Data Science Machine can produce new features that have *some* value.

Even in the IJCAI competition, where the Data Science Machine beat the fewest number of competitors, the Machine captured information about the problem similar to what would be gleaned by human intelligence. During the competition, the sponsors released a benchmark model. The proposed model used a customer's history at shops similar to the merchant in question to make predictions. To do this, they proposed one way to measure merchant similarity. The sponsors claimed that this method would achieve an AUC of at least .65. This is noteworthy, because the Data Science Machine GCP optimized solution achieved an AUC of more than .66. This means that the Data Science Machine was able to automatically derive a model that was at least as good as the human-proposed one. Additionally, if you look at the rate of improvement as a competitor's rank increases, the Data Science Machine 's score sits at the point where score improvement plateaus (see Fig. 8). The Data Science Machine achieved an AUC of 0.6606, while the AUC of the top competitors was 0.704, a difference of approximately 0.04. This suggests that the features chosen by the Data Science Machine captured the major aspects of the dataset, and only missed out on minor improvements.

In the other two competitions, the Data Science Machine features were enough to beat a majority of competitors, 86% in KDD15 and 70% in KDD14. In KDD15 the Data Science Machine also landed on the plateau of score improvement, achieving a final score of 0.8621 while the top competitor stood at 0.90.

This supports our statement that the features synthesized by the Data Science Machine capture important aspects of the prediction problem. In KDD14, the Data Science Machine 's

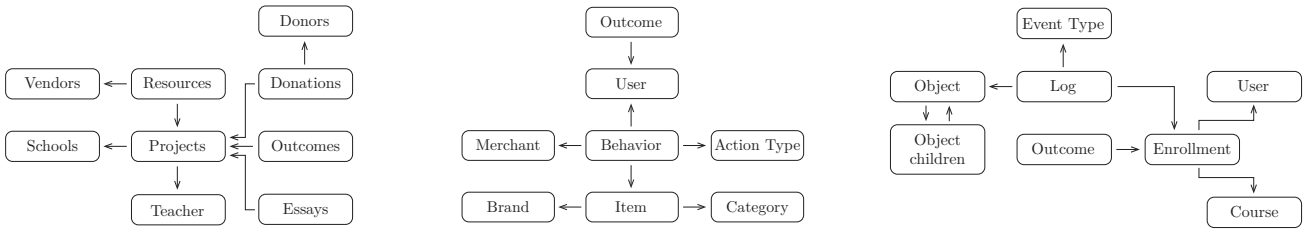

Fig. 6. Three different data science competitions held during the period of 2014-2015 — from left to right: KDD cup 2014, IJCAI, and KDD cup 2015. A total of 906 teams took part in these competitions. Predictions from the Data Science Machine solution were submitted and the results are reported in this paper. We note that two out of three competitions are ongoing at the time of writing.

TABLE II. THE AUC SCORES ACHIEVED BY THE DATA SCIENCE MACHINE. THE "DEFAULT" SCORE USES THE DEFAULT PARAMETERS. THE "LOCAL" SCORE IS THE RESULT OF RUNNING K-FOLDS (K=3) CROSS VALIDATION, WHILE THE "ONLINE" SCORE IS BASED ON SUBMITTING PREDICTIONS TO THE COMPETITION.

Parameter Selection	**KDD cup 2014**		**IJCAI**		**KDD cup 2015**	
	Local	Online	Local	Online	Local	Online
Default	.6059	.55481	.6439	.6313	.8444	.5000
GCP Optimized	.6321	.5863	.6540	.6606	.8672	.8621

TABLE III. THE NUMBER OF ROWS AND THE NUMBER OF SYNTHESIZED FEATURES PER ENTITY IN EACH DATASET. THE UNCOMPRESSED SIZES OF THE KDD CUP 2014, IJCAI, AND KDD CUP 2015 ARE APPROXIMATELY 3.1 GB, 1.9 GB, AND 1.0 GB, RESPECTIVELY.

KDD cup 2014

Entity	# Rows	# Features
Projects	664,098	935
Schools	57,004	430
Teachers	249,555	417
Donations	2,716,319	21
Donors	1,282,092	4
Resources	3,667,217	20
Vendors	357	13
Outcomes	619,326	13
Essays	664,098	9

IJCAI

Entity	# Rows	# Features
Merchants	4995	43
Users	424,170	37
Behaviors	54,925,330	147
Categories	1,658	12
Items	1,090,390	60
Brands	8,443	43
ActionType	5	36
Outcomes	522,341	82

KDD cup 2015

Entity	# Rows	# Features
Enrollments	2,000,904	450
Users	112,448	202
Courses	39	178
Outcomes	120,542	3
Log	13,545,124	263
Objects	26,750	304
ObjectChildren	26,033	3
EventTypes	7	2

chosen features are good enough to perform better than most competitors, but fall short of the top competitors, as the rate of improvement does not plateau.

Autotuning effectiveness: The previous section discussed whether or not the Data Science Machine creates features of value. Another important job of the Data Science Machine involves selecting which of those features to use and how to tune the model to best use them. By using autotuning, the Data Science Machine was able to increase its score across all three datasets, locally and online (see Table II). Through this process, the Data Science Machine is able to design a machine learning pathway that can adapt itself to a range of problems rather than depending on the default parameters.

The *online* submission of predictions for the KDD15 dataset brought about an interesting insight into using autotuning alongside default parameters. With the default parameters, the *local* cross-validation score seemed to produce a good result. However, when the predictions were uploaded to the competition website, the results were not as good as those found locally. We suspected that there was something wrong with the training setup, so to troubleshoot, we manually examined 450 features created by the Data Science Machine and tested many initializations of machine learning pathway parameters. Using an interface we developed, we were able to conclude that the issue was with the default parameters rather than with the training pathway itself.

The process we followed is similar to the process of a practicing data scientist. First, we experimented with what features to include in our system, and then we experimented with parameters of the machine learning process.

This is noteworthy because the auto-tuning process employed by the Data Science Machine did not encounter the issue described above, and the machine performed well relative to competitors in KDD15 when it came to selecting features and tuning parameters. This implies that the Data Science Machine's parameter tuning approach removed hours of debugging from the work flow. Additionally, it highlights the difficulty in picking default parameters for machine learning algorithms. Without the auto-tuning process, the Data Science Machine would not have achieved its goals.

Human value: The Data Science Machine performed well against competitors who were not able to come to the insights that put the "experts" above everyone else. We can see this in Figure 8, where the Data Science Machine scores toward the end of the plateau. At some point, moving up the leader board might not be worth the cost required to do so. The large human effort to move up the leader board is shown in Figure 9.

Implications for Data Scientists: The competitive success of the Data Science Machine suggests it has a role alongside data scientists. Currently, data scientists are very involved in the feature generation and selection processes. Our results show that the Data Science Machine can automatically create features of value and figure out how to use those features in creating a model. Although humans beat the Data Science Machine for all datasets, the machine's success-to-effort ratio suggests there is a place for it in data science.

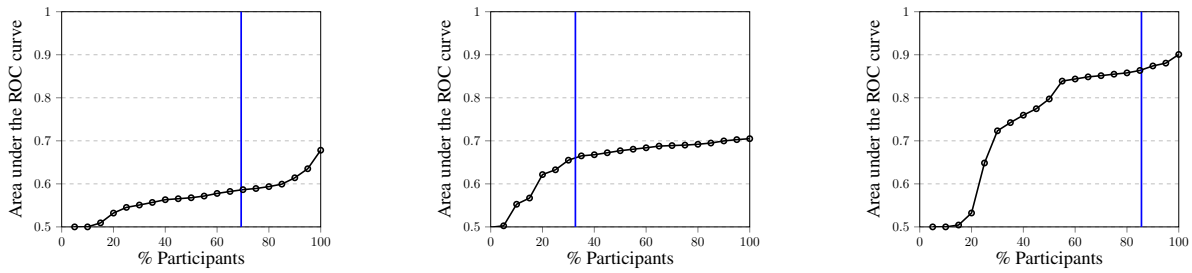

Fig. 8. AUC scores vs % participant achieving that score. The vertical line indicates where the Data Science Machine ranked, from top to bottom: KDD cup 2014, IJCAI, and KDD cup 2015

TABLE IV. HOW THE DATA SCIENCE MACHINE COMPARES TO HUMAN EFFORTS. WE DO NOT HAVE DATA ON NUMBER OF SUBMISSIONS FOR IJCAI. KDD CUP 2015 WAS AN ONGOING COMPETITION AT THE TIME OF WRITING, SO THIS IS A SNAPSHOT FROM MAY 18TH, 2015.

Dataset	# Teams	% of Top Submission's Score	% of Teams Worse	# Submissions worse	# Days Spent on Worse Submissions
KDD cup 2014	473	86.5%	69.3%	3873	929
IJCAI	156	93.7%	32.7%	-	-
KDD cup 2015	277	95.7	85.6%	1319	377

First, the Data Science Machine can be used to set a benchmark. Just as the IJCAI organizers published a benchmark for competitors to use as reference, the Data Science Machine could be a performance reference for practicing data scientists. If the Data Science Machine performance is adequate for the purposes of the problem, no further work is necessary. This situation would save a lot of time if the dataset is similar to KDD15 or IJCAI where most gains are achieved by the Data Science Machine, and further human work has diminishing marginal return. Even in the case of KDD14 where further gains are significant, they appear to come at a high cost. Using the Data Science Machine could significantly lessen the time spent modeling predictive problems.

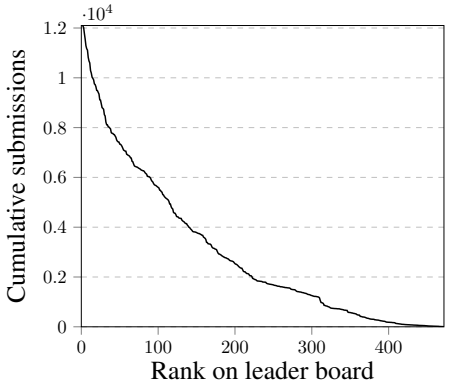

Fig. 9. The cumulative number of submissions made up to each leader board rank in KDD cup 2014. We can see the total number of submissions made by competitors increased exponentially as we move up the leader board.

Second, the Data Science Machine can engender creativity in data science. The Data Science Machine works by exploring a large space of potential features and models for a problem in an attempt to find the best one. Data scientists often face the problem of having more ideas to consider than available resources to test them. Rather than iterating on which features to create, the Data Science Machine could help with this problem by enumerating potential features and letting data scientists iterate on feature selection. In this way, a data scientist can start with the Data Science Machine's solution and then apply their expert knowledge to refine it.

VIII. RELATED WORK

The key components of the Data Science Machine are *Automated feature engineering*, *Working with related data* and *End-to-end system* from data to predictions. Next we present a list of related works that contain some of these components.

A. Automated feature engineering

Having recognized that feature engineering is the difference between success and failure, generalized feature extraction algorithms have been well studied for machine vision and natural language processing.

In machine vision, an early concept was Scale-Invariant Feature Transform (SIFT) [6]. These types of features can be successfully generalized for many problems and applications in machine vision, including object recognition and panorama stitching [7]. Other features such as histograms of oriented gradients [8] have performed well in other situations.

Similarly, in natural language processing, there are generalized features generation techniques such as "term frequency-inverse document frequency (tf-idf)", which is the ratio of how frequently a word shows up in a document to how often it shows up in the whole corpus of documents. This feature type is easy to calculate and performs well [9]. Others include modeling techniques like Latent Dirichlet Allocation (LDA), which transforms a corpus of documents into document-topic mappings [10]. LDA has proven to be general purpose enough to be useful in many document classification tasks, including spam filtering [11] and article recommendation [12].

Importantly, while we consider these generalized algorithms to be useful, they still have to be tuned for best performance. For example, in LDA, choosing the number of topics in the corpus is a challenging enough to encourage further research [10]. The importance of these algorithms is that they are techniques to generically capture information about type of data.

B. Working with related data

While the Data Science Machine focuses on data where we simply know that a relation between entities exists, the field of linked data strengthens these assumptions. In linked data [13], the data is structured such that it can be accessed with semantic queries. The field of automated feature generation for linked data is an active area of research.

Cheng et al. [14] developed an automated feature generation algorithm for data organized in a domain-specific knowledge base. This data is organized as entity-relationship-entity triples. For example, the Yago [15] knowledge base contains the following triple: `(Natalie Portman, hasActedIn, Black Swan)`. Using knowledge bases, the authors create a graph based language for generating features. In one example, they use natural language techniques to extract entities in tweets and relate them to entities in Yago. By relating tweets to entities, they can automatically generate new features with semantic meaning. Using this technique, they show some improvement in prediction results with their automatic semantic features. However, they conclude that further improvement will likely come from more domain-specific knowledge bases and information retrieval techniques.

This work, and other related systems [16], are limited to datasets that are structured in knowledge bases or can be mined for entities that are in knowledge bases. Many datasets, like the ones we present in this paper, do not fit these constraints, yet still have related entities.

C. End-to-end system

The Automatic Statistician project automatically models regression problems and produces reports readable by humans [17]. Their approach uses non-parametric Gaussian processes to model regression functions. They demonstrate their system performing well on 13 different time series datasets, suggesting generalizability. Their system is noteworthy in several regards. First, like the Data Science Machine, it focuses on an input of relatively untransformed data. Second, its approach, composing explanations one by one is similar to how a data scientist might analyze a time series to generate the overall model. Finally, they demonstrate generating a report to explain the resulting model so that the system can be understood by non-experts.

IX. Conclusion

We have presented the Data Science Machine: an end-to-end system for doing data science with relational data. At its core is *Deep Feature Synthesis*, an algorithm for automatically synthesizing features for machine learning. We demonstrated the expressiveness of the generated features on 3 datasets from different domains. By implementing an autotuning process, we optimize the whole pathway without human involvement, enabling it to generalize to different datasets. Overall, the system is competitive with human solutions for the datasets we tested on. We view this success as an indicator that the Data Science Machine has a role in the data science process.

A. Future work

We see the future direction of the Data Science Machine as a tool to empower data scientists. To this end, there are three important directions for future work. First, *Deep Feature Synthesis* itself has its own set of parameters that affect the resulting synthesized features. Future work could focus on selecting these parameters to improve performance and enhance the overall system's abilities. Right now, the system's approach doesn't involve much human interaction. In the future, the Data Science Machine could expose ways for humans to guide and interact with the system, enabling the pairing human and machine intelligence. Finally, the Data Science Machine was tested on 3 different datasets, but more datasets will help generalize and make the Data Science Machine the better tool for data science.

References

[1] P. Domingos, "A few useful things to know about machine learning," *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.

[2] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems*, 2012, pp. 2951–2959.

[3] C. E. Rasmussen, "Gaussian processes for machine learning," *the MIT Press*, 2006.

[4] E. Brochu, V. M. Cora, and N. de Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *CoRR*, vol. abs/1012.2599, 2010.

[5] A. Wilson and Z. Ghahramani, "Copula processes," in *Advances in Neural Information Processing Systems*, 2010, pp. 2460–2468.

[6] D. Lowe, "Object recognition from local scale-invariant features," in *The Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, 1999, pp. 1150–1157.

[7] M. Brown and D. Lowe, "Recognising panoramas," in *Proceedings. Ninth IEEE International Conference on Computer Vision*, vol. 2, Oct 2003, pp. 1218–1225.

[8] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 2005, pp. 886–893.

[9] A. Rajaraman and J. D. Ullman, "Data mining," in *Mining of Massive Datasets*. Cambridge University Press, 2011, pp. 1–17, cambridge Books Online.

[10] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.

[11] I. Bíró, J. Szabó, and A. A. Benczúr, "Latent dirichlet allocation in web spam filtering," in *Proceedings of the 4th International Workshop on Adversarial Information Retrieval on the Web*, ser. AIRWeb '08. New York, NY, USA: ACM, 2008, pp. 29–32.

[12] C. Wang and D. M. Blei, "Collaborative topic modeling for recommending scientific articles," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011, pp. 448–456.

[13] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data-the story so far," *Journal on Semantic Web and Information Systems*, 2009.

[14] W. Cheng, G. Kasneci, T. Graepel, D. Stern, and R. Herbrich, "Automated feature generation from structured knowledge," in *Proceedings of the 20th ACM international conference on Information and knowledge management*, 2011, pp. 1395–1404.

[15] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: a core of semantic knowledge," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 697–706.

[16] H. Paulheim and J. Fümkranz, "Unsupervised generation of data mining features from linked open data," in *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics*, ser. WIMS '12. New York, NY, USA: ACM, 2012, pp. 31:1–31:12.

[17] J. R. Lloyd, D. Duvenaud, R. Grosse, J. B. Tenenbaum, and Z. Ghahramani, "Automatic construction and Natural-Language description of nonparametric regression models," in *Association for the Advancement of Artificial Intelligence (AAAI)*, 2014.